Information Processing Letters 114 (2014) 710–713

Contents lists available at ScienceDirect

Information Processing Letters

www.elsevier.com/locate/ipl

Faster output-sensitive skyline computation algorithm

Jinfei Liu *, Li Xiong, Xiaofeng Xu

Department of Mathematics and Computer Science, Emory University, GA, USA

ARTICLE INFO

Article history:
Received 28 May 2014
Received in revised form 27 June 2014
Accepted 28 June 2014
Available online 2 July 2014
Communicated by Jinhui Xu

Keywords:
Skyline
Output-sensitive
Time complexity
Worst case
Computational complexity

ABSTRACT

We present the second output-sensitive skyline computation algorithm which is faster than the only existing output-sensitive skyline computation algorithm [1] in worst case because our algorithm does not rely on the existence of a linear time procedure for finding medians.

1. Introduction

The *Skyline*, which was also known as *Maxima* in computational geometry or *Pareto* in business management field, is important for many applications involving multi-criteria decision making. The skyline of a set of multi-dimensional objects consists of the objects for which no other object exists that is at least as good along every dimension.

Assume that we have a set of n objects. Each object of d real-valued attributes can be conceptualized as a d-dimensional point $(p[1], p[2], ..., p[d]) \in \mathbb{R}^d$ where $p[i]$ is the i-th attribute of p. We use P to refer to the set of all these points. Fig. 1(a) illustrates a set of eight objects $P = \{p_1, p_2, ..., p_8\}$, each representing the description of a hotel with two attributes: distance to the beach and price. Fig. 1(b) shows the corresponding points in the 2-dimensional space where x and y axes correspond to the value of attributes for distance to the beach and price, respectively. Given two points $p = (p[1], p[2], ..., p[d])$ and $p' = (p'[1], p'[2], ..., p'[d])$ in $\mathbb{R}^d$, p dominates p' iff for every i, $p[i] \leq p'[i]$ and for at least one i, $p[i] < p'[i]$ $(1 \leq i \leq d)$. To illustrate, in Fig. 1(b), point $p_3(2, 150)$ dominates point $p_2(3.5, 175)$ because $2 < 3.5$ and $150 < 175$. Given a set of points P, the *skyline* of P is the set of points in P that are not dominated by any other point in P. The skyline of Fig. 1(b) includes p_1, p_5, and p_8, which offer various tradeoffs between distance and price: p_1 is the nearest to the beach, p_8 is the cheapest, and p_5 may be a good compromise of the two factors. The skyline computation problem is to find the skyline set of the given database P considering attributes of the objects in P as dimensions of the space. Note that not every skyline point needs to dominate a point of P. For example, in Fig. 1(b), while points p_5 and p_8 dominate points $\{p_2, p_3, p_4\}$ and $\{p_2, p_4, p_6, p_7\}$, respectively, point p_1 dominates no point.

Definition 1 *(Skyline).* Given a dataset P of n points in d-dimensional space. Let p and p' be two different points in P, we say p dominates p' iff for all i, $p[i] \leq p'[i]$ and for at least one i, $p[i] < p'[i]$, where $p[i]$ is the i-th dimension of p and $1 \leq i \leq d$. The skyline points are those points in P that are not dominated by any other point in P.

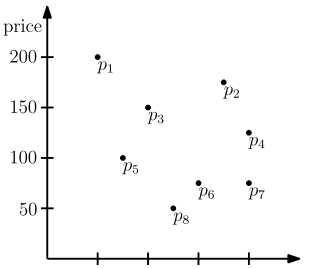
Previous results. The skyline computation problem has been extensively studied in the computational geometry field. The best of existing worst-case algorithms [2–5] are based on divide-and-conquer paradigm which achieves $O(n \log n)$ time complexity in two dimensional space,

* Corresponding author.

http://dx.doi.org/10.1016/j.ipl.2014.06.014
0020-0190/

hotel	distance	price
p_1	1	200
p_2	3.5	175
p_3	2	150
p_4	4	125
p_5	1.5	100
p_6	3	75
p_7	4	75
p_8	2.5	50

(a)



(b)

Fig. 1. A skyline example of hotels.

where n is the number of points. The only existing output-sensitive skyline computation algorithm was presented in [1] by Kirkpatrick and Seidel. Ref. [1] illustrated an algorithm that achieves $O(n \log k)$ time complexity where k is the number of skyline points. Recently, Hu et al. [6] extended the output-sensitive algorithm to external memory. Unfortunately, both algorithms rely on the existence of a linear time median algorithm [7] in the first step which lead to more than $5.4305\dot{n} \log k$ comparisons. Recently, Chan and Lee [8] presented two output-sensitive algorithms that achieve expected comparisons of $n \log k + O(n\sqrt{\log k})$.

2. Output-sensitive skyline computation algorithm

2.1. An output-sensitive algorithm in two dimensions

Let $P \subset \mathbb{R}^2$ be a set of n points and K be the number of skyline points we expect. Since the number of skyline points k is not known in advance, we will show later how to use a sequence of K values to find all k skyline points, that is, increase K until $K \geq k$. The algorithm given K is shown in Algorithm 1. The overall algorithm to find all k skyline points is shown in Algorithm 2.

In Algorithm 1, Step 1 partitions the n points into $\lceil n/K \rceil$ subsets. Then Step 2 computes the skyline points of each subset in $\lceil n/K \rceil \times O(K \log K) = O(n \log K)$ time. We then find a global skyline point by selecting a candidate skyline point in each subset (Step 3) and selecting the point with the global minimum value (Step 4). This point is used to eliminate all points dominated by this point in Step 5. Because of this elimination, we can guarantee that a skyline point is obtained in each subsequent iteration (Lemma 1). Step 3 to Step 5 are repeated for K iterations or until there is no remaining point, in which case all k ($k \leq K$) skyline points will be returned. If $k > K$, Algorithm 1 outputs an empty set since there are still remaining points after K skyline points are found with K iterations.

Example 1. Given 15 points in two-dimensional space, an example of Algorithm 1 is shown in Fig. 2. For simplicity, we assume $k = 5$ is known in advance. In (a), the 15 points are partitioned into 3 subsets (circle, box, and cross) of 5 each. The skyline points of each subset is then computed, shown in (b). Then we choose the point with smallest first dimension value from each subset as the candidate skyline point, shown in (c). From these three candidate points, we choose the point with smallest first dimension value as the skyline point, highlighted in (d). Then all points that are dominated by this skyline point, i.e. the points whose second dimension is equal to or greater than the determined skyline point, are eliminated, shown in (e). Then the next skyline point from the remaining points is selected in next iteration and used to eliminate the dominated pointed, shown in (f). The algorithm continues until all 5 skyline points are found.

Algorithm 1 $O(n \log K)$ SKYLINE(P, K).

Input: a set of n points in two-dimensional space
Output: k skyline points or $\emptyset$

```
1: /*Step 1: partition*/
2: partition P into subsets P_1, P_2, ..., P_⌈n/K⌉ randomly, each of size at most K.
3: /*Step 2: compute skyline points of each subset*/
4: for j = 1, 2, ..., ⌈n/K⌉ do
5:    compute the skyline points of P_j using worst-case O(n log n) skyline algorithm [9].
6: end for
7: for i = 1, 2, ..., K do
8:    /*Step 3: choose the candidate skyline points*/
9:    for j = 1, 2, ..., ⌈n/K⌉ do
10:      choose point p_j with smallest value on first dimension value as a candidate skyline point.
11:   end for
12:   /*Step 4: obtain one skyline point*/
13:   compute the point p_i with smallest first dimension value from p_j, 1 ≤ j ≤ ⌈n/K⌉.
14:   /*Step 5: eliminate non-skyline points*/
15:   for j = 1, 2, ..., ⌈n/K⌉ do
16:      perform a binary search to delete those points whose second dimension value is equal to or greater than p_i[2].
17:   end for
18:   if no point in P then
19:      return SKYLINE = {p_1, p_2, ..., p_i}.
20:   end if
21: end for
22: return ∅.
```

Algorithm 2 $O(n \log k)$ 2-D SKYLINE(P).

Input: a set of n points in two-dimensional space
Output: k skyline points

```
1: for t = 1, 2, ... do
2:    TEMP = O(n log K) SKYLINE(P, K), where K = min{2^(2^t), n}.
3:    if TEMP ≠ ∅ then
4:       return TEMP.
5:    end if
6: end for
```

Lemma 1. *We can obtain one skyline points in each iteration of Step 4.*

Proof. It is easy to see that we can obtain a skyline point from the first iteration because no point can dominate p_i due to the smallest value in the first dimension. For the second iteration, because all the points dominated by p_i are eliminated, the point with smallest first dimension value of remaining points in P should be a skyline point as no other point can dominate it. The analysis for all other skyline points follows similarly. □

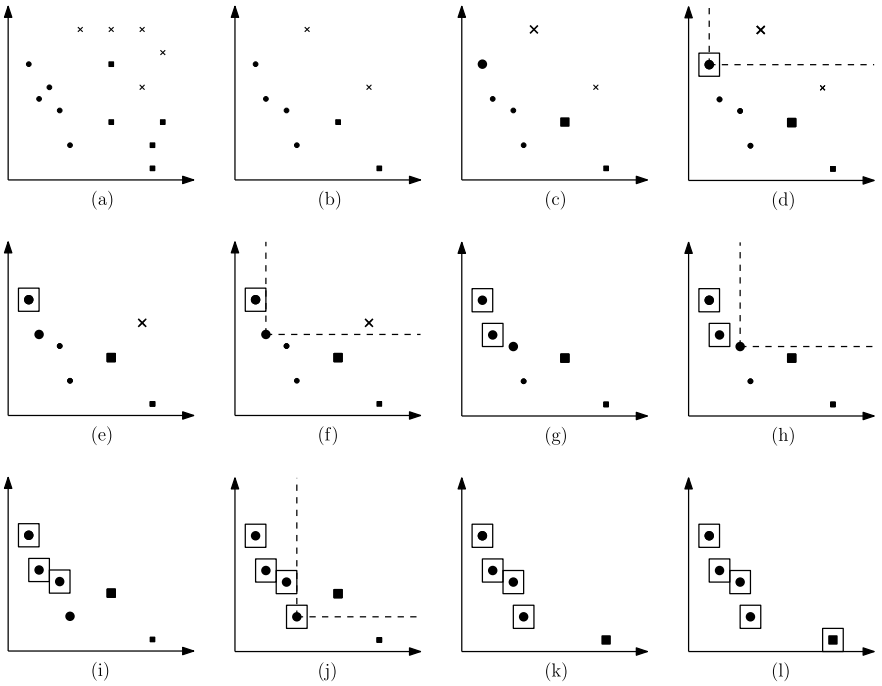


Fig. 2. An example of Algorithm 1. (a) Step 1: partition P into subsets $P_1, P_2, ..., P_{\lceil n/K \rceil}$ randomly. (b) Step 2: compute skyline points of each subset. (c) Step 3: choose candidate skyline points. (d) Step 4: obtain one skyline point. (e) After eliminating non-skyline points (Step 5).

Theorem 1. *Skyline can be computed in $O(n \log K)$ time in worst-case using Algorithm 1 where n is the number of points.*

Proof. Step 1 requires $O(n)$ time to partition. Line 5 requires $O(K \log K)$ time, hence, the total time of Step 2 is $\lceil n/K \rceil \times O(K \log K) = O(n \log K)$. Line 10 requires $O(1)$ time since the skyline points of each subset are already sorted by the first dimension, thus, Step 3 requires $\lceil n/K \rceil \times O(1) = O(\lceil n/K \rceil)$ time. Line 13 requires $O(\lceil n/K \rceil)$ time. Line 16 performs a binary search which requires $O(\log K)$ time. This is possible because if the skyline points of each subset are sorted in their first dimension, they are also sorted (reversely) in their second dimension. Thus, the total time of Step 5 requires $\lceil n/K \rceil \times O(\log K) = O(\lceil n/K \rceil \log K)$ time. In total, Step 3 to Step 5 require $O(\lceil n/K \rceil) + O(\lceil n/K \rceil) + O(\lceil n/K \rceil \log K)$ time. Since there are K iterations at most, the total time is

$$K \times \left(O\left(\lceil n/K \rceil\right) + O\left(\lceil n/K \rceil\right) + O\left(\lceil n/K \rceil \log K\right)\right)$$
$$= O(n) + O(n) + O(n \log K) = O(n \log K). \quad \square$$

Next, we extend Algorithm 1 by iteratively increasing K until all k skyline points are found. The final worst-case optimal $O(n \log k)$ time is shown in Algorithm 2. We note that Algorithm 2 is a classic paradigm for output-sensitive algorithms [1,10].

Theorem 2. *Algorithm 2 requires $O(n \log k)$ time in worst-case.*

Proof. Algorithm 2 stops with the list of skyline points as soon as the value of K in the for-loop reaches or exceeds k. The number of iterations in the loop is $\lceil \log \log k \rceil$, and the t-th iteration takes $O(n \log K) = O(n \log 2^{2^t}) = O(n 2^t)$ time. Hence, the total running time of the algorithm is

$$O\left(\sum_{t=1}^{\lceil \log \log k \rceil} n 2^t\right)$$
$$= O\left(n\left(2^1 + 2^2 + \ldots + 2^{\lceil \log \log k \rceil}\right)\right)$$
$$= O\left(n\left(2^{\lceil \log \log k \rceil + 1} - 2\right)\right) = O(n \log k). \quad \square$$

2.2. Analysis

Although the proposed algorithm has the same big O time complexity as [1], we analyze here that it has a significantly smaller constant factor and hence faster than [1]. Because we used the similar paradigm for Algorithm 2 with [1] and [6], we just focus on Algorithm 1. As noted by Luccio and Prparata [9], it is possible to find skyline by using no more than $S(n) + n$ comparisons for n points in two dimensional space, where $S(n)$ is the number of comparisons for sorting n numbers. If we use merge sort algorithm, then $S(n) = n \log n$. Therefore, we can compute skyline points within $n \log n + n$ comparisons. For Step 2 in our algorithm, it requires $\frac{n}{k}(k \log k + k) = n \log k + n$ comparisons. For Step 4, we use $\frac{n}{k}$ comparisons each time which leads to the total times for Step 4 is $k \times \frac{n}{k} = n$. For Step 5, we need $\log k$ comparisons to maintain the ordered structure. Hence, it requires $k \times \frac{n}{k} \log k = n \log k$ comparisons. In total, it requires $n \log k + n + n + n \log k = 2n \log k + 2n$ comparisons which is less than [1] (more than $5.430\dot{5} n \log k$ comparisons).

Acknowledgements

We would like to thank the reviewers for their helpful comments and suggestions. This research is supported by the AFOSR DDDAS grant FA9550-12-1-0240.

References

[1] D.G. Kirkpatrick, R. Seidel, Output-size sensitive algorithms for finding maximal vectors, in: Symposium on Computational Geometry, 1985, pp. 89–96.
[2] J.L. Bentley, Multidimensional divide-and-conquer, Commun. ACM 23 (4) (1980) 214–229.
[3] J.L. Bentley, K.L. Clarkson, D.B. Levine, Fast linear expected-time algorithms for computing maxima and convex hulls, in: SODA, 1990, pp. 179–187.
[4] J.L. Bentley, H.T. Kung, M. Schkolnick, C.D. Thompson, On the average number of maxima in a set of vectors and applications, J. ACM 25 (4) (1978) 536–543.
[5] H.T. Kung, F. Luccio, F.P. Preparata, On finding the maxima of a set of vectors, J. ACM 22 (4) (1975) 469–476.
[6] X. Hu, C. Sheng, Y. Tao, Y. Yang, S. Zhou, Output-sensitive skyline algorithms in external memory, in: SODA, 2013, pp. 887–900.
[7] M. Blum, R.W. Floyd, V.R. Pratt, R.L. Rivest, R.E. Tarjan, Time bounds for selection, J. Comput. Syst. Sci. 7 (4) (1973) 448–461.
[8] T.M. Chan, P. Lee, On constant factors in comparison-based geometric algorithms and data structures, in: Symposium on Computational Geometry, 2014, p. 40.
[9] F. Luccio, F. Preparata, On finding the maxima of a set of vectors, Instituto di Scienze dell'Informazione, Università di Pisa, 56100 Corso Italia 40, 1973.
[10] T.M. Chan, Optimal output-sensitive convex hull algorithms in two and three dimensions, Discrete Comput. Geom. 16 (4) (1996) 361–368.