ELSEVIER

Information Processing Letters 59 (1996) 255-259

Information Processing Letters

Output-sensitive peeling of convex and maximal layers

Franck Nielsen [a,b,1]

[a] *INRIA, BP93, 06902 Sophia-Antipolis cedex, France*

[b] *University of Nice, Parc Valrose, 06304 Nice cedex, France*

Received 8 August 1995; revised 10 June 1996
Communicated by M.T. Goodrich

Abstract

We give an output-sensitive algorithm to compute the first k convex or maximal layers in $O(n \log H_k)$-time where H_k is the number of points participating in the first k layers. Computing only the first k layers is interesting in various problems that arise in computational geometry (k-sets and dually k-levels, k-hulls and dually k-belts), pattern recognition, statistics, operations research, etc.

Keywords: Convex hulls; Computational geometry

1. Introduction and notations

Let $\mathcal{S}$ be a set of n planar points and $\mathcal{L}(\mathcal{S})$ its first layer, i.e. a subset of $\mathcal{S}$. Let $\mathcal{S}_1 = \mathcal{S}$ and $\mathcal{S}_{i+1} = \mathcal{S}_i \backslash \mathcal{L}(\mathcal{S}_i)$ for integer $i \geqslant 1$. The layers of $\mathcal{S}$ are defined by the following ordered sequence: $\mathcal{L}(\mathcal{S}_1), \ldots, \mathcal{L}(\mathcal{S}_l)$ where l is the first integer such that $\mathcal{S}_{l+1} = \emptyset$. We successively investigate the case where $\mathcal{L}(\mathcal{S})$ is the set $\mathcal{V}(\mathcal{S})$ of points of $\mathcal{S}$ on the boundary of the convex hull $\mathcal{CH}(\mathcal{S})$ (Section 2, Fig. 1) and the maxima $\mathcal{M}(\mathcal{S})$ of $\mathcal{S}$ (Section 3). Let $h_i = |\mathcal{L}_i|$ and $H_i = \sum_{j=1}^{i} h_j$ denote respectively the number of points of $\mathcal{L}(\mathcal{S}_i)$ and the number of points participating in the $\leqslant i$-layers. Let $H_0 = 0$. We have $H_i = H_{i-1} + h_i$ for $i \geqslant 1$. We assume in the sequel that $\mathcal{S}$ is in general position, i.e., all the points are distinct and no three points are colinear. Notice that in that case $H_k \geqslant \min\{3k, n\}$. For a set of n points in the euclidean plane $\mathbb{E}^2$, the algorithms have running time $O(n \log H_k)$ using linear storage.

Computing the first k convex layers of a planar point set has numerous applications [18] and is also used as a first step when one computes k-hulls [4] (or dually k-belts [5]) or k-sets (or dually k-levels [6]). A k-set of $\mathcal{S}$ is a set $\mathcal{S} \cap H$ of size k for some halfspace H. For any set $\mathcal{S}$ of points, the k-hull of $\mathcal{S}$ is the set of points p such that for any line passing through p there are at least $k \leqslant \lfloor n/2 \rfloor$ points in each closed halfspace. It follows from the definition that the k-hull contains the set of points $\mathcal{S}_k$. k-hulls are also used in statistics to measure the *interiorness* of a given set. Using our output-sensitive algorithm (Section 2) for computing the first k layers, we obtain an $O(n \log H_k + (H_k + f_k) \log^2 k)$-time algorithm for computing the k-hull of a n-point set where f_k is the size of the k-hull and H_k is the size of the first k convex layers (the previous algorithm of [4] has running time $O(n \log n + (n + f_k) \log^2 k)$).

As already noticed by Everett et al. [6] in their algorithm for computing the $\leqslant k$-levels: "... *we can remove all layers deeper than k from consideration* ...", so that

[1] Email: franck.nielsen@sophia.inria.fr.

PII S0020-0190(96)00116-0

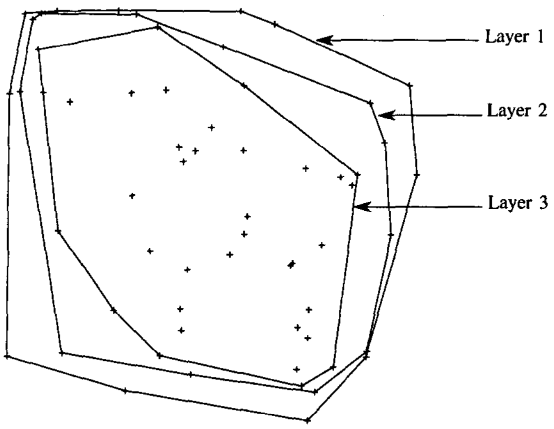

Fig. 1. The first three layers of a set of 60 planar points.

we can replace their $O(n \log n)$ preprocessing time by a $O(n \log H_k)$ preprocessing time to compute the ($\leqslant k$)-sets ($\leqslant k$-levels) of a set of n points (respectively lines) in the euclidean plane. This is to be compared with the result of Chan [2]: a $O(n \log h + h \log^2 n)$-time algorithm is given to compute the h faces of the kth level of n lines using the grouping scheme with ray shooting procedures based on parametric search [1].

We first begin with an overview of the previous algorithms that compute the convex layer decomposition, i.e., all the convex layers of a given set. One can trivially compute the convex layer decomposition in $O(n^2 \log n)$-time by successively peeling $\mathcal{S}$ using any optimal planar convex hull algorithm. Another way to proceed is to first create an x-monotone polygonal chain and then to peel it in $O(n^2)$-time by successively applying any linear-time convex hull algorithm on simple polygons. Overmars and van Leeuwen [16,17] gave a dynamic algorithm to maintain in $O(\log^2 n)$ time per insertion/deletion the convex hull of n points. Their method leads straightforwardly to an $O(n \log^2 n)$-time convex layers algorithm. Chazelle [3] gave an optimal $\Theta(n \log n)$ time algorithm to compute the convex layer decomposition using the hull tree of Overmars and van Leeuwen. His solution relies upon an $\bar{O}(\log n)$ amortized cost procedure that maintains dynamically the convex hull of a set of n points (although the insertion/deletion of a simple point can cost linear time).

We consider the problem of computing only the first k layers of the convex or maximal layers of a planar point set in an output-sensitive manner.

We can apply round after round Jarvis's algorithm [9] to get an output-sensitive algorithm with running time $O(nH_k)$. Kirkpatrick and Seidel [13] gave an optimal output-sensitive algorithm to compute the convex hull of a set of n points in $\Theta(n \log h_1)$-time. We can use successively that algorithm to compute the first k layers in time $(\sum_{i=1}^{k}(n - H_{i-1}) \log h_i) = O(nk \log(H_k/k))$ using Hölder's inequality[2]. This algorithm is therefore optimal if $k = O(1)$. However, in pattern recognition [11], one wants to remove some of the first layers in order to eliminate noise of the input set and to get the real shape of the set of points. The number of layers we want to remove depends on the size of the set (the number of available data) so that generally k is a function of n.

We present below an $O(n \log H_k)$-time algorithm to compute the $\leqslant k$-maximal or convex layers.

2. Computing the first k convex layers

2.1. The power of grouping

2.1.1. Principles

Our algorithm basically uses the grouping scheme, i.e., a general paradigm to obtain output-sensitive algorithms, independently developed by Chan [2] and Nielsen [15]. It consists in grouping the points into balanced-size sets and in preprocessing each group in order to answer queries efficiently. Finally, we build the solution stepwise by querying these groups. For each query, we first determine in each group the local result of that query and then deduce the global result of the query by merging these local results (e.g., selecting one of them).

2.1.2. Answering queries

What is a *query* for the convex hull problem? We consider a group $\mathcal{P}$ of p points. Given a query point Q, with the property that $Q \notin \mathcal{CH}(\mathcal{P})$, we want to determine the two segments issued from Q and tangent to $\partial\mathcal{CH}(\mathcal{P})$, the boundary of the convex hull $\mathcal{CH}(\mathcal{P})$.

We preprocess $\mathcal{P}$ into a hull tree [16,18] in time $O(p \log p)$ (see Fig. 2), so that given a query point Q we can determine in $O(\log p)$ the two tangent seg-

[2] We can state Hölder's inequality as follows: $\max_{p_1+p_2=p}\{\log p_1 + \log p_2\} \leqslant 2\log(p/2)$.

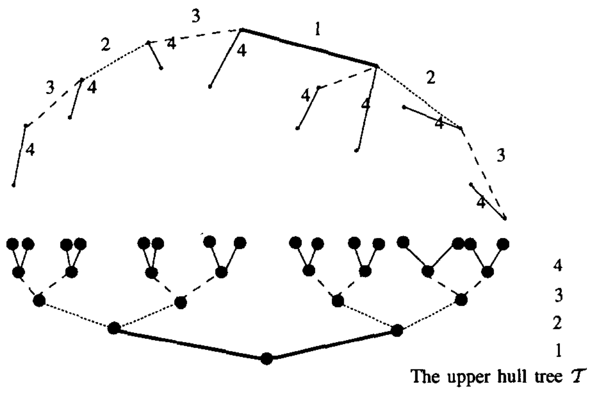

Fig. 2. The upper hull tree of a set of 16 points.

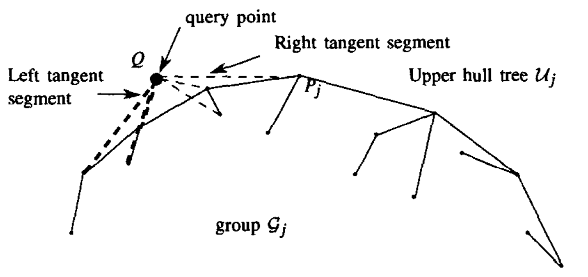

Fig. 3. A query (in one of the groups): given a point Q outside the convex hull, determine its two tangent segments.

ments linking Q to $\partial\mathcal{CH}(\mathcal{P})$ (see Fig. 3). Maintaining dynamically under deletion the group $\mathcal{P}$ requires $O(\log^2 p)$ time per insertion/deletion. (Note that once the data-structures are built, we will only delete points from them in our algorithm).

2.2. *The algorithm given an estimate of the output-size*

Assume that we know a good estimate $h_e \leqslant n^{1/4}$ of the output-size of the $\leqslant k$-layers. Let Layers(k, h_e) be the algorithm described below:

- *Preprocessing.* Group the data points in $\lceil n/h_e \rceil$ balanced-size groups of size at most h_e. For each group $\mathcal{G}_i$, build the upper and lower hull trees. Let $\mathcal{U}_i$ and $\mathcal{L}_i$ be respectively the upper and lower hull trees of the points of $\mathcal{G}_i$.
- *Computing a layer.* If $k = 0$ then HALT. Otherwise, find the two extremal points P_1 and P_2 that have respectively the smallest and largest x-coordinate of the current set of points $(x(P_1) \leqslant x(P_2))$. Let P be initially point P_1. Let $\overrightarrow{v}$ be any vertical vector.
- *Upper Hull.* For each group $\mathcal{G}_j$, find the right tangent segment PP_j given the query point P by querying the upper hull tree $\mathcal{U}_j$. Among the $\lceil n/h_e \rceil$ groups, select the right tangent segment $[PP_i]$ whose angle with the vector $\overrightarrow{v}$ is minimized. Add P_i to the current layer and remove P_i from both the upper and the lower hull trees, $\mathcal{U}_i$ and $\mathcal{L}_i$, of its group $\mathcal{G}_i$. Let $\overrightarrow{v} = \overrightarrow{PP_i}$ and P be P_i. If $P \neq P_2$ then go to step *Upper Hull.*
- *Lower Hull.* symmetric case.
- *Next Layer.* $k \leftarrow k - 1$. Go to step *computing a layer.*

Let $c(n, h)$ denote the time complexity of the above algorithm. Grouping the points into $\lceil n/h_e \rceil$ balanced groups of size at most h_e and computing their upper and lower hull trees cost $O(\lceil n/h_e \rceil h_e \log h_e) = O(n \log h_e)$. We may assume, without loss of generality, that the kth layer is not empty, i.e., $h_k > 0$). Consider the cost of computing the ith layer:

- We first find the two extremal points P_1 and P_2 by computing in each group $\mathcal{G}_i$ the extremal points $P_{1,i}$ and $P_{2,i}$ with $x(P_{1,i}) \leqslant x(P_{2,i})$. Then, we select among those candidates, the extremal points in $O(\lceil n/h_e \rceil)$-time. Finding the $P_{1,i}$s and $P_{2,i}$s costs $O(\lceil n/h_e \rceil \log h_e)$ time by querying, for example, the upper hull trees.
- Computing the upper and lower ith layers costs $O(\lceil n/h_e \rceil h_i \log^2 h_e)$. Indeed, each time we find a new point belonging to the ith layer we answer $\lceil n/h_e \rceil$ queries and remove a point from one of the groups (which has size less than h_e).

Thus, computing the $\leqslant k$-layers costs:

$$O\left(\left\lceil \frac{n}{h_e} \right\rceil h_e \log h_e\right) + O\left(\left\lceil \frac{n}{h_e} \right\rceil \left(k + \sum_{i=1}^{k} h_i\right) \log^2 h_e\right)$$

$$= O\left(n \log h_e\right) + O\left(\frac{nH_k}{h_e} \log^2 h_e\right).$$

In the sequel, we will choose h_e ($h_e \leqslant n$) such that $H_k^2 \leqslant h_e$. Thus, the algorithm runs in time $O(n \log H_k)$ since in that case $O(n(H_k/h_e) \log h_e) = O(n)$. (Notice that if we only choose $h_e \geqslant H_k$ then our algorithm runs in time $O(n \log^2 H_k)$.)

Another interesting feature of the algorithm is that we can determine whether $H_k > p$ or not for a given integer p in $O(n \log p)$. Indeed, we choose $h_e = p^2$

and we stop the queries as soon as we have computed $\min\{p, H_k\}$ points belonging to the first k convex layers. Denote by $\mathsf{Test}(p)$ that algorithm.

2.3. *The final algorithm*

We put together the basic procedures seen so far. The goal is to determine quickly a good estimate h_e of H_k, that is to find h_e such that $H_k^2 \leqslant h_e \leqslant n$. Then, we run the $\mathsf{Layers}(h_e, k)$ algorithm. The final algorithm is described below:

- *Initializing.* Let $i = 0$ and $h_e = (2^{2^0})^2 = 4$.
- *Computing a good estimate.* While $(\sqrt{h_e} < H_k)$ do $i \leftarrow i + 1$ and $h_e = (2^{2^i})^2$. If $h_e > n$ then run Chazelle's algorithm [3].
- *Computing the $\leqslant$ k-layers.* Call algorithm $\mathsf{Layers}(k, h_e)$.

Note that the test $(\sqrt{h_e} < H_k)$ is performed using algorithm $\mathsf{Test}(\sqrt{h_e})$ described above in $O(n \log h_e)$-time. When we get out of the while-loop, we have the desired inequalities: $H_k^2 \leqslant h_e$. We have to assert that $h_e \leqslant n$. This means that in the worst-case when we exit the while-loop, we have $h_e = (H_k^2 - 1)^2 \leqslant H_k^4$. Therefore, if $H_k \leqslant n^{1/4}$ then we get $h_e \leqslant n$.

Let $c(n, H_k)$ denote the running time of the previous algorithm. Clearly, we have:

$$c(n,H_k) = O(1) + O\left(\sum_{i=0}^{\lceil \log\log H_k \rceil} n \log 2^{2^i}\right) + O(n \log h_e),$$

$$c(n,H_k) = O(n \log H_k), \quad \text{if } H_k \leqslant n^{1/4}.$$

If $H_k > n^{1/4}$ we run Chazelle's algorithm [3] and find the $\leqslant k$-layers in $O(n \log n) = O(n \log H_k)$.

3. Computing the first k maximal layers

In this section, we just re-phrase the previous algorithm in a context of maximal layers. More precisely, we change the query problem of the last problem. Notice that in the case of the $\leqslant k$-maximal layers, we have $\mathcal{L}(\mathcal{S}) = \mathcal{M}(\mathcal{S})$ where $\mathcal{M}(\mathcal{S})$ is the set of maxima points of $\mathcal{S}$, that is all the points of $\mathcal{S}$ that are not dominated by any other, where a point P dominates a point Q if $x(P) > x(Q)$ and $y(P) > y(Q)$ (see [19,14,12,7,8,10]). The maxima $\mathcal{M}(\mathcal{S}) = \{M_1 = (x_1, y_1), \ldots, M_i = (x_i, y_i)\}$ of a point set $\mathcal{S}$ can be arranged into a staircase structure as follows: we join two consecutive maxima points $M_k = (x_k, y_k)$ and $M_l = (x_l, y_l)$, with $x_k \leqslant x_j$ or $x_j \geqslant x_l$ by the two iso-oriented segments $[(x_k, y_k), (x_k, y_l)]$ and $[(x_k, y_l), (x_l, y_l)]$ (see Fig. 4).

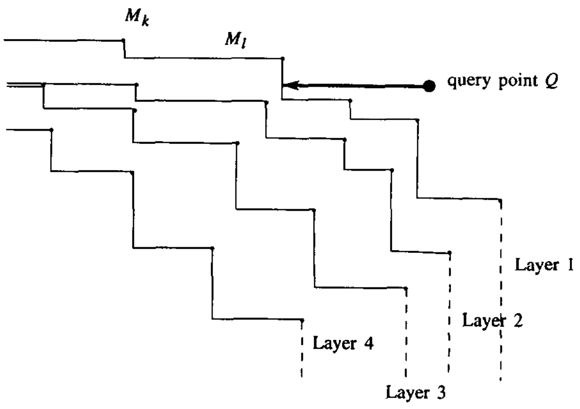

Fig. 4. A query (in one of the groups): given a point Q above the first layer of the maxima points, determine the first (rightmost abscissa) point of the first layer that is to the left of Q.

We group the initial set $\mathcal{S}$ of n points into $\lceil n/p \rceil$ balanced-size groups of size at most p. We compute for each group $\mathcal{G}_i$, $1 \leqslant i \leqslant \lceil n/p \rceil$, a data-structure that allows to maintain dynamically its first maximal layer [10]. The total cost of the preprocessing step is $O(\lceil n/p \rceil p \log p) = O(n \log p)$. The maximal layer of a set of p points can be maintained dynamically both under insertion/deletion in $O(\log p)$ [10]. (Note that we only delete points from our data-structures in the grouping strategy described below). We describe the algorithm $\mathsf{MaximaLayers}(k, p)$ below:

- *Preprocessing.* Group the points into $\lceil n/p \rceil$ groups of size at most p. For each group $\mathcal{G}_i$, compute the data-structure that allow to maintain dynamically the first maximal layer of $\mathcal{G}_i$ using the algorithm of Kapoor [10].
- *Computing a maximal layer.* If $k = 0$ then HALT.
  - *Initializing.* Find the rightmost abscissa point P_{init} of the points clustered into groups by querying each group and selecting the overall rightmost abscissa point. Set P to P_{init}.
  - *Querying.* For each group $\mathcal{G}_i$, "ray shoot" from P horizontally to the left until it hits a staircase

of the first layer of maximas of $\mathcal{G}_i$. Let P_i be the point of $\mathcal{G}_i$ at the top of this staircase (if it does not hit any staircase then set P_i to $(-\infty, +\infty)$). Let P' be the point which has the highest x-coordinate among the P_i's. If $P' = (-\infty, +\infty)$ then skip to step *Next layer*. Otherwise, add P' to the current maximal layer and remove P' from the data-structure of its group. Set $P \leftarrow P'$. Go to *Querying*.

- *Next layer.* $k \leftarrow k - 1$. Go to *Computing a maximal layer.*

Using the structure of Kapoor [10], we can delete a point in a group $\mathcal{G}_i$ in $O(\log p)$ time. The "ray shooting" operations (one per group) used in the querying stage can easily be done in $O((n/p) \log p)$-time. Indeed, for a group $\mathcal{G}_i$ and a query point Q we want to find among the points of the first maximal layer, the one that is above $y(Q)$ and has the highest abscissa. This may be easily implemented using a binary search. (Alternatively, we can also simulate an insertion of Q into the data-structure of $\mathcal{G}_i$ in order to find its left neighbor.)

As before, we can study the complexity of the previous algorithm. Its running time is $O((n/p) H_k \log p) + O(n \log p)$. We then find a good estimate p such that $H_k \leqslant p < H_k^2$ and run the algorithm with that estimate. We finally end up with an $O(n \log H_k)$-time algorithm.

Acknowledgment

I am very thankful to my supervisors, Jean-Daniel Boissonnat and Mariette Yvinec, whose valuable comments on this paper led to a much clearer presentation. I also greatly appreciated the comments of an anonymous referee.

References

[1] P.K. Agarwal and J. Matoušek, Ray shooting and parametric search, *SIAM J. Comput.* **22** (4) (1993) 794–806.

[2] T.M.Y. Chan, Output-sensitive results on convex hulls, extreme points and related problems, in: *Proc. 11th Ann. ACM Symp. on Computational Geometry* (1995) 10–19.

[3] B. Chazelle, On the convex layers of a planar set, *IEEE Trans. Inform. Theory* **31** (1985) 509–517.

[4] R. Cole, M. Sharir and C.K. Yap, On k-hulls and related problems, *SIAM J. Comput.* **16** (1987) 61–77.

[5] H. Edelsbrunner and E. Welzl, Constructing belts in two-dimensional arrangements with applications, *SIAM J. Comput.* **15** (1986) 271–284.

[6] H. Everett, J.-M. Robert and M. van Kreveld, An optimal algorithm for the ($\leqslant k$)-levels, with applications to separation and transversal problems, in: *Proc. 9th Ann. ACM Symp. on Computational Geometry* (1993) 38–46.

[7] G. Frederickson and S. Rodger, A new approach to the dynamic maintenance of maximal points in a plane, *Discrete Comput. Geom.* **5** (1990) 365–374.

[8] R. Janardan, On the dynamic maintenance of maximal points in the plane, *Inform. Process. Lett.* **40** (1991) 59–64.

[9] R.A. Jarvis, On the identification of the convex hull of a finite set of points in the plane, *Inform. Process. Lett.* **2** (1973) 18–21.

[10] S. Kapoor, Dynamic maintenance of maximas of 2-d point sets, in: *Proc. 10th Ann. ACM Symp. on Computational Geometry* (1994) 140–149.

[11] D.G. Kirkpatrick and J.D. Radke, A framework for computational morphology, in: G.T. Toussaint, ed., *Computational Geometry* (North-Holland, Amsterdam, 1985) 217–248.

[12] D.G. Kirkpatrick and R. Seidel, Output-size sensitive algorithms for finding maximal vectors, in: *Proc. 1st Ann. ACM Symp. on Computational Geometry* (1985) 89–96.

[13] D.G. Kirkpatrick and R. Seidel, The ultimate planar convex hull algorithm?, *SIAM J. Comput.* **15** (1986) 287–299.

[14] H.T. Kung, F. Luccio and F.P. Preparata, On finding the maxima of a set of vectors, *J. ACM* **22** (1975) 469–476.

[15] F. Nielsen and M. Yvinec, Output-sensitive convex hull algorithms of planar convex objects, Research Rept. 2575, INRIA, BP93, 06902 Sophia-Antipolis, France, 1995.

[16] M.H. Overmars and J. van Leeuwen, Dynamically maintaining configurations in the plane, in: *Proc. 12th Ann. ACM Symp. on Theory of Computing* (1980) 135–145.

[17] M.H. Overmars and J. van Leeuwen, Maintenance of configurations in the plane, *J. Comput. System Sci.* **23** (1981) 166–204.

[18] F.P. Preparata and M.I. Shamos, *Computational Geometry: An Introduction* (Springer, New York, 1985).

[19] F.F. Yao, On finding the maximal elements in a set of plane vectors, Rept. R-667, Dept. of Computer Science, University of Illinois, Urbana, 1974.