Efficient and Adaptive Distributed Skyline Computation

George Valkanas[1] and Apostolos N. Papadopoulos[2]

[1] Dept. of Informatics and Telecommunications, University of Athens, Athens, Greece
gvalk@di.uoa.gr
[2] Dept. of Informatics, Aristotle University of Thessaloniki, Thessaloniki, Greece
papadopo@csd.auth.gr

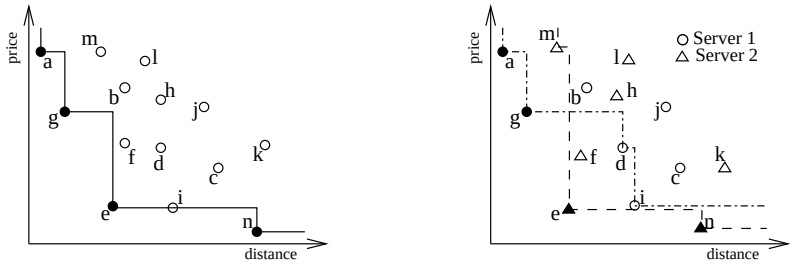
Abstract. Skyline queries have attracted considerable attention over the last few years, mainly due to their ability to return interesting objects without the need for user-defined scoring functions. In this work, we study the problem of distributed skyline computation and propose an adaptive algorithm towards controlling the degree of parallelism and the required network traffic. In contrast to state-of-the-art methods, our algorithm handles efficiently diverse preferences imposed on attributes. The key idea is to partition the data using a grid scheme and for each query to build on-the-fly a dependency graph among partitions which can help in effective pruning. Our algorithm operates in two modes: (i) *full-parallel mode*, where processors are activated simultaneously or (ii) *cascading mode*, where processors are activated in a cascading manner using propagation of intermediate results, thus reducing network traffic and potentially increasing throughput. Performance evaluation results, based on real-life and synthetic data sets, demonstrate the scalability with respect to the number of processors and database size.

1 Introduction

Skyline queries, in the context of databases, were initially proposed in [1] and since then, they have attracted considerable attention from the database community, primarily due to their applicability in multi-criteria decision making, without the requirement of user-defined scoring functions.

The skyline of a data set composed of d-dimensional points returns those points that are not dominated by any other, with respect to some specific preference (e.g., min, max) on each dimension. We say that point p dominates point q if p it is at least as good as q in all dimensions and it is strictly better in at least one. A classic example in the bibliography is that of hotels stored in a database, for which we know their price (y-axis) and distance to the beach (x-axis), as shown in Fig. 1(a). We are interested in those for which there is none cheaper and closer to the beach at the same time, i.e. hotels $\{a, g, e, n\}$.

There is also a current tendency towards addressing problems in a distributed manner, to enable decentralization and faster computation. Peer-to-Peer and Grid computing are two prominent examples and substantial research has focused on answering skyline queries in such environments [2–8]. Answering them efficiently involves query processing and propagation, high parallelism, progressiveness and low network traffic. Achieving all these goals is far from trivial.



(a) centralized case (b) distributed case

Fig. 1. Skyline example of hotels.

Following current trends, we focus on answering skyline queries in a *decentralized* setting. The data is distributed to a set of servers, one of which coordinates the query execution process. Therefore, our technique can be also applied in hierarchical P2P environments, where a superpeer acts as a coordinator which is responsible for a set of peers [9]. An example is given in Fig. 1(b), where points are distributed over two servers. The local skyline set in each server is depicted by using a different line style, whereas filled points are the global skyline.

Our first contribution is to examine the limitations of existing techniques that address skyline queries in such environments [3, 8, 10]. Such methods mainly focus on ways of partitioning the data, instead of developing efficient algorithms.

After justifying our choice of one of the partitioning schemes, we make our second contribution: We devise a new algorithm, *ADISC* (Adaptive DIstributed Skyline Computation), that runs in either *full-parallel* or *cascading* mode, without modification, as it uses the same internal structure in both cases. The algorithm incorporates a set of highly tuned optimizations. Using points from the partitions, we prune areas that will not contribute to the final skyline, following similar research [11]. We exploit these points further, in a novel way, to: i) improve parallelism while maintaining progressiveness, ii) negate relations between partitions which, as a side effect, iii) reduces network traffic and iv) minimizes coordinator workload. Irrelevant points are discarded and *eager* checking [12] is employed to improve performance. Parallel mode is more suitable for lightweighted systems, where response time reduction is desired, while cascading execution is directed towards enhancing progressiveness, reducing network traffic and increasing throughput.

Thirdly, aiming specifically to minimize traffic, we propose the use of *marginal points* as representatives, which is more efficient in the general case than similar techniques [4, 8]. Furthermore, for the special case of 2D data we prove optimality of the approach in that each queried processor i) performs minimum I/Os, ii) returns only global skyline points and with a slight modification iii) we minimize network traffic to the greatest extent.

Fourthly, we devise a data propagation algorithm based on skyline properties to determine the way processors must inform each other. Such a method is crucial as it implicitly defines the order of execution, the degree of parallelism and the

bandwidth consumption. It is interesting to note that for this reason we use the exact same structure as to determine how partitions must be checked with each other, thus minimizing additional overhead at the coordinator.

A basic property of our algorithm is that it handles different query preferences out of the box. Techniques such as [10] assume that skyline queries will always have fixed semantics regarding the criteria on data attributes and apply a specific partitioning method. However, we argue that this is a serious limitation, since different users may pose different preferences. We give two simple examples to illustrate this issue. As a first example, consider a multimedia database where images are 3-dimensional points, with each coordinate being the average value of red, green and blue in it. User U_1 requires images that are more red, but less green and blue, whereas user U_2 is interested in images that are less red but more green and blue. Evidently, U_1 issues a skyline query Q_1 with preferences <max,min,min>, whereas U_2 asks for <min,max,max> in his query Q_2. As a second example, consider a decision making application storing 2-dimensional data with bookstore profits, where each bookstore is represented as a $(x, y) = (time, profit)$ point. A query about bookstores with recent low profit is equally valid to one about bookstores with high profit achieved lately. The first is, of course, a <min,max> skyline query whereas the second is a <max,max> skyline query. According to these observations, handling different preferences is considered important towards supporting a broad range of applications.

The rest of the paper is organized as follows. Section 2 presents an overview of related work. Section 3 gives the definition of the problem in the distributed / parallel setting and explains in detail the partitioning schemes and gives some background information on the topic. Our proposal is studied in detail in Section 4 and evaluated experimentally in Section 5. Finally, Section 6 concludes the work and briefly discusses future research in the area.

2 Related Work

Primarily known as the maximal vector problem, proposed by Kung et al [13], skyline, or Pareto optimal, queries are a well studied problem in the area of Computational Geometry. Several main-memory techniques have been developed [14], based on the assumption that the data set is small enough.

It was not until much later that skyline queries were transferred to the context of databases, when Börzsönyi et al introduced the *skyline operator* [1] and proposed two algorithms: a Block-Nested-Loop (BNL) and a divide-and-conquer (D&C) approach. BNL was later improved by presorting the points according to a monotone scoring function, resulting in *Sort-Filter-Skyline* (SFS) [15]. Kossman et al [16] proposed an algorithm based on nearest-neighbor search. Papadias et al. proposed *Branch-and-Bound Skyline* (BBS) [17], which uses a multidimensional index and it is proven to be I/O optimal.

All of these approaches assume a centralized setting. The first work for distributed skyline queries was by Wu et al [3], where a CAN overlay was used to create grid partitions, each assigned to a processor. The main disadvantages of this approach is its low parallelism and that processors exchange the entire skyline which floods the network. Our technique is also different in that it uses a coordinator and processors are unaware of their neighbors' coordinates.

Additional distributed approaches have been developed, with applicability in the Web [2, 5], Peer-To-Peer [6, 7] and MANETs [4]. Web techniques partition the data vertically, assigning a single dimension to each server instead of a portion of the data, as we do. Peer-To-Peer systems lack any notion of a coordinator and keep distributed indexes. MANETs have limited resources and ad-hoc connectivity, contrary to our wired connection and more resourceful setting.

Parallel approaches include multiprocessor [18] and multi-disk environments [19]. Our setting differs in that we assume a share-nothing architecture in contrast with the shared-memory and shared-disk architecture respectively.

The works mostly related to ours is [8] and [10]. Vlachou et al [10] proposed a partitioning scheme based on hyper-spherical coordinates. Local skylines are computed using BBS, while the global result with SFS. Despite its increased parallelism, the approach has several limitations. First, as it has been noted previously, it assumes that the preference criteria (min or max) are known in advance. Secondly, due to SFS, the technique lacks progressiveness and all processors must report their skyline before any output is returned. This also burdens the coordinator with excessive points that must be kept in main or secondary memory. Finally, all processors must be activated for a skyline query, which may have a negative impact on throughput.

In [8], parallelism of distributed partitions is examined, where processors exchange representative points. Despite the similarities, a major difference is that partitions overlap, unlike our setting. Any parallelism achieved stems only from the initial bounds, whereas we increase parallelism in an innovative way. Partitions that cannot be computed in parallel are assigned a coordinator, hence multiple coordinators exist. Each coordinator creates a linearized execution plan for its partitions, though the details were not given. Finally, our representative points performs better in the general case.

3 Background

Given a data set D of d-dimensional points, $p = \{p_1, p_2, .., p_d\}$, the skyline contains those points that are not dominated by any other. For simplicity, we assume *min* criteria on all dimensions and w.l.o.g. coordinates are normalized in $[0, 1]^d$. For the remainder of the paper we use the notation shown in Table 1.

Definition 1 *Let D be a set of d-dimensional points. The skyline of D, denoted $SKY(D)$, contains those points that are not dominated by any other. We say that p dominates p', $p \prec p'$, if $p_i \leq p'_i$ $\forall i = 1, 2, .., d$ and $\exists j \in \{1, 2, .., d\}$ such that $p_j < p'_j$. The points in $SKY(D)$ are also referred to as skyline points.*

In the distributed/parallel environment, there are N available processors $P = \{P_1, P_2, .., P_N\}$ and, potentially, a distinct node, termed the *coordinator*, which is responsible for the query execution strategy. The data set D is partitioned offline in k subsets D_i, $D = \bigcup_{1 \leq i \leq k} D_i$ and $\bigcap_{1 \leq i \leq k} D_i = \oslash$. Each D_i is randomly[3] assigned to a processor P_j, which is responsible for computing its skyline. The P_is are deployed in a share-nothing architecture and may communicate

[3] We note that other assignment techniques (even dynamic ones) could also be applied, since the proposed techniques are orthogonal to the assignment strategy employed.

Table 1. Frequently used symbols.

Symbol	Interpretation
D	data set
d	data set dimensionality
n	data set cardinality (number of points)
p	d-dimensional point
p_i	i-th coordinate of p
D_i	i-th partition of D
k	number of partitions
P_i	the i-th processor
N	number of processors
$SKY(S)$	skyline of point set S

with each other. We assume one partition per processor, although multiple partitions may be assigned to each processor. The coordinator propagates the query to the P_is, which report back their local skyline, and computes the global skyline without indexing the received data, though such an approach could improve performance [20]. The property $SKY(D) = SKY(\bigcup_{1 \leq i \leq k} SKY(D_i))$ [1, 18] ensures that the coordinator correctly reports the global skyline by performing a skyline algorithm on the local results.

There are several factors affecting the overall performance: (i) local and global skyline computation, (ii) degree of parallelism, (iii) network traffic, (iv) throughput. Although communicating the entire skyline minimizes I/O, it also floods the network, results in more collisions during data transfer and hinders efficiency. Given that the coordinator also accounts for other tasks, e.g. load balancing, we are interested in minimizing its workload and resource consumption. It is evident that many of these issues are contradictory. For example, high parallelism offers low response times, but in a heavy-loaded system with many concurrent queries, it will impact system throughput, due to increased network traffic. Therefore, answering skyline queries efficiently involves issues such as:

- *response time*: This is the time needed to return the full skyline to the user since the query was submitted.
- *parallelism*: This is the number of parallel executing processors and greatly impacts response time.
- *network traffic*: Traffic is measured by the number of points that travel across the network and wish to minimize it, as much as possible.
- *progressiveness*: results should be returned as they are found and not after all partitions have been checked. However, we also view progressiveness in terms of iterativeness. Skyline queries may return a large number of points on occasions, therefore it would be of interest for the algorithm to produce results upon request, each time returning new skyline points.
- *diversity*: Diverse criteria on the data attributes must be supported and no assumptions should be made on the type of queries that will be invoked.

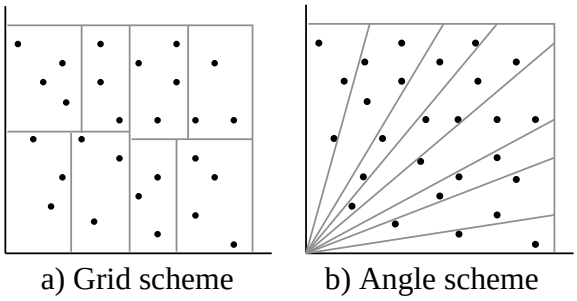


Fig. 2. Partitioning schemes.

3.1 Grid-Based Partitioning

Grid partitioning was used in [3] by applying a CAN overlay on the initial data, with recursive splits on the coordinates in a round robin fashion. Even workload is ensured by creating partitions with fairly the same amount of points, each one assigned to a processor. An example of grid partitioning is shown in Fig. 2(a).

The scheme has several advantages, with the most basic being that partitions preserve skyline properties. Therefore, we can safely exclude those that will certainly not contribute to the result, thus triggering fewer processors. A convenient execution order can be established from partition coordinates which also ensures progressiveness, while parallelism can be achieved for partitions that are certain not to dominate points from each other. Finally, the scheme makes no assumptions regarding the query criteria. This is a very interesting property, since we simply need to develop efficient algorithms to support skyline queries.

Despite its advantages, the scheme has some inherent shortcomings. The most prominent is its low parallelism, which is not tackled by existing techniques. Secondly, activating all processors simultaneously returns a lot of unnecessary points from partitions that are certain not to contribute to the final skyline.

3.2 Angle-Based Partitioning

Angle partitioning was proposed in [10] to simultaneously execute skyline queries on all processors. To do so, it relies on the hyper-spherical coordinates of the data and creates partitions of correlated-like distribution. An example of angle partitioning is given in Fig. 2(b).

Despite its advantage of parallelism, the scheme has several limitations. Firstly, because of its nature, the coordinator cannot exploit skyline properties on partitions. This leaves SFS as the only option for the global phase, when the received data is not indexed. This may be viable for independent distributions, but it is inefficient for high dimensionality or anticorrelated data, since its complexity is $O(m^2)$ on the number of received data and easily becomes a bottleneck.

Secondly, it assumes that the criteria of the queries are known in advance. As shown in Fig. 2(b), the beginning of the axis for the polar coordinates is

the same as the beginning of the criteria on the dimensions. Therefore, if *min* preferences are assumed, the scheme cannot answer efficiently a query with a *max* preference, because the underlying partitions were not designed for such a case. This results in a static partitioning and contradicts our goal of diversity.

Finally, the scheme is designed to answer skyline queries only. It is unclear how to efficiently answer other query types such as top-k, range, k-nearest neighbor queries, or even skyline variants [21, 22], since the partitions do not maintain Cartesian properties and pruning properties in general, which are required by the aforementioned queries.

4 Proposed Approach

Based on the previous discussion, we choose the grid-based partitioning scheme and optimize it, so that its disadvantages are smoothed enough to the point that its advantages are elevated.

4.1 Skyline Dependencies

Definition 2 *Let $A = [ll^A, ur^A]$ and $B = [ll^B, ur^B]$ be two hyper-rectangles. We say that B depends on A, $A \prec B$, if there may be a point from A that dominates any point from B.*

Lemma 1. *If A and B are two hyper-rectangles, then $A \prec B$ iff $ll^A \leq ur^B$, where for two d-dimensional points p and p', we say $p \leq p' \Leftrightarrow p_i \leq p'_i \ \forall i = 1, 2, .., d$.*

Proof. First we will prove sufficiency. If $ll^A \not\leq ur^B \Rightarrow \exists j \in \{1, 2, .., d\}$ such that $ll^A_j > ur^B_j \Rightarrow \forall a \in A, \forall b \in B, a_j > b_j$. Therefore, there is at least one dimension for which B is better, hence no points in B can be dominated by any point in $A \Rightarrow A \not\prec B$. For necessity, the proof is that ll^A has the highest dominating region among points from A that dominate ur^B. Any points in B that fall in that area are dominated by ll^A, assuming it is an actual point. For the same reason, points in A above ll^A, below ur^B, may also dominate points from B.

In Fig. 3(a), $E \prec A$ whereas $D \not\prec B$. Note that relation $\prec$ does not state that such a point exists; only that there might be one. For instance, although $A \prec B$, none of the points in A dominate any of the points in B. An immediate result of Lemma 1 is that for any A and B, $A \not\prec B$ and $B \not\prec A$, their points are definitely skyline with respect to each other. This is of great advantage as such partitions can be computed in parallel. We exploit this observation further so that such partitions are not checked against each other during the global merging phase.

Dependencies can be organized in a graph, where partitions are nodes and there is a directed edge from B to A iff $A \prec B$. This creates a *directed acyclic graph* of dependencies, or a *dependency graph*, as the one in Fig. 3(b) for the partitioning of Fig. 3(a). Such a graph can also be used during data propagation, because if an edge exists from B to A, then A may contain data that B can use for pruning, thus A should send its data to B. This also illustrates that dependencies define the workflow of query execution.

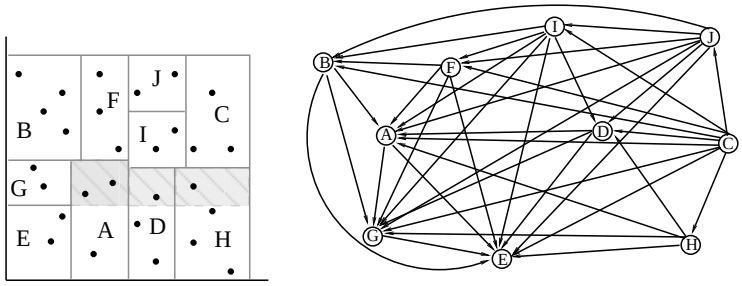


Fig. 3. Dependencies and dominating region.

4.2 Cutting-off Partitions

As in [17], where an R-tree node is not expanded if it is dominated, we can also exclude partitions from being queried if they are dominated by known points. This technique is also used in [11] and it is based on the following lemma.

Lemma 2. *Let p be a point and $A = [l^A, u^A]$ an MBR in the d-dimensional space. If $p \prec l^A$ (also written as $p \prec A$) then none of the points $a \in A$ will be in the final skyline.*

Proof. Since $\forall a \in A$, $a_i \geq l_i^A$ they are all dominated by p and therefore, there is no need to query this partition.

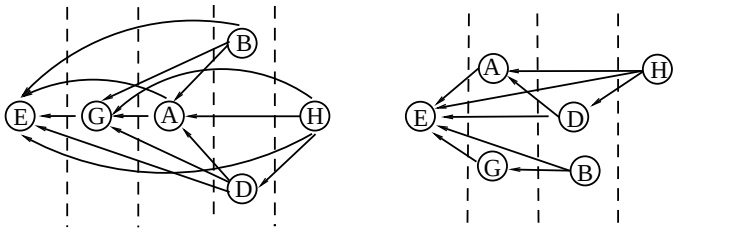
4.3 Increasing Parallelism

Up to this point any achieved parallelism is derived from the partitioning scheme. For example, partitions B and D in Fig. 3(a) will be computed in parallel, because no edges between them exist in the graph. As already mentioned, this does not result in great parallelism.

Definition 3 *Let $A = [l^A, u^A]$ and $B = [l^B, u^B]$ be two non intersecting MBRs and $A \prec B$. We define the dominated region of B from A, $dr \equiv dr_B(A)$, as the part of B that is dominated by l^A.*

The *dominated region dr* is the part of B that is potentially dominated by a point in A and is, in fact, the only reason why the relation $A \prec B$ exists between the two. Points in B outside of dr are skyline with regards to points in A, because $\exists j \in \{1, 2, .., d\}$ such that $b_j < l_j^A, b \in B$. Therefore, dr is the sole reason why parallelism is not achieved between them. As an example in Fig. 3(a), the gray region of A, D, H is the dominating area of G on those partitions.

Lemma 3. *Let $A = [l^A, u^A]$ and $B = [l^B, u^B]$ be two non intersecting MBRs and $A \prec B$. Then A and B can be computed in parallel if $\exists$ point p, $p \notin A$ such that $p \prec dr$.*



a) Contributing grids with defaut parallelism

b) Contributing grids with enhanced parallelism

Fig. 4. Example of enhanced parallelism.

Proof. By definition, $A \prec B \Rightarrow l^A \leq u^B \Rightarrow dr$ exists. If $\exists p$, $p \notin A$, $p \prec dr$ then all of the points in dr are dominated according to Lemma 2. Therefore, the points from B that are the reason for $A \prec B$ are no longer present, which negates the dependency of A, B and allows them to be computed in parallel.

Several observations derive from Lemma 3. First, for the two MBRs A and B, it cannot be that $l^A \leq l^B$ because then $dr \equiv B$, in which case B is dominated and will not contribute according to Lemma 2. Second, $p \notin A$ in any case, as that is the reason for the dependency in the first place. Third, assume $p \in C \Rightarrow l^C \leq p$. Since p dominates $dr \Rightarrow p \leq l^{dr} \leq u^B \Rightarrow l^C \leq u^B \Rightarrow C \prec B$. No particular relation exists between A and C. This gives us an indication of where to look for such points. Fourth, data will not be exchanged, as the initial dependency has been removed, which reduces traffic. Finally, because p negates the dependency, parallelism is increased and response time is minimized as B no longer has to wait data from A. Moreover, points from A and B will not be checked against each other during the global merging phase, thus reducing coordinator workload.

Figure 4(a) shows the dependency graph of the example in Fig. 3(a), after removing non contributing partitions. The new graph is much simpler, mostly because the removed partitions had a lot of dependencies. From left to right, the dashed lines indicate when a partition is computed with respect to the others. B and D are by construction independent of each other, hence their parallelism. Figure 4(b) shows the dependency graph after applying our technique. The graph is obviously simpler, even compared to the one in Fig. 4(a). The lower left point of E negates the dependencies of A, D and H on G and also removes the dependency of B on A. This results in parallelism between G and A, which is also more intuitive by looking at the partitioned space. Finally, there is one less dashed line because computations will end earlier than before.

4.4 Marginal Points

When processors exchange points, sending the entire skyline may become too cumbersome, not because of the skyline size alone, but because each processor may send it to many others. To alleviate this problem, we use representative

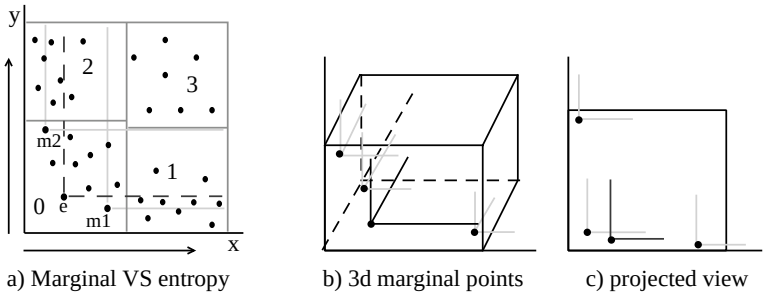


Fig. 5. Marginal versus entropy points.

points. However, instead of selecting the k points with the highest domination probability [8], also known as entropy points [15], we select targeted points with respect to the partitions they will be sent to. We term these points *marginal*.

Definition 4 *The marginal points of a partition are the ones closer to the* $d-1$ *coordinates of the partition's lower left corner based on the L1 distance.*

Assume the partitioning in Fig. 5(a). The query begins at partition 0 and it is propagated to 1 and 2 (3 is pruned). Although e has the highest probability, it dominates very few points from 1 and 2, whereas m_1 and m_2 dominate many more (each one separately). Since the receiving partition has at least one greater coordinate, all of its points have greater values in that coordinate. The only way for them to be in the skyline is to have a smaller value on another coordinate. For example, since partition 1 has greater x values, points will be in the skyline only if their y coordinate is better than the current best. Therefore, m_1 and m_2 are the points closest to x and y axis respectively.

We denote by m_i the marginal point for which the i-th coordinate is ignored. There are d such points in a d-dimensional space. Explaining the technique for the 3D case gives some additional insight. Assume the marginal points m_1, m_2, m_3 of Fig. 5(b), with gray lines, and the entropy point e, with black lines defining their dominating area. Disregarding the z coordinate to find m_3 (bottom left in the back), is like projecting on the xy plane, as in Fig. 5(c). On the projected space m_3 has a higher domination area than e, hence the technique works as if we are selecting the d points with the highest dominating region on the $d-1$ projected subspace. From another perspective, the corner that m_3 is closer to can be viewed as the beginning of the axis for partitions of greater z. In that sense, we choose points closer to the origin of the partitions that will use them and not of the one that sends them. Finally, it is interesting to note that [13] used projection on 2D in order to estimate 3D skyline size.

4.5 Data Propagation

Our algorithm is also able to operate in cascading mode, where processors propagate intermediate results, after computing the local skyline. If a partition sends

data to all others that depend on it, the network may be flooded and, most importantly, ignores the fact that representatives are refined thus rendering previous ones unnecessary. Taking also into account the fact that skylines propagate mainly along the axis, because inner partitions will be cut-off, we devise a data dissemination algorithm to minimize traffic.

The coordinator determines how data will propagate using the dependency graph and skyline properties and informs processors appropriately. If an edge $B \rightarrow A$ exists, the coordinator informs A it must send data to B and also informs B it must wait data from A. Each processor begins computations only after receiving data from all its predecessors.

Multiple dependencies exist between partitions and reducing them minimizes traffic. However, reducing them in a greedy way could have the opposite effect, because less punning will occur and many non skyline points will be returned. Therefore, a natural question is what dependencies can be ignored. A partition should be processed after all of its priors have, thus only edges between nodes that also have an indirect path may be removed, e.g., if $A \prec B$, $A \prec C$ and $C \prec B$, then $A \prec B$ can be neglected. This is as if trying to find the longest path between any two partitions, visiting nodes once all of its dependencies have been visited. Note that this action does not negate dependencies as when increasing parallelism; it only does not take them into account when finding how points must be exchanged. However, not all such relations can be ignored. Finally, the coordinator only knows partition MBRs and not the actual coordinates of propagated points, so the heuristics should depend on that information alone.

We ignore dependencies $A \prec B$ with an intermediate node C, if A, B and C share at least one propagation axis i.e. $u_i^A < l_i^C < u_i^C < l_i^B$ as with E, A and D in Fig. 3(a). The reason is that by projecting on the remaining $d-1$ dimensions C would send to B the m_i point from A or a refined point from itself.

Another heuristic is to check if $l^A < l^C$, regardless of B's coordinates. This is based on the same assumption as above, but the partitions need not share a propagation axis. This applies in the case of 3 or more dimensions. If neither heuristic applies, the dependency remains and A must send to B.

The 2-dimensional case One important issue about marginal points is their optimal behavior in the case of 2D data. Given a skyline query and the available information, we prove that each queried processor: (i) performs the minimum number of I/O operations and (ii) returns only global skyline points. We also prove that by combining the two points, no fewer information can be exchanged without invalidating the above behavior. To prove these, a basic lemma is needed.

Lemma 4. *Let M be a 2D MBR. Its marginal points m_1 and m_2, $m_1 \neq m_2$, dominate the same regions outside M as its entire skyline.*

Proof. Let $p \in SKY(M)$, $p \neq m_1$. Then, the dominating area of p outside M for greater x, begins at (u_x^M, p_y). Then $m_1 \prec (u_x^M, p_y)$, otherwise m_1 would not be marginal. We will prove by contradiction. Since $m_1 \in M \Rightarrow x_{m_1} \leq u_x^M$, so if $m_1 \not\prec (u_x^M, p_y) \Rightarrow y_{m_1} > p_y$. That sets p closer to the x axis, therefore p is the marginal point and not m_1, hence the contradiction. The proof is analogous for greater y's and m_2. By composition, p dominates a subset of the area dominated

by m_1, m_2 and therefore, m_1, m_2 jointly dominate the same regions as the entire skyline of M.

Theorem 1. *a) Let D be a 2-dimensional data set, partitioned in disjoint MBRs. If processors communicate only the marginal points of a partition, each activated processor: (1) will perform the minimum number of I/Os, (2) will report to the coordinator only global skyline points.*

b) Let each partition send a single point to subsequent ones, with coordinates the minimum of its marginal and any other points received. No fewer points can be sent so that (a) still holds.

Proof. a) Having the entire skyline available, BBS discards all non-contributing areas, which is the reason for its I/O optimality. For the same reason, only global skyline points are retrieved. Since marginal points dominate the same external area as the entire partition skyline (Lemma 4), the number of I/Os of subsequent partitions will be identical in the two cases, hence minimal (1) and only global skyline points will be retrieved and reported to the coordinator (2).

b) Being in the 2D case, points from partitions of at least one greater dimension have only one coordinate to best to be in the skyline, i.e. points of greater x (y) will be in the skyline iff their y (x) is less than the minimum of received points. Thus, the x (y) value of received points is irrelevant and can be safely substituted by any $x \leq u_x^A$ ($y \leq u_y^A$). So we can send (x_{m_2}, y_{m_1}) and simultaneously define the maximum allowed coordinate for both cases. Combined with the property $SKY_{A \cup B} = SKY(SKY_A \cup SKY_B)$, we derive that the point to send is in fact (min(x), min(y)) of local marginal and received points. This incorporates the skyline of previous partitions, which is the global so far (according to a). This property and Lemma 4 prove the validity of the technique. Therefore, each partition sends a single point to each subsequent. Due to our first heuristic, each partition receives data from at most two partitions, one for each propagation axis. Since each partition sends a single representative, (b) holds as fewer points means that no points will be sent. As a result no pruning is performed, with unnecessary I/Os and non global skyline points returned.

In fact we can prove, in a similar manner, that for the 2D case all non-contributing partitions are pruned, by constructing the marginal points from the partition MBR (which we do), even though we do not explicitly know them. This means that the global minimum of I/Os and traffic is achieved in that setting. Processors, of course, return only global skyline points. We omit the proof due to space restrictions.

4.6 Additional Optimizations

Additional optimizations are studied so that the overall efficiency is improved, both at the coordinator and processor side. A first one is *point exclusion*. Assume a point p and a partition $A = [l^A, u^A]$. If $p \not\leq u^A \Rightarrow \exists i$ such that. $p_i > u_i^A$. Such a point will not prune any areas of A, hence a processor that receives it can safely discard it. Likewise, if $p \not\geq l^A \Rightarrow \exists j$ such that $p_j < l_j^A$, which means that the coordinator can save time by not checking p against the points from A.

Another optimization is *eager checking*, proposed for continuous skyline [12]. Points are checked for domination upon arrival, thus storing only skyline points. The alternative of storing everything until all partitions have reported accumulates data and more time is spent afterwards to discard already stored points.

4.7 The ADISC Algorithm

The previous ideas are integrated into the ADISC algorithm. The outline of ADISC is given in Fig. 6. In the sequel, we describe briefly its most important aspects. At first, the coordinator C partitions the data offline and assigns them to processors P_i, each of which indexes its local copy with an R-tree and reports back arbitrary representatives. For each grid cell, C stores the MBR and points received. Given a query, C determines all affected partitions (Line 1), by executing a BBS-like pass over the MBRs, since no assumptions are made regarding

Algorithm ADISC (qry, P, rp, $mode$)
qry: the skyline query, P: partitions
rp: the representative points from each grid cell
$mode$: the mode in which to operate

1. $F \leftarrow \bigcup p \in P$, s.t. $\forall r \in rp \bigcup$ (other points from F), $r \not\prec p$;
2. $dg \leftarrow$ create dependency graph(F, $qry.criteria$);
3. $\forall f, g \in F$, $f \prec g$
4. try negating it with $r \in rp \bigcup$ (other points from F)
5. **if** ($mode$ = "cascading") **then**
6. find propagation order(dg, $qry.criteria$);
7. $\forall f \in F$
8. inform(processor(f), qry, $wait(f)$, $send(f)$)
9. $RPTD \leftarrow \{\}$; // no partitions have reported
10. **while** (not all $f \in F$ reported)
11. wait(); //until a partition reports
12. $nprt \leftarrow$ get next reporting partition();
13. $\forall prt \in RPTD$
14. checkGrids(dg, $nprt$, prt); //check $nprt$ with previous
15. **if** ($mode$ = "full-parallel") **then** checkGrids(dg, prt, $nprt$);
16. $RPTD \leftarrow RPTD \bigcup nprt$;
17. **if** ($\exists prtn \in RPTD$ s.t. dg.depends($prtn$) = 0) **then**
18. report points from $prtn$;
19.
20. **function** checkGrids(dg,A,B)
21. **if** dg.depends(A, B) **then**
22. dg.removeDepends(A, B);
23. $\forall a \in A$, **if** (a isAbove B.lowerleft) **then**
24. $isDomd \leftarrow$ check if $\exists b \in B$, s.t. $b \prec a$;
25. **if** ($isDomd$ = true) **then** discard a;

Fig. 6. Outline of ADISC algorithm.

how such information is organized. Partitions dominated by representatives of P_i are excluded. We also find additional points for pruning, by substituting the i-th coordinate of the upper right MBR corner with that of the lower left corner.

For the non-dominated partitions, we create the dependency graph, dg, based on the criteria imposed on the attributes (Line 2). At this phase, representative points are used to negate dependencies (Lines 3-4). If in *cascading* mode, the propagation algorithm is executed (Line 5). Each P_i is then informed of the query (Lines 7-8), including its priors and subsequent P_js. In *parallel* mode that information is empty. Then, C waits for results from queried P_is.

Each P_i stores the information of prior partitions and subsequent P_js. Whenever data is received from P_j, P_i removes P_j from its priors, until the list becomes empty and begins computations. The received data are used for pruning during BBS. The local skyline is reported to back to C. If subsequent P_js must be informed, representatives are selected and propagated as instructed.

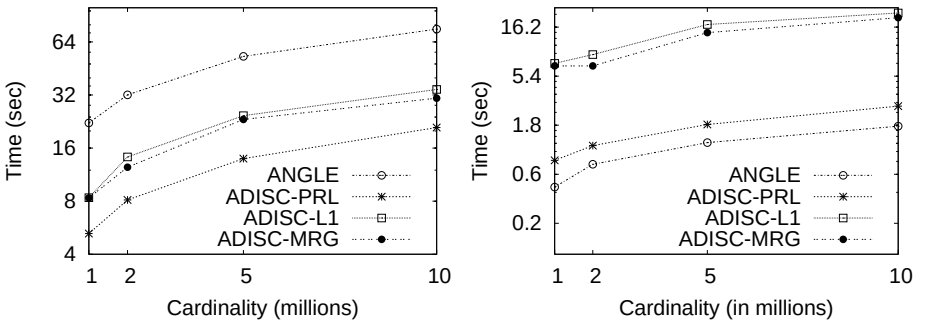
When C receives data, it checks if they are dominated by previously received points. Using dg, it finds reported partitions that the new one depends on (Line 14) and checks the points between them. If in *parallel* mode, the symmetric action must also be performed (line 15), since partitions report in unspecified order. Once checked, the dependency is removed, as they do not need to be checked again. A partition without dependencies (Line 17) has been checked against all partitions that could dominate any of its points, thus it contains only global skyline points. Consequently, its data are reported to the user.

5 Performance Evaluation Study

The experiments have been conducted on both real and synthetic data, on an Intel Core Duo @1.86GHz, Ubuntu Linux machine with 1GB RAM. R-tree page size has been set to 4KBytes and each processor has 20% of its blocks in buffers. We assume 8ms per page fault and a 100Mbps, collision-free, wired network. Timings show execution time in seconds, averaged over multiple data sets, including network time, whereas traffic graphs refer to the number of points communicated among processors. Synthetic data are generated with independent (IND) and anticorrelated (ANT) distributions. Cardinality varies from 1M to 10M and dimensionality from 2 up to 7, with default values of n=5M and d=4. The number of processors varies between 25 and 200, with a default value of 100. For the real data set, we used *forest cover* (FC) (http://kdd.ics.uci.edu).

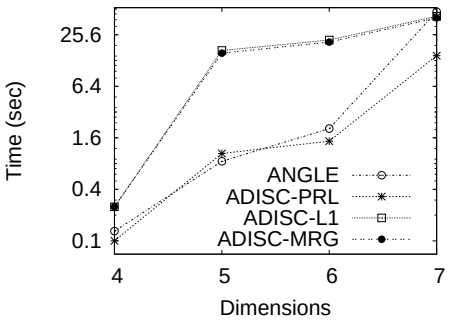
We compare ANGLE (angle partitioning with *min* criteria and SFS), ADISC-PRL (parallel ADISC), ADISC-L1 (cascading ADISC, entropy representatives) and ADISC-MRG (marginal representatives). For fairness between ADISC-L1 and ADISC-MRG, processors may exchange at most the same number of points, equal to the data set dimensionality. The darker portion of ADISC-L1/MRG traffic bars show the points exchanged between processors alone.

The effect of cardinality. The impact of cardinality on response time is given in Fig. 7(a), (b). ANGLE performs slightly better for IND and *min* criteria (Fig. 7(b)), since it is especially designed for this case, whereas ADISC-PRL requires more time for some processors to compute their local skyline. High response

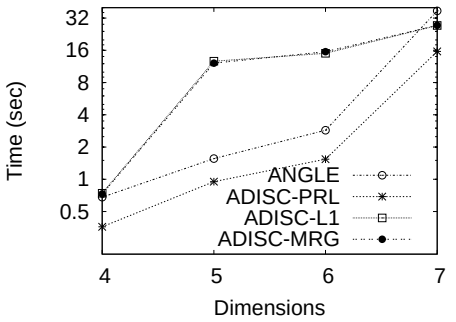


(a) ANT, min preferences

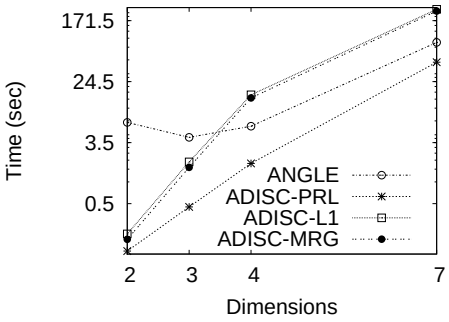
(b) IND, min preferences



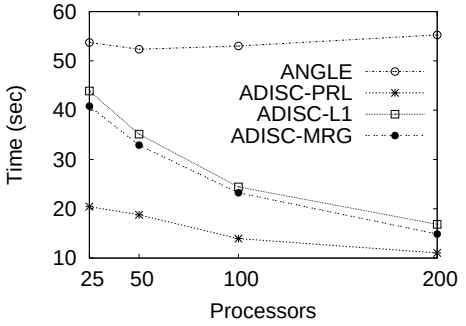
(c) FC, min preferences



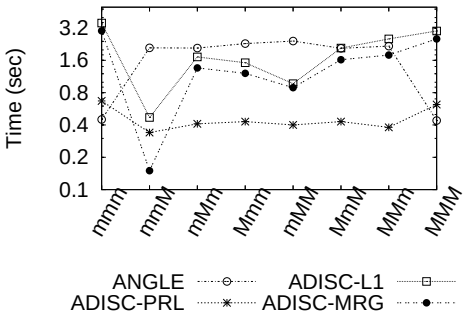
(d) FC, mixed preferences



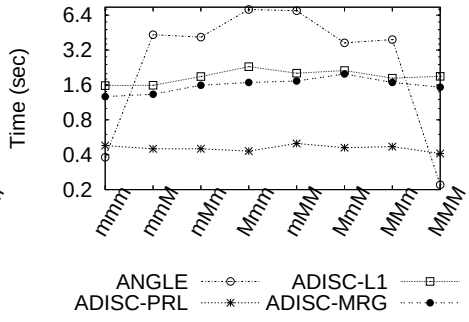
(e) IND, mixed preferences



(f) ANT, min preferences

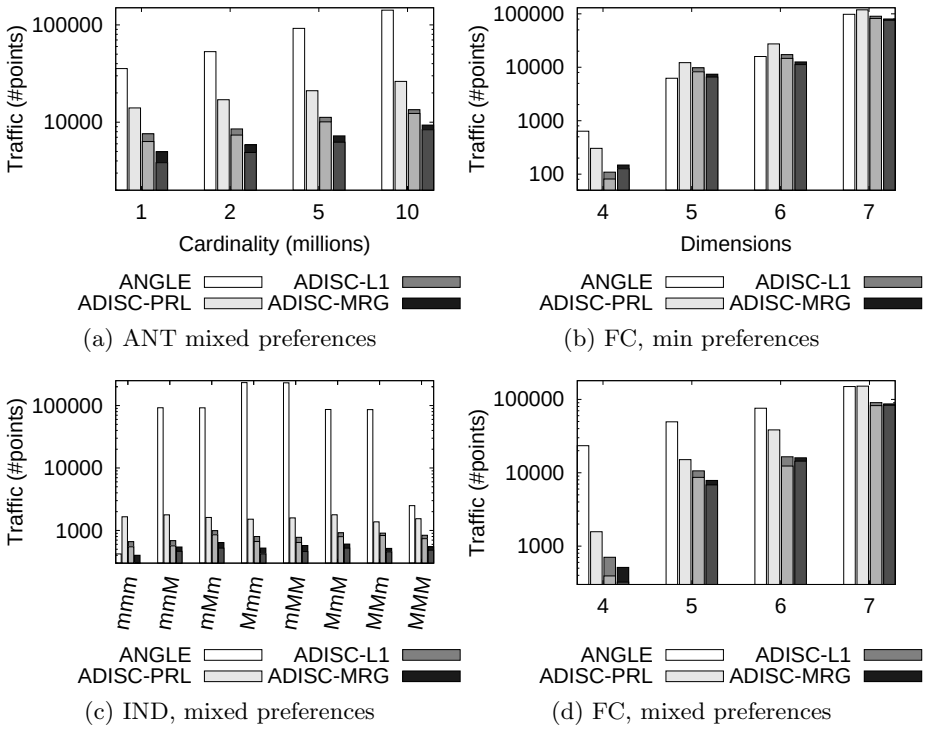


(g) ANT, diverse query criteria



(h) IND, diverse query criteria

Fig. 7. Response time results.



(a) ANT mixed preferences (b) FC, min preferences

(c) IND, mixed preferences (d) FC, mixed preferences

Fig. 8. Network traffic results.

times of ADISC-L1/MRG are due to the almost sequential execution of some partitions, a result of the data distribution. ADISC-MRG constantly performs slightly better than ADISC-L1, and exchanges fewer points (Fig. 8), due to better pruning of marginal points, which reduces I/O and processing time.

However, ANGLE does not behave well in all other cases, to the point that even ADISC-L1 and ADISC-MRG outperform it in the all-*min*, ANT case (Fig. 7(a)). SFS is the reason for this, as more than 90% of the time is spent on merging local skylines. On the other hand, ADISC's dependency graph and optimizations save valuable time. ANGLE also performs worse than ADISC for mixed criteria.

The effect of dimensionality. The impact of dimensionality on response time is shown in Fig. 7(c), (d) and (e). ANGLE performs again slightly better than ADISC-PRL, for few dimensions, all-*min* (Fig. 7(g), (h)). However, for more dimensions (Fig. 7(c), (d)), or even average dimensionality and ANT (Fig. 7(a)), its performance deteriorates. This is due to the fact that skyline computation becomes CPU-bound for high dimensionality and the problems imposed by SFS.

Regarding the 2D IND case with diverse criteria, ADISC-PRL is better than ANGLE up to 2 orders of magnitude (Fi.g 7(e)). Processors in ANGLE execute an anticorrelated like skyline, and visit almost all RTree nodes. This also results

in exceedingly high traffic (Fig. 8(c)). For more dimensions, the difference is reduced (Fig. 7(c)), as the effect of a single diverse criterion is normalized by the large number of other default criteria.

ADISC-PRL also performs better than ANGLE in FC, for almost all cases, with diverse or all-*min* criteria. The problems of SFS are easily seen for 6 dimensions and more, for the all-*min* case, for a data set of low cardinality (∼600K) (Fig. 7(c)). With a single *max* criterion, the difference is even greater, for as few as 4 dimensions (Fig. 7(d)). Regarding traffic, ANGLE sends many more points in most, if not all of these occasions. For the all-*min* case, the number of points is comparable in the two occasions. In fact ADISC-MRG sends less points than ANGLE, though our primary concern was to send less than ADISC-PRL.

The effect of the number of processors. For IND and FC, response time is reduced as the number of processors increases. However, an interesting result is observed from Fig. 7(f): ANGLE response time slightly decreases at first but then increases to a point greater than when having fewer processors. On the contrary, all ADISC variants decrease their response time. More processors mean more partitions which do not dominate points from each other. Therefore, time is saved both at the processors and the coordinator, which significantly decreases total response. This could also lead us to the assumption that skyline of ANT data may in fact be easier to compute than for IND data.

The effect of diverse criteria. The most interesting results are those of Fig. 7(g),(h), where all 8 combinations of criteria on a 3D data set are examined. Apart from *mmm*, ANGLE performs slightly better than ADISC-PRL for MMM, as partitions retain the correlated-like distribution for the most part.

The sudden drop of ADISC-L1/MRG in Fig. 7(g), mmM case, is due to the generators. By construction, the last dimension is the most likely to have a high value, w.r.t. the other two, to achieve anticorrelation. Inverting the criterion to *max* is like ignoring the dimension, since almost all points will have a high value. This results in less anticorrelation, more partitions are pruned, returning less points to C and the total response is lower. However, ANGLE will execute on all processors, even though many will not contribute, hence many points are returned (Fig. 8(a), (c)). Another interesting observation, is that ADISC-PRL shows a particularly steady behavior, almost invariant to the criteria imposed on the attributes, with minor fluctuations due to the different skyline sizes.

A similar observation can be made for network traffic. ANGLE varies greatly, depending not only on the number, but even on the position of a diverse criterion (Fig 8(c)). It should also be noted that such a high number of points does not only flood the network, but also degrades coordinator performance, since these need to be locally stored, until all points are received. On the other hand, ADISC stores only non-dominated points due to *early checking*.

6 Conclusions

In this work, we presented ADISC, an algorithm for efficient and adaptive distributed skyline computation. ADISC may operate in either full-parallel or cascading mode, balancing between the degree of parallelism and network traffic

according to system load. The algorithm runs on top of a grid partitioning scheme and integrates several optimizations for efficient computation. It efficiently handles different criteria on the attributes, since the dependency graph is constructed dynamically. We also proved optimality of our approach for the case of 2D data. We finally demonstrated the efficiency of our scheme with experimental results on real-life and synthetic data sets. Future research may include: (i) the derivation of a cost model, allowing ADISC to autonomously swap between modes, (ii) the study of an *hybrid* mode, where some partitions execute in parallel and others sequentially, for the same query, according to estimated cost and (iii) the design of randomized algorithms trading accuracy for speed.

References

1. Börzsönyi, S., Kossmann, D., Stocker, K.: The skyline operator. In: Proc. of thye 17th ICDE. (2001) 421–430
2. Balke, W.T., Güntzer, U., Zheng, J.X.: Efficient distributed skylining for web information systems. In: EDBT. (2004) 256–273
3. Wu, P., Zhang, C., Feng, Y., Zhao, B.Y., Agrawal, D., Abbadi, A.E.: Parallelizing skyline queries for scalable distribution. In: EDBT. (2006) 112–130
4. Huang, Z., Jensen, C.S., Lu, H., Ooi, B.C.: Skyline queries against mobile lightweight devices in manets. In: Proc. of the 22nd ICDE. (2006)
5. Lo, E., Yip, K.Y., Lin, K.I., Cheung, D.W.: Progressive skylining over web-accessible databases. Data Knowl. Eng. **57**(2) (2006) 122–147
6. Li, H., Tan, Q., Lee, W.C.: Efficient progressive processing of skyline queries in peer-to-peer systems. In: Proc. of the 1st Int. Conf. on Scalable Inf. Sys. (2006)
7. Wang, S., Ooi, B.C., Tung, A.K.H., Xu, L.: Efficient skyline query processing on peer-to-peer networks. In: Proc. of the 23rd ICDE. (2007) 1126–1135
8. Cui, B., Lu, H., Xu, Q., Chen, L., Dai, Y., Zhou, Y.: Parallel distributed processing of constrained skyline queries by filtering. In: Proc. of the 24th ICDE. (2008)
9. Vlachou, A., Doulkeridis, C., Kotidis, Y.: Skypeer: Efficient subspace skyline computation over distributed data. In: Proc. of the 23rd ICDE. (2007) 416–425
10. Vlachou, A., Doulkeridis, C., Kotidis, Y.: Angle-based space partitioning for efficient parallel skyline computation. In: Proc. of the 2008 Int. Conf. ACM SIGMOD. (2008) 227–238
11. Wang, S., Vu, Q.H., Ooi, B.C., Tung, A.K., Xu, L.: Skyframe: a framework for skyline query processing in peer-to-peer systems. The VLDB Journal **18**(1) (2009) 345–362
12. Tao, Y., Papadias, D.: Maintaining sliding window skylines on data streams. IEEE Trans. on Knowl. and Data Eng. **18**(3) (2006) 377–391
13. Kung, H.T., Luccio, F., Preparata, F.P.: On finding the maxima of a set of vectors. Journal of the ACM **22** (1975) 469–476
14. Preparata, F.P., Shamos, M.I.: Computational Geometry: An Introduction. Springer-Verlag (1985)
15. Chomicki, J., Godfrey, P., Gryz, J., Liang, D.: Skyline with presorting (2002)
16. Kossmann, D., Ramsak, F., Rost, S.: Shooting stars in the sky: an online algorithm for skyline queries. In: Proc. of the 28th Int. Conf. on VLDB. (2002) 275–286
17. Papadias, D., Tao, Y., Fu, G., Seeger, B.: An optimal and progressive algorithm for skyline queries. In: Proc. of the 2003 Int. Conf. ACM SIGMOD. (2003) 467–478
18. Cosgaya-Lozano, A., Rau-Chaplin, A., Zeh, N.: Parallel computation of skyline queries. In: Proc. of the 21st Int. Symp. on HPCS and Applications. (2007) 12
19. Gao, Y., Chen, G., Chen, L., Chen, C.: Parallelizing progressive computation for skyline queries in multi-disk environment, DEXA (2006) 697–706
20. Morse, M., Patel, J.M., Grosky, W.I.: Efficient continuous skyline computation. Inf. Sci. **177**(17) (2007) 3411–3437
21. Dellis, E., Seeger, B.: Efficient computation of reverse skyline queries. In: Proc. of the 33rd Int. Conf. on VLDB. (2007) 291–302
22. Yiu, M.L., Mamoulis, N.: Multi-dimensional top-k dominating queries. The VLDB Journal **18**(3) (2009) 695–718