Efficient Sort-Based Skyline Evaluation

ILARIA BARTOLINI, PAOLO CIACCIA, and MARCO PATELLA
Alma Mater Studiorum—Università di Bologna

Skyline queries compute the set of Pareto-optimal tuples in a relation, that is, those tuples that are not *dominated* by any other tuple in the same relation. Although several algorithms have been proposed for efficiently evaluating skyline queries, they either necessitate the relation to have been indexed or have to perform the dominance tests on *all* the tuples in order to determine the result. In this article we introduce SaLSa, a novel skyline algorithm that exploits the idea of presorting the input data so as to effectively *limit* the number of tuples to be read and compared. This makes SaLSa also attractive when skyline queries are executed on top of systems that do not understand skyline semantics, or when the skyline logic runs on clients with limited power and/or bandwidth. We prove that, if one considers symmetric sorting functions, the number of tuples to be read is minimized by sorting data according to a "minimum coordinate," minC, criterion, and that performance can be further improved if data distribution is known and an asymmetric sorting function is used. Experimental results obtained on synthetic and real datasets show that SaLSa consistently outperforms state-of-the-art sequential skyline algorithms and that its performance can be accurately predicted.

Categories and Subject Descriptors: H.2.4 [**Database Management**]: Systems—*Query processing*

General Terms: Algorithms, Performance

Additional Key Words and Phrases: Skyline query, Monotone functions

ACM Reference Format:

Bartolini, I., Ciaccia, P., and Patella, M. 2008. Efficient sort-based skyline evaluation. ACM Trans. Datab. Syst. 33, 4, Article 31 (November 2008), 49 pages. DOI = 10.1145/1412331.1412343. http://doi.acm.org/10.1145/1412331.1412343

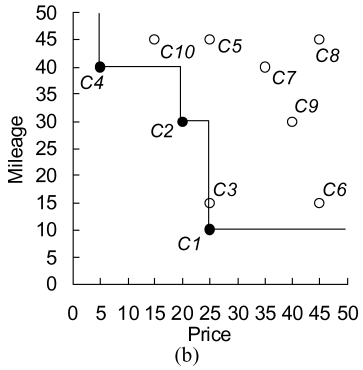
1. INTRODUCTION

Skyline queries are a popular and powerful paradigm for extracting interesting objects from a multidimensional dataset. Given a set r of d-dimensional objects (or *points*), the skyline of r is the set of Pareto-optimal, or *undominated*, points in r. According to the Pareto principle, a point $\mathbf{p}$ dominates point $\mathbf{p}_i$ if $\mathbf{p}$ is at least as good as $\mathbf{p}_i$ on all the attributes of interest, and strictly better than $\mathbf{p}_i$ on at least one attribute.

Authors' address: Alma Mater Studiorum, Università di Bologna, DEIS, Viale Risorgimento, 2 - 40136, Bologna, Italy; email: {i.bartolini, paolo.ciaccia, marco.patella}@unibo.it.

CarID	Price ($\times 10^3$)	Mileage ($\times 10^3$)
C1	25	10
C2	20	30
C3	25	15
C4	5	40
C5	25	45
C6	45	15
C7	35	40
C8	45	45
C9	40	30
C10	15	45

(a)



(b)

Fig. 1. (a) An instance of the UsedCars relation, and (b) its skyline over attributes Price and Mileage.

Example 1. Consider the relation UsedCars(CarID, Price, Mileage, ...) and the instance shown in Figure 1(a). A skyline query over the attributes Price and Mileage (both to be minimized) will return cars C1, C2, and C4, as also shown in Figure 1(b). For instance, C1 dominates C3, since the two cars have the same price but C1 has a lower mileage.

Skyline queries are a specific, yet relevant, example of preference queries [Chomicki 2003; Kießling 2002], and have been recognized as a useful and practical way to make database systems more flexible in supporting user requirements [Chaudhuri et al. 2006]. This has motivated the development of many algorithms for the efficient skyline evaluation on large datasets [Börzsönyi et al. 2001; Chomicki et al. 2003; Papadias et al. 2003; Godfrey et al. 2005]. Skyline algorithms, which we review in Section 2, can be broadly classified into two categories, depending on whether or not they consider that data are preprocessed. The first category, which includes all index-based methods, intuitively leads to superior performance, yet its applicability is limited by the necessity to have an indexed dataset. Generic (or *sequential*) skyline algorithms, on the other hand, do not require any preprocessing step and therefore can be applied to any input relation. The obvious problem with sequential algorithms is that they have to scan all the input data to compute the skyline. Nonetheless, as argued by Godfrey et al. [2005], designing a good sequential skyline algorithm is far from easy, since many factors, besides input scan, can ultimately affect the performance. Among these, it is the observation that computing a skyline is a CPU-intensive task, since many comparisons among points might be needed to determine the result.

In this article we consider the problem of sequential skyline evaluation and introduce a novel algorithm, called SaLSa (for *Sort and Limit Skyline algorithm*), which differs from other generic algorithms in that it consistently *limits* the number of points on which dominance tests need to be executed. The design of SaLSa is based on two key concepts: first, a sorting step of the input data and, second, the observation that, for suitably chosen sorting functions, it is indeed possible to compute the skyline by looking only at a (hopefully small)

prefix of the sorted input stream. While the idea of presorting is not new, since it is at the heart of the SFS algorithm by Chomicki et al. [2003], in that algorithm it was mainly advocated as a way to bring in the first positions those points that are likely to dominate many other points, thus leading to a reduction in the number of dominance tests. On the other hand, sorting data in SaLSa is mainly used as a means to stop fetching points from the input stream. In other terms, SaLSa relies on sorting functions that can guarantee that *all* points beyond a certain level in the input stream are dominated by some already seen point, which we conveniently call the *stop point*.

The limiting behavior of SaLSa makes it an ideal candidate when data are managed by a system that does not support skyline queries (e.g., current database servers and web data sources) and communication costs can therefore become a dominant cost factor. In scenarios like these, a nonlimiting algorithm running on the client side will be forced to fetch from the underlying data source *all* the data in the target relation. For instance, evaluating the query in Example 1 would require all the (presumably many) cars managed by the server to be transmitted to the client. It is evident that this strategy, no matter how smart the skyline algorithm is, will pay an excessive communication overhead, which is particularly true for clients with a limited bandwidth connection. In a situation like this, SaLSa would exploit the capability of the server to sort tuples and then would correctly determine the skyline by fetching only part of the whole dataset.

Example 2. Consider a wireless client (e.g., a PDA) connected to a server through an 802.11b network, in which the effective data transfer rate is 1Mb/sec (1 Megabit per second).[1] The client issues a three-dimensional skyline query over the NBA dataset, which consists of 17, 791 points, each of 100 bytes.[2] Just shipping all the points to the client would require about 13.6 seconds. For this query SaLSa is able to determine the 10 points in the skyline by reading only 3, 783 tuples, thus saving 79% of the transfer cost, i.e., about 10.7 seconds. Note that state-of-the-art algorithms can compute this skyline in less than 1 second, that is one order of magnitude faster.

This article is a substantially extended and revised version of Bartolini et al. [2006], in which the basic SaLSa concepts were introduced. Here we provide a more rigorous analysis of sorting functions and formally demonstrate that ordering points based on their *minimum coordinate* (minC) value is the optimal choice if one considers *symmetric* functions. We also introduce a novel variant of the minC function that aims to reduce the number of dominance tests. For the case of symmetric sorting functions, we present an original analytical cost model for predicting the number of points that the algorithm will have to fetch to compute the result, and show how the model can be applied when data distribution is represented through a multidimensional histogram. We then move on to consider the case in which an *asymmetric* function is used to sort points,

[1]The nominal 802.11b bandwidth is 11Mb/sec, but this assumes no traffic at all in the network. An effective 1Mb/sec transfer rate is a realistic estimate for the case of moderate traffic.
[2]This and all other datasets used in our experiments are described in Section 7.

and introduce a sorting function, called $\Delta\mathsf{minC}_{opt}$, that is provably optimal over all sorting functions and for all input relations. Since $\Delta\mathsf{minC}_{opt}$ has a purely theoretical interest, being based on detailed information that cannot be available in realistic scenarios, we show how histograms can be exploited to choose, on-the-fly, a good asymmetric function to use for a given dataset. An analytical model that provides a limit to SaLSa performance completes our investigation on asymmetric functions. The experimental results we obtain on both synthetic and real datasets confirm that SaLSa is indeed highly effective in limiting the number of points to be read, and that the model's estimates are highly accurate.

The rest of the article is organized as follows. In Section 2, we review the semantics of a skyline query and algorithms for its evaluation. Section 3 introduces the SaLSa algorithm, and Section 4 analyzes in detail specific sorting functions. Section 5 presents the analytical cost model for predicting SaLSa performance. In Section 6, we consider the case of asymmetric sorting functions and prove a basic fact about the optimal theoretical performance of SaLSa. Section 7 presents experimental results. Before concluding, in Section 8 we detail how the basic principles of SaLSa can be extended so as to use *multiple* stop points, rather than a single one.

Some of the proofs in the article are included in the Electronic Appendix that is accessible in the ACM Digital Library.

2. SKYLINE ALGORITHMS

This section reviews the definition of skyline and existing algorithms for its evaluation.

Let $R(A_1, \ldots, A_d)$ be a relation schema, and let $dom(A_j)$ be the domain of attribute A_j. Further, let $\mathcal{D} = dom(A_1) \times \cdots \times dom(A_d)$. In this article, we assume that each domain $dom(A_j)$ is linearly ordered. A relation r over R is a set of n tuples, or d-dimensional *points*, from $\mathcal{D}$, that is, $r = \{\mathbf{p}_1, \ldots, \mathbf{p}_n\}$, where $\mathbf{p}_i = (p_i[1], \ldots, p_i[d])$, and $p_i[j]$ denotes the value of attribute A_j in $\mathbf{p}_i$, that is $p_i[j] \equiv \mathbf{p}_i.A_j$. In the following, the terms tuple and point will be used interchangeably.

Definition 1. Without loss of generality, assume that on each attribute lower values are better. Then, point $\mathbf{p}$ *dominates* point $\mathbf{p}_i$, written $\mathbf{p} \succ \mathbf{p}_i$, if and only if the following holds:

$$\mathbf{p} \succ \mathbf{p}_i \Leftrightarrow \left(\bigwedge_{j=1}^{d} p[j] \leq p_i[j]\right) \wedge \left(\bigvee_{j=1}^{d} p[j] < p_i[j]\right). \tag{1}$$

The *skyline* $\mathcal{S}(r)$ of r is the subset of points in r that are undominated, or Pareto-optimal:[3]

$$\mathcal{S}(r) = \{\mathbf{p} \in r : \nexists \mathbf{p}_i \in r : \mathbf{p}_i \succ \mathbf{p}\}. \tag{2}$$

[3]For simplicity, we consider that all attributes are involved in the skyline. Generalization to the case where R also includes nonskyline attributes is immediate.

Thus a point $\mathbf{p}$ belongs to the skyline of r if and only if there is no other point $\mathbf{p}_i \in r$ that is not worse than $\mathbf{p}$ on all attributes (or *coordinates*) and strictly better than $\mathbf{p}$ on at least one attribute.

Computing the skyline is equivalent to determining the *maxima* of a set of vectors, a well-known problem in computational geometry [Preparata and Shamos 1985]. However, algorithms developed in that field, and based on a *divide and conquer* approach, cannot be directly applied in a database scenario, since they do not take into account main memory limitations that prevent the whole dataset being loaded before the actual skyline computation starts. This was observed in [Börzsönyi et al. 2001], where a divide and conquer algorithm, D&C, adapted to work with external memory was proposed. However, a recent analysis by Godfrey et al. [2005] shows that the average performance of D&C deteriorates with increasing dimensionality, d. As Godfrey et al. [2005] argue, the problem with divide and conquer algorithms is that, if the dataset gets partitioned into smaller parts and *local* skylines are computed for each of them, the chance of early discarding nonskyline points reduces. This is because, if the size of the skyline is a sublinear function of the dataset size (it is $\Theta((\ln n)^{d-1}/(d-1)!)$ when attributes are independent of each other [Buchta 1989; Godfrey 2004]), then the relative size of the skyline grows in smaller datasets (and consequently the number of dominated points diminishes).

Börzsönyi et al. [2001] introduced another algorithm, called *Block Nested Loops* (BNL), for sequential skyline evaluation. BNL allocates in main memory a *window* W and sequentially scans the input relation. When a point $\mathbf{p}$ is read, it is compared to points in W. If $\mathbf{p}$ is dominated by a point in W, then $\mathbf{p}$ is discarded, otherwise $\mathbf{p}$ is inserted in W. If $\mathbf{p}$ dominates some points in W, these are removed from W. In case the window saturates, a temporary file is used to store points that cannot be placed in W. This file is used as the input to the next pass. Eventually the algorithm terminates, since at the end of each pass the size of the temporary file can only decrease.

The SFS (*Sort Filter Skyline*) algorithm by Chomicki et al. [2003] improves over BNL by first sorting the input data using a *monotone* function $\mathcal{M}$. This guarantees that if $\mathcal{M}(\mathbf{p}) \leq \mathcal{M}(\mathbf{p}_i)$, then $\mathbf{p}_i$ will not dominate $\mathbf{p}$, written $\mathbf{p}_i \not\prec \mathbf{p}$. In other terms, using a monotone function corresponds to performing a *topological sort* with respect to the Pareto dominance criterion. Similarly to BNL, SFS keeps in W the undominated points seen so far. However, the monotonicity of $\mathcal{M}$ now guarantees that in the filter phase a new point $\mathbf{p}_i$ will never dominate an already seen point $\mathbf{p}$, thus a point will never be dropped from the window. This leads to three major improvements with respect to BNL: 1) The management of W largely simplifies, 2) points in the skyline can be *progressively* returned without having to wait for all the input to be read, and 3) the number of passes of the filter phase is optimal, that is, $\lceil |\mathcal{S}|/|W| \rceil$. From the last observation it follows that, even for moderately large skylines, SFS will likely complete in a single pass. Experimental results in Chomicki et al. [2003] indeed show that SFS runs (much) faster than BNL, and that it executes fewer dominance tests. In particular, this is achieved by sorting data using their *volume* or, equivalently, their *entropy* (see Section 4 for details).

LESS [Godfrey et al. 2005] is an improvement of SFS that integrates, in the first step of a standard external sort-merge algorithm, an *elimination-filter* window, so as to earlier discard some dominated tuples. Further, LESS combines the last merge pass of the sorting algorithm with the first skyline-filter pass.

These algorithms are *generic*, in the sense that they do not require any specialized access structure to compute the skyline and can therefore be applied even when r is the result of some other operation (e.g., a selection on a nonskyline attribute or a join). On the other hand, algorithms like NN [Kossmann et al. 2002], BBS [Papadias et al. 2003], and Index [Tan et al. 2001], which rely on the existence of an index over the dataset, or Bitmap [Tan et al. 2001], which encodes all points' coordinates using a bitmap structure, can only be applied to stored relations.

Finally, it is worth mentioning skyline algorithms based on a *distributed* access model [Balke et al. 2004; Bartolini et al. 2004; Bartolini et al. 2007]. Such algorithms work by querying d independent subsystems. Each subsystem is in charge of a specific skyline attribute and is able to return objects ordered according to the preference on that attribute (e.g., minimize the price). By iterating on the streams of incoming results, objects that are returned by at least one subsystem are candidates to be part of the skyline. Missing attribute values for candidate objects, needed to perform the dominance tests, are obtained through random accesses to the subsystems using object identifiers. Finally, each algorithm exploits a specific stopping condition to terminate the search, so as to avoid a full scan of the d streams.

3. THE SALSA ALGORITHM

The algorithm we introduce, called SaLSa, is a generic skyline algorithm that builds on the basic idea that, if the input relation r is sorted according to a suitably chosen monotone function, then it is possible to determine the skyline of r *without applying the skyline filter to all the points*. In general, this might drastically reduce the number of points to be read and, depending on the specific instance and sorting function, it might reduce the number of dominance tests as well. Since SaLSa shares with SFS the idea of presorting the input relation, it also keeps all the SFS strengths: simplified management of the window, incremental delivery of results, and optimal number of passes of the filter phase.

In the following, we describe how SaLSa works when a single pass is sufficient to complete the evaluation, that is, the skyline of r fits into main memory. Extension to the case where the skyline size exceeds the available main memory is managed as in BNL and SFS, and not reported here.

SaLSa, whose general logic is described in Algorithm 1, takes as input a stream of points r that has been sorted using a monotone function $\mathcal{M}$, that is, a function such that $\mathcal{M}(\mathbf{p}) \leq \mathcal{M}(\mathbf{p}_i)$ implies $\mathbf{p}_i \not\prec \mathbf{p}$. The specific choice of $\mathcal{M}$ does not influence the correctness of SaLSa but only its performance, as will be extensively discussed in the following. Denote with u the suffix of r that, at any given step, has not been processed yet by SaLSa. Each time a new point $\mathbf{p}$ is

Algorithm 1. SaLSa[$\mathcal{M}$]

Input: input stream r sorted using a monotone function $\mathcal{M}$
Output: the skyline $\mathcal{S}(r)$ of r

1: $\mathcal{S} \leftarrow \emptyset$, $stop \leftarrow false$, $\mathbf{p}_{stop} \leftarrow$ undefined, $u \leftarrow r$
2: **while** not $stop \wedge u \neq \emptyset$ **do**
3: $\quad \mathbf{p} \leftarrow$ get next point from u, $u \leftarrow u \setminus \{\mathbf{p}\}$
4: $\quad$ **if** $\mathcal{S} \not\succ \mathbf{p}$ **then** $\mathcal{S} \leftarrow \mathcal{S} \cup \{\mathbf{p}\}$, update $\mathbf{p}_{stop}$
5: $\quad$ **if** $\mathbf{p}_{stop} \succ u$ **then** $stop \leftarrow true$
6: return $\mathcal{S}$

read from u, $\mathbf{p}$ is compared against the current skyline $\mathcal{S}$. If none of the points in $\mathcal{S}$ dominates $\mathbf{p}$ (i.e., $\mathcal{S} \not\succ \mathbf{p}$), $\mathbf{p}$ is inserted into $\mathcal{S}$. This might possibly trigger the update of the so-called *stop point*, $\mathbf{p}_{stop}$ (step 4). At step 5 SaLSa checks if it has gained sufficient evidence to conclude that no further point in u can be part of the skyline, that is, all points in u are dominated by $\mathbf{p}_{stop}$ ($\mathbf{p}_{stop} \succ u$). If this is so, the algorithm can correctly terminate, returning $\mathcal{S}$ as result.

Two factors ultimately determine the actual performance of SaLSa: the choice of the sorting function, and the strategy for choosing the stop point. Before analyzing both issues, we introduce some basic terminology.

Assume that during the execution of SaLSa the last point $\mathbf{p}$ read so far has value $\mathcal{M}(\mathbf{p}) = l$. We say that $\mathcal{M}$ is at *level* l after having read $\mathbf{p}$, and denote with $u(\mathcal{M}, l)$ the set of unread points at this stage of the execution. Note that $\mathcal{M}(\mathbf{p}_i) \geq l$ holds for each $\mathbf{p}_i \in u(\mathcal{M}, l)$. To safely stop the execution, one should guarantee that $\mathbf{p}_{stop} \succ \mathbf{p}_i$ holds for all such points. This is actually done in SaLSa by considering the *unread domain* at level l, defined as:

$$\mathcal{D}(\mathcal{M}, l) = \{\mathbf{p}_i \in \mathcal{D} : \mathcal{M}(\mathbf{p}_i) \geq l\}. \tag{3}$$

Clearly, it is $u(\mathcal{M}, l) \subseteq \mathcal{D}(\mathcal{M}, l)$. Note that $\mathcal{D}(\mathcal{M}, l)$, unlike $u(\mathcal{M}, l)$, does not depend on the specific input relation r. Then, SaLSa can safely stop if and only if the following is true:

$$\forall \mathbf{p}_i \in \mathcal{D}(\mathcal{M}, l) : \mathbf{p}_{stop} \succ \mathbf{p}_i. \tag{4}$$

We denote with $l_{stop}(\mathcal{M})$ the level at which SaLSa eventually halts, also called the *stop level of* $\mathcal{M}$. Similarly, $u_{stop}(\mathcal{M})$ and $\mathcal{D}_{stop}(\mathcal{M})$ are the values of $u(\mathcal{M}, l)$ and $\mathcal{D}(\mathcal{M}, l)$, respectively, when the algorithm terminates.

Example 3. Consider the UsedCars relation introduced in Example 1, and assume that points are ordered by using the sum of their Price and Mileage values, $\mathcal{M}(\mathbf{p}) = \mathbf{p}.\text{Price} + \mathbf{p}.\text{Mileage}$. The first point to be read is car C1, which is at level $l = 25 + 10 = 35$. At this stage of execution the unread domain corresponds to the shaded region in Figure 2(a). Figure 2(b) shows the unread domain after the last skyline point, C2, has been read, whose level is $l = 20 + 30 = 50$.

Clearly, although SaLSa can be applied with any monotone function, only *limiting* sorting functions are worth considering. We say that function $\mathcal{M}$ limits

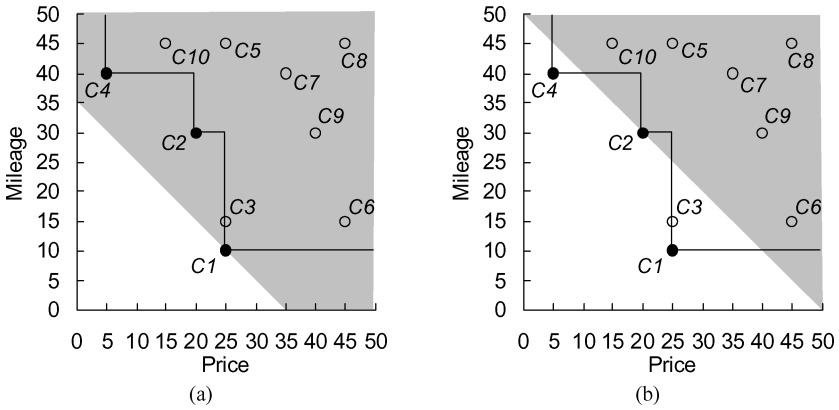


Fig. 2. Points are sorted using the function $\mathcal{M}(\mathbf{p}) = \mathbf{p}.\text{Price} + \mathbf{p}.\text{Mileage}$. The shaded region is the unread domain after points C1 (a) and C2 (b) have been read, respectively.

a relation r when $u_{stop}(\mathcal{M}) \neq \emptyset$, and that $\mathcal{M}$ is limiting if there exists at least a relation r for which $u_{stop}(\mathcal{M})$ is not empty. Note that this is the same as saying that $\mathcal{D}_{stop}(\mathcal{M})$ is not always empty.

We start by focusing on a well-defined class of functions that exhibit the key property of *symmetry*. Section 6 extends the analysis to also include asymmetric functions.

Definition 2. A function $\mathcal{M}$ on d variables is *symmetric* if and only if $\mathcal{M}(x_1, \ldots, x_d) = \mathcal{M}(x_{\pi(1)}, \ldots, x_{\pi(d)})$ for any permutation π of $\{1, \ldots, d\}$, that is, the value of $\mathcal{M}$ is invariant under any rearrangement of its variables.

Intuitively, a symmetric function $\mathcal{M}$ does not privilege any attribute over the others. This seems to be a very natural requirement if one uses $\mathcal{M}$ for computing the skyline, since by definition all skyline attributes are equally important.

3.1 Stopping SaLSa Execution

A key factor affecting the performance of SaLSa is the choice of the stop point, $\mathbf{p}_{stop}$. In turn, this is tightly related to how the stop condition, as expressed by Equation 4, can be effectively checked. In order to avoid unnecessary complications, in the sequel we will consider that attributes' values are normalized, namely for all j it is $dom(A_j) = [0, 1]$. We will deviate from this assumption only to illustrate how our results apply to the UsedCars relation, that will be used as a driving example throughout the article.

We start with a basic fact about symmetric monotone functions.

LEMMA 1. *Let $\mathcal{M}$ be a symmetric monotone function and l be any value of $\mathcal{M}$. Define Low to be the minimum value such that $\mathcal{M}(\mathsf{Low}, 1, 1, \ldots, 1) \geq l$ holds. If no such finite Low exists, then set Low $= 0$. Then, there is no point $\mathbf{p}$ for which both the following are true: 1) $\mathcal{M}(\mathbf{p}) \geq l$, and 2) there exists j such that $p[j] <$ Low.*

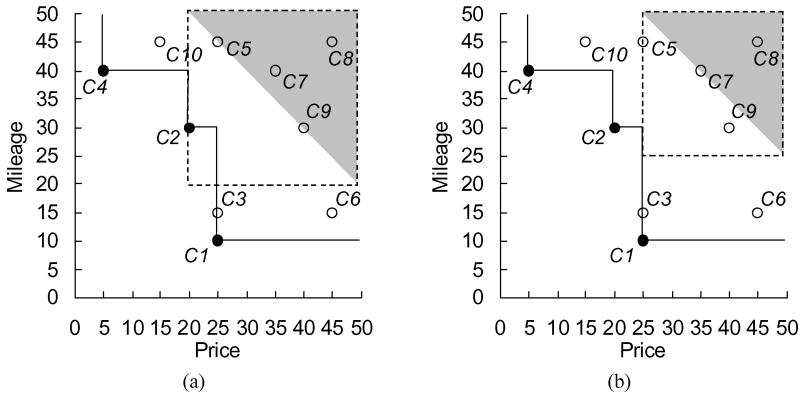


Fig. 3. Illustrations of Lemma 1 (a) and Theorem 1 (b). See Examples 4 and 5 for details, respectively.

PROOF. Clearly, the result applies when $\mathsf{Low} = 0$, since for all j it is $dom(A_j) = [0, 1]$. When $\mathsf{Low} > 0$, assume that 2) holds. Since $\mathcal{M}$ is symmetric, it is possible to arbitrarily choose the attribute A_j for which it is $p[j] < \mathsf{Low}$. Then, let $j = 1$, and consider the point $\mathbf{p}' = (p[1], 1, 1, \ldots, 1)$, that is, $p'[1] = p[1] < \mathsf{Low}$ and, for all $j \neq 1$, it is $p'[j] \geq p[j]$. By monotonicity of $\mathcal{M}$, it is $\mathcal{M}(\mathbf{p}') \geq \mathcal{M}(\mathbf{p})$, where equality can only occur if $\mathbf{p}' = \mathbf{p}$. If $\mathcal{M}(\mathbf{p}) \geq l$ also holds, it is therefore $\mathcal{M}(\mathbf{p}') \geq l$. This contradicts the hypothesis that Low is the minimum value such that $\mathcal{M}(\mathsf{Low}, 1, 1, \ldots, 1) \geq l$ holds. □

Intuitively, this lemma says that for all symmetric monotone functions, $\mathcal{D}(\mathcal{M}, l)$ is included in the hypercube whose opposite vertices are the points $(\mathsf{Low}, \ldots, \mathsf{Low})$ and $\mathbf{1} = (1, \ldots, 1)$, where Low is as in the lemma definition.

Example 4. Assume that both Price and Mileage attributes have values in the range $[0, 50]$ and that the symmetric function used to sort the UsedCars relation is $\mathcal{M}(\mathbf{p}) = \mathbf{p}.\text{Price} + \mathbf{p}.\text{Mileage}$. After point C5 has been read, the level of this function is $l = 25 + 45 = 70$. The minimum value of Low such that $\mathcal{M}(\mathsf{Low}, 50) \geq 70$ is $\mathsf{Low} = 20$. Thus, according to Lemma 1, all points that SaLSa will read after C5 will lie in the hypercube (drawn with a dashed line in Figure 3(a)) whose opposite vertices are the points $(20, 20)$ and $(50, 50)$.

Lemma 1 provides an effective way to determine when SaLSa can be stopped. In the following, we denote with $p^+ = \max_j\{p[j]\}$ the maximum coordinate value of a point $\mathbf{p}$.

THEOREM 1. *Let $\mathcal{M}$ be any symmetric monotone function, and assume $\mathcal{M}$ reaches level l at a certain stage of the execution. Let Low be defined as in Lemma 1. Then, SaLSa can be stopped if and only if the maximum coordinate value of the stop point $\mathbf{p}_{stop}$ satisfies $p^+_{stop} \leq \mathsf{Low}$. Only if $p_{stop}[j] = p^+_{stop}$ holds for all j, then SaLSa should also read all (if any) $\mathbf{p}_{stop}$'s duplicates.*

PROOF. (If) From Lemma 1 we know that no point $\mathbf{p}$, such that $\mathcal{M}(\mathbf{p}) \geq l$, has an attribute value lower than Low. This is sufficient to conclude that either $\mathbf{p}$ is dominated by $\mathbf{p}_{stop}$ or $\mathbf{p} = \mathbf{p}_{stop}$.

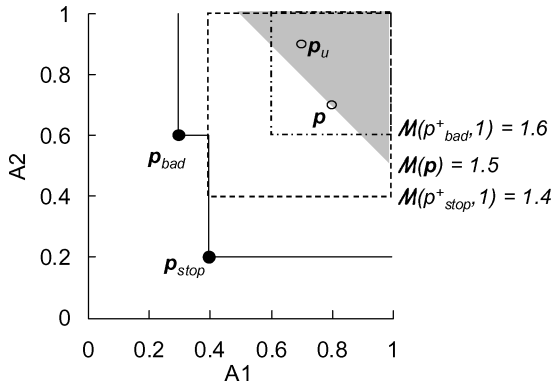


Fig. 4. Illustration of the proof of Theorem 2. The sorting function is $\mathcal{M}(\mathbf{p}_i) = \mathbf{p}_i.A_1 + \mathbf{p}_i.A_2$.

(Only if) Assume that SaLSa is halted when $p^+_{stop} > \mathsf{Low}$. Let A_j be any attribute such that $p_{stop}[j] = p^+_{stop}$. Lemma 1 says that the unread domain at level l includes a point $\mathbf{p}$ such that $p[j] = \mathsf{Low}$, thus $\mathbf{p}$ is not dominated by $\mathbf{p}_{stop}$. □

This theorem shows how SaLSa can be stopped just by looking at the level of the function $\mathcal{M}$ and at the current stop point. Further, it immediately leads to a simple rule for optimally choosing the stop point, which is independent of the specific function used to sort the points. We first define in which sense a rule for choosing the stop point is optimal.

Definition 3. Given a monotone function $\mathcal{M}$, a rule for choosing the stop point is optimal for $\mathcal{M}$ if and only if, for any relation r, there is no other rule that allows SaLSa to stop earlier when points are ordered using $\mathcal{M}$.

THEOREM 2. *The following* MiniMax *rule for choosing the stop point is optimal for* any *symmetric monotone function* $\mathcal{M}$*:*

$$\mathbf{p}_{stop} = \arg\min_{\mathbf{p}_i \in \mathcal{S}}\{p^+_i\}. \tag{5}$$

PROOF. Consider another rule that eventually chooses as stop point $\mathbf{p}_{bad} \in \mathcal{S}$, such that $p^+_{bad} > p^+_{stop}$, where $\mathbf{p}_{stop}$ is as in Equation 5. From Theorem 1 we have that if SaLSa stops at level l with $\mathbf{p}_{bad}$, then so it does with $\mathbf{p}_{stop}$. Further, for any limiting function $\mathcal{M}$, we can show that there exists a relation r on which SaLSa, using $\mathbf{p}_{stop}$, reads strictly fewer points than when using $\mathbf{p}_{bad}$. The relation includes four points: $\mathbf{p}_{stop}$, $\mathbf{p}_{bad}$, $\mathbf{p}$, and $\mathbf{p}_u$ (refer to Figure 4 for an example). The skyline consists of points $\mathbf{p}_{stop}$ and $\mathbf{p}_{bad}$, each of which dominates both $\mathbf{p}$ and $\mathbf{p}_u$. Further, it is $\mathcal{M}(\mathbf{p}) < \mathcal{M}(\mathbf{p}_u)$, that is, $\mathbf{p}$ would be read before $\mathbf{p}_u$. Since $\mathbf{p}_{stop} \succ \mathbf{p}$ and $\mathbf{p}_{bad} \succ \mathbf{p}$, the first two points that SaLSa reads are $\mathbf{p}_{stop}$ and $\mathbf{p}_{bad}$ (the actual order is immaterial). Point $\mathbf{p}$ is chosen so that $\mathcal{M}(p^+_{bad}, 1, 1, \ldots, 1) > \mathcal{M}(\mathbf{p}) > \mathcal{M}(p^+_{stop}, 1, 1, \ldots, 1)$. Note that this is always possible if $p^+_{bad} > p^+_{stop}$ and does not contradict the hypothesis that $\mathbf{p}_{bad}$ dominates $\mathbf{p}$. Setting $\mathcal{M}(\mathbf{p}) = \mathcal{M}(\mathsf{Low}, 1, 1, \ldots, 1)$, we have that $p^+_{stop} < \mathsf{Low} < p^+_{bad}$. This is sufficient to conclude that SaLSa using $\mathbf{p}_{stop}$, can halt just after reading $\mathbf{p}$, whereas this would not be possible if $\mathbf{p}_{bad}$ were used. □

Example 5. For the UsedCars relation, the MiniMax rule leads to choosing C1 as stop point, for which it is $C1^+ = 25$. If UsedCars is sorted using the function $\mathcal{M}(\mathbf{p}) = \mathbf{p}.\text{Price} + \mathbf{p}.\text{Mileage}$, then SaLSa can be halted as soon as it reads point C7, whose level is $l = 75$ (see Figure 3(b)). Indeed, the minimum value of Low that satisfies the equation $\mathcal{M}(\text{Low}, 50) \geq 75$ is $25 \geq C1^+$, which is the stop condition established by Theorem 1.

Theorem 2 provides an efficient, $\mathcal{O}(1)$, method for incrementally updating the stop point. Let $\mathbf{p}_{stop}$ be the current stop point. When a new point $\mathbf{p}$ is added to the skyline, the stop point either remains unchanged or it is set to $\mathbf{p}$. This only depends on which among p^+ and p^+_{stop} is minimum.

4. PROPERTIES OF SOME SYMMETRIC SORTING FUNCTIONS

Although Theorem 2 shows that the stop point can be chosen independently of the sorting function, this does not mean that all functions will behave equally well. In the following we consider some relevant cases and arguments about their limiting capabilities. To avoid making the problem trivial to solve, we consider only skylines over at least two dimensions, that is, $d \geq 2$, and exclude from the analysis those (unrealistic) instances that include the target (ideal) point $\mathbf{0} = (0, \ldots, 0)$. The latter implies that it will always be $p^+_{stop} > 0$.

4.1 Sorting by Volume

The first symmetric function we consider is the one based on the *volume* of the points, as originally proposed by the authors of SFS in Chomicki et al. [2002]:[4]

$$\text{vol}[0](\mathbf{p}) = 1 - \prod_{j=1}^{d}(1 - p[j]).$$

We will shortly explain why we denote the function as vol[0] rather than simply as vol. The rationale of using vol[0], which also justifies its name, is that $\prod_{j=1}^{d}(1-p[j])$ is the volume of the *dominance region* of $\mathbf{p}$, that is, the set of points in $[0, 1]^d$ dominated by $\mathbf{p}$. If points are uniformly distributed over $\mathcal{D}$, then this also corresponds to the expected fraction of points in r that $\mathbf{p}$ dominates. Then, fetching first points with higher volume (thus, lower values of vol[0]) increases the chance of early discarding many other points, thus reducing the overall number of comparisons. Unfortunately, the following result shows that vol[0] is not a good choice for limiting the number of points to be read.

LEMMA 2. *The* vol[0] *sorting function is not limiting.*

PROOF. The range of vol[0] is the $[0, 1]$ interval. For any value of l, it is $\text{Low} = 0$, thus $\mathcal{D}(\text{vol}[0], l)$ always includes points that are minimal on one or more attributes, and could therefore be in the skyline. □

[4]Since we assume that attributes have to be minimized, rather than maximized as Chomicki et al. [2002] do, the original definition, $\prod_{j=1}^{d} p[j]$, has been accordingly modified.

Note that vol[0], besides being nonlimiting, is truly monotone only if one considers points whose level is strictly less than 1, since $\mathsf{vol}[0](\mathbf{p}) = \mathsf{vol}[0](\mathbf{p}') = 1$ does not rule out the possibility that $\mathbf{p} \succ \mathbf{p}'$.

It is possible to modify the volume function, so as to make it limiting, by avoiding that factors of the product vanish when $p[j] = 1$. To this end, consider the function $\mathsf{vol}[1](\mathbf{p}) = 2^d - \prod_{j=1}^{d}(2 - p[j])$, whose range is $[0, 2^d - 1]$. If we have, say, a two-dimensional skyline, now there is a chance of halting SaLSa execution as soon as the level of vol[1] reaches a value $l > 2^2 - 2 = 2$. This is because $l > 2$ guarantees that no point with minimal values either on A_1 or on A_2 is still unread. Such arguments are generalized as follows.

LEMMA 3. *Consider the function:*

$$\mathsf{vol}[m](\mathbf{p}) = (m+1)^d - \prod_{j=1}^{d}(m + 1 - p[j]) \quad (m > 0),$$

whose range is $[0, (m+1)^d - m^d]$. *Then,* $l_{stop}(\mathsf{vol}[m]) > (m+1)^d - (m+1)m^{d-1}$ *for any relation* r.

PROOF. For any value of $l \leq (m+1)^d - (m+1)m^{d-1}$, it is Low $= 0$. On the other hand, if $l > (m+1)^d - (m+1)m^{d-1}$, then Low > 0. □

From a geometric point of view, each vertex $\mathbf{v}$ of the hypercube $[0, 1]^d$ with coordinates $(1, \ldots, 1, 0, 1, \ldots, 1)$, for which it is $\mathsf{vol}[m](\mathbf{v}) = (m+1)^d - (m+1)m^{d-1}$, needs to be excluded from $\mathcal{D}(\mathsf{vol}[m], l)$.

From Lemma 3 the following is derived.

COROLLARY 1. *For any dimensionality* d, *the maximum of* $\mathcal{D}_{stop}(\mathsf{vol}[m])$ *monotonically increases with* m.

PROOF. Lemma 3 shows that $\mathcal{D}_{stop}(\mathsf{vol}[m])$ can at most equal the open set defined as $\{\mathbf{p} \in \mathcal{D} : \mathsf{vol}[m](\mathbf{p}) > (m+1)^d - (m+1)m^{d-1}\}$, that is, $\{\mathbf{p} \in \mathcal{D} : \prod_{j=1}^{d}(m+1-p[j]) < (m+1)m^{d-1}\}$. Consider now the midpoint $\mathbf{p}$ of the closure of the above set, that is, $\mathbf{p}$ lies on the main diagonal of the hypercube and $\mathsf{vol}[m](\mathbf{p}) = (m+1)^d - (m+1)m^{d-1}$. For each attribute A_j, it is derived that $p[j] = 1 - m(\sqrt[d]{(m+1)/m} - 1)$. This monotonically decreases with m, which is also to say that $\mathbf{p}$ moves away from the worst possible point, that is, $\mathbf{1} = (1, \ldots, 1)$. □

We conclude by observing that the *entropy* function, $E(\mathbf{p}) = \sum_{j=1}^{d} \log(p[j] + 1)$, also used by the SFS method, indeed corresponds to vol[1]. Since SFS assumes that higher attribute values are better, and fetches points by decreasing values of E, to prove the equivalence we need to complement attribute values, that is, $p[j] = 1 - p[j]$, and change the sign of E. This yields $-\sum_{j=1}^{d} \log(2 - p[j])$. Clearly, this leads to order points as $-\exp(\sum_{j=1}^{d} \log(2 - p[j])) = -\prod_{j=1}^{d}(2 - p[j])$, which equals vol[1] modulo a constant term.[5]

[5]Chomicki et al. [2002] erroneously assert that E yields the same order as vol[0], which is not true, since vol[1] and vol[0] do not order points in the same way.

4.2 Sorting by Sum of Coordinates

An obvious alternative to vol is to sum (rather than to multiply) attribute values, that is:

$$\mathsf{sum}(\mathbf{p}) = \sum_{j=1}^{d} p[j].$$

There is a simple connection between sum and the vol$[m]$ family.

LEMMA 4. *The* sum *sorting function is strictly more limiting than* vol$[m]$*, for any finite* m*, with* $l_{stop}(\mathsf{sum}) > d - 1$.

PROOF. Since vol$[m](\mathbf{p})$ is a polynomial in $(m+1)$ of degree $d-1$, where the coefficient of $(m+1)^{d-1}$ is $\sum_{j=1}^{d} p[j]$, we have that $\lim_{m\to\infty} \mathsf{vol}[m](\mathbf{p})/(m+1)^{d-1} = \mathsf{sum}(\mathbf{p})$. Due to Corollary 1, this proves the first part of the lemma. That $l_{stop}(\mathsf{sum}) > d-1$, can be proved with similar arguments. Alternatively, from Theorem 1 we have that SaLSa using sum can be halted as soon as it is $p^{+}_{stop} \leq \mathsf{Low}$, where now Low is the solution of $\mathsf{Low} + (d-1) = l$. Since $p^{+}_{stop} > 0$, it follows that $l_{stop}(\mathsf{sum}) > d-1$, as required. □

4.3 Minimum Coordinate Sort

It is clear that many (infinite) limiting functions exist. It is therefore natural to ask if there is an optimal function that can limit any input relation r more than other functions do. The answer is affirmative, and such optimal function, which we call minC (for *minimum coordinate*), is defined as:

$$\mathsf{minC}(\mathbf{p}) = (\min_{j}\{p[j]\}, \mathsf{sum}(\mathbf{p})).$$

Therefore, minC first sorts points considering for each of them their minimum coordinate value. Then, a sum of the skyline attributes is used in order to guarantee monotonicity in case of ties. To this end other tie-breaking rules could in principle be used (e.g., any vol$[m]$ function).

Clearly, minC is limiting, since SaLSa can stop as soon as it fetches a point $\mathbf{p}$ for which it is $\min_j\{p[j]\} \geq p^{+}_{stop}$.

THEOREM 3. *For any relation* r *and symmetric sorting function* $\mathcal{M}$ *different from* minC*, it is* $\mathcal{D}_{stop}(\mathcal{M}) \subset \mathcal{D}_{stop}(\mathsf{minC})$*, thus* $u_{stop}(\mathcal{M}) \subseteq u_{stop}(\mathsf{minC})$.

PROOF. According to Theorem 1, when SaLSa halts, minimal attribute values in the unread domain $\mathcal{D}_{stop}(\mathcal{M})$ are equal to a value, $\mathsf{Low} \geq p^{+}_{stop}$, independent of $\mathcal{M}$, and Theorem 2 guarantees that p^{+}_{stop} is also independent of $\mathcal{M}$. Further, Lemma 1 shows that $\mathcal{D}_{stop}(\mathcal{M}) \subseteq [\mathsf{Low}, 1]^d$, where equality is attained only by minC. The second part of the theorem immediately follows. Here the equality case has also to be considered, since it is possible that r has no point in $\mathcal{D}_{stop}(\mathsf{minC}) \setminus \mathcal{D}_{stop}(\mathcal{M})$. □

Intuitively, the optimality of minC stems from the observation that, if one considers the point $\mathbf{p}_{\text{diag}}$ that lies on the main diagonal of the data space and

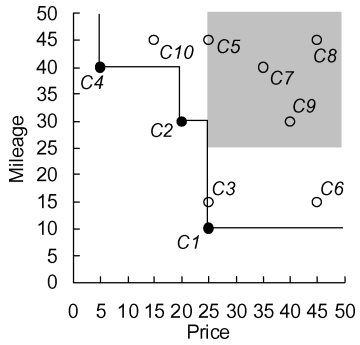


Fig. 5. Points are sorted using the optimal minC function, which leads to prune points C7, C8, and C9. The stop point is C1.

whose coordinates equal p^+_{stop}, that is, $\mathbf{p}_{\text{diag}} = (p^+_{stop}, \ldots, p^+_{stop})$, then it is true that $\mathbf{p}_{\text{diag}} \succ \mathbf{p}$ whenever $\text{minC}(\mathbf{p}) > \text{minC}(\mathbf{p}_{\text{diag}})$, whereas this does not hold for any other symmetric monotone function.

The reader may have noticed that minC is not a scalar function, since it yields a *pair* of reals. However, unless otherwise specified, we slightly abuse notation and terminology by treating $\text{minC}(\mathbf{p})$ as a real value, given by the primary sorting criterion of minC, that is, $\min_j\{p[j]\}$, and refer to this real value as the level of point $\mathbf{p}$. This has no practical consequences and has the advantage of keeping the presentation simple and of avoiding unnecessary formal complications.

Example 6. When the minC function is applied to the UsedCars relation, for which the stop point is C1, SaLSa can be halted after reading point C5, whose level is $l = 25 \geq \text{C1}^+$. The unread domain is shown as a shaded region in Figure 5.

Algorithm 2 describes SaLSa when the minC sorting function is used. With respect to the generic SaLSa template (Algorithm 1), here we immediately check if the stop condition has been reached before comparing the new read point $\mathbf{p}$ with those in the current skyline $\mathcal{S}$. Note that the stop condition at step 4 correctly takes into account the particular case in which $\mathbf{p}_{stop}$ lies on the main diagonal of the data space and duplicates are possible (that is $\mathbf{p} = \mathbf{p}_{stop}$).

Algorithm 2. SaLSa[minC]

Input: input stream r sorted using the minC function
Output: the skyline $\mathcal{S}(r)$ of r

1: $\mathcal{S} \leftarrow \emptyset$, $stop \leftarrow false$, $\mathbf{p}_{stop} \leftarrow$ undefined, $u \leftarrow r$
2: **while** not $stop \wedge u \neq \emptyset$ **do**
3: $\quad \mathbf{p} \leftarrow$ get next point from u, $u \leftarrow u \setminus \{\mathbf{p}\}$
4: $\quad$ **if** $p^+_{stop} \leq \text{minC}(\mathbf{p})$ and $\mathbf{p}_{stop} \neq \mathbf{p}$ **then** $stop \leftarrow true$
5: $\quad$ **else if** $\mathcal{S} \not\succ \mathbf{p}$ **then**
6: $\quad\quad \mathcal{S} \leftarrow \mathcal{S} \cup \{\mathbf{p}\}$,
7: $\quad\quad$ **if** $p^+ < p^+_{stop}$ **then** $\mathbf{p}_{stop} \leftarrow \mathbf{p}$
8: return $\mathcal{S}$

5. ANALYSIS OF SALSA

In this section, we derive a cost model able to estimate the expected number of points that SaLSa needs to fetch before halting. For the sake of definiteness, we consider only the optimal minC function, but our results can be extended to other sorting functions as well. We start by deriving a formula that is valid for any given data distribution. Then we show how the formula simplifies under the assumption of independent and identically distributed (IID) attributes, which include the uniform distribution as a particular case. Although the IID assumption is untenable for real datasets, it provides us with the basic clue to derive a cost model for the case in which the data distribution is described through a multidimensional histogram. As a corollary of our analysis, we also provide a novel result on the asymptotic size of skylines for nonindependent attributes. Finally, in Section 5.3, we consider the problem of reducing the number of dominance tests when the minC function is used.

5.1 A Cost Model for Predicting the Number of Fetched Points

Given a relation r of n d-dimensional points, let $F_{\mathbf{p}}$ denote the underlying data distribution, that is, $F_{\mathbf{p}}(x) = \Pr\{p[1] \leq x[1], \ldots, p[d] \leq x[d]\}$, and let $f_{\mathbf{p}}(x)$ be the corresponding density function. For any point $\mathbf{p}$ let $DR(\mathbf{p})$ be the *dominance region* of $\mathbf{p}$, that is, the subset of $[0, 1]^d$ dominated by $\mathbf{p}$. We call *dominance number* of $\mathbf{p}$, denoted $DN(\mathbf{p})$, the mass of probability in $DR(\mathbf{p})$, that is, $DN(\mathbf{p}) = \int_{DR(\mathbf{p})} f_{\mathbf{p}}(x)dx$. In other terms, $DN(\mathbf{p})$ represents, for any value of n, the fraction of points that $\mathbf{p}$ dominates.

Let $\boldsymbol{l} = (l, \ldots, l)$ and consider the probability that SaLSa halts at level $\leq l$. This is easily derived to equal:

$$F_{SL}(l; n, d) = 1 - (1 - F_{\mathbf{p}}(\boldsymbol{l}))^n, \tag{6}$$

since at least one of the n points should be present in the hypercube $[0, l]^d$ having as opposite corners the origin of the space and the point $\boldsymbol{l}$. The density function of the stop level is the derivative of F_{SL} with respect to l, that is:

$$f_{SL}(l; n, d) = n \frac{d\, F_{\mathbf{p}}(\boldsymbol{l})}{dl} (1 - F_{\mathbf{p}}(\boldsymbol{l}))^{n-1}.$$

From this we obtain the following basic result:

THEOREM 4. *Given a relation r with n d-dimensional points distributed according to $F_{\mathbf{p}}$, the expected fraction of points, FP, to be fetched when using* minC *is:*

$$FP(n, d) = 1 - \int_0^1 DN(\boldsymbol{l}) f_{SL}(l; n, d)\, dl \tag{7}$$

PROOF. Immediate, since when the stop level is l, a fraction of $DN(\boldsymbol{l})$ points is pruned by the minC function,[6] therefore the integral is the expected fraction of points that can be pruned. □

[6]Note that this is the only part of the model that depends on the specific sorting function.

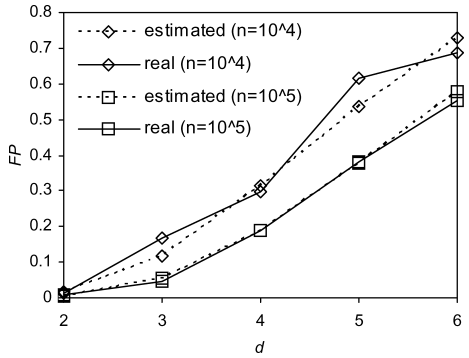


Fig. 6. Estimated and actual values of $FP(n, d)$ when the minC function is used to sort data (uniform datasets).

Equation 4 simplifies under the assumption of independent and identically distributed (IID) attributes. In this case it is $F_{\mathbf{p}}(\mathbf{x}) = \prod_{j=1}^{d} F_A(x[j])$, where F_A denotes the common distribution of the attributes. The stop level distribution and its density can then be written as:

$$F_{SL}(l; n, d) = 1 - (1 - F_A(l)^d)^n \tag{8}$$

$$f_{SL}(l; n, d) = n \cdot d \cdot F_A(l)^{d-1} f_A(l)(1 - F_A(l)^d)^{n-1} \tag{9}$$

As to the dominance number of point $\boldsymbol{l}$, this is $DN(\boldsymbol{l}) = (1 - F_A(l))^d$. Substituting everything into Equation 7 one obtains:

$$FP(n, d) = 1 - \int_0^1 (1 - F_A(l))^d \cdot n \cdot d \cdot F_A(l)^{d-1} f_A(l)(1 - F_A(l)^d)^{n-1} dl.$$

Operating the change of variable $y = F_A(l)$, $dy = f_A(l)dl$, above reduces to:

$$FP(n, d) = 1 - \int_0^1 (1 - y)^d \cdot n \cdot d \cdot y^{d-1}(1 - y^d)^{n-1} dy. \tag{10}$$

Figure 6 shows model predictions together with actual results obtained on uniform datasets. It can be observed that the model accuracy is extremely good and that, although the percentage of points that can be pruned reduces when d grows, SaLSa performs better with larger datasets. For instance, when $d = 6$, one needs to read about 70% of the points if $n = 10^4$, only 58% if $n = 10^5$, and Equation 10 predicts a value less than 44% when $n = 10^6$. This behavior holds for arbitrary data distributions.

LEMMA 5. *For any data distribution $F_{\mathbf{p}}$, the fraction of fetched points, $FP(n, d)$, monotonically decreases with n.*

This result, whose proof is given in the Electronic Appendix, formalizes the intuition that the more points in the dataset, the higher the probability that at least one of them has a low maximum coordinate value p^+, so that the stop level decreases. However, the lemma says nothing about the limiting behavior of SaLSa when the number of points grows arbitrarily large. This is made precise by the following theorem, for which it must be remembered that the support of

a probability distribution is the smallest closed set whose complement has zero probability.

THEOREM 5. *If the stop level distribution has support* $[l_0, 1]$, $0 \leq l_0 \leq 1$, *it is*

$$\lim_{n\to\infty} FP(n, d) = 1 - DN(\boldsymbol{l}_0). \tag{11}$$

where $\boldsymbol{l}_0 = (l_0, \ldots, l_0)$.

Theorem 5, whose proof appears in the Electronic Appendix, demonstrates that all the mass of probability of the stop level concentrates on the lowest possible value, l_0, as n grows arbitrarily large. This is a direct consequence of the form of the stop level distribution given by Equation 6, and as such applies to *any* data distribution. Since $1 - DN(\boldsymbol{l}_0) = 0$ implies that the number of fetched points is a sublinear function of n, $FP(n, d) \cdot n = o(n)$, we have the following corollary, which provides a novel insight on the asymptotic size of skylines.

COROLLARY 2. *If* $DN(\boldsymbol{l}_0) = 1$, *where* $\boldsymbol{l}_0$ *is as in Theorem* 5, *then* $|\mathcal{S}| = o(n)$.

PROOF. Immediate, since the skyline is always included in the set of points fetched by SaLSa. □

It is well known that the size of the skyline is $\Theta((\ln n)^{d-1}/(d-1)!) = o(n)$ when attributes are independent [Buchta 1989; Godfrey 2004], however, Corollary 2 provides a much weaker condition for having "small" skylines. In particular, Corollary 2 is of interest when attribute independence does not hold, an example of which is given as follows.

Example 7. Let $d = 2$ and consider the (non-independent) data distribution:

$$F_{\mathbf{p}}(\mathbf{x}) = \frac{x[1] \cdot x[2] \cdot (x[1] + x[2])}{2} \qquad f_{\mathbf{p}}(\mathbf{x}) = x[1] + x[2].$$

It is plain to see that the support of the stop level distribution is $[0, 1]$, and that $DN(\mathbf{0}) = 1$. From this we can immediately conclude that any relation with n tuples drawn from this distribution will have a skyline of size $o(n)$.

Going the other way in Corollary 2 is not possible. This is to say that there exist data distributions whose skyline has size $o(n)$, yet SaLSa might well fetch a number of points that grows as n. In particular, this holds when $l_0 > 0$ and there is a non-null probability, equal to $1 - DN(\boldsymbol{l}_0)$, to have points for which it is $\mathsf{minC}(\mathbf{p}) < l_0$.

Example 8. Figure 7 shows a dataset with a skyline of only two points, $\mathcal{S} = \{\mathbf{p}_1, \mathbf{p}_2\}$. Assume that no point can lie in the square $[0, .4) \times [0, .4)$, yet there are $(1 - \alpha) \cdot n$ points for which it is $\mathsf{minC}(\mathbf{p}) < .4$, for some $\alpha < 1$. Thus it is $l_0 = .4$, $DN(.4, .4) = \alpha$, and SaLSa will necessarily read at least $(1 - \alpha) \cdot n$ points.

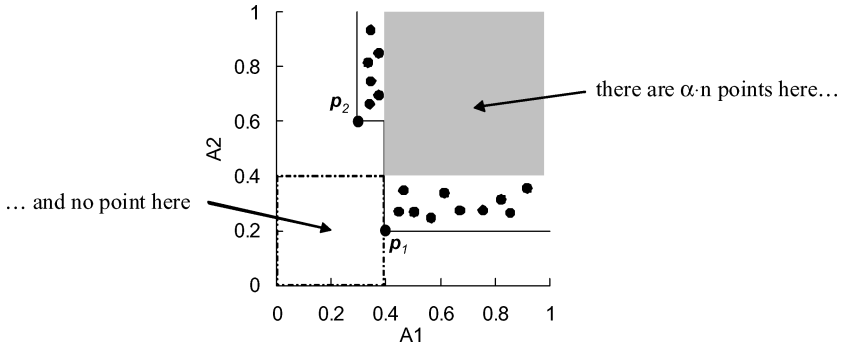


Fig. 7. A 2-d dataset for which the skyline has size $o(n)$, yet SaLSa needs to read at least $(1-\alpha)\cdot n$ points.

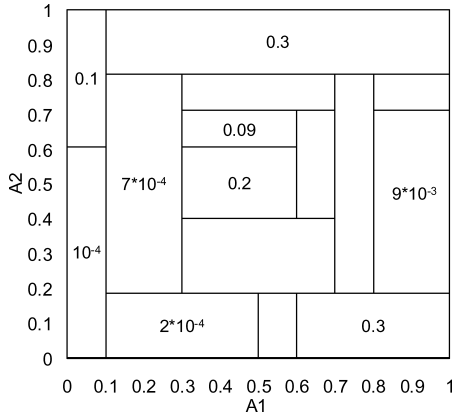


Fig. 8. A 2-d histogram. The number within each bucket represents the fraction of points in the bucket.

5.2 Estimates using Multi-Dimensional Histograms

When attributes are not IID, which is the rule in real datasets, it is still possible to predict SaLSa performance if the data distribution is described through a multidimensional histogram. Without loss of generality, we consider histograms made up of a set of buckets, where bucket B_k represents a hyper-rectangular region of the data space and stores the number of points, n_k, falling in that region. This is illustrated in Figure 8, where the selectivity of each bucket, n_k/n, is also shown (empty buckets have no points at all).

To evaluate the integral in Equation 7, one needs to set an integration step, Δl, and, for each resulting value of l, compute the probability of stopping at that level. The latter can be split into two parts:

(1) Evaluate $DN(\boldsymbol{l})$: This is directly obtained from the histogram by estimating the selectivity of the range query $[l, 1]^d$.

(2) Evaluate $f_{SL}(l)$: By definition, this equals $F_{SL}(l;n,d) - F_{SL}(l - \Delta l;n,d)$. We can estimate $F_{SL}(l;n,d)$ as follows:
   (a) For each bucket B_k intersecting $[0,l]^d$, we compute the probability, denoted $\mathsf{Out}_k(l)$, that all its n_k points fall *outside* the $[0,l]^d$ hypercube. This is done by assuming that within each bucket, data are uniformly distributed, so that $\mathsf{Out}_k(l)$ only depends on the number of points in the bucket, on the bucket extension, and on the size of the intersection, that is:

$$\mathsf{Out}_k(l) = \left(1 - \frac{\text{size of the intersection of } B_k \text{ and } [0,l]^d}{\text{size of } B_k}\right)^{n_k}.$$

   (b) We multiply all the $\mathsf{Out}_k(l)$ probabilities and then complement to 1, thus:

$$F_{SL}(l;n,d) = 1 - \prod_k \mathsf{Out}_k(l).$$

   This follows, since one can stop at a level $\leq l$ if at least one intersecting bucket has a point in the $[0,l]^d$ hypercube.

As an example, using the values in Figure 8 with $n = 10^5$ and $\Delta l = 0.1$, the model predicts $FP \approx 0.28$, and almost certainly the stop level will be ≤ 0.2.

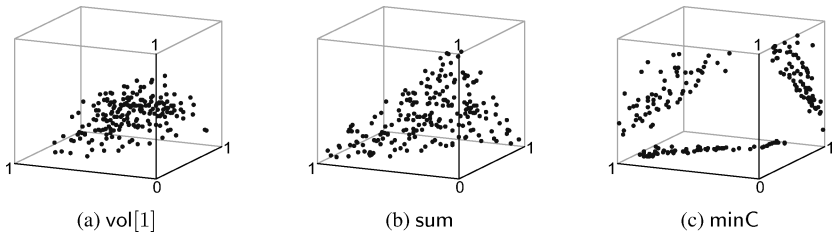
5.3 Dominance Tests

Having demonstrated that minC is the symmetric function that minimizes the number of points to be fetched, it would be interesting to also characterize the *filtering* power of this function, that is, the impact it can have on the effectiveness of the filter phase in which dominance tests are executed.

Unlike the problem of estimating the number of fetched points, that of predicting the number of dominance tests is much more complex. To gain some insight on the difficulty of the problem, observe that the number of dominance tests depends on the (distribution of the) skyline size after i points have been read, and this distribution should be evaluated for any value of i up to the number of fetched points. However, even for the simplest case of independent attributes, there is no known analytical result for the skyline distribution when $d > 2$ [Godfrey 2004]. Further, in our scenario, we should consider how this distribution varies not only with i but also with the specific order in which points are read, which represents a formidable challenge to deal with.[7]

From a more pragmatic viewpoint, it would at least be useful to understand if minC is better than other functions in reducing the complexity of the filter phase. Unfortunately, this does not seem to be the case. The intuition is that functions like vol[m] and sum (which, recall, behaves like vol[∞]) are likely to still enjoy the desirable property of vol[0], that is, that of fetching first points that have a high chance of filtering out many other points. However, this is not necessarily the case with minC, since this function might well retrieve first

[7]Chaudhuri et al. [2006] show that sorting data on the values of a single attribute, say A_1, is approximately equivalent, in terms of dominance tests, to compute the skyline over the remaining $d - 1$ attributes. However, this result holds only if A_1 is independent of the other attributes and does not extend to multiattribute sorting functions.



(a) vol[1] (b) sum (c) minC

Fig. 9. Distribution of the first 200 skyline points out of 810, as obtained by sorting data with different symmetric functions ($n = 500K$, mixed dataset).

points that are bad on all coordinates but one, thus likely to dominate only a few points, if any. This is also suggested by the graphs in Figure 9, which show the distribution of the first 200 skyline points returned by vol[1], sum, and minC, respectively, for a three-dimensional *mixed* dataset, a detailed description of which is given in Section 7, and whose full skyline consists of 810 points. It is apparent that minC first retrieves points that are close to the boundary of the space, and as such less likely to be effective in pruning other points.

A key observation to improve the performance of minC is that the number of dominance tests also depends on the order in which points in the (partial) skyline $\mathcal{S}$ are compared against a new read point. This issue was first investigated by Börzsönyi et al. [2001], who proposed heuristic methods, such as managing $\mathcal{S}$ as a self-organizing list, besides considering the basic forward strategy in which $\mathcal{S}$ is scanned by respecting the order in which skyline points have been fetched. However, these methods introduce additional overhead, which might well lead to nullifying the possible reduction of dominance tests so obtained.

An alternative that is trivial to implement and does not incur any additional cost is to consider points in $\mathcal{S}$ in the *reversed* (or backward) order in which they have been retrieved, that is, the last point added to $\mathcal{S}$ will be the first one against which a new read point will be compared, and so on. The rationale for adopting this backward strategy is given by the following result, whose proof is presented in the Electronic Appendix.

THEOREM 6. *Assume points are sorted using the* minC *function. If attributes are independent and identically distributed (IID), then the expected number of dominance tests executed by scanning the current skyline in* backward *order is less than that needed by the* forward *version.*

For arbitrary data distributions, the relative performance of the backward strategy with respect to the forward one is hard to predict, and it will be experimentally evaluated in Section 7.1.

6. ASYMMETRIC SORTING FUNCTIONS

In this section, we extend our analysis to the case where an asymmetric function is used to sort points. In particular, we still consider that data have been normalized in the $[0, 1]^d$ domain and that asymmetry is defined with respect to this domain. It is not difficult to see that this also covers the case where,

say, data are *not* normalized and one uses a *symmetric* function on the original attributes' domains (e.g., Price + Mileage − Year).

Why should one even consider using an asymmetric function? We foresee three basic reasons for this choice:

Interface limitations. The skyline algorithm runs on the client side, and the server has some limitations that preclude the possibility of using a symmetric function. For instance, data cannot be normalized and one has to work by, say, linearly combining original attributes' values.

User preferences. The user might want to obtain the skyline points according to a specific order, for example, $A_1 + 2A_2$. In this case, especially for large skylines, it might not be convenient, for performance reasons, to first sort points using a symmetric function and then resort the result according to the user preferences. Rather, the user sorting function should be directly used, which would also preserve the incremental behavior of the algorithm.

Performance. As demonstrated in Theorem 3, no symmetric function can do better than minC in limiting the input. However, as will be shown in Section 6.2, if one has some knowledge of the actual data distribution, then using an asymmetric sorting function can lead to considerable performance improvements.

Our analysis is organized as follows. We first show how SaLSa can be extended so as to also work with asymmetric functions by generalizing results derived in Section 3.1. Then, in Section 6.2 we introduce an asymmetric function, called ΔminC_{opt}, that is provably optimal over all monotone functions and for *all* input relations. Our theoretical analysis on the performance of ΔminC_{opt}, summarized in Theorem 11, leads to a rather unexpected conclusion: even if no intrinsic asymmetry is present in the dataset, using ΔminC_{opt} in place of minC can improve SaLSa performance by more than 40% in terms of number of points to be read. However, ΔminC_{opt} assumes a detailed knowledge of the data distribution that cannot be available in realistic scenarios. For this reason in Section 6.3 we investigate how multidimensional histograms can help in approximating the optimal sorting criterion established by ΔminC_{opt}. This results in a practical algorithm for choosing the function to use for any given dataset.

6.1 Extending SaLSa to Asymmetric Sorting Functions

When $\mathcal{M}$ is asymmetric, it is no longer true that minimal values of points in the unread domain at level l, $\mathcal{D}(\mathcal{M}, l)$, are the same for all attributes, thus Lemma 1 no longer holds and all subsequent results in Section 3.1 are therefore invalidated. We start by showing how Lemma 1 can be easily extended to arbitrary (that is not necessarily symmetric) functions.

LEMMA 6. *Let $\mathcal{M}$ be* any *monotone function and assume $\mathcal{M}$ is at level l. Further, let $\mathsf{Low}[j]$ be the minimum value such that $\mathcal{M}(1, \ldots, 1, \mathsf{Low}[j], 1, \ldots, 1) \geq l$ holds ($j = 1, \ldots, d$). If no such finite $\mathsf{Low}[j]$ exists, then set $\mathsf{Low}[j] = 0$. Then, there is no point $\mathbf{p}$ for which both the following are true: 1) $\mathcal{M}(\mathbf{p}) \geq l$, and 2) there exists j such that $p[j] < \mathsf{Low}[j]$.*

PROOF. The result immediately follows from the definition of $\mathsf{Low}[j]$ and the monotonicity of $\mathcal{M}$. □

Above lemma says that, for any monotone function, $\mathcal{D}(\mathcal{M}, l)$ is included in the hyper-rectangle whose opposite vertices are $(\mathsf{Low}[1], \ldots, \mathsf{Low}[d])$ and $\mathbf{1} = (1, \ldots, 1)$.

Given a skyline point $\mathbf{p}$, now it is no longer true that its maximum coordinate value, p^+, is the only relevant information that has to be used for stopping SaLSa execution. Rather, one should look at the specific function $\mathcal{M}$ and determine which is the largest (hyper-rectangle enclosing) $\mathcal{D}(\mathcal{M}, l)$ that $\mathbf{p}$ dominates. To this end, define the j-th stop level of point $\mathbf{p}$ as:[8]

$$l^j = \mathcal{M}(1, \ldots, 1, p[j], 1, \ldots, 1). \tag{12}$$

and let $l^+ = \max_j\{l^j\}$ be the *maximum stop level* of $\mathbf{p}$. Theorem 1 can be generalized by considering the maximum stop level of the stop point rather than just its maximum coordinate value.

THEOREM 7. *Let $\mathcal{M}$ be any monotone function, and assume $\mathcal{M}$ reaches level l at a certain stage of the execution. Then, SaLSa can be stopped if the maximum stop level l^+_{stop} of the stop point $\mathbf{p}_{stop}$ satisfies $l^+_{stop} \leq l$. Only in the particular case that $l^j_{stop} = l^+_{stop}$ holds for all j, SaLSa should also read all (if any) $\mathbf{p}_{stop}$'s duplicates.*

PROOF. From Lemma 6 we know that if $\mathcal{M}(\mathbf{p}) \geq l$, then for all j it is $p[j] \geq \mathsf{Low}[j]$. Since $l \geq l^+_{stop}$ it is $l \geq l^j_{stop}$, thus $\mathcal{M}(1, \ldots, 1, \mathsf{Low}[j], 1, \ldots, 1) \geq \mathcal{M}(1, \ldots, 1, p_{stop}[j], 1, \ldots, 1)$ holds for all j. If $l = l^j_{stop}$, then it is necessarily $\mathsf{Low}[j] = p_{stop}[j]$, whereas when $l > l^j_{stop}$, the monotonicity of $\mathcal{M}$ guarantees that $\mathsf{Low}[j] > p_{stop}[j]$, thus $p[j] > p_{stop}[j]$. If at least one strict inequality occurs, the result clearly holds. If equality occurs on all coordinates, then $\mathbf{p}_{stop}$ dominates all points in $\mathcal{D}(\mathcal{M}, l)$ but its duplicates. □

Intuitively, the theorem gives a condition that, if satisfied, guarantees that on each coordinate the stop point is no worse than the point $(\mathsf{Low}[1], \ldots, \mathsf{Low}[d])$, thus $\mathbf{p}_{stop}$ dominates all the points that have not been read yet.

The MiniMax rule for choosing the stop point (Theorem 2) can be generalized in a similar way.

THEOREM 8. *The following* generalized MiniMax *rule for choosing the stop point is optimal for* any *monotone function $\mathcal{M}$:*

$$\mathbf{p}_{stop} = \arg\min_{\mathbf{p}_i \in \mathcal{S}}\{l^+_i\}. \tag{13}$$

PROOF. Immediate from Theorem 7. □

Example 9. Let points in the UsedCars relation be sorted using the function $\mathcal{M}(\mathbf{p}) = 2 \cdot \mathbf{p}.\text{Price} + \mathbf{p}.\text{Mileage}$ and refer to Figure 10(a). For the 3 skyline points it is derived, respectively: $l^+_1 = 110$, $l^+_2 = 130$, and $l^+_4 = 140$, which leads

[8]Denoting the jth stop level of $\mathbf{p}$ as $l[j]$ would be inconsistent when $\mathcal{M}$ is a vectorial function, in which case the jth stop level is a vector.

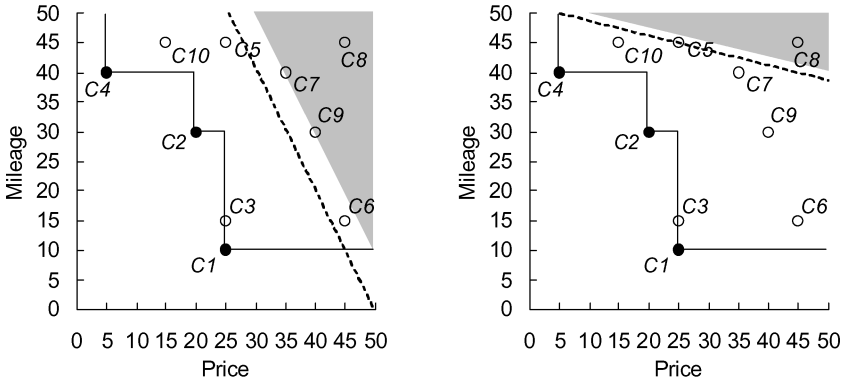


Fig. 10. In (a) points are sorted using the function $2 \cdot \mathbf{p}.\text{Price} + \mathbf{p}.\text{Mileage}$, which leads to prune points C8 and C9 (or C8 and C7). The nonmaximal stop level of C1, $l_1^{\text{Price}} = 100$, is shown as a dashed line. In (b) the sorting function is $\mathbf{p}.\text{Price} + 4 \cdot \mathbf{p}.\text{Mileage}$, and no point can be pruned.

to choose C1 as the stop point. As an example of how the maximum stop levels are derived, consider point C1 $= (25, 10)$ and remember that both attributes have domain $[0, 50]$. It is $l_1^{\text{Price}} = 100$, since $\mathcal{M}(25, 50) = 100$, and $l_1^{\text{Mileage}} = 110$, since $\mathcal{M}(50, 10) = 110$. Both points C7 and C9 are at level 110. If, say, point C7 is read first, point C9 can be pruned, as well as point C8. On the other hand, if the sorting function is $\mathcal{M}(\mathbf{p}) = \mathbf{p}.\text{Price} + 4 \cdot \mathbf{p}.\text{Mileage}$, the stop point is C4, with $l_4^+ = 210 = \max\{\mathcal{M}(5, 50), \mathcal{M}(50, 40)\}$. In this case, as Figure 10(b) shows, no point can be pruned, since C5 is at level $205 < l_{stop}^+ = 210$, thus SaLSa cannot halt before reading even C8.

Theorem 7 proves that $l \geq l_{stop}^+$ is a *sufficient* condition for SaLSa to halt. Unlike the corresponding stop condition in Theorem 1 ($\text{Low} \geq p_{stop}^+$), $l \geq l_{stop}^+$ is *not necessary* for all monotone functions. This is to say that there are functions for which SaLSa can be halted earlier. To see why this is the case, it should be remembered that Theorem 7 also applies to nonscalar, thus *vectorial*, functions, such as minC and the asymmetric versions of this function we are going to introduce. When $\mathcal{M}$ is vectorial, level l is a vector, $l = (l[1], l[2], \ldots)$, and sorting is based first on $l[1]$, then on $l[2]$, and so on. The key to strengthen the stop condition is based on the observation that for many functions, which we call *1-monotone*, it is enough to consider just the primary sorting criterion to halt (thus all scalar sorting functions are also 1-monotone).

Definition 4. Let $\mathbf{p}_i$ and $\mathbf{p}_k$ be any two points such that $\mathbf{p}_i = (1, \ldots, 1, p_i[j], 1, \ldots, 1)$ and $\mathbf{p}_k = (1, \ldots, 1, p_k[j], 1, \ldots, 1)$, with $p_i[j] < p_k[j]$. Let $l_i = \mathcal{M}(\mathbf{p}_i)$ and $l_k = \mathcal{M}(\mathbf{p}_k)$. If the primary sorting criterion is always sufficient to order $\mathbf{p}_i$ and $\mathbf{p}_k$, regardless of which is the only coordinate j on which they are not maximal, that is, $l_i[1] < l_k[1]$ holds for each j, then $\mathcal{M}$ is called a 1-monotone function.

THEOREM 9. *For any monotone function $\mathcal{M}$ it is necessary that $l[1] \geq l^+_{stop}[1]$ holds before* SaLSa *can be stopped. The condition is also sufficient if $\mathcal{M}$ is 1-monotone.*[9]

PROOF. The condition is necessary, since $l[1] < l^+_{stop}[1]$, thus $l \prec l^+_{stop}$ implies that the unread domain at level l includes a point $\mathbf{p}$ that is not dominated by $\mathbf{p}_{stop}$.

To prove the second part of the theorem, observe that the only difference with the proof of Theorem 7 concerns the hypothesis needed to derive that $\mathsf{Low}[j] \geq p_{stop}[j]$ holds for all j. Since $l[1] > l^+_{stop}[1]$ implies $l \succ l^+_{stop}$, it remains to show that $l[1] = l^+_{stop}[1]$ cannot lead to having $\mathsf{Low}[j] < p_{stop}[j]$. But this is now ruled out by the assumption of 1-monotonicity of $\mathcal{M}$. □

Example 10. The minC function is 1-monotone, thus SaLSa can be stopped as soon as it fetches a point $\mathbf{p}$ such that $\min_j\{p[j]\} \geq p^+_{stop}$, that is, only the primary sorting criterion is needed. On the other hand, consider the "bucketed" version of minC defined as:

$$\mathsf{b\_minC}(\mathbf{p}) = (\min_j\{\lfloor p[j] \cdot b \rfloor\}, \mathsf{sum}(\mathbf{p})),$$

in which the range $[0, 1]$ is partitioned into $b+1$ intervals (the last one including only the value $p[j] = 1$). Let $d = 2$, $b = 4$, and consider point $\mathbf{p} = (.5, 1)$, for which it is $l = \mathsf{b\_minC}(\mathbf{p}) = (2, 1.5)$. Assume the stop point is $\mathbf{p}_{stop} = (.6, .2)$. From the definition of j-th stop level it is derived $l^1_{stop} = \mathsf{b\_minC}(.6, 1) = (2, 1.6)$ and $l^2_{stop} = \mathsf{b\_minC}(1, .2) = (0, 1.2)$, thus $l^+_{stop} = (2, 1.6)$ since $(2, 1.6) \succ (0, 1.2)$. Although $l[1] \geq l^+_{stop}[1]$ $(2 \geq 2)$ SaLSa *cannot* be halted after reading $\mathbf{p}$, since $\mathbf{p}_{stop} \not\prec \mathbf{p}$. Indeed b_minC is not 1-monotone, since the primary sorting criterion assigns a same value (i.e., 2) to points $\mathbf{p} = (.5, 1)$ and $(.6, 1)$. This implies that the secondary sorting criterion, $\mathsf{sum}(\mathbf{p})$, is also relevant to determine when execution can be stopped.

For non 1-monotone functions, the necessary and sufficient conditions to halt do not necessarily coincide. It is therefore legitimate to ask if there exists a function $\mathcal{M}$ such that SaLSa using $\mathcal{M}$ can halt when $l[1] = l^+_{stop}[1]$, yet $l \prec l^+_{stop}$. The well-known *lexicographic sort*, which first orders data using A_1, then breaks ties using A_2, and so on, is a such function. Lexicographic sort can be compactly defined as $\mathsf{lex}(\mathbf{p}) = \mathbf{p}$, that is, the level of point $\mathbf{p}$ coincides with $\mathbf{p}$ itself. Clearly, lexicographic sort is not 1-monotone, since it is $l_i[1] = l_k[1]$ whenever it is $\mathbf{p}_i.A_1 = \mathbf{p}_k.A_1$.

For lex, the jth stop level of point $\mathbf{p}$ is $l^j = (1, \ldots, 1, p[j], 1, \ldots, 1)$, and the maximum stop level of $\mathbf{p}$ is therefore $l^+ = (1, \ldots, 1, p[d])$. Thus Theorem 7 asserts that SaLSa using lex can be halted after reading a point $\mathbf{p}$ whose first $(d-1)$ coordinates are 1 and the dth one is $p[d] \geq p_{stop}[d]$. Although correct, this stop condition is not necessary.

Consider the point $\mathbf{p}_{stop} = (p_{stop}[1], \ldots, p_{stop}[k], 0, \ldots, 0)$, that is, $\mathbf{p}_{stop}$ is minimal (thus, optimal) on attributes $k+1, \ldots, d$. Then, the execution can be

[9]As usual, the particular case when $p_{stop}[j] = p^+_{stop}$ for all j should be considered, so that possible duplicates of $\mathbf{p}_{stop}$ are fetched before halting.

halted as soon as SaLSa reads a point $\mathbf{p}$ such that $p[j] = 1$, $1 \leq j < k$, and $p[k] \geq p_{stop}[k]$.[10] This follows since, for any other point $\mathbf{p}_i$ such that $\mathbf{p} < \mathbf{p}_i$, it is true that $p_i[j] = 1$, $1 \leq j < k$. The minimality of $\mathbf{p}_{stop}$ on the remaining coordinates is then sufficient to conclude that $\mathbf{p}_{stop} \succ \mathbf{p}_i$. The rule for choosing the stop point when using lex can be found in the Electronic Appendix.

In spite of its popularity, lexicographic sort is not a good choice for sort-based skyline algorithms. Indeed, experimental results in Chomicki et al. [2002] have already shown that it performs much worse than sorting by volume when all points are read. This is because of the nested sort it requires and of the higher number of dominance tests it leads to executing.

6.2 An Optimal Asymmetric Sorting Function

In order to derive an optimal asymmetric function, the following observation is fundamental: Since SaLSa prunes the input stream by means of a single stop point, the best one can hope to obtain by using an asymmetric function is to avoid reading a fraction of points equal to the maximum value of $DN(\mathbf{p})$ in the dataset (remember that $DN(\mathbf{p})$ is the dominance number of $\mathbf{p}$, that is, the fraction of points in r that $\mathbf{p}$ dominates). To this end, let $\mathbf{p}_{opt}$ be the point that dominates the highest number of points in r, that is:

$$\mathbf{p}_{opt} = \arg\max_{\mathbf{p}_i \in \mathcal{S}} \{DN(\mathbf{p}_i)\}. \tag{14}$$

For instance, in the UsedCars relation, it is $\mathbf{p}_{opt} \equiv$ C1, since C1 dominates 6 points (C3, C5, C6, C7, C8, and C9) whereas both C2 and C4 dominate only 4 points.

An asymmetric function such that:

(1) $\mathbf{p}_{opt}$ is the stop point when such function is used, and
(2) SaLSa will stop without reading all the points dominated by $\mathbf{p}_{opt}$

would be necessarily optimal by definition. In order to gain some intuition on how a function with such features can be derived, consider Figure 11(a), in which the shaded area is the dominance region of $\mathbf{p}_{opt} \equiv$ C1. Since this dominance region is a hyper-rectangle, it is reasonable to guess that the optimal asymmetric function should be a generalization of minC, which is only able to prune hypercubic regions. This generalization can be obtained by moving $\mathbf{p}_{opt}$ to the main diagonal of a *transformed* data space in which all the dominance relationships are preserved. The simplest way of achieving this is to introduce an (optimal) *shift vector*, Δ_{opt}, defined as:

$$\Delta_{opt}[j] = p^{+}_{opt} - p_{opt}[j] \qquad j = 1, \ldots, d$$

and to add to each coordinate value of a point the corresponding shift component.

Figure 11 (b) shows this transformation for the UsedCars relation. Since C1 $= (25, 10)$, the optimal shift vector is $\Delta_{opt} = (0, 15)$, that is, the 2nd coordinate (Mileage) of each point is increased by 15. It follows that sorting points

[10]The last should be a strict inequality only if $p_{stop}[j] = 1$, $1 \leq j < k$.

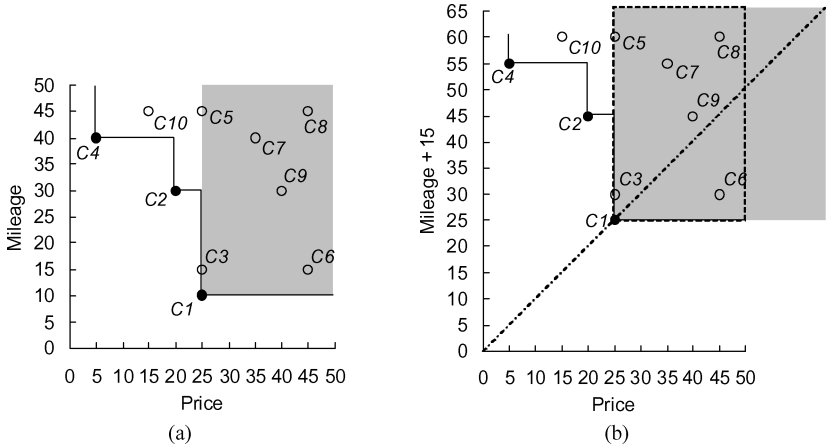


Fig. 11. The shaded area is the dominance region of point C1 in the original (a) and the transformed (b) space, respectively. The transformation is obtained by applying to points the (optimal) shift vector $\Delta_{opt} = (0, 15)$.

by the minimum of their so-transformed coordinate values makes it possible to prune all the points dominated by $\mathbf{p}_{opt}$.[11]

THEOREM 10. *Let $\mathbf{p}_{opt}$ be the point in relation r with the highest dominance number, and let $\Delta_{opt}[j] = p^{+}_{opt} - p_{opt}[j]$, $j = 1, \ldots, d$. Define the $\Delta minC_{opt}$ function as:*

$$\Delta minC_{opt}(\mathbf{p}) = (\min_{j}\{p[j] + \Delta_{opt}[j]\}, sum(\mathbf{p})). \tag{15}$$

Then, for any sorting function $\mathcal{M}$ it is $\mathcal{D}_{stop}(\mathcal{M}) \subseteq \mathcal{D}_{stop}(\Delta\text{minC}_{opt})$, thus ΔminC_{opt} is optimal over all *monotone functions.*

PROOF. We first prove that sorting points using ΔminC_{opt} leads to having $\mathbf{p}_{stop} \equiv \mathbf{p}_{opt}$. Then we show that all points dominated by $\mathbf{p}_{opt}$ can be safely pruned, from which the result follows, since $\mathbf{p}_{opt}$ has the highest dominance number among the points in r.

According to Equation 13, $\mathbf{p}_{stop}$ is the point that minimizes the maximum of its stop levels. In the case of ΔminC_{opt}, for any point $\mathbf{p}$ it is:[12]

$$\begin{aligned} l^{j} &= \min\{1 + \Delta_{opt}[1], \ldots, p[j] + \Delta_{opt}[j], \ldots, 1 + \Delta_{opt}[d]\} \\ &= \min\{1, p[j] + \Delta_{opt}[j]\}, \end{aligned}$$

where the second equality follows, since $p[j] \leq 1$ holds for all j and there exists at least one coordinate j such that $\Delta_{opt}[j] = 0$. Then, l^{+} equals:

$$l^{+} = \max_{j}\{\min\{1, p[j] + \Delta_{opt}j[]\}\} = \min\{1, \max_{j}\{p[j] + p^{+}_{opt} - p_{opt}j[]\}\}. \tag{16}$$

[11]Actually, all but one if duplicates are possible.

[12]As done for the minC function, even here we simplify the presentation by considering only the primary sorting criterion of ΔminC_{opt}. This has no harmful consequences, mainly because ΔminC_{opt} is 1-monotone.

For point $\mathbf{p}_{opt}$ the above yields $l^+_{opt} = p^+_{opt}$. For any other skyline point $\mathbf{p}$, let j be any coordinate such that $p[j] > p_{opt}[j]$. Note that such j must exist, otherwise $\mathbf{p}_{opt}$ would not be in the skyline. Then, $l^+ > p^+_{opt}$ is immediately derived, thus $l^+_{opt} = l^+_{stop}$ as claimed.

For the second part of the proof it is sufficient to show that if $\mathbf{p}$ is any point dominated by $\mathbf{p}_{opt}$, then the level of $\mathbf{p}$ is $l \geq l^+_{opt} \equiv l^+_{stop}$. But this is obvious, since $\mathbf{p}_{opt} \succ \mathbf{p}$ guarantees that, for each j, $p[j] + \Delta_{opt}[j] = p[j] + p^+_{opt} - p_{opt}[j] \geq p^+_{opt} = l^+_{opt}$, thus $l \geq l^+_{opt}$. □

The behavior of SaLSa with ΔminC_{opt} is characterized by a sudden stop after $\mathbf{p}_{opt}$ has been read. In fact, $\mathbf{p}_{opt}$ will always be the *last* skyline point to be read. After reading $\mathbf{p}_{opt}$, further points are to be read only if $\mathbf{p}_{opt}$'s duplicates are present, thus SaLSa can safely halt as soon as it discovers that this is not the case (anymore). For instance, in Figure 11(b) SaLSa would halt just after reading point C3, thus pruning C5, C6, C7, C8, and C9.

It has to be remarked that *any* dominance-preserving transformation that moves $\mathbf{p}_{opt}$ to the main diagonal of the data space would generate an optimal sorting function. For instance, one could consider the *scale* (rather than shift) vector, Δ^{scale}_{opt}, whose jth component is $\Delta^{scale}_{opt}[j] = p^+_{opt}/p_{opt}[j]$, and then transform the jth coordinate of point $\mathbf{p}$ into the scaled value $p[j] \cdot \Delta^{scale}_{opt}[j]$.[13] However, ΔminC_{opt} is apparently the simplest way to achieve optimality without introducing any unnecessary computational overhead.

When points are sorted using the ΔminC_{opt} function, the performance of SaLSa can be accurately predicted on the assumption that attributes are independent.

THEOREM 11. *Given a relation r with n d-dimensional points and independent attributes, the expected fraction of points to be fetched when SaLSa uses the optimal ΔminC_{opt} sorting function is:*

$$FP_{opt}(n,d) = \int_0^1 z^n \left(\sum_{j=0}^{d-1} \frac{(-\ln z)^j}{j!} \right)^n dz. \tag{17}$$

Note that the theorem, whose proof is given in the Electronic Appendix, holds regardless of the specific attributes' distributions, which might also be different for each attribute. This is perfectly reasonable, since the skyline dominance relationship does not depend on attributes' scales.

Figure 12(a) plots estimates of $FP_{opt}(n,d)$ provided by Equation 17 and contrasts them with actual results, as obtained by executing SaLSa with the optimal ΔminC_{opt} function. Figure 12(b) compares $FP_{opt}(n,d)$ with $FP(n,d)$, that is, the basic SaLSa version using minC and modelled by Equation 10. The key observations are that: 1) The model given by Equation 17 is highly accurate, 2) the gain in performance of ΔminC_{opt} with respect to minC is appreciable regardless of the specific dimensionality, and 3) such gain increases with

[13]However, this works only if for all j it is $p_{opt}[j] > 0$.

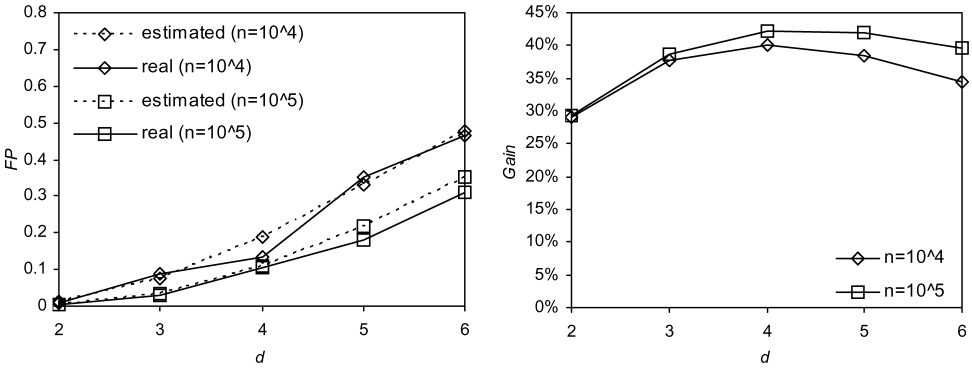


Fig. 12. (a) Estimated and actual values of $FP_{opt}(n, d)$; (b) Percentage gain of ΔminC_{opt} wrt minC, that is, $(1 - FP_{opt}(n, d)/FP(n, d)) \cdot 100$ (uniform datasets).

larger datasets, exceeding 40% for a four-dimensional uniform dataset with 10^5 points.

6.3 An Algorithm for Choosing a Good Asymmetric Function

In this section, we investigate how multidimensional histograms can be exploited so as to make the sorting criterion adaptive to the given dataset. More precisely, we aim to determine a "good" shift vector Δ with which to perturb point coordinates so as to minimize the number of points to be read. Ideally, one would like to have $\Delta = \Delta_{opt}$, but the granularity with which data are modelled in a histogram is likely to yield only a suboptimal shift vector.

There are two issues to address for solving the problem: how to cope with the infinite number of possible shift vectors, and how, given two shift vectors Δ_1 and Δ_2, to determine which one is likely to lead to a better performance.

As in Section 5.2, we consider that points within each bucket are uniformly distributed, that is, each bucket B_k only provides information on its hyperrectangle, $[L_k[1], H_k[1]] \times \cdots \times [L_k[d], H_k[d]] = [\mathbf{L}_k, \mathbf{H}_k]$, and on the number of points, n_k, in $[\mathbf{L}_k, \mathbf{H}_k]$. First, it is plain to see that only *undominated* buckets need to be considered, where bucket B_s dominates B_k if $\mathbf{H}_s \succ \mathbf{L}_k$. This guarantees that B_k cannot contain any skyline point.

For any undominated bucket B_k, in principle, one should guess the number and *the actual positions* of skyline points within B_k, and for each of them, evaluate the advantage of using it as a stop point (that is using its coordinates to compute the shift vector). Given the bucket's information, this is not feasible at all, which is why we pursue an approximate approach based on the concept of *hypothetical skyline point* [Papadias et al. 2005]. This assumes that the skyline restricted to the points in B_k consists of a *single* point, $\mathbf{p}_k$, whose coordinates

Algorithm 3. SaLSa[ΔminC]

Input: relation r and a multidimensional histogram describing r distribution
Output: the skyline $\mathcal{S}(r)$ of r

```
1:  estimate p̂_opt using the histogram
2:  compute Δ[j] = p̂_opt^+ − p̂_opt[j] (j = 1, ..., d)
3:  sort r with the function ΔminC
4:  S ← ∅, stop ← false, p_stop ← undefined, l+_stop ← 1, u ← r
5:  while not stop ∧ u ≠ ∅ do
6:      p ← get next point from u, u ← u \ {p}
7:      if l+_stop ≤ ΔminC(p) and p_stop ≠ p stop ← true
8:      else if S ⊁ p then
9:          S ← S ∪ {p},
10:         l+ = max_j{p[j] + Δ[j]},
11:         if l+ < l+_stop then l+_stop ← l+, p_stop ← p
12: return S
```

are:

$$p_k[j] = L_k[j] + \frac{H_k[j] - L_k[j]}{n_k + 1}, \tag{18}$$

which is just the expected value of the minimum of n_k points uniformly distributed over $[L_k[j], H_k[j]]$.[14]

Equation 18 provides us with m hypothetical skyline points, one for each undominated bucket. However, even if B_s does not dominate B_k, it might well be the case that $\mathbf{p}_s \succ \mathbf{p}_k$. After dropping dominated points, this leaves us with a number of undominated hypothetical skyline points. To estimate the advantage of using one of them for setting up the sorting function, we just perform a range query on the histogram, that is, for point $\mathbf{p}_s$ we compute $DN(\mathbf{p}_s)$. The point for which this is maximum, denoted $\widehat{\mathbf{p}_{opt}}$, is then chosen as an approximation of the optimal stop point and its coordinates used for computing the Δ shift vector, that is:

$$\Delta[j] = \widehat{p_{opt}}^{+} - \widehat{p_{opt}}[j] \qquad j = 1, \ldots, d.$$

Then, points are sorted using the function:

$$\Delta\mathsf{minC}(\mathbf{p}) = (\min_j\{p[j] + \Delta[j]\}, \mathsf{sum}(\mathbf{p})). \tag{19}$$

Algorithm 3 details the steps needed to execute SaLSa with the ΔminC function. Step 10, in which the maximum stop level of point $\mathbf{p}$ is computed, exploits the observation that, since $l^+ = \min\{1, \max_j\{p[j] + \Delta[j]\}\}$ (see Equation 16), the only relevant case to consider for updating the stop point is when $l^+ < 1$.

7. EXPERIMENTAL ANALYSIS

In this section, we analyze the performance of SaLSa and compare it against SFS. In particular, for SaLSa we consider the sorting functions described in

[14]Papadias et al. [2005] use $1/(d \cdot n_k + 1)$ rather than $1/(n_k + 1)$ in the denominator.

Section 4, namely minC, sum, and vol[1], hereafter referred to simply as vol. Since, as demonstrated in Section 4.2, sum behaves like vol[∞], the performance of vol[m], for $1 < m < \infty$, will always be between that of sum and vol. For the SFS algorithm, we consider that it sorts points based on their entropy, that is, using vol (see Section 4.1). In Section 7.4, we analyze the effects of accessing points according to the ΔminC asymmetric sorting function.

In our study, we are mainly interested in measuring the following cost metrics:

Fraction of fetched points (FP). This represents the fraction of points in the dataset (that is $FP \leq 1$) that must be read for correctly computing the skyline. This metric is shown only for SaLSa variants, since $FP = 1$ always holds for SFS.

Dominance tests (DT). This measures the total number of dominance tests that an algorithm executes, normalized to n, the cardinality of the dataset. Since SFS fetches all points, for this algorithm, it is always $DT \geq 1$, whereas $DT < 1$ is possible for SaLSa.[15]

Besides above machine-independent metrics, in Section 7.3 we also report results on actual execution times.

For our experiments, we used both synthetic and real datasets. Synthetic datasets with dimensionality d in the range $[2, 6]$ were generated as in Börzsönyi et al. [2001], each following a particular data distribution, namely uniform, correlated, or anti-correlated. In an anti-correlated dataset, a point with a low value on a dimension is likely to have a high value on at least one other dimension, which leads to having large skylines [Börzsönyi et al. 2001]. We also generated so-called *mixed* datasets, in which 50% of the points are anti-correlated and the remaining 50% are uniformly distributed in the $[0.5, 1]^d$ domain. Thus a mixed dataset simulates the case where the "front" of the dataset is anti-correlated, after which several other points follow. Their actual distribution is indeed almost immaterial, the relevant thing being that such points are all likely to be dominated by some other point in the front. We found this kind of mixed distribution more similar than any of the other three synthetic ones to what some real data actually look like, an example of which is given in Figure 13.

The real datasets we used are:

NBA. This dataset (available at `www.basketballreference.com`) consists of 19, 112 tuples containing statistics of basketball players during regular seasons in the period [1946–2005]. Multiple tuples are present for players that switched teams during the season (one for each team + one total), thus we deleted the "total" tuples from the set, obtaining 17, 791 tuples. For each player, we only retained six of his statistics, namely number of games played during the season (*gp*), points scored (*pts*), total rebounds (*reb*), assists (*ast*), field goals made (*fgm*), and free throws made (*ftm*). Other statistics were not used, since they were not available for all players, for example, steals were not considered before 1973.

[15]Actually, for SFS it is $DT \geq (n-1)/n$, but this makes no practical difference.

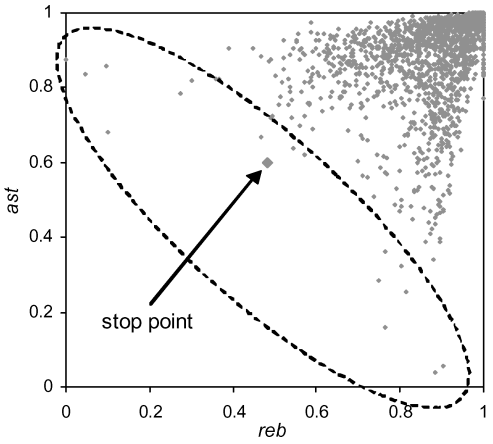


Fig. 13. Normalized distribution of points in the *reb-ast* subspace of the NBA dataset. The ellipse encloses the anti-correlated skyline front. Also shown is the stop point used by symmetric sorting functions.

Color. The nine-dimensional Color dataset, which we downloaded from the UCI KDD site at `kdd.ics.uci.edu`, include 68, 040 objects, where each tuple represents the first three moments of the RGB color distribution of a color image.

Household. This dataset (available at `www.ipums.org`) consists of 127, 931 six-dimensional tuples. Each tuple represents the money spent in one year by an American family for six different types of expenditures (e.g., electricity, gas, phone). Apparently, this is a positively correlated dataset, since attribute values are all likely to assume higher values for families with a higher income.

EEG. Our largest dataset, available at the UCI KDD site, contains 2, 824, 483 64-dimensional vectors, where each component represents a measurement obtained from an EEG electrode placed on a subject's scalp. For our experiments, we only retained the first six components of the vectors.

All real datasets were normalized on-the-fly in the [0, 1] range: for each attribute A_j normalization was achieved by using in any sorting function the expression $(A_j - A_{j,\min})/(A_{j,\max} - A_{j,\min})$ in place of A_j (or complementing the expression to 1 if A_j had to be maximized), where $A_{j,\max}$ and $A_{j,\min}$ are the maximum and the minimum value, respectively, of attribute A_j in the dataset.

Unless otherwise stated, all experiments on real datasets were performed by averaging results on all subspaces at a given dimensionality d. For instance, results at $d = 4$ for the NBA dataset are the average of executing the experiments on all the $\binom{6}{4} = 15$ four-dimensional subspaces of this dataset.

All datasets were bulk-loaded into the DB2 Universal DataBase V8.1, running on a Pentium IV 3.4 GHz PC equipped with 512 MB of main memory and an 80GB Samsung SP0812C hard drive, under the Windows XP Professional operating system. SaLSa and SFS were implemented in Java on a client machine. Since in all the experiments the skyline was sufficiently small to fit in main memory, no temporary file was ever needed (i.e., both SaLSa and SFS always completed in a single pass).

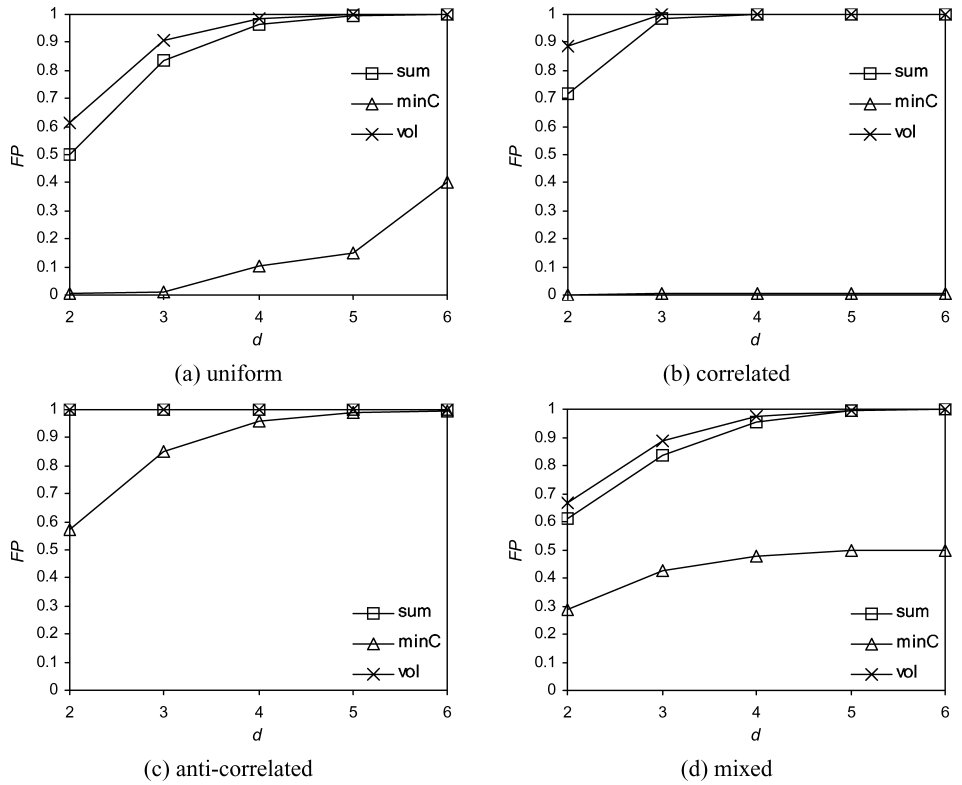


Fig. 14. Fraction of fetched points (*FP*) versus dimensionality d ($n = 500K$, synthetic datasets).

7.1 The Effect of Sorting Functions

In this first experiment, we aim to establish the performance of the vol, sum, and minC functions in limiting the input relation. Figures 14 (synthetic datasets) and 15 (real datasets) show the fraction of fetched points, *FP*, as a function of the number of skyline attributes, d.

From the figures, it is evident that the minC function largely outperforms its competitors in all scenarios, and that even at six dimensions it is able to prune a large part of the input stream. For instance, on the Color dataset, only 42.9% of the points (i.e., 29, 217 out of 68, 040) have to be read to determine a complete six-dimensional skyline. At lower dimensionalities, results are more impressive: a two-dimensional skyline on the NBA dataset requires only 3.8% of the input to be fetched, that is, 678 points out of 17, 791. Results on Household show that this dataset is indeed positively correlated, as can be argued from the comparison of graphs in Figure 15(c) and Figure 14(b). On datasets of this kind, for which finding the skyline can be considered a relatively easy task, the performance of minC is indeed excellent and almost independent of the number of attributes. Even at $d = 6$, SaLSa with minC reads only 0.93% (=1,197/127,931) and 0.62% (=3,116/500,000) of the Household and the correlated synthetic data, respectively. On EEG, which is the largest dataset we use,

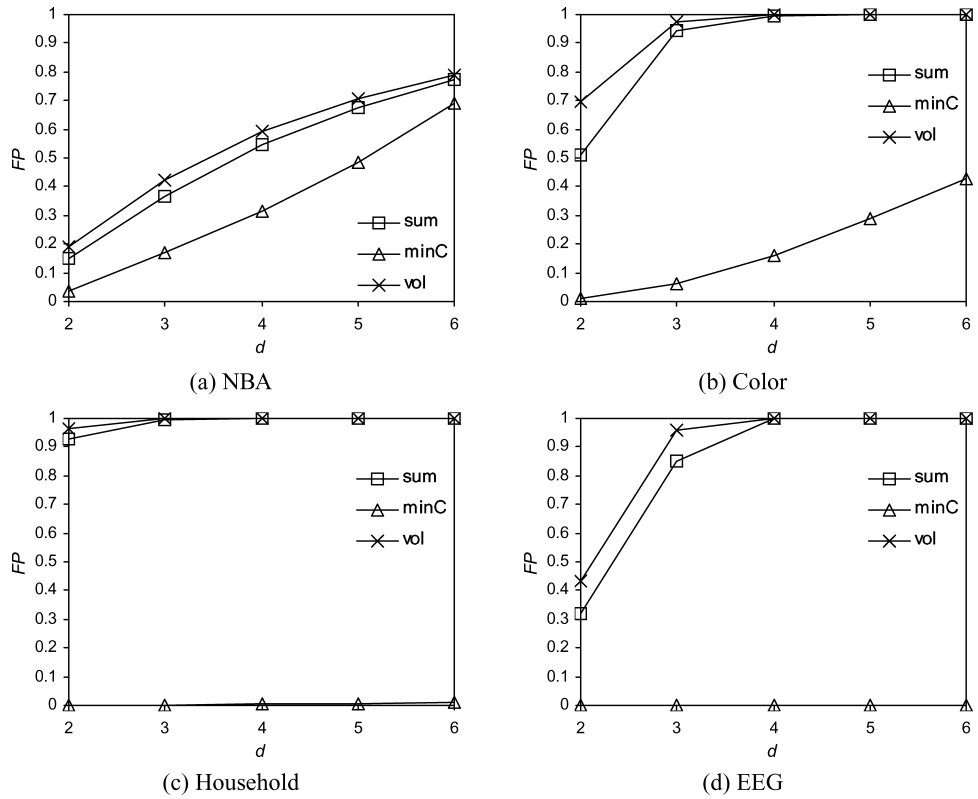


Fig. 15. Fraction of fetched points (*FP*) versus dimensionality d (real datasets).

the pruning capability of minC reaches its maximum: for instance, when $d = 6$ only 1,418 points out of more than 2.8 millions are fetched to compute a skyline that contains 56 points!

The only dataset on which minC is not effective in limiting the input stream is the anti-correlated one, starting from $d = 4$ (see Figure 14(c)). However, it is known that skyline queries on anti-correlated datasets are almost meaningless for medium-large values of d, since they are likely to return far too many points. For instance, when $d = 6$, the skyline contains about 70,000 points. Nonetheless, if the dataset consists of an anti-correlated "front" and of other worse data, SaLSa is still applicable with success. This is derived from the results on the mixed dataset (Figure 14(d)), where the minC function is able to get rid of *all* the data behind the anti-correlated front, which in our setting accounts for 50% of the input.

Concerning the behavior of the sum and vol functions, from all graphs it is evident that the former always limits more than the latter, even if differences are not so impressive as if compared with minC. This confirms the analysis in Section 4.1 and suggests that any vol[m] function, with $1 < m < \infty$, will yield similar results.

Figure 16 shows how *FP* depends on the cardinality of the dataset, n, when $d = 4$ (similar results are observed for other dimensionalities). On all datasets it

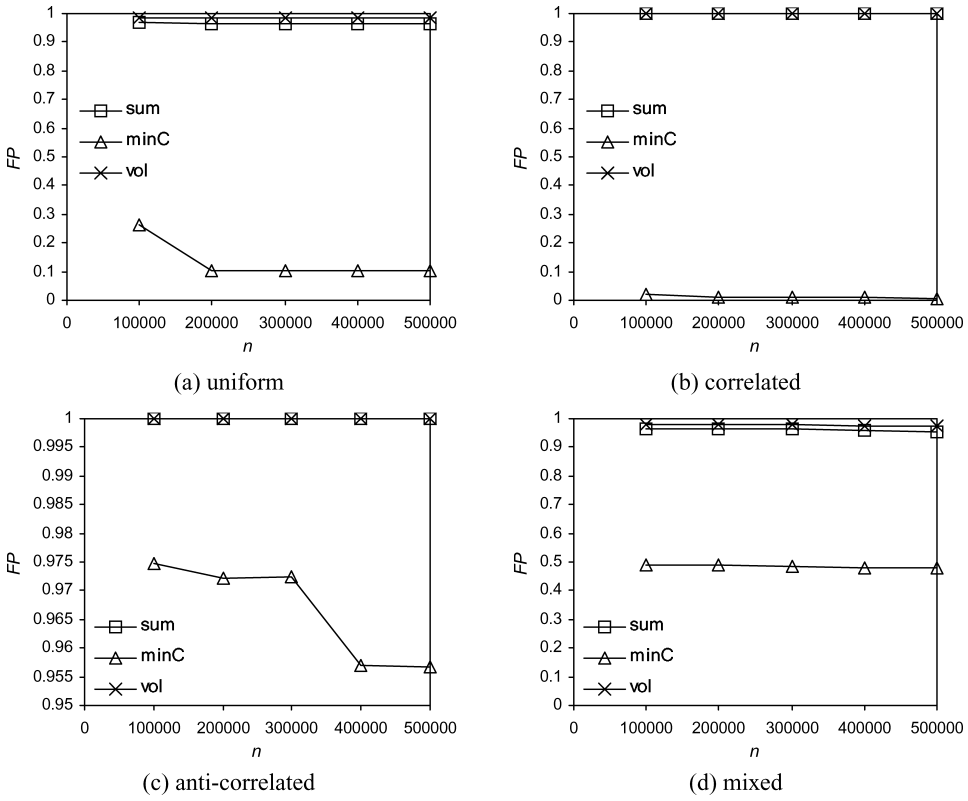


Fig. 16. Fraction of fetched points (FP) versus dataset cardinality n ($d = 4$, synthetic datasets).

is confirmed that having more points makes SaLSa more effective, as explained in Section 5.1.

We now turn to studying how a different sorting criterion can influence the number of dominance tests, DT. Our first experiment aims to investigate if the variant of minC, in which current skyline points are scanned in backward order, as described in Section 5.3, is indeed effective in reducing the number of dominance tests with respect to the basic (forward) case. Results for all the eight datasets at $d = 4$ are shown in Table I.

In all cases but one, the backward version is highly beneficial, being able to save up to 89% of the overall number of comparisons. The only dataset for which reversing the scan order of skyline points leads to increase the number of dominance tests is EEG. Although the relative increase is high (about 200%), in absolute terms, differences are almost irrelevant, even considering that the EEG dataset consists of more than 2.8 millions points.

It is interesting to observe that the relative performance of the backward variant improves with the "difficulty" of the filter phase, as can be argued by looking at the results for synthetic datasets. Similar trends are observed at all dimensionalities. For this reason, in the following, we only consider minC in which current skyline points are scanned using the backward variant.

Table I. Number of Dominance Tests: minC Variants for Scanning the Current Skyline Set ($d = 4$)

	Datasets							
	uniform	corr.	anti-corr.	mixed	NBA	Color	House.	EEG
forward	1,317,935	45,723	431,529,348	212,458,072	11,604	631,785	2,998	2,918
backward	368,434	22,127	47,199,335	28,594,851	10,164	135,369	2,310	8,760
saving	72.0%	51.6%	89.1%	86.5%	12.4%	78.6%	22.9%	−200.2%

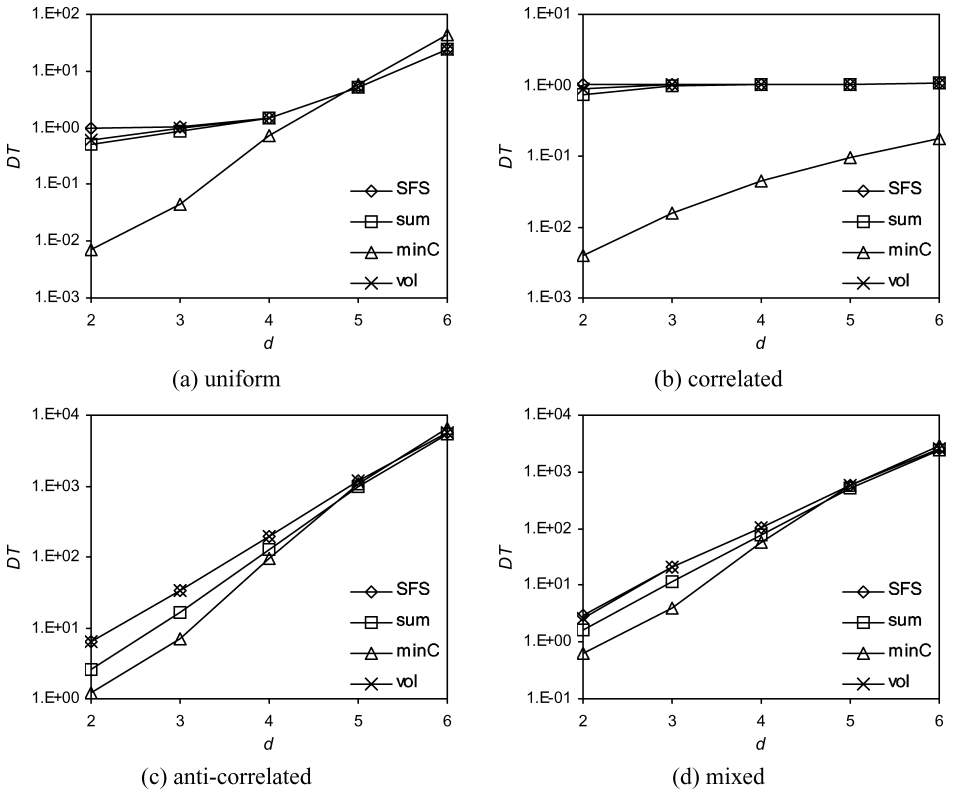


Fig. 17. Number of dominance tests (DT) versus dimensionality d ($n = 500K$, synthetic datasets).

Figure 17 shows that on synthetic datasets the minC sorting function, which is the clear winner when limiting the input is concerned, is also the best alternative for minimizing the number of comparisons. This is evident for the correlated datasets. For other datasets this holds as long as $d \leq 4$, whereas at higher dimensionalities differences are hardly appreciable and all methods perform similarly. Considering the sum and vol functions, the former is always slightly better than the latter.

Results on real datasets, shown in Figure 18, provide a similar picture. On NBA, minC is always far better than other sorting criteria, even if relative differences reduce when d grows. For instance, at $d = 2$, minC performs 823 tests and SFS 18,643, a relative 2,163% overhead, whereas at $d = 6$, the overhead of SFS reduces to 38.7% (53,105 versus 38,293 tests). Similar, but with much

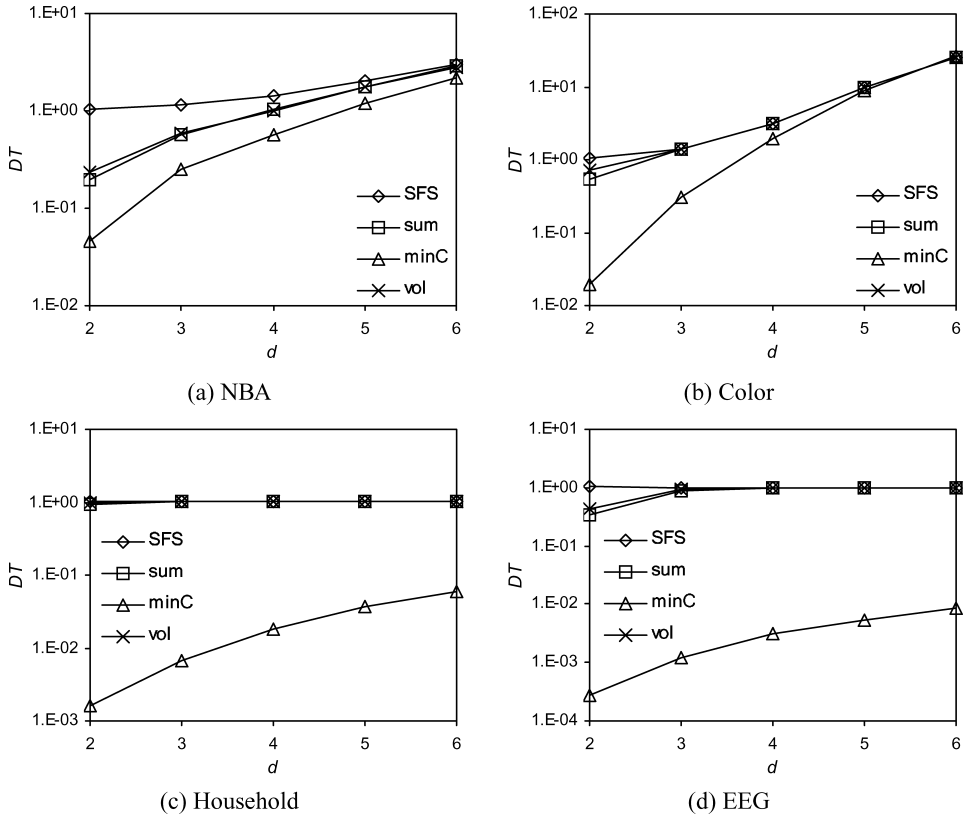


Fig. 18. Number of dominance tests (DT) versus dimensionality d (real datasets).

more marked differences, is the behavior on Household, for which minC at six dimensions, say, performs only 7,657 dominance tests against the 129,999 executed by SFS and the other sorting functions. Analogous considerations can be done for the EEG dataset. Finally, on the Color dataset, we observe that minC behaves as with uniform datasets, being the best method up to $d \leq 4$ and comparable to the others for higher d values.

Finally, Figure 19 analyzes how DT varies with the cardinality of the dataset ($d = 4$ is used in this experiment). Remind that DT is the average number of dominance tests per object in the dataset. For all methods the general trend is that DT decreases with n, which is also to say that the overall number of tests, that is, $DT \cdot n$, grows less than linearly with n.

7.2 Accuracy of the Model

Now we turn to analyzing the accuracy of the model, introduced in Section 5.2, in predicting the fraction of fetched points when using the minC function. To this end, we compare actual results with estimates obtained from a *minskew* histogram [Acharya et al. 1999] in which each coordinate is initially split into 10 intervals. All experiments are run using an integration step $\Delta l = 0.01$.

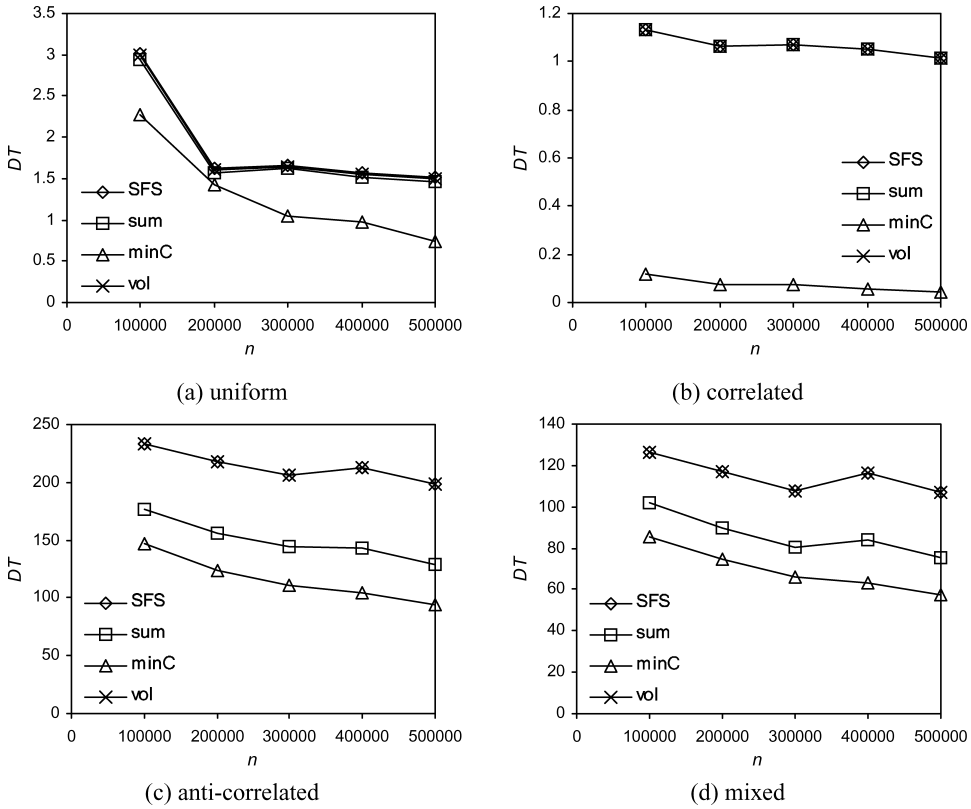
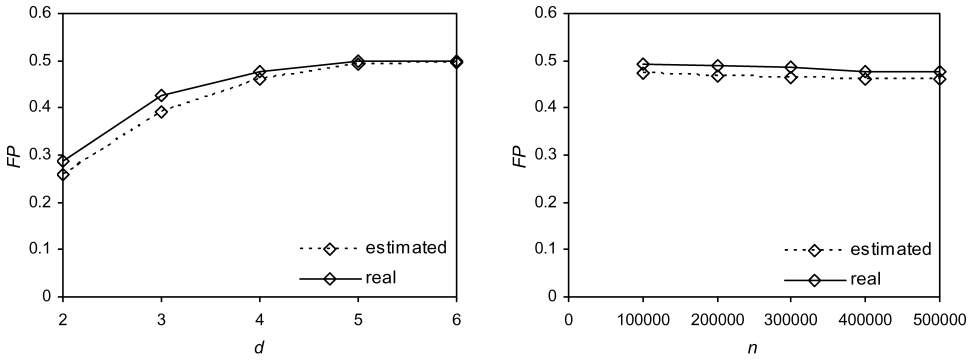


Fig. 19. Number of dominance tests (DT) versus dataset cardinality n ($d = 4$, synthetic datasets).

Figure 20 shows that the model is extremely accurate, errors never exceeding 10%. This holds regardless of the dimensionality and of the cardinality of the dataset. Results refer to mixed-distributed data, accuracy on other synthetic datasets being similar (for the uniform datasets refer to Section 5.1).

Figure 21 validates the model on the real datasets. For these experiments, rather than averaging results over all possible subspaces as done in other experiments, we selected one specific subspace for each value of d in the range $[2, 6]$. This is also to say that the graphs show the actual model's predictions (on the selected subspaces), rather than average ones. From these premises, we can indeed conclude that the model's estimates are excellent. The only case in which the model yields poor *relative* estimates is for the Household dataset. The basic reason lies in the assumption of uniform distribution of points within each bucket (refer to Section 5.2), which for correlated datasets (as Household appears to be) leads to pessimistic estimates. Nonetheless, *absolute*, as contrasted to relative, errors of the model are low even for the Household dataset, at $d = 6$ being only 0.06.

The applicability of our model in a real scenario depends on the dimensionality of the dataset and on the actual data distribution. This is because memory requirements of *minskew*, as well as of any other state-of-the-art histogram,



(a) FP versus d (b) FP versus n

Fig. 20. Estimated and actual fraction of fetched points (mixed dataset, minC sorting function); (a) versus d ($n = 500K$); (b) versus n ($d = 4$).

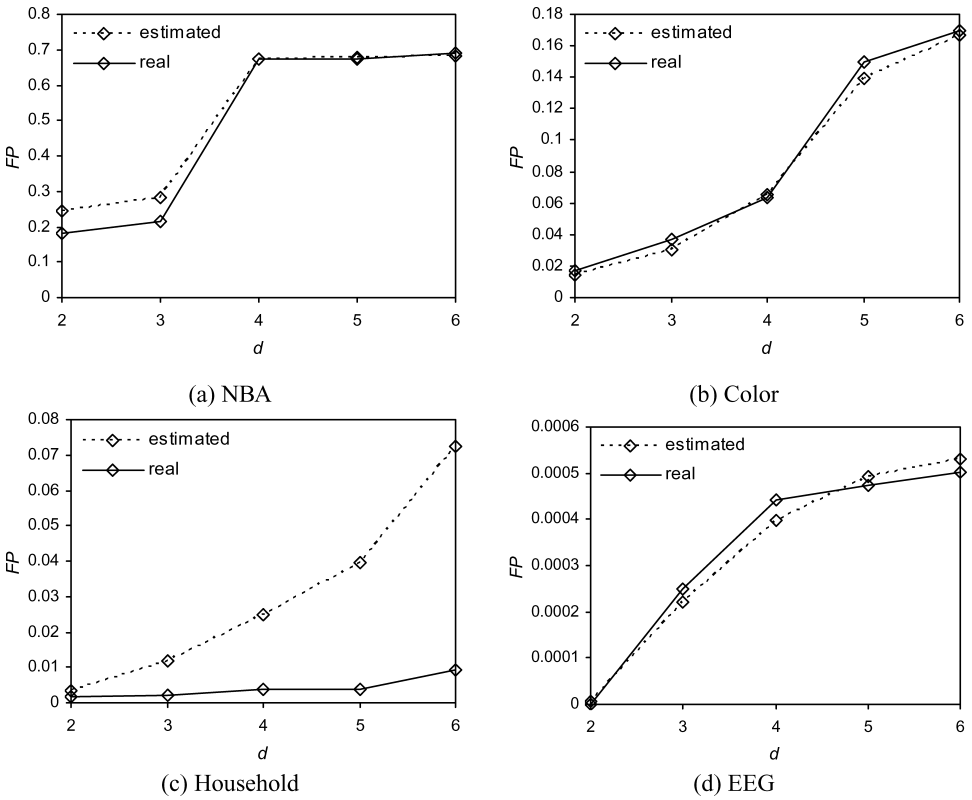


Fig. 21. Estimated and actual fraction of fetched points versus d (real datasets, minC sorting function).

Table II. Elapsed Time on Synthetic Datasets ($n = 500K$, $d = 4$. Times are in Seconds)

	Uniform			Mixed		
	sum	minC	vol	sum	minC	vol
Sorting	1.57	1.79	1.64	2.15	2.40	2.44
Fetching	3.85	0.42	3.94	3.82	1.91	3.90
Filtering	2.11	0.10	2.21	5.02	2.57	6.26
Total	7.53	2.31	7.79	10.99	6.88	12.60

Table III. Elapsed Time on Real Datasets ($d = 4$)

	NBA			Color			Household			EEG		
	sum	minC	vol	sum	minC	vol	sum	minC	vol	sum	minC	vol
Sorting	0.02	0.03	0.02	0.07	0.11	0.07	0.34	0.40	0.36	39.97	48.65	40.33
Fetching	0.08	0.03	0.09	0.54	0.08	0.55	1.02	0.01	1.02	22.59	0.01	22.59
Filtering	0.03	0.01	0.04	0.35	0.06	0.34	0.52	0.01	0.52	23.39	0.01	24.46
Total	0.13	0.07	0.15	0.96	0.25	0.96	1.88	0.42	1.90	85.95	48.67	87.38

in the worst case, grow exponentially with d. Our experiments, whose results confirm those by Papadias et al. [2005], who used *minskew* histograms to compute an *approximate skyline*, show that good accuracy *and* moderate memory consumption can be obtained for all datasets as long as $d \leq 3 \sim 4$, whereas, for higher d values, the number of buckets can become impractical for some datasets (e.g., anti-correlated). We remark that this is not a limit of our model, yet it depends on the intrinsic complexity of accurately modelling multidimensional datasets.

7.3 Elapsed Times

Now we present results on total elapsed times for the different symmetric sorting functions, which also allows a comparison between the different cost components to be made. For these experiments, the size of each point becomes relevant, since it can influence both sorting and communication costs. For all datasets, as done in Börzsönyi et al. [2001], we added a dummy attribute so as to have a point size of 100 bytes.

Table II shows how the total elapsed time spent for the four-dimensional uniform and mixed datasets breaks down into sorting, fetching, and filtering components. Similar results are shown in Table III for the real datasets. For these experiments, we used a client with the same processing capabilities as the server and the average throughput between the two machines was about 100Mb/sec.

Looking at results, one can draw the conclusion that the fetching time can have a major impact on the overall performance, in our setting even superior, for all datasets but mixed and EEG, to that of the filtering phase. On the other hand, sorting, unless the other two phases are cheap to execute, only accounts for a minor percentage of the overall time. It has to be observed that sorting times of minC are usually slightly higher than those of both sum and vol. This is due to the overhead of invoking a DB2 User Defined Function (UDF) for computing the minimum of d values and also to the secondary sorting criterion of minC.

Table IV. Fetching+Filtering Times of minC with Respect to vol ($d = 4$)

Uniform	Mixed	NBA	Color	Household	EEG
8.46%	44.13%	30.78%	16.00%	0.30%	0.02%

Table V. Simulation of Different Network and Client Scenarios (Color dataset, $d = 4$)

	Slower (10%) client connection			Slower (10%) client CPU speed		
	sum	minC	vol	sum	minC	vol
Sorting	0.07	0.11	0.07	0.07	0.11	0.07
Fetching	5.44	0.84	5.48	0.54	0.08	0.55
Filtering	0.35	0.06	0.34	3.46	0.58	3.43
Total	5.86	1.01	5.89	4.07	0.77	4.05

Table IV summarizes these results by showing the sum of fetching and filtering times for the minC function, normalized to that obtained when using vol. The table ignores sorting times so as to emphasize the speedup obtainable from a sorting criterion that allows most of the input data to be pruned.

In the next experiment, we simulate two different scenarios in order to better appreciate how the different cost components vary with them. Table V shows results for the Color dataset at $d = 4$. In the first case, we simulate a network connection with an effective transfer rate of 10Mb/sec, that is, 10 times slower than that in Table III, whereas in the second scenario we have a fast 100Mb/sec connection, yet the client processing power is one-tenth of the reference case. In both cases it is evident how minC can substantially limit the total time, which is only the 17.3% and the 19%, respectively, of that required by the other two functions.

7.4 Asymmetric Sorting Functions

We now turn to analyzing the effects of using the ΔminC asymmetric function for sorting points. For histogram construction, we consider the same setting as in Section 7.2.

In Figure 22, we report values of fetched points on the real datasets for the ΔminC (denoted Delta-minC in the figure) and the minC functions. With the only exception of Household, for which values of *FP* are extremely low even when using minC, and no difference can be appreciated, it can be seen that adapting the sorting criterion to the actual data distribution is indeed highly effective, in particular as d grows. For instance, at six dimensions, ΔminC fetches only 21.4% of the NBA dataset, whereas minC needs to read 69.1% of the points. Values change to 31.5% and 42.9%, respectively, for the Color dataset, and to 0.02% and 0.05%, respectively, for the EEG dataset.

The hypothesis that asymmetric sorting is more effective with "harder" datasets is also supported by results in Figure 23 on the number of dominance tests. In particular, while for Color and Household datasets no significative improvements with respect to minC can be observed, for the NBA dataset, ΔminC is indeed able to save a large percentage of comparisons among points (92.9% at 2 and 56.7% at six dimensions, respectively).

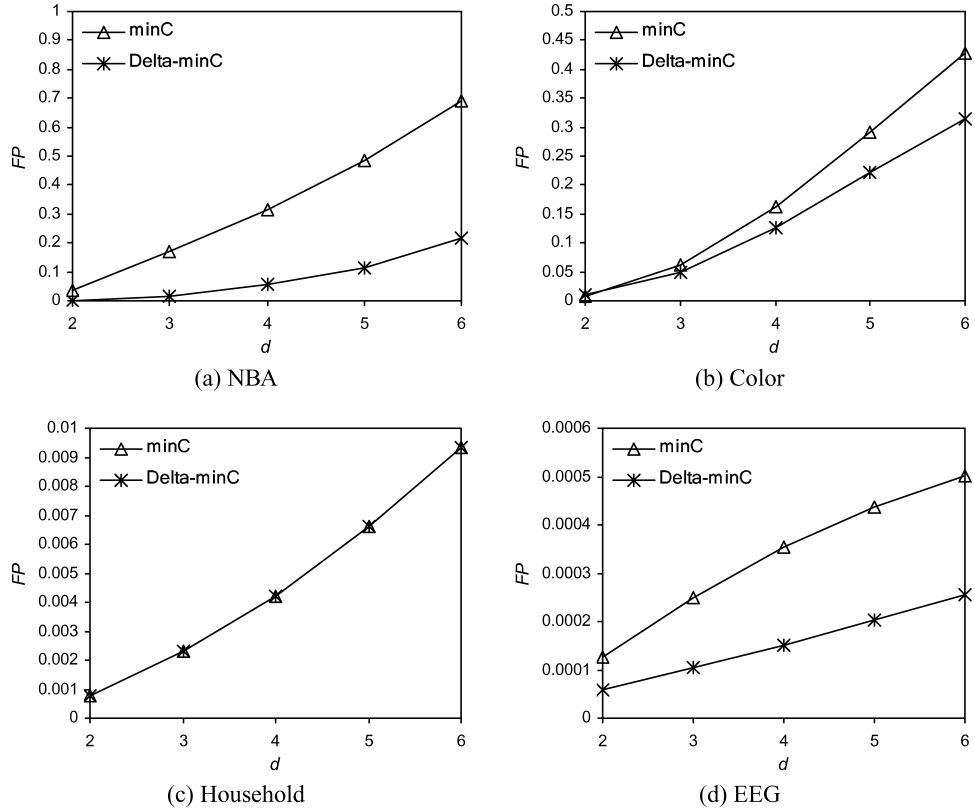


Fig. 22. Fraction of fetched points (*FP*) versus dimensionality d for the symmetric minC and the asymmetric ΔminC functions (real datasets).

Figure 24 analyzes how good the estimates provided by histograms are in guessing the optimal shift vector. As in Figure 21, for each value of d in the range $[2, 6]$, results refer to a single subspace of that dimensionality. Graphs labelled Delta-minC-opt (ΔminC_{opt}) refer to the ideal case in which the exact location of $\mathbf{p}_{opt}$ is known, and provide a lower bound to the number of points to be fetched by ΔminC. For all datasets but Household, ΔminC is quite close to the intrinsic SaLSa performance limit, which is about one order of magnitude less than can be obtained from the (symmetric) minC function.

The relatively poor behavior of ΔminC with Household has the same explanation as the one given to comment on the results in Figure 21, that is, the high positive correlation exhibited by this dataset invalidates the assumption of uniform distribution within histogram buckets.

8. EXTENDING SALSA WITH MULTIPLE STOP POINTS

Before concluding this article, in this section, we provide some clues on how the basic principle exploited by SaLSa to prune points from a data stream can be generalized so as to work with *multiple stop points* (MSP's). In particular, we describe which changes are needed to the logic of SaLSa when MSP's are

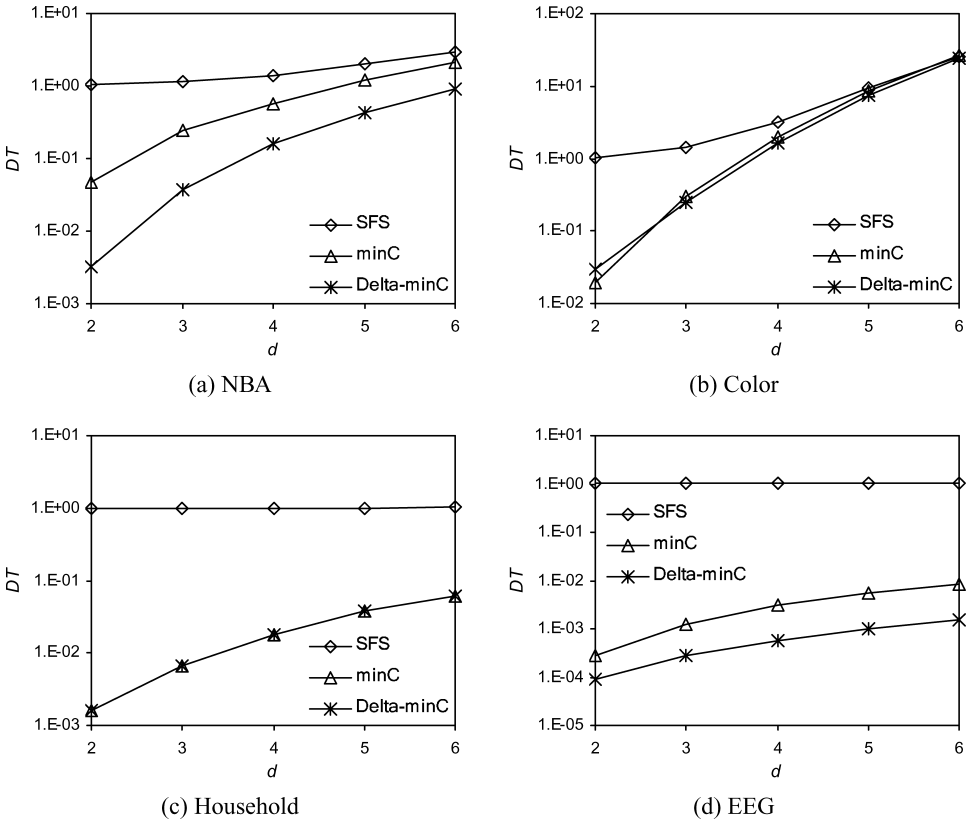


Fig. 23. Number of dominance tests (DT) versus dimensionality d for the symmetric minC and the asymmetric ΔminC functions (real datasets).

used, discuss what this extension, which we call SaLSa-MSP, requires from a computational point of view, and present some experimental results. To keep the discussion simple, we will consider only symmetric functions, the generalization to the asymmetric case presenting no particular difficulty, and we will continue to use $\mathbf{p}_{stop}$ to denote the point chosen according to the MiniMax rule in Theorem 2.

To start with, let $\mathcal{S}_{stop} \subseteq \mathcal{S}$ be a nonempty subset of the complete skyline of relation r that is used to stop SaLSa-MSP execution. We call $\mathcal{S}_{stop}$ the *stop set*. Clearly, the case $\mathcal{S}_{stop} = \{\mathbf{p}_{stop}\}$ yields the basic SaLSa version we have considered until now. Then, SaLSa-MSP can be halted when the sorting function $\mathcal{M}$ has reached a level l such that the stop set dominates the unread domain at level l, that is, $\mathcal{S}_{stop} \succ \mathcal{D}(\mathcal{M}, l)$. This is to say that, for each point $\mathbf{p} \in \mathcal{D}(\mathcal{M}, l)$, there exists a point $\mathbf{p}_i \in \mathcal{S}_{stop}$ such that $\mathbf{p}_i \succ \mathbf{p}$.

Let $DR(\mathcal{S}_{stop})$ be the region dominated by points in $\mathcal{S}_{stop}$, that is:

$$DR(\mathcal{S}_{stop}) = \bigcup_{\mathbf{p}_i \in \mathcal{S}_{stop}} DR(\mathbf{p}_i). \tag{20}$$

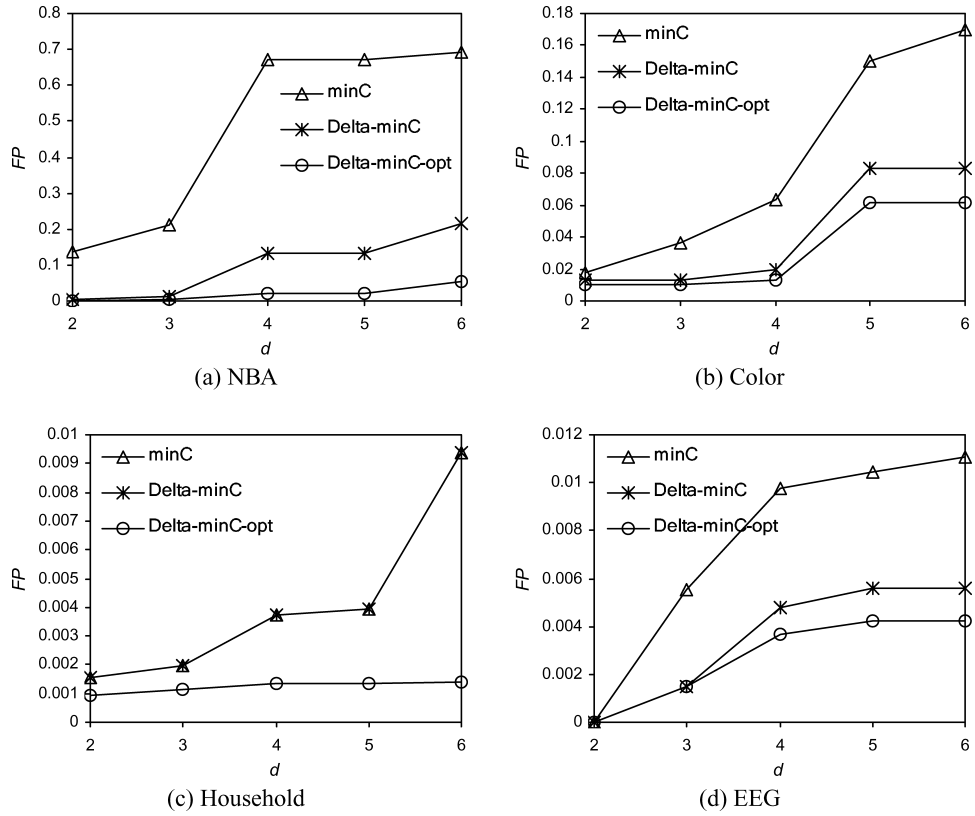


Fig. 24. Fraction of fetched points (*FP*) versus dimensionality d (real datasets). Graphs of ΔminC are derived from histogram estimates, those of ΔminC_{opt} are obtained by using the optimal shift vector defined in Theorem 10.

and denote with $l_{stop}(\mathcal{M}; \mathcal{S}_{stop})$ the stop level corresponding to $\mathcal{S}_{stop}$, that is, the minimum value of l such that $\mathcal{D}(\mathcal{M}, l) \subseteq DR(\mathcal{S}_{stop})$ holds.

Our first result on SaLSa-MSP concerns the optimal minC function.

LEMMA 7. *When the input relation r is sorted using* minC*, it is:*

$$l_{stop}(\text{minC}; \mathcal{S}_{stop}) \geq l_{stop}(\text{minC}; \{\boldsymbol{p}_{stop}\}) = p^+_{stop},$$

with equality attained if and only if $\mathcal{S}_{stop}$ includes $\boldsymbol{p}_{stop}$ or another point $\boldsymbol{p}_i$ such that $p^+_i = p^+_{stop}$.

PROOF. Without loss of generality, assume that $\mathbf{p}_{stop}$ does not lie on the main diagonal of the data space, that is, there exists j such that $p_{stop}[j] < p^+_{stop}$.[16] Consider the point $\mathbf{p}_{\text{diag}} = (p^+_{stop}, \ldots, p^+_{stop})$ and observe that $\mathbf{p}_{stop} \succ \mathbf{p}_{\text{diag}}$. Further, SaLSa can halt just after reading $\mathbf{p}_{\text{diag}}$. For SaLSa-MSP to stop at level p^+_{stop} there must be a point $\mathbf{p}_i \in \mathcal{S}_{stop}$ such that $\mathbf{p}_i \succ \mathbf{p}_{\text{diag}}$. This implies that

[16]If $p_{stop}[j] = p^+_{stop}$ holds for all j, arguments in the proof still apply with minimal changes.

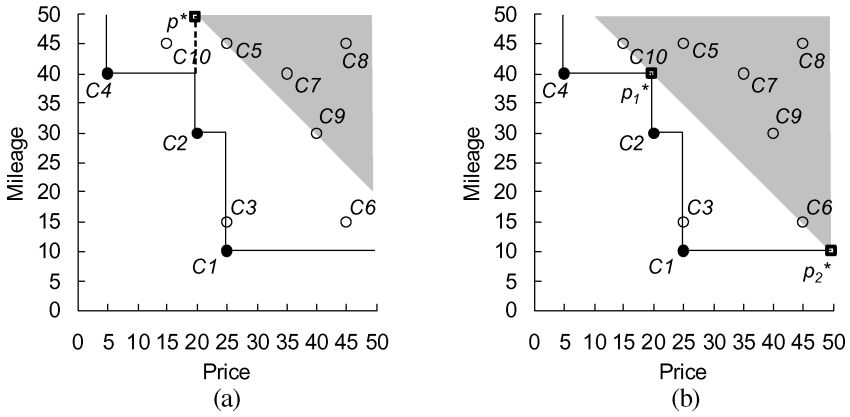


Fig. 25. SaLSa-MSP uses a set $\mathcal{S}_{stop}$ with multiple stop points: In (a) it is $\mathcal{S}_{stop} = \{C1, C2\}$, whereas the complete skyline is used in (b). Solution points of the StopLevel problem are drawn as small squares.

$p_i^+ \leq p_{stop}^+$. The case $p_i^+ = p_{stop}^+$ is covered in the Theorem formulation. The case $p_i^+ < p_{stop}^+$ is ruled out by the very definition of $\mathbf{p}_{stop}$. □

The lemma asserts that having MSP's with the minC function is useless. This is because minC always prunes a hypercubic region of the data space, and the maximal hypercube contained in $DR(\mathcal{S}_{stop})$ coincides with the one dominated by $\mathbf{p}_{stop}$ (if $\mathbf{p}_{stop} \in \mathcal{S}_{stop}$).

On the other hand, for sorting functions other than minC, using more than one stop point can indeed be beneficial. For instance, Figure 25 shows the effects of using MSP's on the UsedCars relation, when the sum function is used (more details on this are given in Example 11). Intuitively, the effectiveness of SaLSa-MSP in limiting the input stream increases by adding more points to the stop set. However, this comes at the price of additional resources needed to determine the value of the stop level $l_{stop}(\mathcal{M}; \mathcal{S}_{stop})$. Note that this evaluation needs to be repeated several times, since the stop level might change (decrease) whenever new skyline points are discovered and added to the stop set.

If one decides to have a stop set with only a few points, how to effectively choose such points becomes an issue. The solution that guarantees that the stop level gets minimized would rebuild $\mathcal{S}_{stop}$ from scratch whenever a new skyline point is discovered. However, the complexity of this approach is prohibitive, even for moderately large skylines. At the other extreme, one could build $\mathcal{S}_{stop}$ in an incremental way and, whenever a new skyline point $\mathbf{p}$ is added to $\mathcal{S}$, check if the stop level can be decreased by inserting $\mathbf{p}$ also in $\mathcal{S}_{stop}$ in place of another point. However, this heuristic approach provides no guarantee that the stop level gets minimized.

Considering the observations, we conclude that choosing a "good" stop set seems to be a difficult problem, which we plan to deeply investigate in a future work. For this reason, in the following, we omit considering such issues and concentrate on the problem that *any* strategy for choosing a stop set has to deal with, that is, how to determine the value of the stop level. The basic scenario to analyze is when points are sorted using the linear sum function, in which case

computing the stop level amounts to solve a linear optimization problem. The proof of the following result appears in the Electronic Appendix.

THEOREM 12. *When the input relation r is sorted using the* sum *function and the stop set is $\mathcal{S}_{stop}$, the stop level $l_{stop}(\mathsf{sum}; \mathcal{S}_{stop})$ is the result of the following* StopLevel *disjunctive linear optimization problem:*

$$\textit{Maximize}: \quad l = \sum_{j=1}^{d} p[j], \tag{21}$$

$$\textit{subject to}: \quad \bigvee_{j=1}^{d} p[j] \leq p_i[j] \qquad \forall \mathbf{p}_i \in \mathcal{S}_{stop} \tag{22}$$

$$0 \leq p[j] \leq 1 \qquad \forall j = 1, \ldots, d. \tag{23}$$

Example 11. Figure 25 shows the effects of using MSP's on the Used-Cars relation. In Figure 25(a) it is $\mathcal{S}_{stop} = \{\text{C1, C2}\}$, and the stop level is $l_{stop}(\mathsf{sum}; \mathcal{S}_{stop}) = 70$. The (unique) solution point of the StopLevel problem is point $\mathbf{p}^* = (20, 50)$. When all skyline points are used, that is, $\mathcal{S}_{stop} = \mathcal{S} = \{\text{C1, C2, C4}\}$, the stop level drops down to 60, as shown in Figure 25(b). In this case, there are two solution points, $\mathbf{p}_1^* = (20, 40)$ and $\mathbf{p}_2^* = (50, 10)$, respectively.

How complex is it to determine the stop level? As the following theorem proves, in the general case the problem is a difficult one.

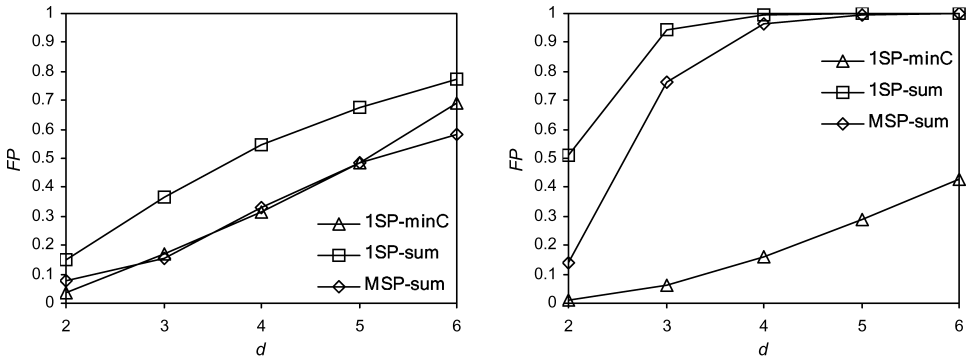
THEOREM 13. *The* StopLevel *problem is NP-hard.*

Although the StopLevel problem is intrinsically difficult, from the proof of Theorem 13 presented in the Electronic Appendix, one sees that NP-hardness is due to the number of dimensions d, rather than to the cardinality of the stop set. Thus, for moderately low values of d, one could still expect that the stop level can be efficiently computed.

We implemented SaLSa-MSP by using the CPLEX optimization package[17] for solving the StopLevel problem and decided to always have the stop set coincident with the complete skyline, that is, $\mathcal{S}_{stop} = \mathcal{S}$. The latter aims to measure the maximum benefit, in terms of pruned points, obtainable from SaLSa-MSP. In this way, we are able to restrict the range in which to search for optimal trade-offs, the other extreme being the basic SaLSa version with 1 stop point.

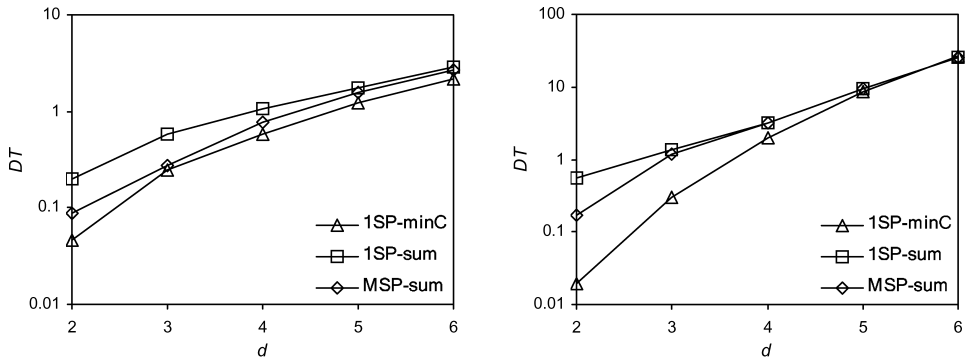
Figures 26 and 27 show, respectively, the fraction of fetched points and the number of dominance tests for SaLSa-MSP using sum (denoted MSP-sum in the figures), SaLSa using sum (1SP-sum), and SaLSa using minC (1SP-minC). Results are rather unexpected, in that they lead to the overall conclusion that using minC with a single stop point is definitely the best choice on the Color dataset, and it leads to performance comparable to that of SaLSa-MSP on NBA data. Note that at six dimensions, the skyline of NBA includes 102 points, that of Color 1160, and all of them are also in the stop set $\mathcal{S}_{stop}$. Even disregarding the unavoidable overhead due to solving StopLevel with so many points, it is apparent that, on these datasets, using MSP's is not effective at all. We conjecture that this is mainly due to the poor limiting power (with respect to minC) the

[17] CPLEX is a commercial product distributed by ILOG, see www.ilog.com.



(a) NBA (b) Color

Fig. 26. Fraction of fetched points (*FP*) versus dimensionality d for SaLSa and SaLSa-MSP.



(a) NBA (b) Color

Fig. 27. Number of dominance tests (*DT*) versus dimensionality d for SaLSa and SaLSa-MSP.

function sum has on such datasets. Further, how skyline points are distributed on the data space could be another relevant factor to explain the bad behavior of SaLSa-MSP. Although we do not rule out the possibility that using MSP's can outperform SaLSa using minC on some datasets (possibly using a sorting function other than sum), understanding when this can be the case remains an open problem.

9. CONCLUSIONS

Skyline queries have recently emerged as a major tool for extracting interesting objects from multidimensional datasets, and many algorithms have been proposed so far for their efficient evaluation. Among them, generic (sequential) algorithms play a major role because of their general applicability, both in the context of a database system, as well as when the skyline logic runs in a standalone application. With the aim of providing efficient support to the evaluation of skylines, in this article, we have introduced the SaLSa algorithm, whose innovative feature is the ability of computing the result without having to apply dominance tests to all the objects (points) in the input relation. This is

achieved by presorting the data using a monotone *limiting* function, and then checking that unread data are all dominated by a so-called *stop point*.[18]

Experimental results show that SaLSa is indeed effective in reducing the number of points to be read, thus also particularly attractive when the skyline logic runs on a client with a limited bandwidth connection. We have experimentally and analytically shown that sorting on the minimum coordinate value of points (the minC function) is optimal, as long as one considers symmetric sorting functions, and that performance can be accurately predicted. We have also provided clear evidence that *reversing* the order in which current skyline points are considered for filtering new read data, is a highly effective strategy for reducing the number of comparisons performed by minC, which makes this function perform strictly better than the entropy method used by the SFS algorithm [Chomicki et al. 2003]. In the course of investigating the performance limits of SaLSa, we have also analyzed the general case in which an asymmetric function is used to sort points, and shown how performance can be further improved given information on the actual data distribution.

An interesting extension of our work would be to combine SaLSa's principles with those of the LESS algorithm [Godfrey et al. 2005], which integrates part of the skyline computation within the initial sort phase. Since LESS shares with SFS the entropy-based sorting criterion, it is expected that using minC (as well as its asymmetric version) would lead to substantial performance gains, even in this integrated scenario. Along this direction, trying to also incorporate principles of *incremental sorting* [Paredes and Navarro 2006] appears to be promising. Another interesting research direction would be to investigate an ad hoc method for histogram construction, so as to make our analytical model applicable even to higher-dimensional datasets. Finally, a deeper investigation of the multiple stop points variant we have introduced in Section 8 is needed to properly understand its full potentialities.

ELECTRONIC APPENDIX

The Electronic Appendix for this article can be accessed in the ACM Digital Library.

ACKNOWLEDGMENTS

We would like to thank the anonymous referees for their insightful suggestions.

REFERENCES

ACHARYA, S., POOSALA, V., AND RAMASWAMY, S. 1999. Selectivity estimation in spatial databases. In *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data (SIGMOD)*. Philadelphia, PA, ACM, New York, NY, 13–24.

[18]Incidentally, the idea of limiting the amount of data to be read by exploiting the value of a monotone function is also used by the SUBSKY algorithm [Tao et al. 2006] for computing skylines in subspaces. However, SUBSKY is based on a apriori, thus fixed, ordering for each attribute, thus it cannot be used for arbitrary preference specifications (e.g., distance to a target point), nor can it be used when the input is the result of some other relational operation.

BALKE, W.-T., GÜNTZER, U., AND ZHENG, J. X. 2004. Efficient distributed skylining for web information systems. In *Proceedings of the 6th International Conference on Extending Database Technology (EDBT)*. Lecture Notes in Computer Science, vol. 2992, Springer, Berlin, Heidelberg, New York, 256–273.

BARTOLINI, I., CIACCIA, P., ORIA, V., AND ÖZSU, T. 2004. Integrating the results of multimedia subqueries using qualitative preferences. In *Proceedings of the 10th International Workshop on Multimedia Information Systems (MIS)*. College Park, MD. Lecture Notes in Computer Science, vol. 2992, Springer, Berlin, Heidelberg, New York, 66–75.

BARTOLINI, I., CIACCIA, P., ORIA, V., AND ÖZSU, T. 2007. Flexible integration of multimedia subqueries with qualitative preferences. *Multimed. Tools Appl. 33*, 3, 275–300.

BARTOLINI, I., CIACCIA, P., AND PATELLA, M. 2006. SaLSa: computing the skyline without scanning the whole sky. In *Proceedings of the 2006 ACM CIKM International Conference on Information and Knowledge Management*. Arlington, VA, ACM, New York, NY, 405–414.

BÖRZSÖNYI, S., KOSSMANN, D., AND STOCKER, K. 2001. The skyline operator. In *Proceedings of the 17th International Conference on Data Engineering (ICDE)*. Heidelberg, Germany, IEEE Computer Society, Washington, DC, 421–430.

BUCHTA, C. 1989. On the average number of maxima in a set of vectors. *Inform. Process. Lett. 33*, 2, 63–65.

CHAUDHURI, S., DALVI, N., AND KAUSHIK, R. 2006. Robust cardinality and cost estimation for skyline operator. In *Proceedings of the 22nd International Conference on Data Engineering (ICDE)*. Atlanta, GA. IEEE Computer Society, Washington, DC, 64.

CHOMICKI, J. 2003. Preference Formulas in Relational Queries. *ACM Trans. Database Syst. 28*, 4, 427–466.

CHOMICKI, J., GODFREY, P., GRYZ, J., AND LIANG, D. 2002. Skyline with presorting. Tech. Rep. CS-2002-04, York University, Toronto, ON.

CHOMICKI, J., GODFREY, P., GRYZ, J., AND LIANG, D. 2003. Skyline with presorting. In *Proceedings of the 19th International Conference on Data Engineering (ICDE)*. Bangalore, India. IEEE Computer Society, Washington, DC, 717–816.

GAREY, M. R. AND JOHNSON, D. S. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman.

GODFREY, P. 2004. Skyline cardinality for relational processing. In *Proceedings of the 3rd International Symposium on Foundations of Information and Knowledge Systems (FoIKS)*. Lecture Notes in Computer Science, vol. 2942, Springer, Berlin, Heidelberg, New York, 78–97.

GODFREY, P., SHIPLEY, R., AND GRYZ, J. 2005. Maximal vector computation in large data sets. In *Proceedings of the 31st International Conference on Very Large Data Bases (VLDB)*. Trondheim, Norway. Morgan Kaufmann, San Francisco, CA, 229–240.

KIESSLING, W. 2002. Foundations of preferences in database systems. In *Proceedings of the 28th International Conference on Very Large Data Bases (VLDB)*. Hong Kong, China. Morgan Kaufmann, San Francisco, CA, 311–322.

KOSSMANN, D., RAMSAK, F., AND ROST, S. 2002. Shooting stars in the sky: an online algorithm for skyline queries. In *Proceedings of the 28th International Conference on Very Large Data Bases (VLDB)*. Hong Kong, China. Morgan Kaufmann, San Francisco, CA, 275–286.

PAPADIAS, D., TAO, Y., FU, G., AND SEEGER, B. 2003. An optimal and progressive algorithm for skyline queries. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data (SIGMOD)*. San Diego, CA. ACM, New York, NY, 467–478.

PAPADIAS, D., TAO, Y., FU, G., AND SEEGER, B. 2005. Progressive skyline computation in database systems. *ACM Trans. Database Syst. 30*, 1, 41–82.

PAREDES, R. AND NAVARRO, G. 2006. Optimal incremental sorting. In *Proceedings of the 8th Workshop on Algorithm Engineering and Experiments (ALENEX) and the 3rd Workshop on Analytic Algorithmics and Combinatorics (ANALCO)*. Miami, FL. SIAM Press, Philadelphia, PA, 171–182.

PREPARATA, F. P. AND SHAMOS, M. I. 1985. *Computational Geometry—An Introduction*. Springer.

SAKAMOTO, H. 1943. On the distributions of the product and the quotient of the independent and uniformly distributed random variables. *Tohoku Math. J. 49*, 243–260.

TAN, K.-L., ENG, P.-K., AND OOI, B. C. 2001. Efficient progressive skyline computation. In *Proceedings of the 27th International Conference on Very Large Data Bases (VLDB)*. Rome, Italy, Morgan Kaufmann, San Francisco, CA, 301–310.

TAO, Y., XIAO, X., AND PEI, J. 2006. SUBSKY: efficient computation of skylines in subspaces. In *Proceedings of the 22nd International Conference on Data Engineering (ICDE)*. Atlanta, GA. IEEE, Computer Society, Washington, DC, 65.

Received June 2007; revised March 2008; accepted July 2008