Indexing for Skyline Computation
A Comparison Study

Markus Endres(✉) and Erich Glaser

Faculty of Computer Science and Mathematics, University of Passau,
Innstr. 33, 94032 Passau, Germany
markus.endres@uni-passau.de, erichglaser@gmail.com

Abstract. Skyline queries enable satisfying search results by delivering best matches, even if the filter criteria are conflictive. Skyline algorithms 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 paper, we give an overview on the most recent index-based Skyline algorithms *BBS*, *ZSky*, and *SkyMap*. We conducted comprehensive experiments on different data sets and present some really interesting outcomes.

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

1 Introduction

Preferences in databases are a well established framework to create personalized information systems [7]. Skyline queries [1] are the most prominent representatives of these queries; they model equally important preferences.

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.

As an example consider 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*. 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.

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*.

There exist several algorithms for Skyline processing which, in general, can be divided into *generic* and *index-based* techniques.

A. Cuzzocrea et al. (Eds.): FQAS 2019, LNAI 11529, pp. 31–42, 2019.
https://doi.org/10.1007/978-3-030-27629-4_6

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

Generic algorithms are often capable to evaluate each kind of preference (modeled as irreflexive and transitive order [4]) 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. 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 [2,11], 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 paper, we compare the best known index algorithms for Skyline computation, namely *BBS* [12], *ZSky* [8,9], and *SkyMap* [14] w.r.t. their performance, since search efficiency is the most important performance criteria using this kind of queries. 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 the paper is organized as follows: Sect. 2 presents background on Skylines and we introduce the index-based Skyline algorithms used in this paper in Sect. 3. Section 4 contains our comprehensive experiments and in Sect. 5 we give some final remarks.

2 Preliminaries

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)* [3]. 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:*

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

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.

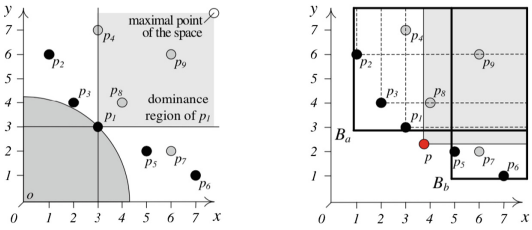
In the introductory example we have $Sky(D) = \{p_1, p_2, p_3, p_5, p_6\}$.

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) [12,13] is based on a nearest neighbor (NN) search and uses R-trees for data partitioning. As an example consider Fig. 1a taken from [8]. The object p_1 is the first Skyline object, since it is the NN to 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.



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

Fig. 1. The BBS algorithm, cp. [8].

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

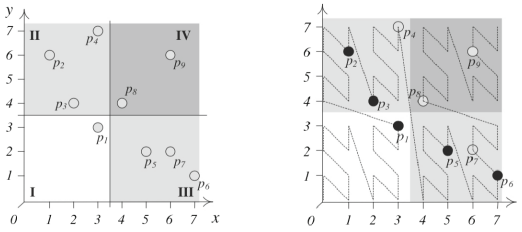
is entirely enclosed by any Skyline candidate's dominance region, it is dominated and can be discarded. For example, in Fig. 1b, p_8 is compared with the minimum bounding rectangles (MBR) B_a and B_b. Since p_8 is in B_a, it is possibly dominated by some data objects enclosed by B_a. Hence, p_8 is compared 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 Fig. 2a 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 Fig. 2b.

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 [14] 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 [1] 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 $\mathcal{O}(n^2)$ [1].

The major advantage of a BNL-style algorithm is its simplicity and suitability for computing the Skyline of arbitrary partial orders [4]. 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 [9,10,12] and [14]. 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 [1] and that one was also used by the original papers [9,12,14]. 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. Due to the restricted space in this paper we only present some characteristic results. More experiments and details can be found in our Technical Report [5].

4.1 Effect of Data Dimensionality

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

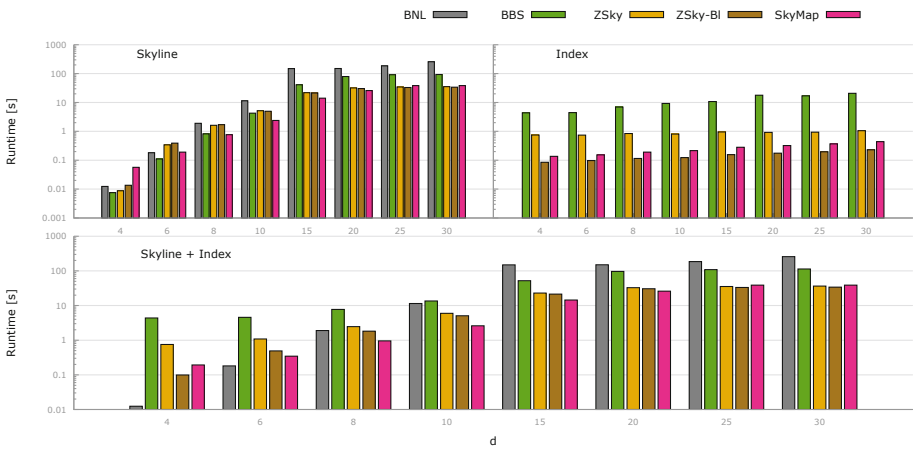


Fig. 3. Independent data. Runtime w.r.t. dimensionality.

Table 2(a) summarizes some statistics for the evaluation, e.g., the size of the Skyline, and the number of dominance tests. The dominance tests also include 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 etter performance and less comparisons.

Table 2. Dominance tests $\cdot 10^6$ w.r.t. dimensionality.

Dim	Skyline	BNL	BBS	ZSky	ZSky-Bl	SkyMap
4	246	0.472	0.211	0.199	0.258	0.229
6	2486	8	5	7	7	2
8	9671	88	51	32	31	9
10	25673	465	336	95	94	33
15	76944	3265	2967	411	409	168
20	97034	4794	4709	602	599	285
25	99806	4988	4980	649	647	315
30	99995	4999	4999	650	648	316

(a) Independent data.

Dim	Skyline	BNL	BBS	ZSky	ZSky-Bl	SkyMap
4	3465	17	76	31	34	16
6	14076	175	507	139	139	46
8	34278	823	1741	325	324	123
10	58508	2108	3346	612	610	415
15	94400	4603	5892	1066	1063	804
20	99669	4979	6295	1169	1166	876
25	99933	4995	6242	1193	1189	924
30	99978	4999	6187	1185	1182	921

(b) Anti-correlated data.

Anti-correlated Data. Figure 4 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 2(b). SkyMap uses the lowest number of dominance tests.

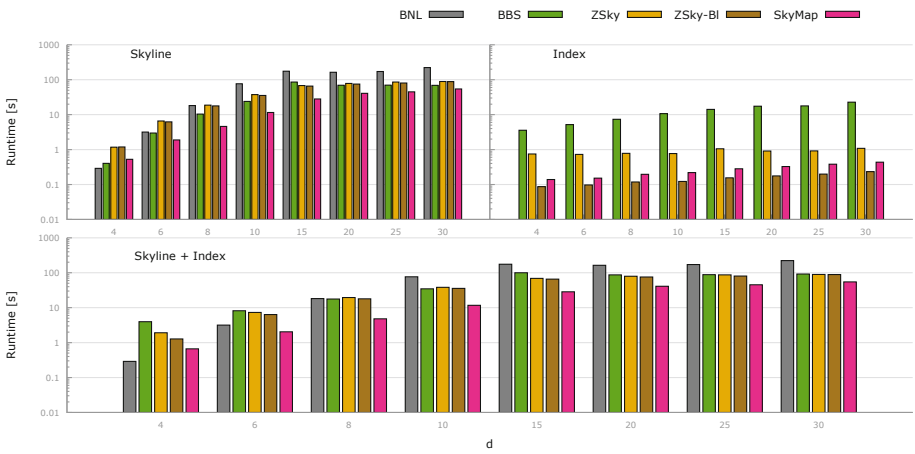


Fig. 4. Anti-correlated data. Runtime w.r.t. dimensionality.

4.2 Effect of Data Cardinality

In the next 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 5 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.

Table 3(a) shows the number of dominance tests, where SkyMap clearly outperforms all other algorithms. It is notable that in ZSky the number of index nodes increase. Therefore, the algorithm builds larger Z-regions, which in the end lead to a higher runtime.

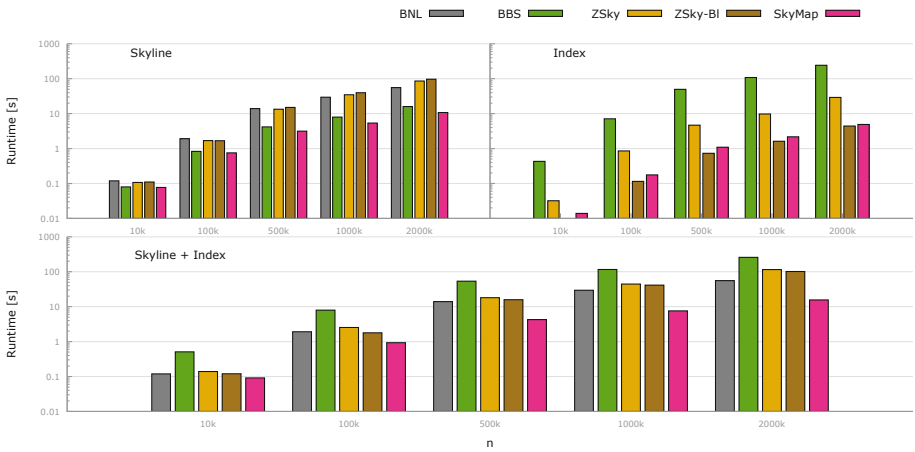


Fig. 5. Independent data. Runtime w.r.t. input size.

Anti-correlated Data. Figure 6 and Table 3(b) 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 object 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.

Table 3. Dominance tests $\cdot 10^6$ w.r.t. input size.

n	Skyline	BNL	BBS	ZSky	ZSky-Bl	SkyMap
10k	2591	5.5	3.5	2.2	2.1	1.8
100k	9671	88	51	32	31	9
500k	22302	539	287	239	243	48
1000k	30332	1086	556	537	562	77
2000k	39301	2048	994	1215	1300	132

(a) Independent data.

n	Skyline	BNL	BBS	ZSky	ZSky-Bl	SkyMap
10k	5754	21.8	32.4	8.7	8.7	5.7
100k	34278	823	1741	325	324	123
500k	103719	8403	23265	2890	2877	1284
1000k	164304	21457	70307	7594	7569	3683
2000k	250442	53123	199088	1890	18829	10561

(b) Anti-correlated data.

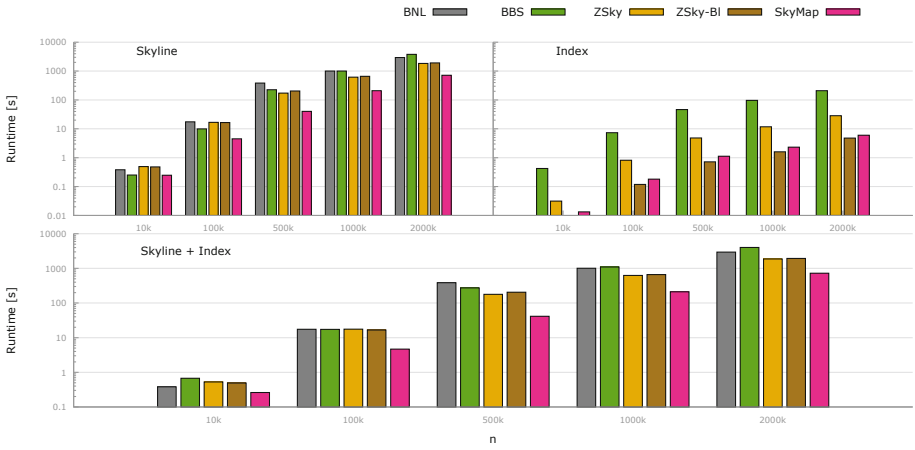


Fig. 6. Anti-correlated data. Runtime w.r.t. input size.

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 7 shows our results. 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 4 shows the number of dominance tests. SkyMap is better than its competitors in most cases w.r.t. the dominance tests, but performs worse w.r.t. the runtime.

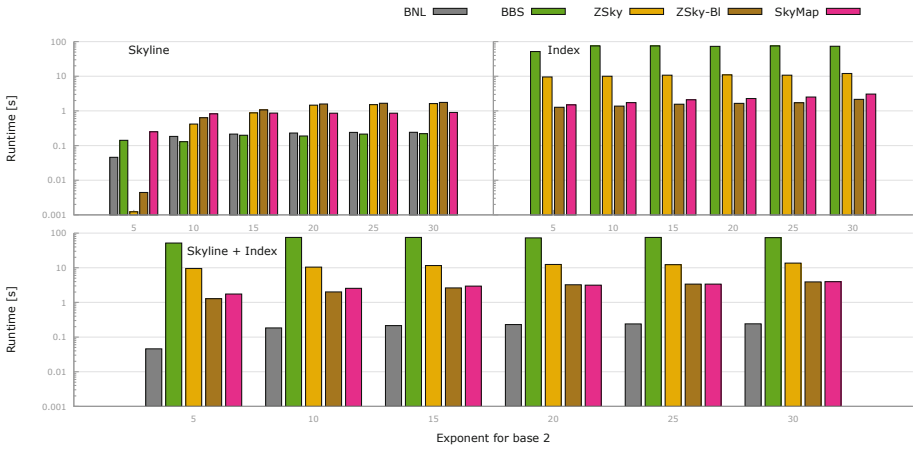


Fig. 7. Independent data. Runtime w.r.t. domain size.

Table 4. Independent data. Dominance tests $\cdot 10^6$ w.r.t. the domain size.

Domain size	Skyline	BNL	BBS	ZSky	ZSky-Bl	SkyMap
2^5	25	2.1	0.2	0.009	0.03	1.0
2^{10}	1277	9.2	5.3	8.5	11.3	2.8
2^{15}	1787	12.1	7.7	17.7	20.0	3.4
2^{20}	1842	12.3	7.5	27.8	28.6	3.5
2^{25}	1843	12.3	7.6	29.0	29.5	3.5
2^{30}	1843	12.3	7.6	29.0	29.5	3.5

Table 5. 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
	Dominance tests $\cdot 10^3$		
BNL	1.289	24.945	412
BBS	36	23.669	765
ZSky	0.794	24.305	798
ZSky-Bulk	1.5	23.585	833
SkyMap	1.288	5.389	533

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 8 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 5 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.

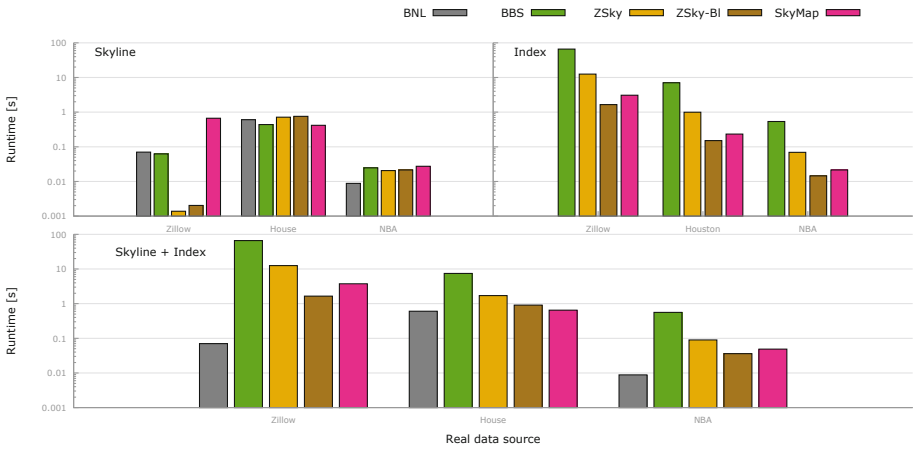


Fig. 8. Real data.

5 Summary and Conclusion

In this paper 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.

Based on these results, it will be necessary to develop a cost-based algorithm selection, which automatically decides which approach should be used. But this remains future work.

References

1. Börzsönyi, S., Kossmann, D., Stocker, K.: The skyline operator. In: Proceedings of ICDE 2001, pp. 421–430. IEEE, Washington (2001)
2. Chaudhuri, S., Dalvi, N., Kaushik, R.: Robust cardinality and cost estimation for skyline operator. In: Proceedings of ICDE 2006, p. 64. IEEE Computer Society, Washington (2006)
3. Chomicki, J., Ciaccia, P., Meneghetti, N.: Skyline queries, front and back. Proc. SIGMOD Rec. **42**(3), 6–18 (2013)
4. Endres, M.: The structure of preference orders. In: Morzy, T., Valduriez, P., Bellatreche, L. (eds.) ADBIS 2015. LNCS, vol. 9282, pp. 32–45. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-23135-8_3
5. Endres, M., Glaser, E.: Evaluation of index-based skyline algorithms. Technical report 2019–01, University of Augsburg, Institute of Computer Science (2019). https://opus.bibliothek.uni-augsburg.de/opus4/49414
6. Endres, M., Weichmann, F.: Index structures for preference database queries. In: Christiansen, H., Jaudoin, H., Chountas, P., Andreasen, T., Legind Larsen, H. (eds.) FQAS 2017. LNCS (LNAI), vol. 10333, pp. 137–149. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-59692-1_12
7. Kießling, W., Endres, M., Wenzel, F.: The preference SQL system - an overview. Bull. Tech. Commitee Data Eng. **34**(2), 11–18 (2011)
8. Lee, K., Zheng, B., Li, H., Lee, W.C.: Approaching the skyline in Z Order. In: Proceedings of VLDB 2007, pp. 279–290. VLDB Endowment (2007)
9. Lee, K.C.K., Lee, W.C., Zheng, B., Li, H., Tian, Y.: Z-SKY: an efficient skyline query processing framework based on Z-order. VLDB J. **19**(3), 333–362 (2009)
10. Liu, B., Chan, C.Y.: ZINC: efficient indexing for skyline computation. Proc. VLDB Endow. **4**(3), 197–207 (2010)
11. Mandl, S., Kozachuk, O., Endres, M., Kießling, W.: Preference analytics in EXA-Solution. In: Proceedings of BTW 2015 (2015)
12. Papadias, D., Tao, Y., Fu, G., Seeger, B.: An optimal and progressive algorithm for skyline queries. In: Proceedings of SIGMOD 2003, pp. 467–478. ACM (2003)
13. Papadias, D., Tao, Y., Fu, G., Seeger, B.: Progressive skyline computation in database systems. ACM TODS **30**(1), 41–82 (2005)
14. Selke, J., Balke, W.-T.: SkyMap: a trie-based index structure for high-performance skyline query processing. In: Hameurlain, A., Liddle, S.W., Schewe, K.-D., Zhou, X. (eds.) DEXA 2011. LNCS, vol. 6861, pp. 350–365. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-23091-2_30