Parallel Skyline Queries on Multi-Core Systems

Meng-Zong Liou, Yi-Teng Shu, and Wei-Mei Chen

Department of Electronic and Computer Engineering
National Taiwan University of Science and Technology
Taipei 106, Taiwan
{M10002141, M9902109, wmchen}@mail.ntust.edu.tw

***Abstract*—The skyline query is an efficient data analysis tool for multi-criteria decision making that has received significant attention in the database community. As multi-core architectures have gone mainstream, we present a new parallel skyline query algorithm that can be applied to multi-core and multiprocessor systems, to progressively return skyline points as they are identified efficiently. In this paper, we proposed a parallel skyline algorithm which can eliminate redundant computations and improve parallelism of the skyline query. Experimental results show that our algorithm successfully exploits the features of multiple cores to improve the performance of skyline computation for large high-dimensional datasets.**

***Keywords*-Skyline; Dominance; Parallel programming**

I. Introduction

Multi-core processors are going mainstream. Manufacturers are building processor with multiple energy efficient cores instead of one powerful core. The multi-core processor does not necessarily run as fast as the highest performing single-core models, but they improve overall performance by handling more work in parallel. To take advantage of multi-core processors, programmers must redesign applications so that the processor can run them as multiple threads. Programmers find good patterns to break up the applications, divide the work into roughly equal pieces that can run at the same time, and determine the best times for the threads to communicate with one another.

Given a dataset, a skyline query returns skyline points that are not dominated by any other points. The problem of skyline query came from some interesting research topics like contour problem [11], maximum vectors [9], and convex hull [12]. Up to now, several algorithms for sequential skyline query have been developed, such as block-nested-loops (BNL) [1], sort-filter-skyline (SFS) [3], nearest neighbor (NN) [8], and branch and bound skyline (BBS) [13]. BNL compares each point $\mathbf{p}$ against every other point in the given dataset and reports $\mathbf{p}$ as a skyline point if it is never dominated. SFS is based on the same principle as BNL, but improves performance by first sorting the dataset according to a monotone function. Both NN and BBS utilize an R-tree structure [5] to extract skyline points and return progressively. The number of skyline points rapidly increases as the dimensionality of the dataset grows, which is called the curse of the dimensionality problem.

Parallel computation is a critical issue of skyline query [6], [14]. The intensively computational nature of skyline query leads that it has to be parallelized to reduce run time on multi-core architectures. The skyline query is suitable for multi-core architectures since a large number of comparisons between data points can be performed independently. Several parallel algorithms for skyline query are proposed, such as improved distributed skyline algorithm (IDS) [2] and PSkyline algorithm [7]. The IDS algorithm partitions d-dimensional dataset into d lists, where each list contains values of dimension in ascending order, then performs sorted accesses on the d lists in a round-robin manner to find out a point, called an anchor. Finally, utilize anchors to eliminate redundant comparisons between data point. On the other hand, PSkyline algorithm divides the given dataset into several blocks with roughly equal size, and computes local skyline points of each block by using a sequential skyline query algorithm, then merges local skyline points sequentially. PSkyline achieves a speedup approximately proportional to the number of cores.

For multi-core architectures, we develop a new parallel skyline query algorithm, called PSQ, that can be applied to shared memory architectures, to progressively return skyline points as they are identified. The PSQ algorithm employs the feature of a terminating point [2], [10] that efficiently eliminates redundant computations of skyline query. Besides, we introduce a new property that can be applied to prune data effectively. To let the core processor get the information of operations from local memory space to reduce the opportunity to read the data from the shared memory, we try to arrange the points to be processed in the same array. We compare the performance of the PSQ algorithm and the PSkyline algorithm on an eight-core computer, included run time and speedup. Experimental results show that both algorithms make use of multiple cores to improve parallel processing and the PSQ algorithm runs faster than the PSkyline algorithm when the dimensionality of the dataset is high or the number of data points is large.

The remainder of this paper is organized as follows. Section 2 discusses related work. Section 3 presents the preliminaries of this paper. Section 4 introduces the PSQ

978-1-4799-2419-6/13 $31.00 © 2013 IEEE
DOI 10.1109/PDCAT.2013.51

algorithm. Section 5 shows experimental results and Section 6 gives conclusions.

II. Problem description

Given a d-dimensional dataset D, a point $\mathbf{p} \in D$ is said to *dominate* another point $\mathbf{q} \in D$ if $p_j \geq q_j$ for $1 \leq j \leq d$, where $\mathbf{p} = (p_1, \ldots, p_d)$ and $\mathbf{q} = (q_1, \ldots, q_d)$, and is greater than in at least one dimension. For convenience, we write $\mathbf{p} \succ \mathbf{q}$. A point p is a skyline point of the dataset D if there is no point in D that can dominate $\mathbf{p}$. A point is called a non-skyline point if it is not a skyline point. The skyline query is to find out all skyline points from a given dataset. Note the dominance relation preserves the transitivity property; that is, if $\mathbf{p} \succ \mathbf{q}$ and $\mathbf{q} \succ \mathbf{r}$, then $\mathbf{p} \succ \mathbf{r}$.

The definition of the dominance relation brings that the number of skyline points increases rapidly when the number of dimensions is large. To improve the performance, skyline query algorithms usually prune unnecessary comparisons between data points by using specific properties of skyline points. For example, by transitivity property of dominance, skyline query does not check the points dominated by some points.

Consider an example of baseball players in Table I. There are ten baseball players named from A to J. Each player's performance is recorded by three attributes, included batting average (AVG), on base percentage (OBP), and slugging percentage (SLG). These attributes can be used to evaluate the performance of a baseball player. Thus the performance of a baseball player can be interpreted as a 3-dimensional point. By computation, the players A, B, E, H, and I are skyline points.

The main operation for finding skyline points is comparison. By previous studies [2], [10] and our observation, there are three heuristics that can help to reduce the cost of comparisons after sorting the dataset according to each attribute (or component) respectively. As shown the right part in Table I, players A to J are sorted by the AVG attribute, the OBP attribute, and the SLG attribute separately, then the ranks of each player for each attribute are known.

Heuristic 1: Determine the terminating point

The terminating point is the first point with all its attribute values scanned in row-major order. All points not yet encountered cannot be in the skyline [10], which can efficiently eliminate redundant computations of skyline query. Our proposed algorithm suggests to determine the terminating point by identifying the point with the minimum sum of ranks.

For the example in Table I, the sum of all attribute rank values of the player A is $3 + 2 + 1 = 6$, and the value of other players are $0 + 4 + 3 = 7$ (for the player B), $1 + 6 + 5 = 12$ (for the player C), $7 + 3 + 6 = 16$ (for player D), $2 + 1 + 4 = 7$ (for the player E), $8 + 5 + 2 = 15$ (for the player F), $9 + 9 + 7 = 25$ (for the player G), $6+0+8 = 14$ (for player H), $4+7+0 = 11$ (for the player I), $5 + 8 + 9 = 22$ (for the player J), as shown in Table II. The terminating point is the player A in the example and the player A must be a skyline point since it not be dominated by any point in the dataset. Points, such as the players D, F, G and J, whose components are all less than those of the terminating point can be pruned, as they must be dominated by the terminating point. On the other hand, we only have to check points (the players B, C, E, H and I in Table I) that have at least one component is greater than the corresponding component of the terminating point.

Table I
An example of baseball players

player	AVG	OBP	SLG	rank	AVG	OBP	SLG
A	0.295	0.469	0.612	0	B	H	I
B	0.312	0.451	0.576	1	C	E	A
C	0.307	0.445	0.554	2	E	A	F
D	0.254	0.458	0.531	3	A	D	B
E	0.301	0.485	0.564	4	I	B	E
F	0.243	0.451	0.587	5	J	F	C
G	0.223	0.415	0.521	6	H	C	D
H	0.265	0.485	0.487	7	D	I	G
I	0.285	0.432	0.654	8	F	J	H
J	0.276	0.428	0.432	9	G	G	J

Table II
Sum of ranks for the example in Table I

Player	A	B	C	D	E	F	G	H	I	J
Rank sum	6	7	12	16	7	15	25	14	11	22

Heuristic 2: Each point does not need to be compared with non-skyline points.

Using the transitivity property, skyline query algorithms can be more efficient by data pruning. Consider three points $\mathbf{p}$ and $\mathbf{q}$ with the relation $\mathbf{p} \succ \mathbf{q}$. For a point $\mathbf{r}$, if $\mathbf{q} \succ \mathbf{r}$, then we have $\mathbf{p} \succ \mathbf{r}$. Then the comparison of $\mathbf{p}$ and $\mathbf{r}$ can be skipped. Note that if a point is marked as a non-skyline point, it can be removed from the data for further computation. It reveals that each point only be deleted once in the entire process.

Heuristic 3: Organize data properly.

After pruning the data set by Heuristics 1 and 2, we try to organize the remaining data for parallel processing. To let the core processor get the information of operations from local memory space to reduce the opportunity to read the data from the shared memory, we arrange the points to be processed such that the core processors retrieve them all in the same array.

In the example in Table I, the players B, C, E, H and I need to be considered further. For the player E, we have to compare his record with those of the players B, C, and H, respectively, since the players B and C are better in the AVG attribute and the player H is better in the OBP attribute. Let us consider the OBP attribute, there are only a player H is better than E, so the players B and C are worse than the player E in this filed. This means that E can not be marked as a non-skyline point by comparing with the player B and C. These two comparisons can be omitted and thereby we need only to compare the records of E and H. Thus, for a possible skyline point, we suggest to select the dimension with the minimum number of records to be further checked.

Based on the three heuristics, we propose a new parallel skyline query algorithm that can be applied to multi-core architectures, to progressively return skyline points as they are identified.

III. The Proposed algorthm

The proposed parallel skyline query algorithm is divided into four steps, as demonstrated in Fig. 1. The first step sorts the dataset in non-ascending order according to each attribute separately (Fig. 1, Lines 1 to 3).

The second step finds the terminating point (Fig. 1, Lines 4 and 5) which can efficiently eliminate redundant computations of skyline query. According to Heuristic 1, points whose each dimension value is smaller than the corresponding dimension value of the terminating point can be discarded later.

The third step collects points that might be skyline points (Fig. 1, Lines 6 to 13). The PSQ algorithm checks points that have at least one component is greater than the corresponding component of the terminating point. We use $\text{rank}_i(\mathbf{p})$ to denote the rank of point $\mathbf{p}$ for the ith component. Then the PSQ algorithm finds r such that $\text{rank}_r(\mathbf{p}) = \min\{\text{rank}_j(\mathbf{p}) | 1 \leq j \leq d\}$ and puts $\mathbf{p}$ into array B_r.

Finally, the fourth step checks these possible skyline points, and then outputs points progressively if they are skyline points (Fig. 1, Lines 14 to 18). Based on Heuristic 3, we can arrange subarray B_r in local memory for moderate datasets. Once the CheckSkyline procedure reports that a point $\mathbf{p}$ is a skyline point, the answer can be output immediately though there still are some points to be checked. Note that CheckSkyline does not check with the points marked as non-skyline points (as described in Heuristic 2).

```
**Algorithm** PSQ
//**Input**: A list of points $D$
//**Output**: The skyline points of $D$
1:  parallel for $i := 1$ to $d$ do
2:      $S_i \leftarrow$ Sort $D$ in non-ascending order according to
            the $i$-th dimension
3:  end parallel
4:  $\mathbf{t} \leftarrow$ Determine a terminating point
5:  Mark $\mathbf{t}$ as a skyline point and output $t$
6:  parallel for each $\mathbf{p} \in D$ do
7:      for $i := 1$ to $d$ do
8:          if ($\text{rank}_i(\mathbf{p}) < \text{rank}_i(\mathbf{t})$)
9:              Find $r$ with $\text{rank}_r(\mathbf{p}) = \min\{\text{rank}_j(\mathbf{p}) | 1 \leq j \leq d\}$
10:             Insert $\mathbf{p}$ into $B_r$
11:             break
12:     end for
13: end parallel
14: for $i := 1$ to $d$ do
15:     parallel for each $\mathbf{p} \in B_i$ do
16:         CheckSkyline$(i, \mathbf{p})$
17:     end parallel
18: end for

CheckSkyline$(i, \mathbf{p})$
1:  for each $\mathbf{q} \in B_i$ with $\text{rank}_i(\mathbf{q}) < \text{rank}_i(\mathbf{p})$ do
3:      if ($\mathbf{q}$ is not marked as a non-skyline point)
4:          if ($\mathbf{q}$ dominates $\mathbf{p}$)
5:              $\mathbf{p}$ is marked as non-skyline point
6:              break
7:  end for
8:  if ($\mathbf{p}$ is not marked as a non-skyline point)
9:      $\mathbf{p}$ is marked as a skyline point
10:     Output $\mathbf{p}$
```

Figure 1. The PSQ algorithm.

IV. Experimental results

This section presents experimental results of the PSQ algorithm and the PSkyline algorithm on an eight-core machine. For a parallel algorithm on multicore systems, the speedup is defined by the ratio of run time used by one core to that used by multi-cores on the same multi-core architecture. We compare the performances of both algorithms for speedup and run time.

A. Experiment setup

Given a dataset D, let d and n be the dimensionality of the dataset and the number of points in the dataset, respectively. The variable c represents the number of cores used in skyline computation. We use K for 1000 (for example 100K $= 100,000$).

In this study, we generate synthetic datasets according to two major representative random models are hypercubes and simplices [4]. For the hypercube model, samples are generated by $[0,1]^d$, independently and uniformly. For the simplex model, samples are generated from the d-dimensional simplex, $\mathcal{D} = \{\mathbf{x} : x_i \geq 0, \sum_{1 \leq i \leq d} x_i \leq 1\}$. Fig. 3 shows

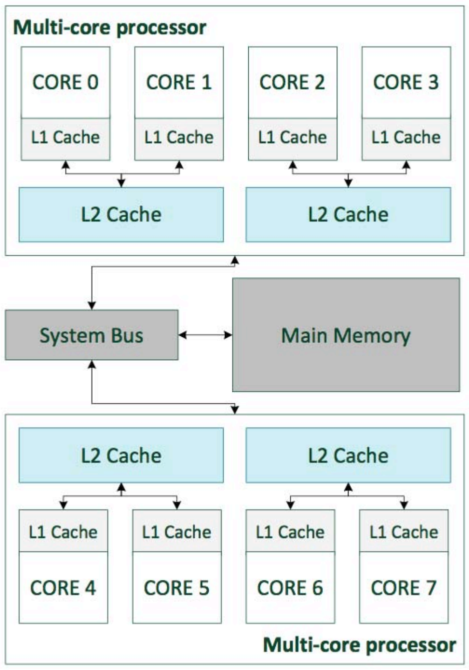


Figure 2. An example of a multi-core system based on Intel Xeon processor X5482.

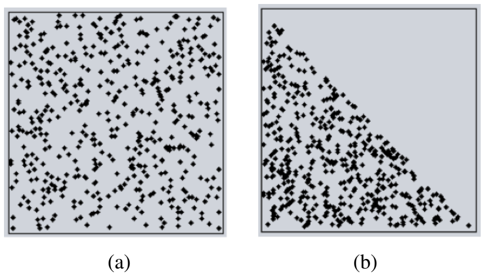


Figure 3. Test datasets for $d = 2$: (a) hypercube model and (b) simplex model.

the distribution of two test samples when $d = 2$. Table III illustrates the percentage of skyline points in different datasets.

All experiments in this study are constructed on a Mac server with Mac OS X Lion 10.7.3, two quad-core Intel Xeon X5482 3.2GHz CPUs and 16G of main memory, whose architecture is shown in Fig. 2. We implement PSQ and PSkyline in C language on the OpenMp programming environment. The compiler used in all the algorithms is LLVM GCC 4.2.

B. *Effect of dimensionality and dataset size*

In this section, we test the effect of PSQ on varying dimensionality and dataset size with two groups of experiment setup. We set $c = 1, 2, 4,$ and 8 in each dataset.

Total running time of a parallel program is composed of actual execution time of a program and elapsed time

Table III
THE PERCENTAGE OF SKYLINE POINTS.

$d = 6$

n	10K	50K	100K	500K	1000K
hypercube	9.08	3.66	2.41	0.89	0.57
simplex	72.77	62.35	57.1	47.88	43.9

$n = 100K$

d	4	6	8	10	12	14	16	18	20
hypercube	0.32	2.41	9.75	25.1	45.8	67.2	82.6	92.2	96.8
simplex	0.56	18.8	57.9	86.1	96.8	99.4	99.9	99.9	100

of overhead such as communication between threads and thread management. For the hypercube model, the run time of skyline query increases when dimensionality of dataset increases and the speedup of PSQ increases when run time of skyline query increases (Fig. 4(a) and (b)). On the other hand, the run time and speedup decrease for the simplex model (Fig. 4(c) and (d)). For large high-dimensional datasets, almost all points are skyline points. In these cases, the capacity of cache memory can not contains all points that computed on skyline query. Thus, cache misses incur memory access latencies in retrieving the corresponding data from the main memory. During this penalty period, processors must stall until the data arrive and the speedup is reduced.

C. *Comparison of run time*

In this section, we compare run time of skyline query by PSkyline and PSQ with two groups of experiment datasets. We test $c = 8$ in each dataset.

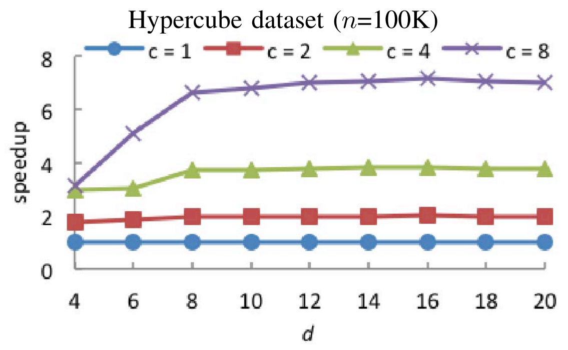
Fig. 5 illustrates that PSQ runs faster than PSkyline does except for low-dimensional datasets and small datasets. PSQ still performs better speedups for small low-dimensional datasets.

V. Conclusion

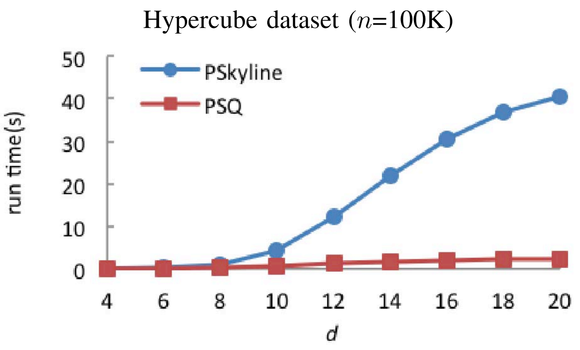
As multi-core architectures have gone mainstream, we present a new parallel skyline query algorithm that can be applied to multi-core and multiprocessor systems, to progressively return skyline points as they are identified efficiently. To reduce the cost of computation, the proposed algorithm sorts the given dataset in descending order for each coordinate component separately, and finds out a terminating point for eliminating redundant computations of skyline query. Experimental results show that our algorithm successfully exploits the features of multiple cores to improve the performance of skyline computation for large high-dimensional datasets.

References

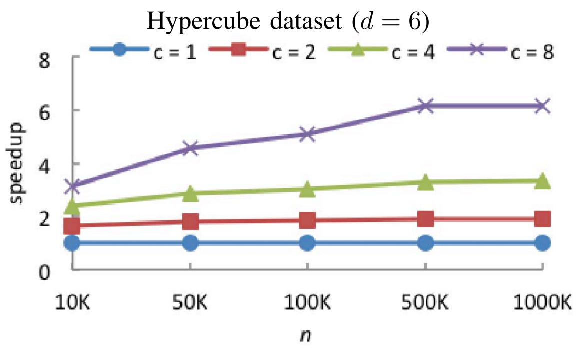
[1] S. Borzsonyi, D. Kossmann and K. Stocker, The skyline operator, *Proceedings of the 17th International Conference on Data Engineering*, 2001, pp 421–430.



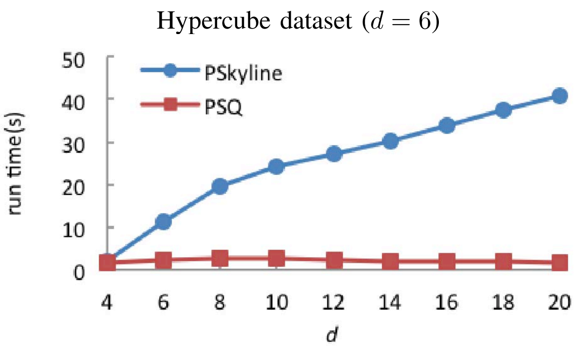
(a)



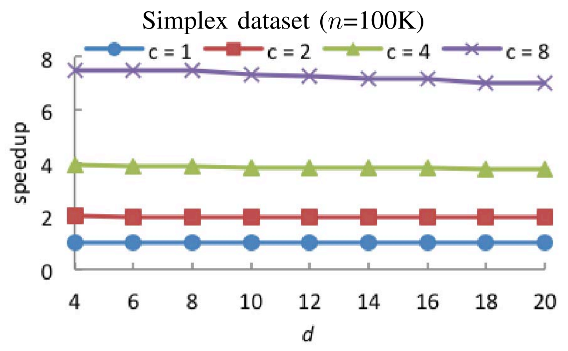
(a)



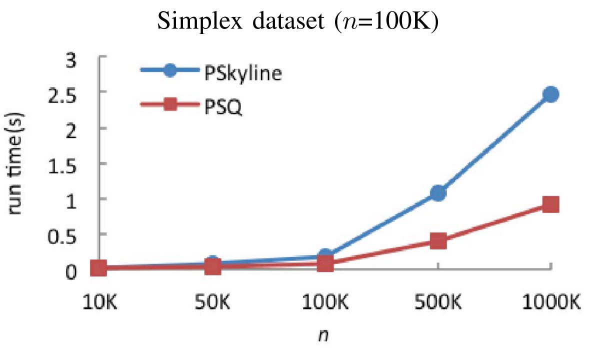
(b)



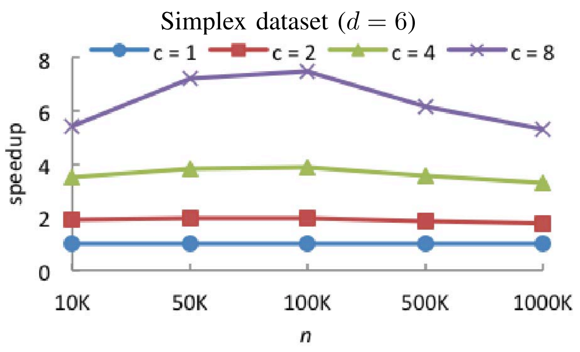
(b)



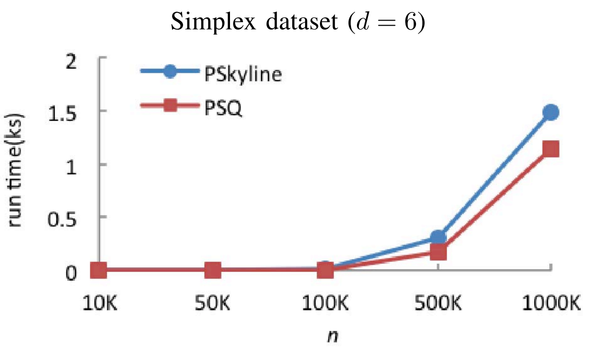
(c)



(c)



(d)



(d)

Figure 4. Performance of PSQ : (a)(c) speedup on varying dimension, (b)(d) speedup on varying data size.

Figure 5. Performance comparison of PSkyline and PSQ : (a)(c) run time comparison on varying dimension, (b)(d) run time comparison on varying data size.

[2] W. T. Balke, U. Guntzer and J. X. Zheng, Efficient Distributed Skylining for Web Information Systems, *Proceedings of the 9th International Conference on Extending Database Technology*, 2004, pp 256–273.

[3] J. Chomicki, P. Godfrey, J. Gryz and D. Liang, Skyline with Presorting, *Proceedings of the 19th International Conference on Data Engineering*, 2003, pp 717–719.

[4] W.-M. Chen, H.-K. Hwang and T.-H. Tsai, Maxima-finding algorithms for multidimensional samples: A two-phase approach, *Computational Geometry: Theory and Applications*, **45**:1-2 (2012), 33–53.

[5] A. Guttman, R-trees: a dynamic index structure for spatial searching, *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 1984. pp 47–57.

[6] Z. Huang, C. S. Jensen, H. Lu and B. C. Ooi, Skyline Queries Against Mobile Lightweight Devices in MANETs, *Proceedings of the International Conference on Data Engineering*, 2006. pp 210–219.

[7] H. Im, J. Park and S. Park, Parallel skyline computation on multicore architectures, *Information Systems*, **36**:4 (2011), 808–823.

[8] D. Kossmann, F. Ramsak and S. Rost, Shooting stars in the sky: an online algorithm for skyline queries, *Proceedings of 28th International Conference on Very Large Data Bases*, 2002, pp 275–286.

[9] H. Kung, F. Luccio and F. Preparata, On finding the maxima of a set of vectors, *Journal of the ACM*, **22**:4 (1975), pp 469–476.

[10] E. Lo, K. Y. Yip , K.-I. Lin, and D. W. Cheung, Progressive skylining over web-accessible databases, *Data & Knowledge Engineering*, **57** (2006) 122–147.

[11] D. H. McLain, Drawing contours from arbitrary data points, *The Computer Journal*, Nov. 1974.

[12] F. P. Preparata and M. I. Shamos, *Computational Geometry: An Introduction*, Springer-Verlag, 1985.

[13] D. Papadias, Y. Tao, G. Fu and B. Seeger, Progressive skyline computation in database systems, *ACM Transactions on Database Systems*, **30**:1 (2005), pp 41–82.

[14] A. Vlachou, C. Doulkeridis, Y. Kotidis and M. Vazirgiannis, Skypeer: Efficient Subspace Skyline Computation over Distributed Data, *Proceedings of the International Conference on Data Engineering*, 2007, pp 416–425.