Efficient Skyline Computation in High-Dimensionality Domains

Rui Liu
University of Tours, France
rui.liu@univ-tours.fr

Dominique Li
LIFAT Laboratory, University of Tours, France
dominique.li@univ-tours.fr

ABSTRACT

We present a dimension indexing based algorithm for skyline computation. We first show that the dominance tests required to determine a skyline tuple can be sufficiently bounded to a subset of the current skyline, and then propose the algorithm SDI, of which the time complexity is better than the best known algorithm in high-dimensionality domains with reasonably low cardinality. Our performance evaluation on synthetic and real datasets shows that SDI outperforms the state-of-the-art skyline algorithm in both low-dimensionality and high-dimensionality domains.

1 INTRODUCTION

The *skyline computation problem* aims at retrieving the complete set of dominating tuples from multidimensional data, with respect to a monotonic *preference order* on all dimensions. Over several past decades, many algorithms have been developed, which can be categorized into sorting based [1, 4, 6, 7, 14] and partitioning based [2–4, 8–13, 15]. Most the existing skyline algorithms have been designed for low-dimensionality domains because of the quadratic issue raised by the worst case time complexity.

In this paper, we present a dimension indexing based skyline algorithm SDI (Scalable Dimension Indexing) that is efficient in high-dimensionality domains as well as in low-dimensionality domains. We show that by indexing all dimensions, it is sufficient to test a tuple only with the existing skyline tuples on an arbitrary dimension instead of with the complete set of skyline tuples. We also show that any skyline tuple can be used as a *stop line* that traverses the indexed dimensions to stop the computation, which is much performant than the calculation of *stop point* mentioned in SaLSa [1]. Furthermore, SDI adopts the *weak incomparability checking* to take the incomparability between tuples into account, which is the most important feature of the state-of-the-art skyline algorithm BSkyTree [10]. Our analysis shows that the worst time complexity of SDI is better than the best known one [13] in high-dimensionality domains with reasonably low cardinality, and our performance evaluation shows that SDI outperforms BSkyTree on both low and high dimensional data, but less efficient than BSkyTree on medium dimensional data.

The rest of this paper is organized as follows. Section 2 presents the SDI algorithm with preliminary definitions. We show our theoretical analysis of the computational complexity of SDI in Section 3. Section 4 reports the performance evaluation of SDI on both synthetic and real datasets. We conclude in Section 5.

2 THE SDI APPROACH

Let t be a d-dimensional tuple, we denote $t[i]$ the *dimension value* of t on the dimension i, where $1 \le i \le d$. We define the *preference order*, denoted by $\prec$, as a total order that covers each dimension such that given two tuples t and u, $t[i]$ is *better than* $u[i]$ if $t[i] \prec u[i]$; $t[i]$ is *equal to* $u[i]$ if $t[i] = u[i]$; $u[i]$ is *not worse* than $t[i]$ if $(t[i] \prec u[i]) \vee (t[i] = u[i])$ holds, denoted by $t[i] \preceq u[i]$. We say that a tuple t *dominates* a tuple u, denoted by $t \prec u$, if and only if for each dimension i, we have $t[i] \preceq u[i]$, and for at least one dimension k we have $t[k] \prec u[k]$. We denote $t \nprec u$ that t does not dominate u, and $t \nsim u$ that t and u are *incomparable*, that is, $(t \nprec u) \wedge (u \nprec t)$. Considering a d-dimensional database $\mathcal{D}$ and a preference order $\prec$ on $\mathcal{D}$, a tuple $t \in \mathcal{D}$ is a *skyline tuple* if and only if $\nexists u \in \mathcal{D}$ such that $u \prec t$. The *skyline* of $\mathcal{D}$ is the complete set $\mathcal{S}$ of skyline tuples such that $\mathcal{S} = \{t \in \mathcal{D} \mid \nexists u \in \mathcal{D}, u \prec t\}$. We have $s \nsim t$ for any two skyline tuples s and t.

Given a database, the *dimension index*, denoted $\mathcal{I}$, is the set of d ordered lists on a preference order $\prec$, in which each list $I_i \in \mathcal{I}$ is a *dimensional subindex* that contains all dimension values sorted with respect to $\prec$.

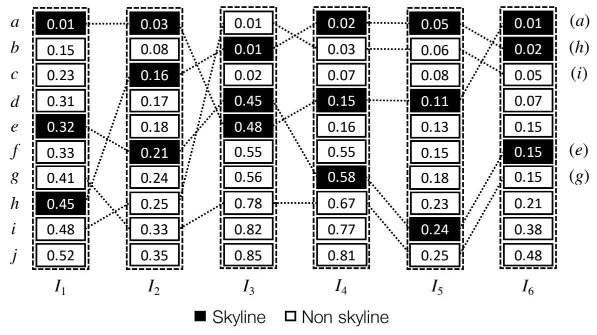


Figure 1: A dimension index example.

Let us consider the 5 tuples $\{a, e, g, h, i\}$ presented in Figure 1 that shows a dimension index example consisting of 10 tuples and 6 subindexes, where the dashed lines link the tuple among the subindexes. Obviously, a is a skyline tuple, which can be independently concluded from I_1, I_2, and I_6 because no tuple can dominate a; if we regard only I_4, h is immediately a skyline tuple and in order to determine whether i, the second tuple in this subindex, is a skyline tuple, it is enough to compare i with h because any tuple x located after the position of i in I_4 cannot dominate i since we have $i[4] \prec x[4]$ in I_4. Nevertheless, if we focus on I_3, we see that $h \prec i$ and $i \prec h$ must be first tested in order to decide whether h or/and i shall be skyline tuple(s) since $h[3] = i[3]$ (indeed we have $h \prec i$ and $i \nprec h$). That is also the case in I_6, where 3 tuples contain the same dimension value 0.15, so all these 3 tuples must be first locally compared in order to filter the potential skyline tuples (in our example, e). We call such tuples as h in I_3 and e in I_6 the *local skyline tuple*. It is easy to see that any tuple is a local skyline tuple on a given dimension if there are no identical dimension values.

Theorem 1. *Let $\mathcal{I}$ be the dimension index of a database $\mathcal{D}$ and $I_i \in \mathcal{I}$ be an arbitrary subindex of the dimension i. Then, a local*

© 2020 Copyright held by the owner/author(s). Published in Proceedings of the 22nd International Conference on Extending Database Technology (EDBT), March 30-April 2, 2020, ISBN 978-3-89318-083-7 on OpenProceedings.org.
Distribution of this paper is permitted under the terms of the Creative Commons license CC-by-nc-nd 4.0.

skyline tuple t is a skyline tuple if and only if (1) there is no tuple u such that $u[i] \prec t[i]$, or (2) for any skyline tuple s such that $s[i] \prec t[i]$, we have $s \nprec t$.

Proof. Let t be a local skyline tuple. If t is the first tuple in I_i, then t is a skyline tuple because no tuple is better than t. Otherwise, let s be a skyline tuple such that $s[i] \prec t[i]$, then if $s \prec t$, t cannot be a skyline tuple since s dominates t; if $s \nprec t$, then there exists at least on dimension $k \neq i$ such that $t[k] \prec s[k]$ so no tuple dominated by s dominates t. Hence, for each skyline tuple s such that $s[i] \prec t[i]$, if we have $s \nprec t$, then no tuple in the database dominates t, thus, t is a skyline tuple. If $t[i] \prec s[i]$, then $s \nprec t$ so it is meaningless to compare t with s. □

Theorem 1 allows to determine whether a tuple is a skyline tuple only with a subset of the existing skyline. Furthermore, once the dimension index has been constructed, Theorem 1 allows to switch among subindexes so that the best one containing the least of known skyline tuples can always be selected. Indeed, let p be a skyline tuple, then any tuple t such that $p \prec t$ can be pruned from the database in order to reduce the computation time. In this paper, we propose the notion of stop line based on the dimension index of which the effectiveness can be guaranteed.

Theorem 2. *Let $\mathcal{I}$ be a dimension index of a d-dimensional database $\mathcal{D}$ and $p \in \mathcal{D}$ be an arbitrary skyline tuple. Let $o_i(p)$ denote the largest offset of any tuple x such that $x[i] = p[i]$ in the dimensional subindex $I_i \in \mathcal{I}$, if all offsets $o_i(p)$, $1 \leq i \leq d$, have been reached by following a top-down traversal on all dimensions, then the complete set of skyline tuples has been identified and the computation can be terminated.*

Proof. Let p and $t \in \mathcal{D} \backslash p$ be two skyline tuples in $\mathcal{D}$, we have: (1) $t \nsim p$; or (2) t and p have identical values on all dimensions. We denote $L_p = \bigcup_{1 \leq i \leq d} o_i(p)$ the set of all offsets $o_i(p)$. In the first case, $t \nsim p \Rightarrow \exists k$ such that $p[k] \prec t[k]$, i.e. $o_k(p) < o_k(t)$, hence, if the index traversal reaches all offsets in L_p, t must have been identified at least in the dimension k. In the second case, we have $p[i] = t[i]$ on any dimension i. In both cases, if all offsets in L_p have been reached, all skyline tuples have been identified. □

We call the set $L_p = \bigcup_{1 \leq i \leq d} o_i(p)$ a *stop line* that can safely terminate the skyline computation of SDI. In theoretical, any skyline tuple can be selected as a stop line, however, different stop lines behave differently in pruning irrelevant tuples. We propose therefore a function min_{stop} to find the best stop line L_p^*, defined as:

$$L_p^* = min_{stop}(p) = \arg\min_p (max\{o_i(p)\}, \sum_{i=1}^{d} o_i(p)),$$

where d is the dimensionality of the data. The function min_{stop} sorts first by the maximum offset, then by the sum of offsets in all dimensions, so the skyline tuple having the minimized value is the best stop line. The best stop line L_p can be dynamically maintained by keeping $min_{stop}(p) < min_{stop}(t)$ for any two skyline tuples p and t.

The *incomparability checking* is taken into account while a dominance test is proceeding. In our approach, we consider that a d-dimensional dominance test runs in $O(d)$ time, so any tuple comparison better than $O(d)$ time shall improve the efficiency of SDI. Indeed, to efficiently determine $s \nsim t$ in stead of testing $s \prec t$ in the case of $s \nprec t$ is an essential time-costly task while comparing t with all existing skyline tuples in a dimensional subindex. In this paper, we propose a *weak* checking mechanism of the incomparability between a skyline tuple s and a testing tuple t in a dimensional subindex I_i as following.

Theorem 3. *Let $\mathcal{I}$ be a dimension index and s be a skyline tuple present in a dimensional subindex $I_i \in \mathcal{I}$. Given a tuple t such that $s[i] \prec t[i]$, we sufficiently have $s \nsim t$ if $max(s) > max(t)$, or $min(s) > min(t)$, or $sum(s) > sum(t)$.*

Proof. The sets L_s and L_p (see the proof of Theorem 2 for the definition) can be considered as the coordinates of two curves in a two-dimensional space. In Euclidean geometry, $max(s) > max(t)$, or $min(s) > min(t)$, or $sum(s) > sum(t)$ are sufficient conditions for the existence of at least one intersection of the curves formed by s and t since we have $s[i] \prec t[i]$, i.e. $o_i(s) < o_i(t)$, that is, according to the definition of dominance, $s \nsim t$. □

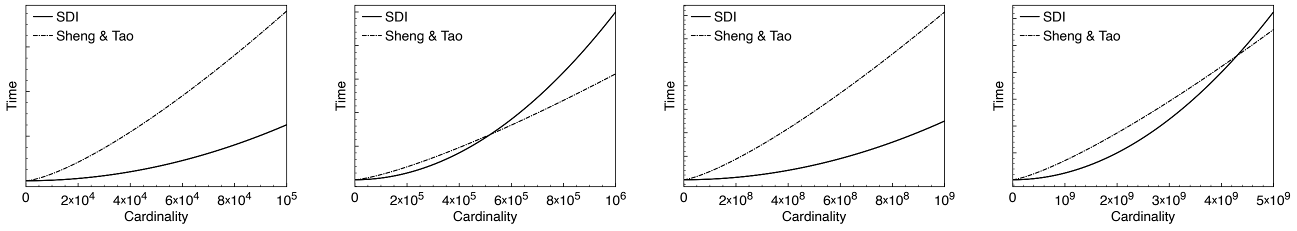
Note that the maximal value, the minimal value, and the sum of a tuple can be pre-calculated while constructing the dimension index, so Theorem 3 can efficiently determine $s \nsim t$. However, Theorem 3 shows in fact 3 sufficient conditions for $s \nsim t$, hence a dominance test is necessary to determine $s \nsim t$ in the cases that are not covered by Theorem 3, for which we call Theorem 3 a weak incomparability checking. The sketch of SDI is listed in Algorithm 1.

Algorithm 1: SDI
```
   Input: Dimension index $\mathcal{I}$
   Output: Skyline $\mathcal{S}$
1  while true do
2      $I_{best} \leftarrow BestSubindex(\mathcal{I})$
3      while $T \leftarrow NextLocalSkyline(I_{best})$ do
4          if $T = null$ then
5              return $\mathcal{S}$
6          foreach $t \in T$ and $t \notin \mathcal{S}$ do
7              if $\mathcal{S}_{best} \nprec t$ then
8                  $\mathcal{S}_{best} \leftarrow \mathcal{S}_{best} \cup t$
9                  $\mathcal{S} \leftarrow \mathcal{S} \cup t$
10         if Found new skyline tuples then
11             Update StopLine
12             break
13     if StopLine is reached then
14         return $\mathcal{S}$
```

Extensions. (1) SDI computes the skyline in the categorical domains as long as the preference order $\prec$ can be defined; (2) SDI can be immediately adapted to subspace skyline computation by skipping unrelated dimensions; (3) SDI can be extended to the skyline maintenance by dynamically constructing the dimension index; (4) SDI can handle the top-k skyline query by finding the skyline tuples having the best positions in the dimension index.

3 THEORETICAL ANALYSIS

We denote d the dimensionality and N the cardinality of the data, and M the size of the skyline. We discuss without duplicate values on any dimension, but if K duplicate values are present in a dimensional subindex, $O(dK^2)$ shall be considered in assuming that BNL is applied to compute local skylines. Note that we consider $O(d)$ time for a d-dimensional dominance test, which implies the tests of $s \prec t$ and $t \prec s$, hence, the dominance test



(a) $d = 6$, $N = 1 \times 10^5$. (b) $d = 6$, $N = 1 \times 10^6$. (c) $d = 8$, $N = 1 \times 10^9$. (d) $d = 8$, $N = 5 \times 10^9$.

Figure 2: Numerical simulation for complexity study in the worst case.

within SDI is considered in $O(d/2)$ time since $s < t$ is sufficient. We also note that the construction of dimension index requires $O(dN \log N)$ time with respect to general $O(N \log N)$ sorting algorithms on d dimensions, and we do not consider the two heuristics of stop line and incomparability checking.

The average time complexity of SDI is measured on M skyline tuples uniformly distributed in d dimensional subindexes.

Theorem 4. *Considering M skyline tuples, SDI computes the skyline of a d-dimensional database with the cardinality N in*

$$O(dN \log N + \frac{M(2N - M - d)}{4}).$$

Proof. With M skyline tuples uniformly distributed on each dimension, $(N - M)/d$ non skyline tuples must be compared with M/d skyline tuples. For each dimension, in the worst case, $(M/d)(M/d - 1)/2$ dominance tests are required by skyline tuples, $((N - M)/d)(M/d)$ dominance tests are required by non skyline tuples, and each dominance test cost $O(d/2)$ time. Therefore, the average time complexity of SDI is $O(((M/d)(M/d - 1)/2 + ((N - M)/d)(M/d))(d/2)d)$, that is the result shown in Theorem 4. □

Theorem 5. *In the worst case, all N tuples in a d-dimensional database are skyline tuples. SDI computes the skyline in*

$$O(dN \log N + \frac{N^2 - dN}{4}).$$

Proof. The proof is immediate if we replace M in Theorem 4 by N. □

Comparing with the best known worst-case time complexity $O(N \log^{max(1,d-2)} N)$ proposed by Sheng and Tao [13], given $d > 2$, the following equation must be resolved:

$$N \log^{d-2} N > dN \log N + \frac{N^2 - dN}{4}.$$

The above equation belongs to transcendental equations that have no closed-form solutions. Our numerical simulation results presented in Figure 2 shows that while $d = 6$, SDI is better than the approach of Sheng and Tao for $N < 5 \times 10^5$; and while $d = 8$, the compared approach beats SDI only if $N > 4 \times 10^9$. SDI performs better in high-dimensionality domains with respect to a reasonable data cardinality.

4 PERFORMANCE EVALUATION

We evaluate the performance of SDI in comparison with BSkyTree implemented in SkyBench[1] [5] on both synthetic datasets and real world datasets. The synthetic datasets are generated by Skyline Benchmark Data Generator[2], including uniform independent (UI), correlated (CO), and anti-correlate (AC) data; the real world datasets include NBA, HOUSE, and WEATHER [5]. We implemented SDI[3] in C++ standard and all executables are compiled by LLVM Clang with -O3 option. All experiments have been conducted on an Intel Core i5 2.8 GHz processor with 16GB 1600 MHz DDR3 RAM, running macOS 10.15.1 operating system. We note that all results reported are the average performance over 5 iterations.

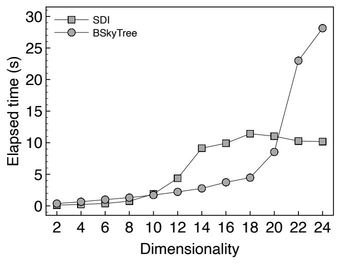
First, the effects of **(1) dimensionality** and **(2) cardinality** of data have been evaluated. For (1), the cardinality is fixed to 100K and for (2), the dimensionality of data is fixed to 24. Due to the space limitation, only overall elapsed time, that is, the sum of data loading time, data structure construction time, and query time, has been reported in this paper. Indeed, we consider that for single-round skyline queries, to focus on total processing time is much important than to focus only on the query time without looking at data structure building time.

Dataset	$d = 2$	$d = 4$	$d = 6$	$d = 8$	$d = 10$	$d = 12$
UI	7	259	2,597	9,960	25,737	46,301
CO	1	5	31	120	449	790
AC	50	4,100	26,713	56,118	75,668	87,857
Dataset	$d = 14$	$d = 16$	$d = 18$	$d = 20$	$d = 22$	$d = 24$
UI	67,676	82,286	92,011	96,832	99,059	99,662
CO	1,439	2,941	5,471	8,936	14,498	16,948
AC	94,053	96,956	98,413	99,205	99,570	99,760

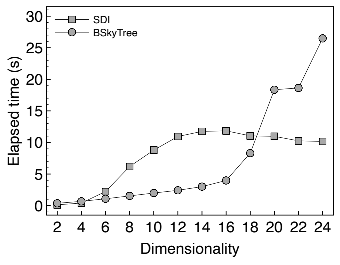
Table 1: Skyline size of synthetic datasets (N = 100K).

Table 1 lists the skyline size of synthetic datasets with the cardinality of 100K. We see that the higher the dimensionality of data, the closer to the worst case is likely to be. For instance, the skyline consists of 92% of tuples in UI data while $d = 18$, however while $d = 14$ of AC data, the skyline rate reaches already 94%. Figure 3 shows the effect of dimensionality on SDI where the cardinality N = 100K is reasonable in the most of use cases. SDI outperforms BSkyTree in both low-dimensionality and high-dimensionality domains on UI ($d \leq 10$ or $d > 20$) and AC data ($d \leq 4$ or $d > 20$), but is less efficient than BSkyTree in other dimensionalities. In fact, we note that the stop line takes no advantage in AC data because of the *anti-correlated* characteristics of data, however SDI systematically outperforms BSkyTree on the CO data because the stop line can efficiently determined with respect to the strong *correlation* in data. Figure 4 shows the effect of cardinality on SDI with the highest dimensionality in our experiments. In high-dimensionality domains, SDI outperforms BSkyTree in most cases except in AC data. Again, we confirm that the stop line does not show any advantage in high-dimensionality

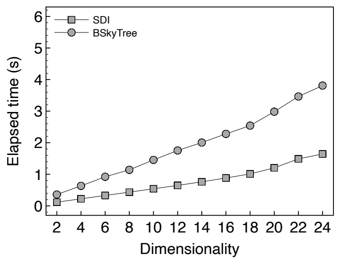
[1]https://github.com/sean-chester/SkyBench
[2]http://pgfoundry.org/projects/randdataset
[3]https://github.com/skyline-sdi/sdi-bench



(a) The UI dataset.

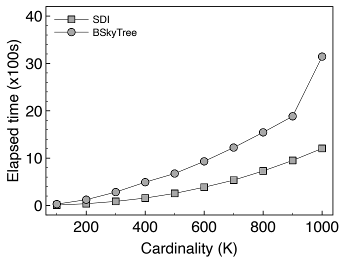


(b) The AC dataset.

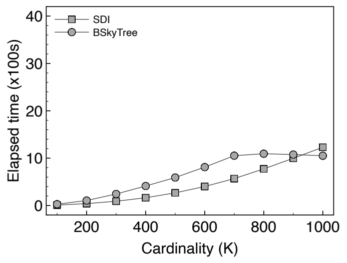


(c) The CO dataset.

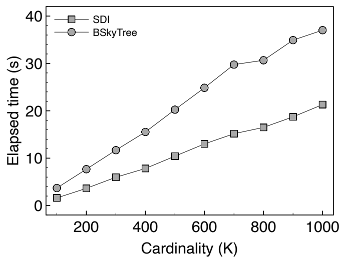
Figure 3: Performance evaluation on the effect of dimensionality (N = 100K).



(a) The UI dataset.



(b) The AC dataset.



(c) The CO dataset.

Figure 4: Performance evaluation on the effect of cardinality (d = 24).

and high-cardinality AC data. Table 2 lists the pruned tuples by the stop line in different synthetic datasets while $N = 100$K, that are relevant to our experimental results.

Dataset	$d = 2$	$d = 4$	$d = 6$	$d = 8$	$d = 10$	$d = 12$
UI	99,849	88,011	63,260	30,576	12,381	8,110
CO	99,997	97,773	98,832	97,656	96,526	92,066
AC	36,731	5,022	2,227	240	30	27
Dataset	$d = 14$	$d = 16$	$d = 18$	$d = 20$	$d = 22$	$d = 24$
UI	3,794	297	933	35	3	8
CO	87,901	80,587	73,837	51,347	39,742	48,239
AC	13	25	0	10	0	0

Table 2: Pruned tuples by the stop line (N = 100K).

Table 3 shows the performance comparaison between SDI and BSkyTree on real datasets. SDI outperforms BSkyTree on NBA and HOUSE datasets but is much slower than BSkyTree on the WEATHER dataset because the huge number of duplicate dimension values in WEATHER makes $O(dK^2)$ (discussed in Section 3) an important factor.

Dataset	d	N	$\lvert\mathcal{S}\rvert$	SDI	BSkyTree
HOUSE	6	127,931	5,774	306 ms	839 ms
NBA	8	17,264	1,796	45 ms	155 ms
WEATHER	15	566,268	26,713	18,680 ms	11,641 ms

Table 3: Performance evaluation on real datasets.

5 CONCLUSION

In this paper, we presented an efficient Skyline computation algorithm. We proved that in multidimensional databases, skyline computation can be conducted on an arbitrary dimensional index which is constructed with respect to a predefined total order that determines the skyline. We further showed that any skyline tuple can be used to stop the computation process by outputting the complete skyline. Our experimental evaluation shows that SDI outperforms the state-of-the-art skyline algorithm in both low-dimensionality and high-dimensionality domains.

REFERENCES

[1] Ilaria Bartolini, Paolo Ciaccia, and Marco Patella. 2008. Efficient sort-based skyline evaluation. *ACM Transactions on Database Systems* 33, 4 (2008), 31.
[2] Jon L. Bentley, Kenneth L. Clarkson, and David B. Levine. 1993. Fast linear expected-time algorithms for computing maxima and convex hulls. *Algorithmica* 9, 2 (1993), 168–183.
[3] Jon L. Bentley, Hsiang-Tsung Kung, Mario Schkolnick, and Clark D. Thompson. 1977. On the average number of maxima in a set of vectors and applications. *J. ACM* (1977).
[4] S. Borzsony, D. Kossmann, and K. Stocker. 2001. The Skyline operator. In *ICDE*. 421–430.
[5] Sean Chester, Darius Šidlauskas, Ira Assent, and Kenneth S Bøgh. 2015. Scalable parallelization of skyline computation for multi-core processors. In *ICDE*. 1083–1094.
[6] Jan Chomicki, Parke Godfrey, Jarek Gryz, and Dongming Liang. 2003. Skyline with presorting. In *ICDE*, Vol. 3. 717–719.
[7] Parke Godfrey, Ryan Shipley, and Jarek Gryz. 2005. Maximal Vector Computation in Large Data Sets. In *VLDB*. 229–240.
[8] Donald Kossmann, Frank Ramsak, and Steffen Rost. 2002. Shooting Stars in the Sky: An Online Algorithm for Skyline Queries. In *VLDB*. 275–286.
[9] Hsiang-Tsung Kung, Fabrizio Luccio, and Franco P. Preparata. 1975. On finding the maxima of a set of vectors. *J. ACM* 22, 4 (1975), 469–476.
[10] Jongwuk Lee and Seung-Won Hwang. 2014. Scalable skyline computation using a balanced pivot selection technique. *Information Systems* 39 (2014), 1–21.
[11] Ken CK Lee, Wang-Chien Lee, Baihua Zheng, Huajing Li, and Yuan Tian. 2010. Z-SKY: an efficient skyline query processing framework based on Z-order. *The VLDB Journal* 19, 3 (2010), 333–362.
[12] Dimitris Papadias, Yufei Tao, Greg Fu, and Bernhard Seeger. 2005. Progressive skyline computation in database systems. *ACM Transactions on Database Systems* 30, 1 (2005), 41–82.
[13] Cheng Sheng and Yufei Tao. 2012. Worst-case I/O-efficient skyline algorithms. *ACM Transactions on Database Systems* 37, 4 (2012), 26.
[14] Kian-Lee Tan, Pin-Kwang Eng, and Beng Chin Ooi. 2001. Efficient Progressive Skyline Computation. In *VLDB*. 301–310.
[15] Shiming Zhang, Nikos Mamoulis, and David W Cheung. 2009. Scalable skyline computation using object-based space partitioning. In *SIGMOD*. 483–494.