Universität Augsburg

Evaluation of Index-Based Skyline Algorithms

M. Endres, E. Glaser

Report 2019-01 February 28, 2019

Institut für Informatik
D-86135 Augsburg

Evaluation of Index-Based Skyline Algorithms

Markus Endres, Erich Glaser

Institut for Computer Science, University of Augsburg, D-86135 Augsburg, Germany
markus.endres@informatik.uni-augsburg.de

Abstract. Skyline queries enable satisfying search results by delivering best matches, even if the filter criteria are conflictive. The result of a Skyline query consists of those objects for which there is no dominating object in the input data set. Algorithms for Skyline computation are often classified into *generic* and *index-based* approaches. While there are uncountable papers on the comparison on generic algorithms, there exists only a few publications on the effect of index-based Skyline computation. In this technical report, we evaluate the most recent index-based Skyline algorithms *BBS*, *ZSky*, and *SkyMap* in order to find out which algorithm performs best. We conducted comprehensive experiments on different data sets and present some really unexpected outcomes.

Keywords: Skyline; Pareto; Index; BBS; ZSky; SkyMap

1 Introduction

Preferences in databases are a well established framework to create personalized information systems [1]. Skyline queries [2] are the most prominent representatives of these queries; they model equally important preferences, and the aim is to find all Pareto-optimal objects. More detailed: Given a data set D, a Skyline query returns all objects that are not dominated by any other object in D. An object p is dominated by another object q, if q is at least as good as p on all dimensions and definitely better in at least one dimension. Thus, a Skyline query computes all Pareto-optimal objects w.r.t. to a preference or feature function and has many applications in multi-criteria optimization problems.

Example 1. Assume the sample data set in Table 1. Imagine that the objects are *hotels* and the x and y coordinates in the 2-dim space correspond to the *price* and *distance to the beach*. The target is to find the *cheapest* hotels which are *close to the beach*, where both preferences should be considered as equally important (a Pareto preference). Then this query would identify the hotels $\{p_1, p_2, p_3, p_5, p_6\}$ as the *Skyline* result. All objects in this set are indifferent and dominate all other objects.

Table 1. Sample data set for Skyline.

object	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8	p_9
x	3	1	2	3	5	7	6	4	6
y	3	6	4	7	2	1	2	4	6

The main problem with Skyline queries is to efficiently find the set of non-dominated objects from a large data set, because Skyline processing is an expensive operation. Its cost is mainly constituted by *I/O costs* in accessing data from a secondary storage (e.g., disks) and CPU costs spent on *dominance tests*. Note that search efficiency is one of the most important performance criteria to Skyline computation. There exist several algorithms for Skyline processing such as window-based, distributed, divide-conquer, lattice-based, and index-based algorithms. In general, these approaches can be divided into *generic* and *index-based* techniques.

Generic algorithms are often capable to evaluate each kind of preference (modeled as irreflexive and transitive order [3]) due to an object-to-object comparison approach. However, in a worst-case scenario the generic algorithms show a quadratic runtime $\mathcal{O}(n^2)$ in the size n of the input relation. Note that there are also linear algorithms, but only for limited applications [4,5]. On the other hand, index-based algorithms tend to be faster, but are less flexible – they are designed for quite static data, flat query structures and have a high maintenance overhead associated with database updates [6]. In general, they cannot deal with complex preference queries, where, e.g., intermediate relations are dynamically produced by a Cartesian product or a join.

As Skyline queries have been considered as an analytical tool in some commercial relational database systems [7,8], and the data sets to be processed in real-world applications are of considerable size, there is definitely the need for improved query performance. And indexing data is one natural choice to achieve this performance improvement. Also, Lee et al. [9] show that a wide variety of special Skyline queries (k-dominant Skylines, Skybands, Subspace Skylines, etc.) can be supported using a single index structure. While indexes can dramatically speed-up retrieval, they also introduce maintenance costs and tend to quickly degenerate on higher dimensional data.

In this report, we compare the best known index algorithms for Skyline computation, namely *BBS* [10], *ZSky* [11,9], and *SkyMap* [12] w.r.t. their performance, since search efficiency is the most important performance criteria using this kind of queries. As a baseline algorithm we use the well-known BNL approach [2]. We will present comprehensive experiments on synthetic and real-world data to evaluate the behavior in different scenarios in order to find the best approach for one's field of application.

The rest of this work is organized as follows: Section 2 presents background on Skylines and we introduce the index-based Skyline algorithms used in this report in Section 3. Section 4 contains our comprehensive experiments. In Section 5 we discuss some related work, and in Section 6 we give some final remarks.

2 Background

Before going into details on index-based Skyline processing, this section will provide some preliminary definitions. The aim of a Skyline query or Pareto preference is to find *the best matching objects* in a data set D, denoted by *Sky(D)* [13]. More formally:

Definition 1 (Dominance and Indifference). *Assume a set of vectors $D \subseteq \mathbb{R}^d$. Given $p = (p_1, ..., p_n), q = (q_1, ..., q_d) \in D$, p* dominates *q on D, denotes as $p \prec q$, if the following holds:*

$$p \prec q \Leftrightarrow \forall i \in \{1, ..., d\} : p_i \leq q_i \wedge \exists j \in \{1, ..., d\} : p_j < q_j \tag{1}$$

Two objects p and q are called indifferent *on D, denoted as $p \sim q$, if and only if $p \not\prec q$ and $q \not\prec p$.*

Note that following Definition 1, we consider a subset $D \subseteq \mathbb{R}^d$ in that we search for Skylines w.r.t. the natural order $\leq$ in each dimension. Characteristic properties of such a data set D are its dimensionality d, its cardinality n, and its Skyline size $|Sky(D)|$.

Definition 2 (Skyline Sky(D)). *The Skyline* Sky(D) *of D is defined by the maxima in D according to the ordering $\prec$, or explicitly by the set*

$$Sky(D) := \{p \in D \mid \nexists q \in D : q \prec p\} \tag{2}$$

In this sense, the minimal values in each domain are preferred and we write $p \prec q$ if p is better than q.

In Example 1 we have $Sky(D) = \{p_1, p_2, p_3, p_5, p_6\}$.

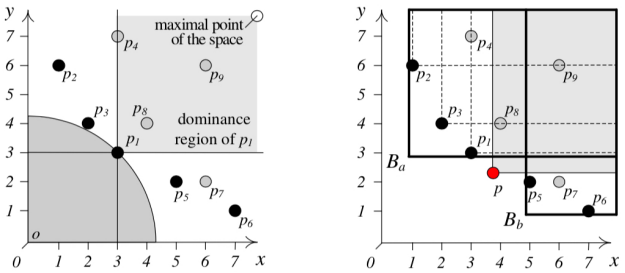
Skylines are not restricted to numerical domains. For any universe Ω and orderings $\prec_i \in (\Omega \times \Omega)(i \in \{1, ..., d\})$ the Skyline w.r.t. $\prec_i$ can be computed, if there exist scoring functions $g_i : \Omega \to \mathbb{R}$ for all $i \in \{1, ..., d\}$ such that $p \prec_i q \Leftrightarrow g_i(p) < g_i(q)$). Then the Skyline of a set $M \subseteq \Omega$ w.r.t. $(\prec_i)_{i=1,...,d}$ is equivalent to the Skyline of $\{(g(p_1), ..., g(p_d)) \mid p_i \in M\}$. That means, categorical domains can easily be mapped to a numerical domain.

3 Algorithms

In this section we review the state-of-the-art index-based Skyline algorithms *BBS*, *ZSky*, and *SkyMap* as well as *BNL* as an object comparison approach.

3.1 BBS

BBS (Branch-and-Bound Skyline) [10,14] is based on a nearest neighbor (NN) search and uses R-trees for data partitioning. As an example consider Figure 1(a) taken from [11]. The object p_1 is the first Skyline object, since it is the NN to



(a) BBS (b) Main-memory R-tree.

Fig. 1. The BBS algorithm, cp. [11].

the origin. The objects p_4, p_8, and p_9 fall into the *dominance region* of p_1 and therefore can be discarded. p_3 is the second NN (not worse than p_1) and hence is another Skyline object. The same idea applies to p_5 (which dominates p_7) and p_2 and p_6. All non-dominated objects build the Skyline.

BBS uses a *main-memory R-tree* to perform dominance tests on every examinee (i.e., data object or index node) by issuing an enclosure query. If an examinee is entirely enclosed by any Skyline candidate's dominance region, it is dominated and can be discarded. For example, in Figure 1(b), p_8 is compared with the minimum bounding rectangles (MBR) B_a and B_b. Since p_8 is in B_a, it is possibly dominated by some data objects enclosed by B_a. Hence, p_8 is compared with the dominance regions of all the data objects inside B_a and found to be dominated by p_1 and p_3.

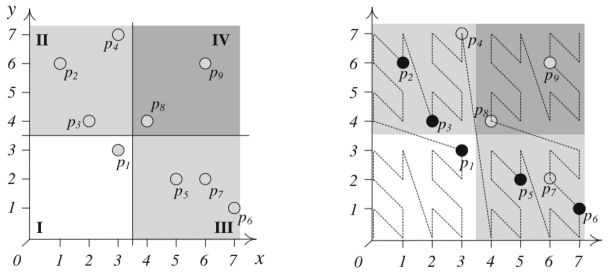
3.2 ZSky

ZSky is a framework for Skyline computation using a Z-order space filling curve [9]. A Z-order curve maps multi-dimensional data objects to one-dimensional objects. Thereby each object is represented by a bit-string computed by interleaving the bits of its coordinate values, called *Z-address*, which then can be used for B-tree indexing. Through the Z-addresses the B-tree imposes a pre-sorting on the data, which can be exploited for dominance tests: No database item can dominate any item having a lower Z-address. These observations lead to the access order of the data objects arranged on a Z-order curve.

In Figure 2(a) the data space is partitioned into four regions I to IV. Region I is not dominated by any other object, and all objects in region IV are dominated by region I. Region II and III are incomparable. These principles also apply to subregions and single coordinates. Using a Z-order curve, region I should be accessed first, followed by region II and III, and finally region IV. The access sequence therefore follows the mentioned Z-order curve as seen in Figure 2(b).

With effective region-based dominance tests, ZSky (more accurate *ZSearch*) can efficiently assert if a region of data objects is dominated by a single object or a region of Skyline objects. In each round, the region of a node is examined against

the current Skyline candidate list. If its corresponding region is not dominated, the node is further explored.



(a) ZSky regions. (b) A Z-order curve.

Fig. 2. ZSky example, cp. [9].

Z-Sky can also be used with *bulkloading*. Bulkloading builds a ZB-tree in a bottom-up fashion. It sorts all data objects in an ascending order of their Z-addresses and forms leaf nodes based on every N data objects. It also puts every N leaf nodes together to form non-leaf nodes until the root of a ZB-tree is formed.

3.3 SkyMap

Selke and Balke [12] proposed *SkyMap* for Skyline query computation. In general, SkyMap is based on the idea of the Z-order curve, but relies on a *trie* (from re*trie*val) indexing structure instead on a ZB-tree. In a trie (also known as Prefix B-tree), internal nodes are solely used for navigational purposes, whereas the leaf nodes store the actual data. SkyMap is a multi-dimensional extension of binary tries, which additionally provides an efficient method for dominance checks. The SkyMap index has primarily been designed to resemble the recursive splitting process of Z-regions.

When traversing a SkyMap index while looking for objects q dominating an object p, one can skip any node (along with all its children) whose corresponding Z-region is worse than p w.r.t. at least one dimension. Navigation within the SkyMap index is particularly efficient by relying on inexpensive bitwise operations only. In this sense, SkyMap promises efficient navigation and index maintenance which should result in a higher performance in comparison to Z-Sky.

3.4 BNL

BNL (Block-Nested-Loop) was developed by Börzsönyi [2] in 2001. The idea of BNL is to scan over the input data set D and to maintain a *window* (or block) of objects in main memory containing the temporary Skyline elements w.r.t. the

data read so far. When an object $p \in D$ is read from the input, p is compared to all objects of the window and, based on this comparison, p is either eliminated, or placed into the window. At the end of the algorithm the window contains *the Skyline*. The average case complexity is of the order $\mathcal{O}(n)$, where n counts the number of input objects. In the worst case the complexity is $\mathcal{O}(n^2)$ [2].

The major advantage of a BNL-style algorithm is its simplicity and suitability for computing the Skyline of arbitrary partial orders [3]. Note that BNL is *not* an index approach, but is used as a *baseline algorithm* in our experiments.

4 Experiments

In this section we show our comprehensive comparison study on index-based Skyline algorithms, i.e., **BBS**, **ZSky**, **ZSky-Bl** (ZSky with *bulkloading*), and **SkyMap**. As a base line algorithm we used the generic **BNL**. In all our experiments the data objects and index structures are held in main memory as has also been done by the original works [10,9,15] and [12]. All experiments were implemented in Java 1.8 and performed on a common PC (Intel i7 4.0 GHz CPU, 16 GB RAM) running Linux. We use a maximum of 4 GB RAM for the JVM.

Similar to most of the related work in the literature, we use *elapse time / runtime* as the main performance metric. Each measurement was based on 16 repetitions from which we neglected the four best and four worst runtimes. From the remaining 8 measurements we used the average runtime in our figures.

Four our synthetic data sets we used the data generator commonly used in Skyline research [2] and that one was also used by the original papers [10,9,12]. We generated *independent* (ind), *correlated* (cor), and *anti-correlated* (anti) data and varied the number of *dimensions* (d) and the number of *input objects* (n). For the experiments on real-data, we used the well-known *Zillow*, *House*, and *NBA* data sets which will be explained in detail later when used.

4.1 Effect of Data Dimensionality

In this section we consider the influence of the *number of dimensions* d on the runtime of the algorithms. We used different data distributions and varied $d \in \{4, 6, 8, 10, 15, 20, 25, 30\}$, where each dimension has the integer domain $[0, 1024)$. We fixed $n = 100K$, and plotted the elapsed time in log scale against the data dimensionality.

Independent Data. Figure 3 shows our results on synthetic independent data. Considering the index construction (on the top right, "Index"), BBS is worst and ZSky-Bl is best, because there are no special computations due to bulkloading. We also observe that the index construction time increases with growing dimensions. For the Skyline computation time (on the top left, "Skyline"), BNL outperforms some index algorithms, but has the highest runtime from 10 dimensions on. Note, that the size of the Skyline is nearly the size of the input data from 20 dimensions on and therefore the computation costs are nearly equal in these cases. In general,

BBS is the slowest algorithm, whereas there is nearly no difference between ZSky and ZSky-Bl. Based on the incremental insert of objects, we only get slightly better Z-regions. In summary, BNL performs well for less number of dimensions, whereas SkyMap performs better with increasing dimensions.

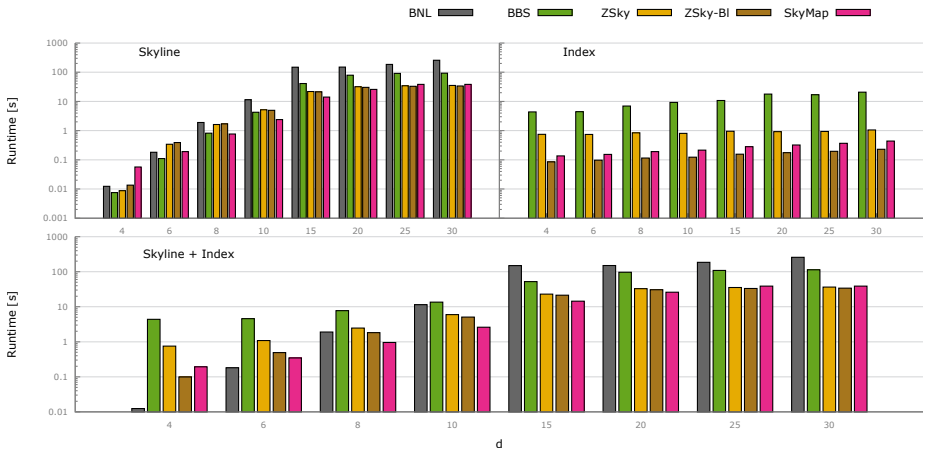


Fig. 3. Independent data. Runtime w.r.t. dimensionality.

Table 2 summarizes some statistics for the evaluation, e.g., the size of the Skyline, and the number of dominance tests. The dominance tests also includes the comparison between regions to objects and other regions in BBS and ZSky. In particular, the number of dominance tests is very high for BNL and BBS, which are mainly based on object-to-object comparisons. On the other hand, ZSky and SkyMap are able to sort out leafs or inner nodes of the index structure, which leads to a better performance and less comparisons.

Table 2. Independent data. Dominance tests w.r.t. dimensionality.

Dim	Skyline	BNL	BBS	ZSky	ZSky-Bl	SkyMap
4	246	472.733	211.994	199.086	258.971	229.151
6	2.486	8.686.399	5.338.050	7.189.352	7.560.025	2.719.943
8	9.671	88.058.715	51.243.124	32.050.250	31.739.487	9.780.642
10	25.673	465.239.329	336.695.082	95.895.490	94.628.219	33.612.608
15	76.944	3.265.808.129	2.967.079.852	411.655.318	409.392.015	168.155.529
20	97.034	4.794.518.883	4.709.122.182	602.389.909	599.937.128	285.420.489
25	99.806	4.988.782.311	4.980.685.164	649.692.554	647.307.956	315.818.979
30	99.995	4.999.628.298	4.999.453.504	650.497.209	648.157.904	316.831.960

Correlated Data. In **Figure 4** we present results on correlated data. The index construction time is very similar to the results of independent data: The data distribution has nearly no influence on the tree construction. It is notable that for the used correlated data exactly one point was in the Skyline. This, of course, speeds-up Skyline computation. As a result, BBS shows a very good behavior for every number of dimensions, because BBS checks the temporary Skyline set (one point) against the points in the R-tree. Since there are many boxes which are dominated early, BBS performs extraordinary good. A similar behavior can be found for Z-Sky, whereas SkyMap performs worst.

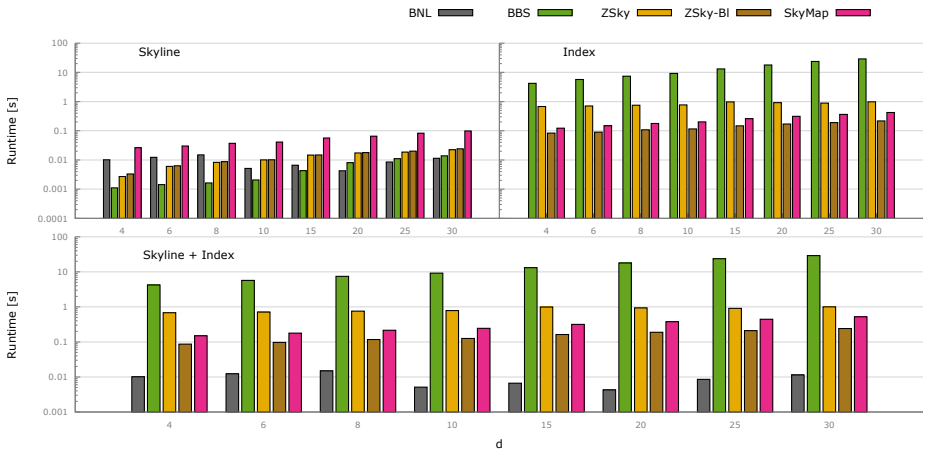


Fig. 4. Correlated data. Runtime w.r.t. dimensionality.

Table 3 shows the number of dominance tests for correlated data. Interestingly, SkyMap uses exactly 99.999 dominance tests for all eight experiments. This is due to the fact that the first point is added to the list and afterwards all other points in the list are tested against this point. All other algorithms need more dominance test with increasing dimensions.

Table 3. Correlated data. Dominance tests w.r.t. dimensionality.

Dim	Skyline	BNL	BBS	ZSky	ZSky-Bl	SkyMap
4	1	100.585	476	24.009	28.200	99.999
6	1	100.761	409	45.510	48.600	99.999
8	1	100.616	380	55.097	56.600	99.999
10	1	105.989	428	58.802	61.000	99.999
15	1	106.876	484	64.031	64.400	99.999
20	1	137.654	698	65.655	65.800	99.999
25	1	136.053	757	66.419	66.400	99.999
30	1	130.694	770	67.645	67.600	99.999

Anti-Correlated Data. Figure 5 shows our results on anti-correlated data. Anti-correlated data is the worst-case for Skyline computation, because there are many indifferent objects and the result set is large. The costs for index creation and Skyline computation is very similar to independent data. Considering the total costs ("Skyline + Index"), BNL is better than all index-based approaches until 6 dimensions. In higher dimensions BBS, ZSky, and ZSky-Bl are nearly equally good and all are outperformed by SkyMap. Furthermore, SkyMap is much better than all other algorithms w.r.t. the pure Skyline computation. These results are also reflected by the numbers in **Table 4**. SkyMap uses the lowest number of dominance tests.

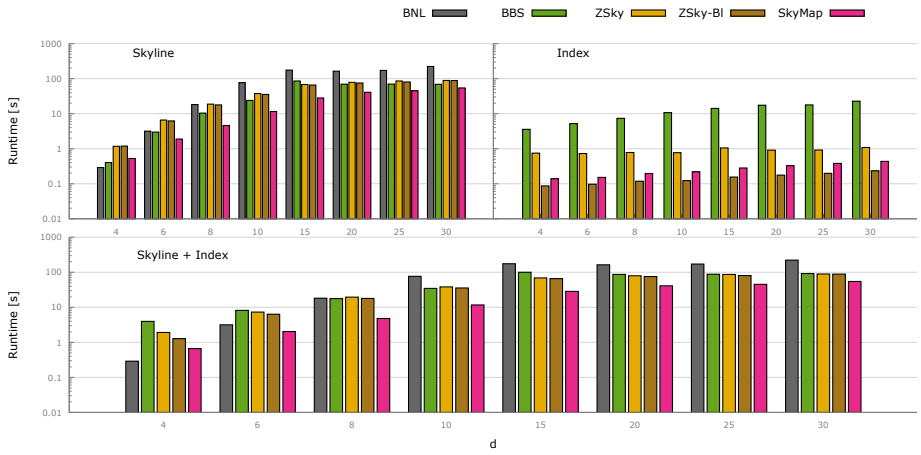


Fig. 5. Anti-correlated data. Runtime w.r.t. dimensionality.

Table 4. Anti-correlated data. Dominance tests w.r.t. dimensionality.

Dim	Skyline	BNL	BBS	ZSky	ZSky-Bl	SkyMap
4	3.465	17.819.929	76.151.238	31.583.925	34.211.658	16.059.065
6	14.076	175.948.579	507.586.113	139.902.463	139.814.862	46.419.955
8	34.278	823.124.157	1.741.955.033	325.706.677	324.366.814	123.623.731
10	58.508	2.108.683.666	3.346.922.807	612.903.441	610.423.362	415.030.123
15	94.400	4.603.053.657	5.892.636.374	1.066.961.134	1.063.518.245	804.763.547
20	99.669	4.979.353.245	6.295.607.429	1.169.862.079	1.166.292.765	876.884.805
25	99.933	4.995.078.177	6.242.921.947	1.193.496.757	1.189.897.954	924.878.532
30	99.978	4.999.135.061	6.187.246.550	1.185.961.162	1.182.256.403	921.935.565

Memory Usage **Figure 6** shows the memory usage (in 10^6 bytes) of BBS, ZSky, ZSky-Bl, and SkyMap for different number of dimensions. BBS and ZSky require the highest amount of memory, whereas ZSky-Bl and SkyMap are more frugal. This is due the small 'inner index structure' resp. the sorted list in SkyMap. Nevertheless, increasing dimensions lead to increasing memory usage.

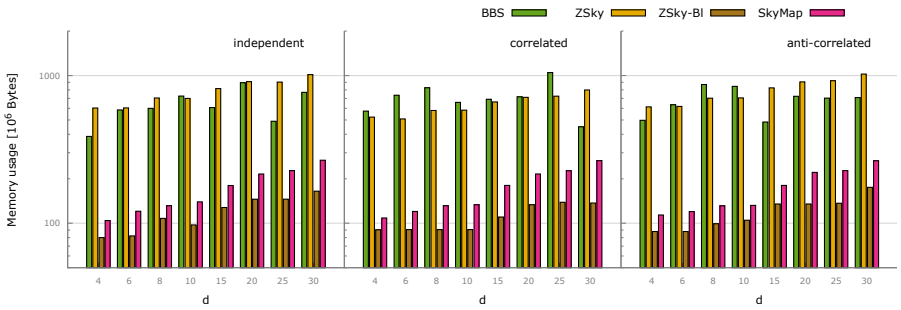


Fig. 6. Memory usage w.r.t. dimensionality.

We also investigated the structure of the index approaches in **Table 5**. We considered the number of *inner nodes*, the number of *leafs*, and the *height* of the trees. It is visible that ZSky-Bl always constructs the same tree due to its bulkloading technique.

Table 5. Independent data. Tree structure w.r.t. dimensionality.

dim	BBS			ZSky			ZSky-Bl		
	#in. nodes	#leafs	height	#in. nodes	#leafs	height	#in. nodes	#leafs	height
4	92	1.626	3	5	683	3	4	500	3
6	37	1.343	3	5	702	3	4	500	3
8	67	1.393	3	5	715	3	4	500	3
10	73	1.461	3	5	693	3	4	500	3
15	41	1.324	3	5	696	3	4	500	3
20	74	1.399	3	5	706	3	4	500	3
25	31	1.268	3	5	706	3	4	500	3
30	36	1.281	3	5	693	3	4	500	3

4.2 Effect of Data Cardinality

In the following experiments we considered the influence of the *data input size* n using the following characteristics: Integer domain in $[0, 1024)$, $d = 8$ dimensions, input size $n \in \{10K, 100K, 500K, 1000K, 2000K\}$.

Independent Data. Figure 7 shows that ZSky and ZSky-Bl perform worse from $n = 500K$ objects on w.r.t. the Skyline computation. Even BNL as an object-to-object comparison algorithm is faster. This is based on the fact that the underlying ZB-tree constructs index nodes very fast, and due to less common prefixes this results in very large Z-regions which must be checked for dominance. SkyMap is definitely better than its competitors, because of its trie index structure. Also BBS is better than the ZSky approaches, although it is the oldest of all algorithms. On the other hand, BBS is really worse w.r.t. the index construction time because of the linear splits. The SkyMap sorting is a bit more costly than the filling of the ZB-trees via bulkloading.

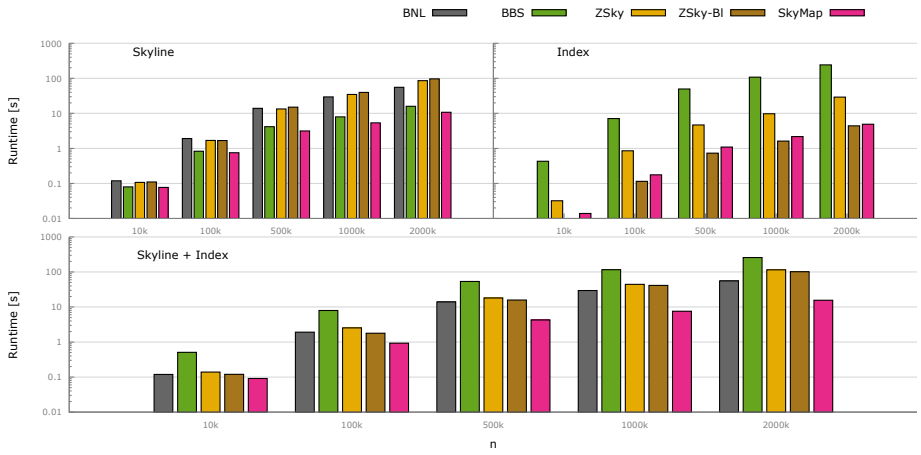


Fig. 7. Independent data. Runtime w.r.t. input size.

Table 6 shows the number of dominance tests, where SkyMap clearly outperforms all other algorithms. It is notable that in ZSky the number of index nodes highly increases and therefore it builds larger Z-regions, which in the end lead to a higher runtime.

Table 6. Independent data. Dominance tests w.r.t. input size.

n	|Skyline|	BNL	BBS	ZSky	ZSky-Bl	SkyMap
10k	2.591	5.495.792	3.532.499	2.138.624	2.110.187	1.187.352
100k	9.671	88.058.715	51.243.124	32.050.250	31.739.487	9.780.642
500k	22.302	539.332.777	287.752.839	239.295.317	243.096.515	48.935.100
1000k	30.332	1.086.388.323	556.550.102	537.592.064	562.808.810	77.029.696
2000k	39.301	2.048.044.727	994.592.267	1.215.101.764	1.300.961.556	132.732.900

Correlated Data. We also considered the input size w.r.t. correlated data. **Figure 8** presents the results. Again, our data generator produced correlated data with exactly *one Skyline point* independently from the cardinality. It can be seen that SkyMap is the slowest algorithm, because SkyMap has to process the complete input list. Even BNL is faster. ZSky and ZSky-Bl are better but outperformed by BBS. However, BBS has very high index construction costs. We also see that the number of dominance tests increases with higher cardinality, **Table 7**.

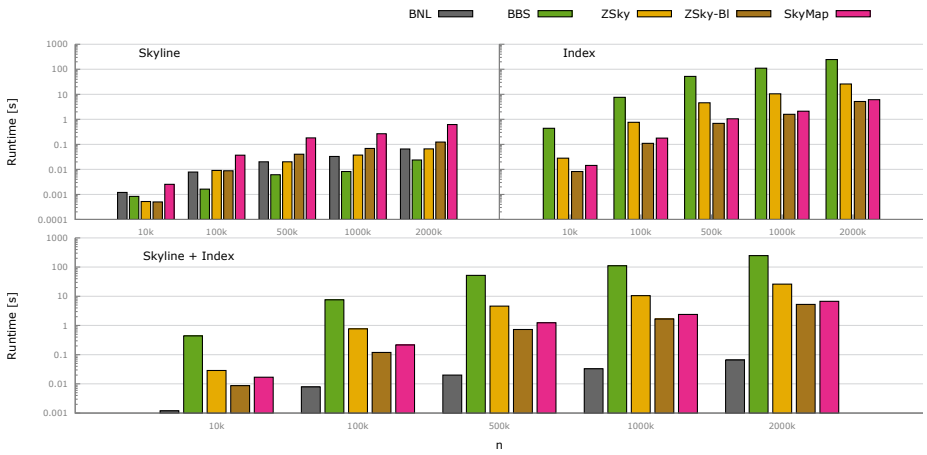


Fig. 8. Correlated data. Runtime w.r.t. input size.

Table 7. Correlated data. Dominance tests w.r.t. input size.

n	|Skyline|	BNL	BBS	ZSky	ZSky-Bl	SkyMap
10k	1	10.616	202	6.222	6.248	9.999
100k	1	100.616	380	55.097	56.600	99.999
500k	1	500.616	1.112	230.943	239.610	499.999
1000k	1	1.000.616	2.255	392.334	414.222	999.999
2000k	1	2.000.616	4.583	671.706	718.047	1.999.999

Anti-Correlated Data. Figure 9 and **Table 8** show our results on anti-correlated data. Anti-correlated data lead to many Skyline objects and therefore are more challenging for Skyline algorithms. Clearly, BNL shows a bad performance because of many objects comparisons. BBS is quite good on less data objects but slows down with increasing number of objects. Even ZSky becomes worse because of larger Z-regions. The winner is definitely SkyMap, which outperforms all other algorithms by far.

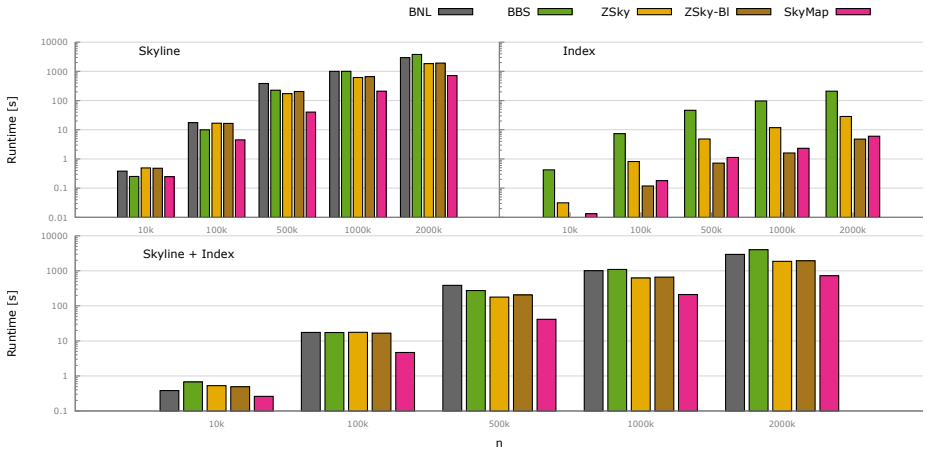


Fig. 9. Anti-Correlated data. Runtime w.r.t. input size.

Table 8. Anti-correlated data. Dominance tests w.r.t. input size.

n	Skyline	BNL	BBS	ZSky	ZSky-Bl	SkyMap
10k	5.754	21.792.379	32.411.570	8.738.591	8.692.718	5.759.944
100k	34.278	823.124.157	1.741.955.033	325.706.677	324.366.814	123.623.731
500k	103.719	8.403.576.644	23.265.561.175	2.890.122.498	2.877.608.350	1.284.343.881
1000k	164.304	21.457.709.801	70.307.815.653	7.594.185.785	7.569.948.205	3.683.825.552
2000k	250.442	53.123.931.360	199.088.948.328	18.903.874.716	18.829.325.685	10.561.851.499

Memory Usage Considering the memory usage, **Figure 10**, there is no clear winner. BBS in general has a very high memory consumption. The memory usage for ZSky and SkyMap is low, but strongly increases with larger input data.

Table 9 shows the structure of the used trees, which is not very spectacular, and does not explain the strange behavior in our memory measurements before.

Table 9. Independent data. Tree structure w.r.t. input size.

	BBS			ZSky			ZSky-Bl		
n	#in. nodes	#leafs	height	#in. nodes	#leafs	height	#in. nodes	#leafs	height
10k	3	132	3	1	66	2	1	50	3
100k	67	1.393	3	5	715	3	4	500	3
500k	299	7.030	4	33	4.081	3	14	2.500	3
1000k	529	13.854	4	65	8.166	3	26	5.000	3
2000k	1.089	27.842	4	129	16.332	3	51	1.000	3

Fig. 10. Memory usage w.r.t. input size

4.3 Effect of Domain Size

We now examine the influence of the domain size. Instead of considering domains in $[0, 1024)$, we utilize a domain size of $[0, \{2^5, 2^{10}, 2^{15}, 2^{20}, 2^{25}, 2^{30}\})$ for each dimension. In addition, we set $d = 5$, $n = 10^6$ and used independent data. **Figure 11** shows our results.

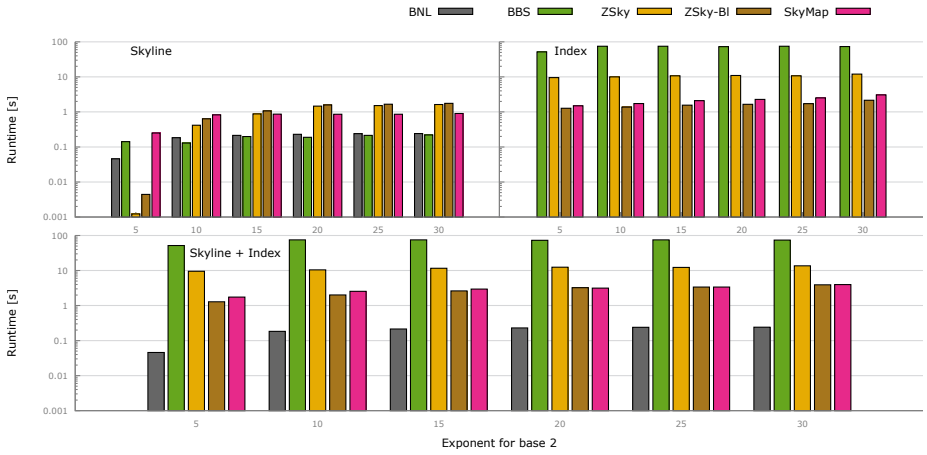


Fig. 11. Independent data. Runtime w.r.t. domain size.

It is notable that ZSky is highly efficient for $[0, 2^5)$, but worse for higher domains w.r.t. Skyline computation runtime. BBS and BNL are much better than ZSky and SkyMap for higher dimensions. This is due to the Z-addresses, which are stored as bits, and these bits are based on the domain values. That means, when using a maximal domain value of 2^5 on 5 dimensions we need 25 bits per Z-address, and 150 bits for 2^{30} values. This leads to the high computation

costs. Therefore, algorithms using Z-addresses are mainly applicable for "low-cardinality" domains. On the other hand, the runtime of BNL and BBS are quite good, because they are based on an object comparison where a high or low cardinality domain does not matter. Considering the index constructions costs, BBS and ZSky are worse than ZSky-Bl and SkyMap.

Table 10 shows the number of dominance tests. SkyMap ist better than its competitors in most cases w.r.t. the dominance tests but performs worse w.r.t. the runtime.

Table 10. Independent data. Dominance tests w.r.t. the domain size.

Werte	|Skyline|	BNL	BBS	ZSky	ZSky-Bl	SkyMap
2^5	25	2.068.282	153.226	9.154	30.160	1.036.345
2^{10}	1.277	9.228.729	5.329.707	8.466.249	11.399.219	2.856.159
2^{15}	1.787	12.082.605	7.783.663	17.743.135	20.071.884	3.476.130
2^{20}	1.842	12.301.452	7.519.476	27.879.935	28.617.562	3.518.037
2^{25}	1.843	12.337.183	7.639.781	29.074.243	29.569.252	3.520.676
2^{30}	1.843	12.337.183	7.639.781	29.074.255	29.569.264	3.520.688

4.4 Real Data

For our experiments on real world data we used the well-known *Zillow* data set, which consists of 5 dimensions and 1.288.684 distinct objects. Zillow represents real estates in the United States and stores information about the number of rooms, base area, year of construction, and so on. The *House* data set is a 6-dimensional database of 127.931 objects and represents the average costs of a family in the USA for water, electricity, etc. Our third real data set is *NBA*, a 5-dimensional data with 17.265 entries about NBA players. For the sake of convenience, we search the objects with the lowest values, i.e., the smallest flat, the thrifty American and the worst basketball player. Note that ZSky is not able to deal with duplicates and hence we reduced all data sets to its essence.

Figure 12 shows that ZSky is best for the Zillow data set. This is obvious, because the Skyline only exists of 1 object. In contrast, the runtime of SkyMap, similar to our other tests, is quite high for small Skyline sets, i.e., Zillow and NBA, whereas it performs better for House. Considering the House data set, BBS and SkyMap perform best when considering the pure Skyline computation, even though BBS is much older than SkyMap. On the other hand, SkyMap produces lower index maintenance costs. In the NBA data set, BNL outperforms its competitors because the input data set is relatively small.

Table 11 presents the number of dominance tests used to find the Skyline. In particular, ZSky uses only a few dominance tests on the Zillow data set. This is due to the early rejection of Z-regions, which avoids many object-to-object comparisons.

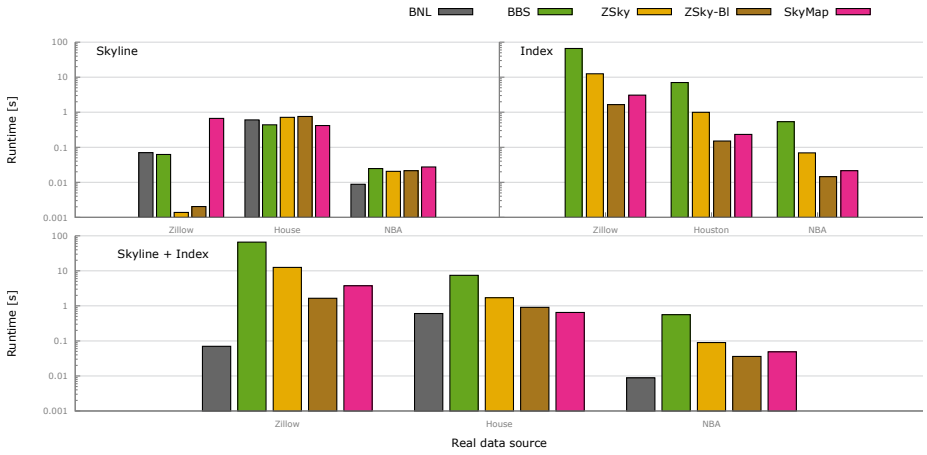


Fig. 12. Real data.

Table 11. Real data. Dominance tests.

Data source	Zillow	House	NBA
n	1.288.684	127.931	17.265
dim	5	6	5
Skyline	1	5.762	493
	Dominanztests		
BNL	1.289.021	24.945.919	412.469
BBS	36.425	23.669.845	765.889
ZSky	794	24.305.540	798.783
ZSky-Bulk	1.504	23.585.699	833.547
SkyMap	1.288.683	5.389.686	533.139

5 Related Work

Since the introduction of Skyline queries by Börzsönyi et al. in 2001 (cp. [2]) to retrieve Pareto-optimal objects from a database, various algorithms have been proposed. These algorithms can be categorized into *index-based* and *generic* approaches.

Index-based algorithms offer very good performance under some circumstances, but the major problem is the lack of applicability for computations with joined relations, cp. [10]. This leads to poor applicability of index-based algorithms if they are not used in well-defined static use cases. Another problem is the index maintenance costs if the data set of the index is constantly changing. The maintenance may generate more computational costs than the Skyline queries themselves. In general, index-based algorithms cannot deal with complex Skyline queries [6]. Despite this lack of generality, index-based Skyline techniques have been a center of interest in the last few years. Examples for index-based

approaches are for instance *Index* by Tan et al. using B^+-trees [16], or Kossmann et al.'s *Nearest Neighbor* [17]. Using R-trees, *BBS* [10,14,18] proved to be even faster, outperforming generic algorithms by far. In [15,11,9] the ZB-tree was presented. This index data structure was specifically designed for Skyline queries and made the *ZINC* and *ZSearch* algorithm using it faster than BBS with less memory consuming. Another state-of-the art algorithm is *SkyMap* [12], which exploits trie-based index structure to solve the major efficiency bottlenecks like fast access and superior scalability for higher dimensions, in particular for realistic applications.

Generic algorithms involve solutions that do not require any pre-processing of the underlying data set. Despite the lower performance compared to index algorithms, *generic algorithms* are capable of processing arbitrary data without any preparations. Each object has to be read and analyzed at least once, something a good index-base algorithm never has to do. Well-known generic Skyline algorithms are BNL [2], SFS [19], LESS [20], SaLSa [21], BSkyTree [22], ARL-S [4], Scalagon [23,24], just to list a few.

6 Summary and Conclusion

In this work we briefly reviewed the well-known index-based Skyline algorithms BBS, ZSky, and SkyMap. In order to apply the most efficient index structure in database systems, we presented comprehensive experiments on synthetic and real-world data to evaluate the performance of the presented algorithms. As expected, none of the algorithms performs best for all experiments. The decision for an algorithm must be based on the application it should be used for.

BNL is quite good for a small number of dimensions, whereas SkyMap shows its advantages for higher dimensions. We have also seen that with increasing data dimensionality the performance of R-trees and hence of BBS deteriorates. On the other hand, BBS and SkyMap outperform the other algorithms with increasing input size, independently from the data distribution. When considering the domain size, BNL and BBS are better than their competitors and therefore should be preferred for high cardinality domains. The Z-Sky approaches do well in the case of real data. However, one of the drawbacks of Z-Sky is its restriction to total orders. Duplicates are not allowed. In addition, in the ZB-tree approach regions may overlap, which hampers effective pruning. Moreover, the maintenance of B-trees is rather expensive in case of frequent updates, in particular due to rebalancing operations caused by node underflows.

In a nutshell, one has carefully to decide which algorithm should be used w.r.t. the application, and this report gives some decision criteria.

References

1. W. Kießling, M. Endres, and F. Wenzel. The Preference SQL System - An Overview. *Bulletin of the Technical Commitee on Data Engineering, IEEE Computer Society*, 34(2):11–18, 2011.
2. S. Börzsönyi, D. Kossmann, and K. Stocker. The Skyline Operator. In *Proceedings of ICDE '01*, pages 421–430, Washington, DC, USA, 2001. IEEE.
3. M. Endres. The Structure of Preference Orders. In *ADBIS '15: The 19th East European Conference in Advances in Databases and Information Systems*. Springer, 2015.
4. M. Endres and W. Kießling. High Parallel Skyline Computation over Low-Cardinality Domains. In *Proceedings of ADBIS '14*, pages 97–111. Springer, 2014.
5. M. Endres and L. Rudenko. *A Tour of Lattice-Based Skyline Algorithms*. In M. K. Habib (eds.): Emerging Investigation in Artificial Life Research and Development, IGI Global, 2018.
6. M. Endres and F. Weichmann. Index Structures for Preference Database Queries. In *FQAS '17: Proceedings of the 12th International Conference on Flexible Query Answering Systems*, Lecture Notes in Computer Science, pages 137–149. Springer-Verlag, 2017.
7. S. Chaudhuri, N. Dalvi, and R. Kaushik. Robust Cardinality and Cost Estimation for Skyline Operator. In *Proceedings of ICDE '06*, page 64, Washington, DC, USA, 2006. IEEE Computer Society.
8. S. Mandl, O. Kozachuk, M. Endres, and W. Kießling. Preference Analytics in EXASolution. In *Proceedings of BTW '15*, 2015.
9. K. C. K. Lee, W.-C. Lee, B. Zheng, H. Li, and Y. Tian. Z-SKY: An Efficient Skyline Query Processing Framework Based on Z-Order. *VLDB Journal*, 19(3):333–362, 2009.
10. D. Papadias, Y. Tao, G. Fu, and B. Seeger. An Optimal and Progressive Algorithm for Skyline Queries. In *Proceedings of SIGMOD '03*, pages 467–478. ACM, 2003.
11. K. Lee, B. Zheng, H. Li, and W.-C. Lee. Approaching the Skyline in Z Order. In *Proceedings of VLDB '07*, pages 279–290. VLDB Endowment, 2007.
12. J. Selke and W.-T. Balke. SkyMap: A Trie-Based Index Structure for High-Performance Skyline Query Processing. In *Proceedings of DEXA '11*, volume 6861 of *LNCS*, pages 350–365. Springer, 2011.
13. J. Chomicki, P. Ciaccia, and N. Meneghetti. Skyline Queries, Front and Back. *Proceedings of SIGMOD '13*, 42(3):6–18, 2013.
14. D. Papadias, Y. Tao, G. Fu, and B. Seeger. Progressive Skyline Computation in Database Systems. *ACM TODS*, 30(1):41–82, 2005.
15. B. Liu and C.-Y. Chan. ZINC: Efficient Indexing for Skyline Computation. *Proc. VLDB Endow.*, 4(3):197–207, 2010.
16. K.-L. Tan, P.-K. Eng, and B. C. Ooi. Efficient Progressive Skyline Computation. In *Proceedings of VLDB '01*, pages 301–310, San Francisco, USA, 2001.
17. D. Kossmann, F. Ramsak, and S. Rost. Shooting Stars in the Sky: An Online Algorithm for Skyline Queries. In *Proceedings of VLDB '02*, pages 275–286.
18. C. Y. Chan, P.-K. Eng, and K.-L. Tan. Stratified Computation of Skylines with Partially-ordered Domains. In *SIGMOD '05*, pages 203–214, New York, NY, USA, 2005. ACM.
19. J. Chomicki, P. Godfrey, J. Gryz, and D. Liang. Skyline with Presorting. In *Proceedings of ICDE '03*, pages 717–816, 2003.

20. P. Godfrey, R. Shipley, and J. Gryz. Maximal Vector Computation in Large Data Sets. In *Proceedings of VLDB '05*, pages 229–240. VLDB Endowment, 2005.
21. I. Bartolini, P. Ciaccia, and M. Patella. SaLSa: Computing the Skyline without Scanning the Whole Sky. In *Proceedings of CIKM '06*, pages 405–414, New York, NY, USA, 2006. ACM.
22. J. Lee and S w. Hwang. BSkyTree: Scalable Skyline Computation Using a Balanced Pivot Selection. In *Proceedings of EDBT '10*, pages 195–206, NY, USA, 2010. ACM.
23. M. Endres, P. Roocks, and W. Kießling. Scalagon: An Efficient Skyline Algorithm for all Seasons. In *Proceedings of DASFAA '15*, 2015.
24. M. Endres and W. Kießling. Parallel Skyline Computation Exploiting the Lattice Structure. *JDM: Journal of Database Management*, 2016.