On the Convex Layers of a Planar Set

BERNARD CHAZELLE

Abstract—Let S be a set of n points in the Euclidean plane. The convex layers of S are the convex polygons obtained by iterating on the following procedure: compute the convex hull of S and remove its vertices from S. This process of peeling a planar point set is central in the study of robust estimators in statistics. It also provides valuable information on the morphology of a set of sites and has proven to be an efficient preconditioning for range search problems. An optimal algorithm is described for computing the convex layers of S. The algorithm runs in $O(n \log n)$ time and requires $O(n)$ space. Also addressed is the problem of determining the depth of a query point within the convex layers of S, i.e., the number of layers that enclose the query point. This is essentially a planar point location problem, for which optimal solutions are therefore known. Taking advantage of structural properties of the problem, however, a much simpler optimal solution is derived.

I. Introduction

LET $S = \{p_0, \cdots, p_{n-1}\}$ be a set of n points in the Euclidean plane. The set of convex layers of S, denoted $C(S)$ in the following, is the set of convex polygons defined iteratively as follows: compute the convex hull of S and remove its vertices from S (Fig. 1). The convex layers of a point set can be seen as a natural extension of its convex hull. In [17] Shamos mentions applications of this concept to pattern recognition and statistics. A central problem in robust estimation is that of evaluating an unbiased estimator that is not too sensitive to outliers, i.e., observations lying abnormally far from the others. To tackle the two-dimensional version of this problem, Tukey has suggested removing the outliers of a point set by peeling or shelling the set in the manner described above, iterating on this process until only a prescribed fraction of the original points remain [9].

Another illustration of the importance of convex layers in computational geometry has come up recently in the context of a well-known retrieval problem. The halfplane range search problem involves preprocessing n points in the Euclidean plane so that for any query line L, the subset of points lying on a given side of L can be reported effectively. The use of convex layers allowed Chazelle, Guibas, and Lee [6] to derive an optimal solution to this problem.

Manuscript received October 28, 1983; revised January 10, 1985. This work was supported in part by the National Science Foundation under Grant MCS 83-03925 and the Office of Naval Research and the Defense Advanced Research Projects Agency under Contract N00014-83-K-0146 and DARPA Order 4786. The material in this paper was partially presented at the 21st Annual Allerton Conference on Communication, Control, and Computing, Monticello, IL, October 1983.

The author is with the Department of Computer Science, Brown University, Providence, RI 02912, USA.

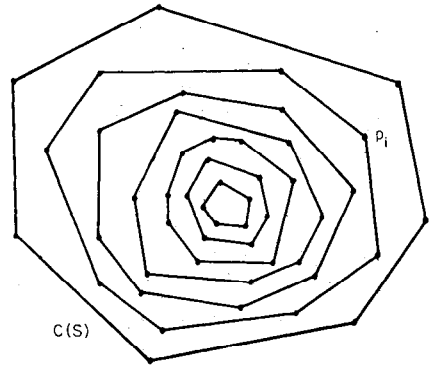

Fig. 1. Convex layers of point set.

Besides its practical relevance, the problem of computing the convex layers of a point set is also interesting in its own right, for it intuitively represents a geometric "equivalent" to sorting. Indeed, considering the various algorithms known for computing the convex hull of a set of points, one is tempted to draw a parallel with sorting algorithms. The Jarvis march [10] resembles selection sort, Bentley and Shamos's method [3] smacks of merge sort, and Eddy's algorithm [7] is strongly reminiscent of quicksort. There is, however, a fundamental difference that often makes computing convex hulls easier than sorting; this is the fact that the output is a convex polygon that may contain only a small fraction of the original points. This is what allows the existence of linear-expected-time algorithms for computing convex hulls under certain distributions of the points [2], [3], [18]. Knowing that similar results are provably impossible to obtain in the case of sorting [1], one can appreciate the intrinsic difference between the two problems. One way of bridging this complexity gap is precisely to require the explicit computation of all the convex layers of the set of points, for it then becomes impossible to take advantage of the possible scarcity of the output in order to bound the time complexity of the problem.

This paper describes an $O(n)$ space, $O(n \log n)$ time algorithm for computing the convex layers of S. Because the convex hull of S is one of the convex layers, computing $C(S)$ requires $\Omega(n \log n)$ time [17], [20]. Our algorithm is therefore optimal. A number of $O(n^2)$ time algorithms for computing convex layers have been found [8], [17], but the most efficient method previously known for this problem requires $O(n \log^2 n)$ time [13]. It is based on a general technique for maintaining the convex hull of a point set in a dynamic environment. Any point can be inserted or

0018-9448/85/0700-0509$01.00

deleted in $O(\log^2 n)$ time, while efficient retrieval of the convex hull is possible at any given time. The main contribution of this paper is to show that the deletions involved in the computation of $C(S)$ can be "batched" together (as well as simplified) to give an $O(n \log n)$ overall running time.

We also address the problem of determining the depth of a query point within the convex layers of S, i.e., the number of layers that enclose the query point. This can be reduced to a planar point location problem and can therefore be solved optimally [11], [12]. The specific structure of the problem at hand, however, allows us to use a new point location technique that is both optimal and practical, and avoids many of the complications of previous algorithms.

This paper is organized as follows. In Section II we introduce the basic ingredients of our method and prove a number of preliminary results. In Section III we give a detailed description of the algorithm for computing convex layers, and in Section IV we attack the problem of determining the depth of a query point.

II. The Ingredients

We endow the Euclidean plane with a Cartesian system of reference (Ox, Oy). Each convex boundary in $C(S)$ can be represented as the concatenation of two convex polygonal lines, called upper and lower chains (Fig. 2). Let a (resp. b) denote the point of the convex layer C with minimum (resp. maximum) x-coordinate. The upper (resp. lower) chain of C runs clockwise (resp. counterclockwise) from a to b. Note that the lower and upper chains are the same if C consists of one or two points only. From now on, we will concentrate on the computation of, say, the upper chains of $C(S)$; the other case is strictly similar. Assume without loss of generality, that $p_0, \cdots, p_{n-1}$ appear in this order by nondecreasing x coordinates. Consider the complete binary tree, denoted T, whose leaves are $p_0, \cdots, p_{n-1}$, from left to right. Let $S(v)$ be the set of points stored at the leaves of the subtree of T rooted at node v, and let $U(v)$ be the upper chain of the convex hull of $S(v)$, also referred to as the upper hull of $S(v)$. The union of all the edges in $U(v)$, for all nodes v in T, forms a planar graph G that is easily shown to be connected and acyclic, i.e., forming a free tree. G is called the hull graph of S. Fig. 3 depicts a hull graph and its correspondence with the binary tree T. Each edge of the hull graph is a so-called tangent to two convex chains, and is in one-to-one correspondence with a node of T.

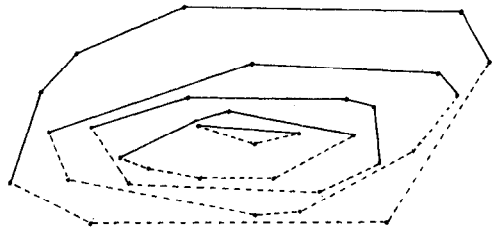

Fig. 2. Upper and lower chains.

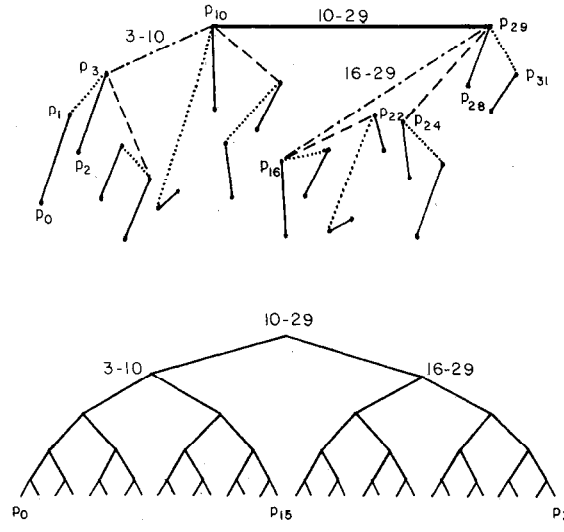

Fig. 3. Hull graph of S.

Note that this data structure is essentially a simplified variant of the structure used by Overmars and van Leeuwen for computing convex hulls dynamically [13]. Similarly to G, we define G' as the hull graph of S with respect to lower chains. Computing G is a straightforward operation, as long as we can efficiently compute the tangent to two upper chains. An algorithm for a very similar task has been described by Preparata and Hong [15], so we only sketch the procedure.

Let U (resp. V) be an upper chain with vertices $u_1, \cdots, u_m$ (resp. $v_1, \cdots, v_p$), in clockwise order. Assume that all the vertices of U have smaller x coordinates than the vertices of V. The tangent to U and V is defined as the unique edge joining U and V in the upper chain of the convex hull of $U_1 \cup U_2$. It is easy to compute this segment s as follows. Set s to $u_1 v_i$, for $i = 1, 2, \cdots$, until both v_{i-1} and v_{i+1} lie below s. The segment s is now tangent to V but, in general, not to U. Next, we take the other endpoint of s to $u_2, u_3, \cdots$ successively, until we reach a vertex u_j for which both u_{j-1} and u_{j+1} lie below s. During the course of this operation, we must be careful to roll the line passing through s around V. Keeping track of the two angles formed by s and U, V allows us to determine the next point to go to in constant time. Since in this process both endpoints move around their respective upper chains in clockwise order, the total running time of the algorithm is $O(m + p)$.

We are now in a position to compute the graph G. Before proceeding, however, let us specify the data structure used for its representation. We use a traditional adjacency-list structure, whereby each vertex p has associated with it a vertex-to-edge list $V(p)$ with the names of its adjacent vertices. Each vertex-to-edge list $V(p)$ consists of two sublists, $V_1(p)$ and $V_2(p)$, defined as follows. $V_1(p)$ (resp. $V_2(p)$) is an angularly sorted doubly linked list containing the vertices adjacent to p with smaller (resp. greater) x coordinates than p (Fig. 4). We define the top