An I/O Optimal and Scalable Skyline Query Algorithm

Yunjun Gao, Gencai Chen, Ling Chen, and Chun Chen

College of Computer Science, Zhejiang University, Hangzhou, 310027, P.R. China
{gaoyj, chengc, lingchen, chenc}@cs.zju.edu.cn

Abstract. Given a set of d-dimensional points, skyline query returns the points that are not dominated by any other point on all dimensions. Currently, BBS (branch-and-bound skyline) is the most efficient skyline processing method over static data in a centralized setting. Although BBS has some desirable features (e.g., I/O optimal and flexibility), it requires large main-memory consumption. In this paper, we present an improved skyline computation algorithm based on best-first nearest neighbor search, called IBBS, which captures the optimal I/O and less memory space (i.e., IBBS visits and stores only those entries that contribute to the final skyline). Its core enables several effective pruning strategies to discard non-qualifying entries. Extensive experimental evaluations show that IBBS outperforms BBS in both scalability and efficiency for most cases, especially in low dimensions.

1 Introduction

Skyline query is one of important operations for several applications involving multi-criteria decision making, and has received considerable attention in the database community. Given a set of d-dimensional points $P = \{p_1, p_2, \ldots, p_n\}$, the operator returns a set of points p_i, which is not dominated by any other point p_j in P on all dimensions, forming the skyline of P. A point dominates another one if it is *as good or better in all dimensions and better in at least one dimension* [19]. Consider, for instance, a common example in the literature, "*choosing a set of hotels that is closer to the beach and cheaper than any other hotel in distance and price attributes respectively from the database system at your travel agents'* [3]". Figure 1(a) illustrates this case in 2-dimensional space, where each point corresponds to a hotel record. The room price of a hotel is represented as the x-axis, and the y-axis specifies its distance to the beach. Clearly, the most interesting hotels are the ones $\{a, g, i, n\}$, called *skyline*, for which there is no any other hotel in $\{a, b, \ldots, m, n\}$ that is better on both dimensions. For simplicity, in this paper, we use the *min* condition on all dimensions to compute the skyline, even though the proposed algorithm can be easily applied to different conditions (e.g., *max* metric). Using the *min* condition, a point p is said to dominate another one q if (i) p is not larger than q in all dimensions, and (ii) p is strictly smaller than q in at least one dimension. This implies that p is preferable to q for the users in real life. Continuing the running example,

D. Bell and J. Hong (Eds.): BNCOD 2006, LNCS 4042, pp. 127–139, 2006.

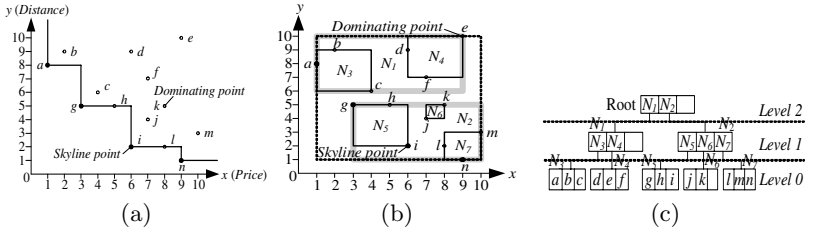


Fig. 1. Example of skyline and an R-tree in 2-dimensional space

hotel a dominates hotels b, d, and e because the former is nearer to the beach and cheaper than the latter.

Skyline query processing has been extensively studied, and a large number of algorithms have been also proposed [1, 3, 5, 7, 9, 12, 13, 15, 16, 19, 21]. These methods can be mainly divided into two categories. Specifically, (i) non-index-structure-based schemes, which do not assume any index structure on the underlying dataset, but compute the skyline through scanning the entire dataset at least once, resulting in expensive CPU overhead; (ii) index-structure-based solutions, which significantly reduce CPU and I/O costs by performing the skyline retrieval on an appropriate index structure (e.g., R*-tree [2]). We concentrate on the second category in this paper. In addition, the skyline computation problem is also closely related to several other well-known problems that have been extensively investigated in the literature, such as *convex hull* [4, 17], *top-k* queries [6, 8, 11, 14], and *nearest neighbor* search [10, 18].

Currently, BBS (branch-and-bound skyline), presented by Papadias *et al.* in [15], is the most efficient skyline query algorithm over static datasets in a centralized setting. It employs a best-first based nearest neighbor search paradigm [10] on dataset indexed by R*-tree. BBS minimizes the I/O overhead, and the considerable experiments of [15] show that it outperforms previous algorithms in terms of CPU and I/O costs for all problem instances. Although BBS has some desirable advantages, it yet needs large memory space. As reported in [15], the heap size of BBS is larger than the *to-do* list size of NN [12] in 2-dimensional space. In fact, we can greatly reduce space consumption used by the heap and speed up the execution of the algorithm via several dominance checking based pruning heuristics (discussed detailedly in Section 3 of this paper) for filtering all the non-qualifying entries that may not contain (become) any skyline point. As known, the less the memory space requires, the more scalable the algorithm is. Thus, in this paper, we present an improved skyline query algorithm, called IBBS, which, like BBS, is depended on best-first nearest neighbor search on R*-tree, whereas (unlike BBS) it enables several effective pruning strategies to discard unnecessary entries. IBBS incorporates the virtues of BBS (e.g., I/O optimal, low CPU cost, etc.), while gaining less main-memory consumption (i.e., smaller heap size). Finally, extensive experiments with synthetic datasets confirm that IBBS outperforms BBS in both efficiency and scalability for most cases, especially in low dimensions.

The rest of the paper is organized as follows. Section 2 reviews existing algorithms for skyline queries, focusing more on BBS as it is more related to our

work. Section 3 describes IBBS, together with some pruning heuristics and a proof of its memory space optimality. Section 4 experimentally evaluates IBBS, comparing it against BBS under various setting. Section 5 concludes the paper with some directions for future work.

2 Related Work

To our knowledge, Borzsonyi *et al.* [3] first introduce the skyline operator in the database context and develop two skyline computation methods including divide-and-conquer (D&C) and block-nested-loop (BNL). Chomicki *et al.* [7] present a sort-first-skyline (SFS) algorithm as an improved version of BNL. Tan *et al.* [19] propose the first *progressive* technique that can return skyline points instantly, and develop two solutions for skyline queries, termed Bitmap and Index, respectively. Another two progressive skyline query algorithms, nearest neighbor (NN) and branch-and-bound skyline (BBS), are proposed by Kossmann *et al.* [12] and Papadias *et al.* [15], respectively, based on nearest neighbor search [10, 18] on datasets indexed by R-trees. The great difference of both algorithms is that NN requires multiple nearest neighbor queries, but BBS executes only a single retrieval of the tree. Furthermore, BBS guarantees the minimum I/O cost. Since our work in this paper is more related to BBS, the following discussion describes its executive steps, using an illustrative example.

Table 1. Execution of BBS

Action	**Heap Contents**	S
Visit root	$(N_2, 4)$,$(N_1, 7)$	Ø
Expand N_2	$(N_5, 5)$,$(N_1, 7)$,$(N_7, 9)$,$(N_6, 11)$	Ø
Expand N_5	$(N_1, 7)$,$(g, 8)$,$(i, 8)$,$(N_7, 9)$,$(h, 10)$,$(N_6, 11)$	Ø
Expand N_1	$(N_3, 7)$,$(g, 8)$,$(i, 8)$,$(N_7, 9)$,$(h, 10)$,$(N_6, 11)$,$(N_4, 13)$	Ø
Expand N_3	**$(g,8)$**,**$(i,8)$**,**$(a,9)$**,$(N_7, 9)$,$(c, 10)$, $(h, 10)$,$(b, 11)$,$(N_6, 11)$,$(N_4, 13)$	$\{g, i, a\}$
Expand N_7	~~$(c,10)$~~,~~$(h,10)$~~,**$(n,10)$**,~~$(b,11)$~~, ~~$(N_6,11)$~~,~~$(N_4,13)$~~	$\{g, i, a, n\}$

As an example, suppose that we use the 2-dimensional dataset of Figure 1(a), organized in the R-tree of Figure 1(c), together with the minimum bounding rectangles (MBRs) of the nodes whose capacity is 3. Note that the distances from an intermediate entry (e.g., N_3, N_4, etc.) or a data point (e.g., a, b, etc.) to the beginning of the axes are computed according to L_1 norm, that is, the *mindist* of a point equals the sum of its coordinates (e.g., $mindist(g) = 3 + 5 = 8$) and the *mindist* of a MBR (i.e., intermediate entry) equals the *mindist* of its lower-left corner point (e.g., $mindist(N_5) = 3 + 2 = 5$). Initially, all the entries in the root node are inserted into a heap H sorted in ascending order of their *mindist*. Then, BBS circularly computes the skyline until H becomes empty. Each circulation, it first removes the top entry E with the minimum *mindist* from H and accesses it. Here are two cases. (i) If E is an intermediate entry and

not dominated by the existing skyline points, the algorithm en-heaps its child entries there. (ii) If E is a data point and not dominated by the skyline points obtained, the algorithm inserts it into the list S of the skyline as a skyline point. Table 1 illustrates the executive steps of BBS. Also notice that skyline points discovered are bold and pruned entries are shown with strikethrough fonts.

Recently, Balke *et al.* [1] extend the skyline computation problem for the web information systems. Lin *et al.* [13] study continuous skyline monitoring on data streams. Chan *et al.* [5] consider skyline computation for partially-ordered domains. Godfrey *et al.* [9] present an algorithm, called linear-elimination-sort for skyline (LESS), which has attractive worst-case asymptotical performance. Pei *et al.* [16] and Yuan *et al.* [21] independently present the skyline cube consisting of the skylines in all possible subspace.

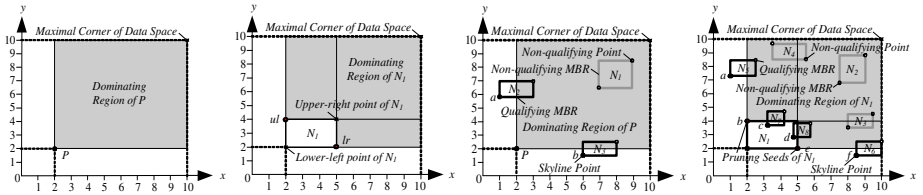
3 Improved Branch-and-Bound Skyline Algorithm

3.1 Dominance Checking Based Pruning Strategy

Consider the execution of BBS demonstrated in Table 1 of Section 2, some non-qualifying entries for the skyline are existed in the heap. For example, entry N_6 inserted into the heap after expanding entry N_2 is such one because it completely falls into the dominating region (DR for short) of entry N_5, and may not contain any skyline point. Similarly, entries h, N_4, b, and c are all redundant ones. Thus, they must be discarded, and not be en-heaped there. In fact, only those entries that may potentially contain (or become) the skyline points (e.g., N_3, g of Figure 1(b)) are needed to be kept in the heap. Based on this observation, it may be helpful to identify these entries and prevent them from being inserted in the heap. Fortunately, we can achieve this goal by careful dominance checking for each entry before it is inserted in the heap. Next, we present several pruning heuristics that is inspired by the analysis of per entry's DR.

Let S be a set of the skyline points, there must be at least one entry E_j ($j \neq i$) that dominates E_i, if an entry E_i does not appear in S. If we can know E_j when inserting E_i into the heap, E_i can be discarded immediately as it is an unnecessary entry for the skyline. However, the problem is now how to get such E_j in order to safely prune E_i. Since such a E_j must come from all the entries that have been visited by the algorithm, we can pick possible E_j from them. To address this problem, we need to consider for an entry E_j its ability to dominate and then prune other entries. For simplicity, we take 2-dimensional data space into account in the following discussion. However, similar conclusions also hold for d-dimensional ($d > 2$) data space.

Toward a point entry P with coordinates (x_1, x_2), its ability to dominate other entries, including point and intermediate entries (i.e., MBRs), is determined by its own values and the boundaries of the data space, that is, the rectangle whose diagonal is the line segment with P and the maximum corner of the data space as coordinates. Any other entry that resides in that region is dominated by P and it excluded from the final skyline. For this reason, we call that rectangle



(a) DR of P (b) DR of N_1 (c) Pruning with P (d) Pruning with N_1

Fig. 2. Illustration of DRs of one point, one MBR, and their pruning ability

the DR of P. Intuitively, the larger P's DR is, the more other entries are dominated by P because a larger rectangle covers more entries in the data space, especially for the entries following independent (uniform) distribution. For ease of comprehension, we use a 2-dimensional illustration as shown in Figure 2 in this discussion. Specifically, Figure 2(a) shows the DR of P (represented as the shaded rectangle), assuming that the maximum corner of data space is 10. The P's ability to dominate other entries is plotted in Figure 2(c) under the same supposition as Figure 2(a). Clearly, in Figure 2(c), entry N_1 is dominated by point P since it fully falls into the DR of P. Hence, N_1 can be pruned instantly, and need not to be inserted into the heap for further consideration. However, entries N_2 and N_3 that intersect the P's DR must en-heap there, because they may contain some skyline points (e.g., a and b). Therefore, we call that N_1 the non-qualifying MBR, but N_2 and N_3 the qualifying MBRs.

Assume that the boundaries of an intermediate entry N are $[{x_1}^l, {x_1}^h] \times [{x_2}^l, {x_2}^h]$, then the coordinates of its lower-left, lower-right, upper-left, and upper-right corners are the points $({x_1}^l, {x_2}^l)$, $({x_1}^h, {x_2}^l)$, $({x_1}^l, {x_2}^h)$, and $({x_1}^h, {x_2}^h)$, respectively. Thus, the DR of N is defined by its own boundaries and the boundaries of the data space. As an example, the DR of entry N_1 is shown in Figure 2(b) (specified the shaded area). From the diagram, we can also see that the N_1's DR is determined by the upper-left, bottom-right corners of N_1 (denoted as two red points), and the maximal corner of data space, respectively. Specifically, let ul be the upper-left vertex of N_1 and lr the lower-right vertex of N_1, then the DR of N_1 is the union of the dominating regions of ul and lr, i.e., formally, DR (N_1) = DR (ul) DR (lr). So, it implies that the N_1's ability to dominate other entries can be done by dominance checking with ul and lr. For this reason, we term that ul and lr the pruning seeds of N_1.

Those entries, including points and MBRs, which completely fall into the DR of N_1 are dominated by N_1, and they must not appear in the final skyline. As known, an intermediate entry N is dominated by a point entry P with coordinates (x_1, x_2) only if its bottom-left corner is dominated by P. Similarly, N is dominated by another intermediate entry N' only if the lower-left vertex of N resides in the DR of N'. Figure 2(d) demonstrates the pruning with a MBR N_1, where the pruning seeds of N_1 are denoted as two red points (i.e., b and e). Evidently, entries N_2, N_3, and N_4 are non-qualifying ones since they are dominated by N_1. As a result, they can be discarded in security, and have not to be inserted in the heap. Entries N_5, N_6, N_7, and N_8, however, need be stored in the heap

in order to access them later, because their lower-left corners are not dominated by N_1, and they may contain some skyline points, such as points a, c, d, and f of Figure 2(d) are such ones.

In summary, we can derive the following pruning heuristics to prune the non-qualifying entries for the skyline based on the above discussion. Suppose that, in d-dimensional data space, two point entries P and P' with coordinates $(x_1, x_2, \ldots, x_d)$ and $(x'_1, x'_2, \ldots, x'_d)$ respectively, and two intermediate entries N and N' with boundaries $[{x_1}^l, {x_1}^h] \times [{x_2}^l, {x_2}^h] \times \ldots \times [{x_d}^l, {x_d}^h]$ and $[{x'_1}^l, {x'_1}^h] \times [{x'_2}^l, {x'_2}^h] \times \ldots \times [{x'_d}^l, {x'_d}^h]$, respectively. Then several pruning heuristics are developed as follows.

Heuristic 1. If P' is dominated by P, i.e., (i) $x_i \leq x'_i$ for i $\in [1, d]$, and (ii) $x_i < x'_i$ in at least one dimension, then it can be safely pruned immediately and not be inserted into the heap, since it must not appear in the skyline.

Heuristic 2. If the bottom-left corner of N is dominated by P, then N can be also safely discarded and not be en-heaped there, as it must not contain any skyline point.

Heuristic 3. If P is dominated by N, that is, P fully falls into the N's DR, then P can be safely removed instantly and excluded from the heap, because it may not become a skyline point.

Heuristic 4. If N dominates N', namely, the lower-left corner of N' fully resides in the DR of N, then N' can be also safely discarded immediately and not be kept in the heap, since it may not contain any skyline point.

3.2 Algorithm Description

Like BBS, IBBS is also based on best-first nearest neighbor search. Although IBBS can be applied to various multi-dimensional access methods, in this paper, we assume that the dataset is indexed by an R*-tree due to its efficiency and popularity in the literature. Unlike BBS, IBBS enables several effective pruning heuristics to discard non-qualifying entries in order to greatly decrease the memory space and speed up its execution. In particular, IBBS incorporates two pruning strategies. The first one is that when expanding an intermediate entry, all entries dominating each other in its child nodes are removed according to heuristics 1 to 4 (proposed in Section 3.1 of this paper). The other one involves pruning strategy that checks the contents of the heap H before the insertion of an entry E. If E is dominated by some entry in H, it is pruned immediately and not en-heaped there. In contrast, E is stored in H, and all entries in H that are dominated by it are also discarded accordingly.

The pseudo-code of IBBS is shown in Figure 3. Note that an entry is checked for dominance twice: before it is inserted into the heap and before it is expanded. Furthermore, the algorithm also implements pruning twice. Specifically, line 13 filters all entries dominated by some entry in the heap. Line 15 excludes all entries dominating each other from the heap. Thus, only those entries that contribute to the final skyline are maintained in the heap, such that the maximum heap size (i.e., memory consumption) is reduced by factors, as well as the CPU cost is decreased accordingly.

```
Algorithm IBBS (R-tree R)
/* S is used to keep the final skyline. */
1. S = Ø;
2. Insert all entries in the root of R into the heap H;
3. While H is not empty do;
4.    Remove the first entry E from H;
5.    If E is dominated by any point in S then
6.       Discard E;
7.    Else   // E is not dominated by any point in S
8.       If E is a data point then
9.          Insert E into S;
10.      Else   // E is an intermediate entry
11.         For each child entry E_i of E do
12.            If E_i is not dominated by any point in S then
13.               If E_i is not dominated by any entry in H then
14.                  Insert E_i into H;
15.         Prune all entries dominating each other in H by heuristics 1 to 4;
16. End while
End IBBS
```

Fig. 3. Pseudo-code of an IBBS algorithm

Table 2. Execution of IBBS

Action	Heap Contents	S
Visit root	$(N_2, 4)$,$(N_1, 7)$	Ø
Expand N_2	$(N_5, 5)$,$(N_1, 7)$,$(N_7, 9)$	Ø
Expand N_5	$(N_1, 7)$,$(g, 8)$,$(i, 8)$,$(N_7, 9)$	Ø
Expand N_1	$(N_3, 7)$,$(g, 8)$,$(i, 8)$,$(N_7, 9)$	Ø
Expand N_3	$(g,\mathbf{8})$,$(i,\mathbf{8})$,$(a,\mathbf{9})$,$(N_7, 9)$	$\{g, i, a\}$
Expand N_7	$(n,\mathbf{10})$	$\{g, i, a, n\}$

Continuing the example of Figure 1, for instance, let us give an illustrative example of IBBS to simulate its executive steps for skyline query. Initially, all the entries in the root node are inserted into a heap H sorted in ascending order of their *mindist*, resulting in $H = \{(N_2, 4), (N_1, 7)\}$. Then, the algorithm removes the top entry (i.e., N_2) from H, visits it, and en-heaps its children there, after which $H = \{(N_5, 5), (N_1, 7), (N_7, 9)\}$. Here N_6 is discarded since it is dominated by N_5. Similarly, the next expanded entry is N_5 with the minimum *mindist*, in which the data points are added into $H = \{(N_1, 7), (g, 8), (i, 5), (N_7, 9)\}$. Also notice that h is pruned as it is dominated by g. The algorithm proceeds in the same manner until H becomes empty. The final list S of skyline points becomes $S = \{g, i, a, n\}$. As with the settings of Table 1, Table 2 illustrates the execution of IBBS. From Table 2, we can see that the heap size of IBBS is smaller significantly than that of BBS, which is also verified by the experiments in the next section of this paper.

3.3 Discussion

In this section, we focus on proving the memory space optimality of IBBS, and omit the proofs of its correctness and I/O optimality because they are similar to those of BBS in [15].

Lemma 1. *If an entry E, either an intermediate entry or a data point entry, does not inserted into the heap H, then there must exist another entry E' in H or some skyline point discovered that dominates E.*

Proof. The proof is straightforward since our proposed pruning heuristics discard all entries that are dominated by any other entry in the given dataset before they are en-heaped there. □

Lemma 2. *All entries that may contain (or become) skyline points must be kept in the heap H.*

Proof. Clear, by Lemma 1, all entries in H must not be dominated by any other entry in the given dataset. Also, they act on the skyline, that is, they either contain some skyline points or are skyline points. Thus, Lemma 2 holds. □

Theorem 1. *The main-memory consumption for IBBS is optimal.*

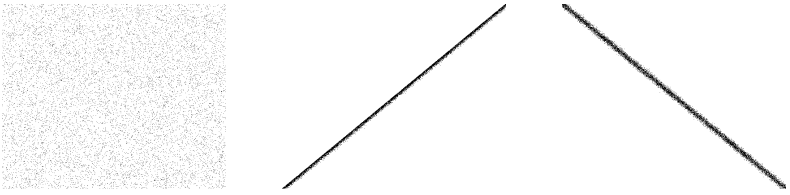
Proof. The Theorem 1 trivially holds, since Lemmas 1 and 2 ensure that the heap H used by IBBS only stores those entries that may contain (or be) skyline points, as well as they are inserted into H at most once by their *mindist*. □

4 Experimental Evaluation

This section experimentally verifies the efficiency of IBBS by comparing it with BBS under a variety of settings. We implemented two versions of IBBS, called IBBS-OS and IBBS-WOS, respectively. Specifically, IBBS-OS incorporates two pruning strategies (described in Section 3.2), but IBBS-WOS employs only the first one, i.e., when expanding an entry, all its child entries dominating each other are removed by heuristics 1 to 4. All algorithms (involving IBBS-OS, IBBS-WOS, and BBS) were coded in C++ language. All experiments were performed on a Pentium IV 3.0 GHz PC with 1024 MB RAM running Microsoft Windows XP Professional. We considered L_1 norm to compute *mindist* from the origin of the data space in all experiments.

4.1 Experimental Settings

Following the common methodology in the literature, we generated three synthetic datasets conforming the independent (uniform), correlated, and anti-correlated, respectively. Figure 4 illustrates such datasets with cardinality $N = 10000$ and dimensionality $d = 2$. We utilized these datasets with d varied between 2 and 5, and N in the range [100k, 10M]. All datasets are indexed by R*-tree [2], whose node size was fixed to 4096 bytes resulting in node capacities altered 204 ($d = 2$), 146 ($d = 3$), 113 ($d = 4$), and 92 ($d = 5$), respectively. All experiments examined several factors, including d, N, and *progressive* behavior, which affect the performance of the algorithms.



(a) Independent (b) Correlated (c) Anti-correlated

Fig. 4. Illustration of three synthetic datasets

4.2 Experimental Results

The first set of experiments studies the effect of dimensionality d using the datasets with $N = 1$M and d varied from 2 to 5. Figure 5 shows the maximum size of the heap (in Kbytes) as a function of d. Clearly, the maximal heap size (MHS for short) of both IBBS-OS and IBBS-WOS almost equals under various dimensionalities, which implies that most of non-qualifying entries are pruned after doing the first pruning strategy, and few entries can be further discarded via the second one. As validated again in the following experiments. For all datasets, however, the MHS of IBBS-WOS is greatly smaller than that of BBS, especially in low dimensions. Notice that the difference of both algorithms decreases gradually with the dimensionality, since the larger overlap among the MBRs at the same level of R-trees occurs in the high-dimensionality [20]. Despite the gain of IBBS-WOS reduces in this case (e.g., $d = 5$), it is yet less than BBS, which is also pointed out by the number at the side of each polyline in the diagrams.

Figure 6 illustrates the number of node access versus d. From these graphs, we can see that three algorithms display the same efficiency for all datasets. This explains the I/O overhead of IBBS is the same as that of BBS. Similar to Figure 6, Figure 7 compares the algorithms in terms of CPU time (in secs). By and large, the CPU costs of both IBBS-WOS and BBS are similar. However, as shown in Figure 7, IBBS-WOS slightly outperforms BBS in the lower dimensionality. The CPU time of IBBS-OS increases fast as d increases, and it is clearly higher than other algorithms. The reason is that IBBS-OS introduces some CPU overhead for implementing the second pruning strategy. Additionally, it is expected that the performance of all algorithms degrades because the overlapping among the MBRs of R-tree increases and the number of skyline points grows.

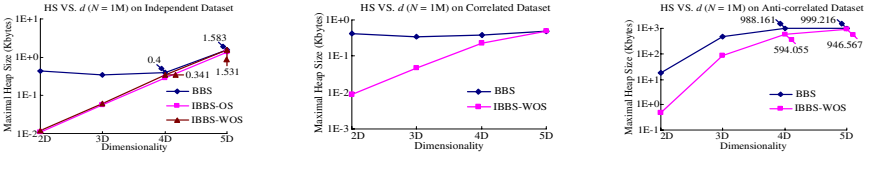


Fig. 5. Maximal Heap Size (Kbytes) VS. d ($N = 1$M)

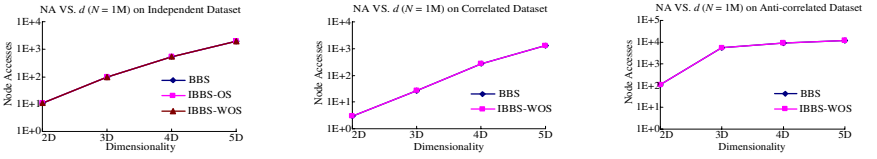


Fig. 6. Node Accesses VS. d (N = 1M)

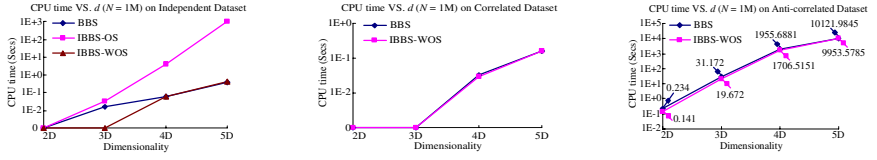


Fig. 7. CPU time VS. d (N = 1M)

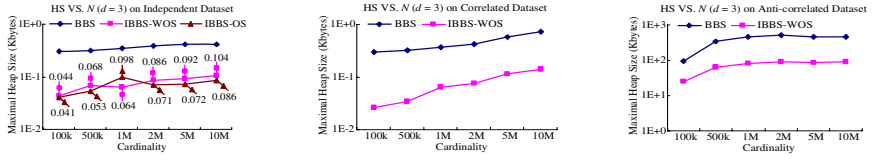


Fig. 8. Maximal Heap Size (Kbytes) VS. N (d = 3)

Next, we investigate the influence of cardinality N. Toward this, we deployed the 3-dimensional (the parameter $d = 3$ is the median value used in Figures 5 to 7) datasets whose cardinality range varies between 100k and 10M. Figures 8-9 show the MHS and CPU cost, respectively, versus N. Obviously, IBBS-WOS exceeds BBS in all cases. Specifically, the heap of IBBS-WOS is several orders of magnitude smaller than that of BBS. For CPU time, IBBS-WOS is also faster than BBS, especially in low dimensions. In addition, as the above experiments, the heap of IBBS-OS is similar to that of IBBS-WOS, but its CPU cost is greatly larger than other algorithms. Also note that, as shown in Figure 9(c), the difference increases with the cardinality, which is due to the positions of the skyline points and the order in which they are discovered.

Finally, we also inspect the *progressive* behavior of the algorithms for skyline query on 3-dimensional datasets. Figure 10 compares the size of the heap as a function of the number of skyline points (NSP for short) for datasets with N = 1M (for dependent and correlated datasets) or 100k (for anti-correlated dataset) and $d = 3$. Note that the NSP in the final skyline is 94, 26, and 13264, for independent, correlated, and anti-correlated datasets, respectively. From the diagrams, we see that IBBS-WOS clearly exhibits smaller heap size than BBS (over orders of magnitude) in all cases, since most of non-qualifying entries are pruned by IBBS-WOS. As expected, the heap size of both IBBS-OS and IBBS-WOS is highly adjacent. On the other hand, notice that the heaps reach their

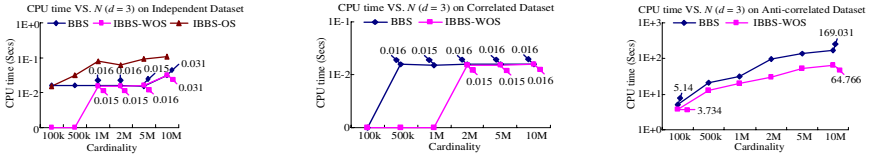


Fig. 9. CPU time VS. N ($d = 3$)

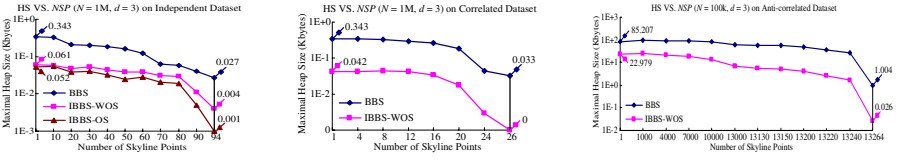


Fig. 10. Maximal Heap Size (Kbytes) VS. NSP (N = 1M or 100k, $d = 3$)

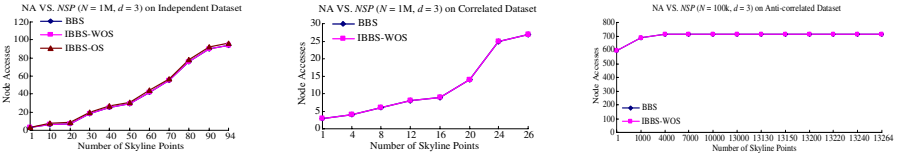


Fig. 11. Node Accesses VS. NSP (N = 1M or 100k, $d = 3$)

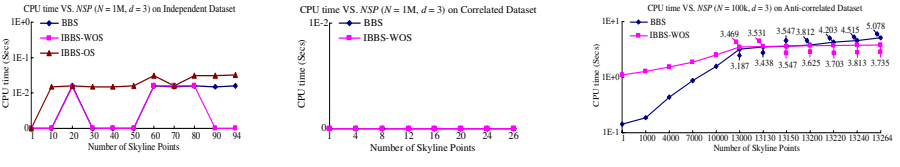


Fig. 12. CPU time VS. NSP (N = 1M or 100k, $d = 3$)

maximum size at the beginning of all algorithms, and stepwise decrease with the growth of NSP, which is also shown in Figure 10. The reason of this phenomenon is these algorithms insert respective all necessary entries visited in the heap (due to no any skyline point found) before they discover the first skyline point.

Figures 11 and 12 show all experimental results on the number of node accesses and CPU time, respectively, versus NSP under the same settings as Figure 10. Similar to Figure 6, all algorithms are I/O optimal, and their I/O costs grow as the skyline points returned increase. For CPU cost, both IBBS-WOS and BBS are similar in most cases, as well as they are faster than IBBS-OS. Additionally, in Figure 12(c), notice that BBS outperforms IBBS-WOS initially, which is caused mainly that IBBS-WOS need expend some time to remove non-qualifying entries at the beginning of it. However, the difference gradually decreases with

the NSP, and then IBBS-WOS faster than BBS. This happens because the heap of BBS keeps some redundant entries (that can be removed in IBBS-WOS using our proposed pruning heuristics of this paper).

5 Conclusion and Future Work

Although BBS has some desirable features such as I/O optimal and flexibility, it requires large main-memory consumption. Motivated by this problem, an improved skyline query algorithm based on best-first nearest neighbor search, termed IBBS, is proposed in this paper. It enables several dominance checking based pruning strategies to eliminate non-qualifying entries, thus reducing significantly the memory space and speed up slightly the skyline computation. Extensive experiments with synthetic datasets confirm that the proposed algorithm is efficient and outperforms its alternative in both space overhead (i.e., heap size) and CPU cost for most cases, especially in low dimensions. In the future, we plan to study new algorithms for skyline queries relied on breadth-first (or depth-first) nearest neighbor retrieval paradigm. Another interesting topic is to explore parallel skyline query processing methods for various parallel environments (e.g., multi-disk or multi-processor setting).

Acknowledgement. This research was supported by the National High Technology Development 863 Program of China under Grant No. 2003AA4Z3010-03.

References

1. Balke, W.-T., Gntzer, U., Zheng, J.X.: Efficient Distributed Skylining for Web Information Systems. In: EDBT. (2004) 256-273
2. Beckmann, N., Kriegel, H.-P., Schneider, R., Seeger, B.: The R*-tree: An Efficient and Robust Access Method for Points and Rectangles. In: SIGMOD. (1990) 322-331
3. Borzsony, S., Kossmann, D., Stocker, K.: The Skyline Operator. In: ICDE. (2001) 421-430
4. Böhm, C., Kriegel, H.-P.: Determining the Convex Hull in Large Multidimensional Databases. In: DaWaK. (2001) 294-306 265–318
5. Chan, C.-Y., Eng, P.-K., Tan. K.-L.: Stratified Computation of Skylines with Partially-Ordered Domains. In: SIGMOD. (2005) 203-214
6. Chang, Y.-C., Chang, Y.-C., Bergman, L.D., Castelli, V., Li, C.-S., Lo, M.-L., Smith, J.: The Onion Technique: Indexing for Linear Optimization Queries. In: SIGMOD. (2000) 391-402
7. Chomicki, J., Godfrey, P., Gryz, J., Liang, D.: Skyline with Presorting. In: ICDE. (2003) 717-719
8. Fagin, R.: Fuzzy Queries in Multimedia Database Systems. In: PODS. (1998) 1-10
9. Godfrey, P., Shipley, R., Gryz, J.: Maximal Vector Computation in Large Data Sets. In: VLDB. (2005) 229-240
10. Hjaltason, G.R., Samet, H.: Distance Browsing in Spatial Databases. ACM TODS **24** (1999) 265-318

11. Hristidis, V., Koudas, N., Papakonstantinou, Y.: PREFER: A System for the Efficient Execution of Multi-parametric Ranked Queries. In: SIGMOD. (2001) 259-270
12. Kossmann, D., Ramsak, F., Rost, S.: Shooting Stars in the Sky: An Online Algorithm for Skyline Queries. In: VLDB. (2002) 275-286
13. Lin, X., Yuan, Y., Wang, W., Lu, H.: Stabbing the Sky: Efficient Skyline Computation over Sliding Windows. In: ICDE. (2005) 502-513
14. Natsev, A., Chang, Y.-C., Smith, J.R., Li., C.-S., Vitter. J.S.: Supporting Incremental Join Queries on Ranked Inputs. In: VLDB. (2001) 281-290
15. Papadias, D., Tao, Y., Greg, F., Seeger, B.: Progressive Skyline Computation in Database Systems. ACM TODS **30** (2005)41-82
16. Pei, J., Jin, W., Ester, M., Tao, Y.: Catching the Best Views of Skyline: A Semantic Approach Based on Decisive Subspaces. In: VLDB. (2005) 253-264
17. Preparata, F., Shamos, M. Computational Geometry: An Introduction. Springer-Verlag (1985)
18. Roussopoulos, N., Kelley, S., Vincent, F.: Nearest neighbor queries. In: SIGMOD. (1995) 71-79
19. Tan, K.-L., Eng, P.-K., Ooi, B.C.: Efficient Progressive Skyline Computation. In: VLDB. (2001) 301-310
20. Theodoridis, Y., Sellis, T.K: A Model for the Prediction of R-tree Performance. In: PODS. (1996) 161-171
21. Yuan, Y., Lin, X., Liu, Q., Wang, W., Yu, J.X., Zhang, Q.: Efficient Computation of the Skyline Cube. In: VLDB. (2005) 241-252