Enumerating Extreme Points in Higher Dimensions*

Th. Ottmann† S. Schuierer† S. Soundaralakshmi†

Abstract

We consider the problem of enumerating all extreme points of a given set P of n points in d dimensions. We present an algorithm with $O(n)$ space and $O(nm)$ time where m is the number of extreme points of P.

We also present an algorithm to compute the depth of each point of the given set of n points in d-dimensions. This algorithm has complexity $O(n^2)$ which significantly improves the $O(n^3)$ complexity of the previously best known deterministic algorithm. It also improves the best known randomized algorithm with expected running time $O(n^{3-\frac{2}{1+\lfloor d/2 \rfloor}+\delta})$ (for any fixed $\delta > 0$).

Topics: Computational Geometry

1 Introduction

The convex hull of a set of points is the smallest convex set that contains the points. The construction of the convex hull is a central problem in computational geometry, with applications to pattern recognition, image processing, generation of Voronoi diagrams and a number of other problems, where the generation of the convex hull is often a preprocessing step for a main algorithm.

*This research is supported by the DFG-Project "Diskrete Probleme", No. Ot 64/8-1.
†Universität Freiburg, Institut für Informatik, Am Flughafen 17, 79110 Freiburg, FRG. e-mail: {ottmann,schuiere}@informatik.uni-freiburg.de, Fax: (0761) 203-8122

Given a set P of n points in d dimensions we say that point p in P is an *extreme point* of P if $conv(P) \neq conv(P \setminus \{p\})$ where $conv(P)$ denotes the convex hull of P. The set of extreme points of P can also be characterized as the minimal subset E of P with $conv(P) = conv(E)$. In particular, the extreme points of P are the vertices of the convex hull of P. In dimensions two and three the problem of determining the extreme points of a given set P can be solved in optimal time $O(n \log n)$ by computing the convex hull of P. The problem of finding the convex hull of a set of n points has been extensively studied and many elegant algorithms exist for its solution [2, 3, 11, 13, 14, 15, 16]. The method for identifying the points on the convex hull in two and three dimensions cannot be extended to dimensions greater than three since the number of facets of the convex hull of P has an exponential growth in higher dimensions, i.e., there can be up to $\Omega(n^{\lfloor d/2 \rfloor})$ facets [5].

Deciding whether a point in P is an extreme point or not can be formulated as a linear programming problem. There are many deterministic and randomized algorithms that solve a linear program with n constraints in time $O(n)$ [5, 8, 10, 18, 22, 23]. Unfortunately, the constants are exponential functions of d. The best known deterministic algorithm has a running time of $O(2^{O(d \log d)} n)$ [5] while one of the fastest randomized algorithms is a combination of the approach of Clarkson [8] and Sharir and Welzl [23] with a running time of $O(d^2 n + d^3 \sqrt{n} \log n + d^5 2^d \log n)$ [23].

Given one of the algorithms for linear programming, the extreme points can be determined if one linear program is solved for each of the n points of P and, therefore, the extreme points of P can be enumerated in $O(n^2)$ time.

Seidel and Welzl present a randomized algorithm which improves the time complexity to $O(n^{3/2} \log n)$ expected running time in dimension 4 and to $O(n^{2-c_d})$ expected time in any fixed dimension $d \geq 5$, where $1/c_d = O(\lfloor d/2 \rfloor !^2)$ [20]. They use a randomized divide and conquer approach to partition the problem into suitable smaller subproblems, which are then solved by linear programming. Their result is further improved by Matousek and Schwarzkopf [17]. They present a randomized algorithm that determines the extreme points of a set of n points in d dimensions in $O(n^{2 - \frac{2}{1+\lfloor d/2 \rfloor} + \delta})$ expected time, for any fixed $\delta > 0$. Their approach has the disadvantage that the hidden constant is at least $d!$ since the algorithm works by recurring on smaller dimensions.

In the first part of this paper we present a simple deterministic algorithm for enumerating the extreme points of a finite set of n points in d dimensions. It is based on an interesting modification of the naive approach to determine the extreme points of P. We enumerate all the extreme points of a point set P of n points in d dimensions in $O(nm)$ time where m is the number of extreme points of P. If $m = o(n)$, then our method improves the previous best known deterministic complexity of $O(n^2)$ for enumerating extreme points in d dimensions where $d \geq 4$ [12]. In many real world applications, like for example a uniform distribution of the points in the d-dimensional unit cube, the number of expected extreme points is considerably less than n [4], in the example of the unit cube $O(\log^{d-1} n)$ [21].

A closely related problem to the enumeration of extreme points is the computation of the *convex layers* of a set P. Starting with the convex hull as the first layer, the ith layer is defined as the convex hull of the set of points remaining after removing all the points from the previous j layers with $j \leq i-1$. The *depth* of a point in P is i if it belongs to the ith convex layer.

The second part of this paper is concerned with the problem of computing the depth of each point p of P in d dimensions. The two dimensional problem was first considered by Overmars and van Leeuwen which provide an $O(n \log^2 n)$ time solution using a planar configuration maintenance technique [19]. This algorithm has been further refined by Chazelle and he has shown that the convex layers and, therefore, the depth of a set of n points can be computed in optimal $O(n \log n)$ time [6]. In 3-dimensional space Agarwal and Matousek give an $O(n^{1+\epsilon})$ time algorithm to compute the convex layers of a point set [1]. For dimensions $d > 3$, the problem has not yet been considered and the existing (naive) deterministic algorithm takes $O(n^3)$ time.

We show in Section 3 that our method of enumerating the extreme points can be extended to compute all the convex layers of P in $O(n^2)$ time. Hence, the depth of all the points of a finite set of n points in d dimensions can be computed in $O(n^2)$ time, thus solving an open problem posed by Edelsbrunner [12, Problem 8.11].

2 Extreme Points

This section concerns the enumeration of extreme points of a finite set P of n points in d dimensions where the dimension d is fixed. We are only interested in the case $d \geq 4$ and we assume that $n \geq d+1$ and that the points are in general position, i.e., no $d+1$ points lie in a common hyperplane. If it cannot be guaranteed that each hyperplane contains at most d points, then the general technique of *simulating simplicity* can be used which ensures that algorithms developed for points in general position produce a correct output even if the point set contains degenerate configurations [12, Section 9.4].

The problem of deciding whether a given point $p \in P$ is an extreme point is equivalent to the problem of determining whether $p \in conv(P \setminus \{p\})$. This problem can be dualized into a linear programming problem that involves n constraints and d variables [9]. Before we give our algorithm, we review the concepts of linear programming in relation to the computation of extreme points.

2.1 Linear Programming and Extreme points

A natural and frequently studied problem regarding convex polyhedra is the linear programming problem. We are given a set H of halfspaces in d dimensions. The halfspaces are called *linear constraints*. In addition, we are given a linear function f which is called the *objective function*. The tuple (H, f) is called a *linear program*. The linear programming problem is to determine a point of the polyhedron formed by the intersection of the given halfspaces at which the maximum of f is attained.

In order to see how linear programming can be employed to determine extreme points we make use of the concept of *duality*. Consider the geometric transform D which maps each point p of $I\!R^d \setminus \{\mathbf{0}\}$ into a halfspace $D(p) = \{x \in I\!R^d \mid <x, p> \ \leq 1\}$ ($<x, y>$ denotes the inner product of the two vectors x and y) and vice versa. We denote the halfspace $D(p)$ also by h_p. The point p and the halfspace h_p are called *duals* of each other (see Figure 1).

Let P be a set of n points $p_1, \ldots, p_n$ in d dimensions and H_P be the set $\{h_{p_1}, \ldots, h_{p_n}\}$ of the halfspaces that are dual to the points of P. If H_Q is a subset of H_P, then we assume in the following that Q is the subset of P such that $D(Q) = H_Q$.

We denote the polyhedron formed by the intersection of the halfspaces of H_P by $\Pi(H_P)$, i.e. $\Pi(H_P) = \bigcap H_P$. A halfspace that contains $\Pi(H_P)$ in its interior is called *redundant*. In other words, a halfspace is redundant if it can be discarded without affecting the intersection $\Pi(H_P)$. Otherwise, the halfspace is called *non-redundant*. The hyperplanes bounding non-redundant

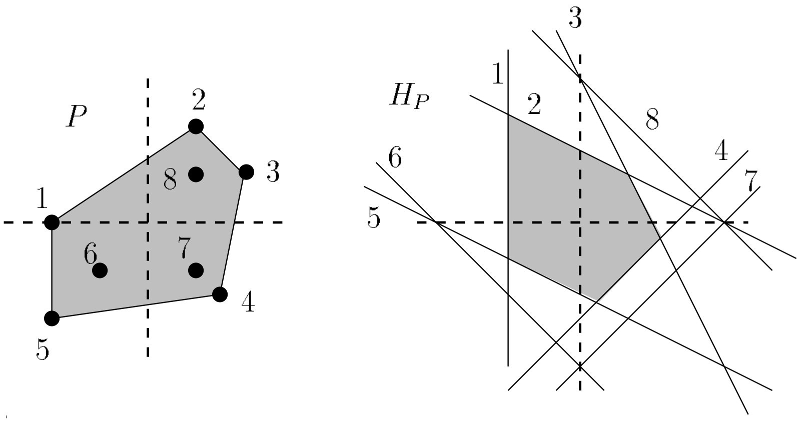

Figure 1: A point set P and its dual H_P.

halfspaces are called *bounding hyperplanes* of $\Pi(H_P)$.

If the origin belongs to $conv(P)$, then p in P is an extreme point of P if and only if h_p is a non-redundant hyperplane of H_P [12, Corollary 1.9]. If the origin does not belong to $conv(P)$, the $d+1$ points $p_1, \ldots, p_{d+1}$ span a d-simplex (since the points are in general position). If we translate P by the vector $-1/(d+1)\sum_{i=1}^{d+1} p_i$, then the convex hull of the translated point set $\overline{P}$ contains the origin while the extreme point of $\overline{P}$ are the translations of the extreme points of P. Since this transformation requires only $O(n)$ additional time we can assume in the following that the origin belongs to $conv(P)$ and to compute the extreme points of P it suffices to decide, for each halfspace h_p in H_P, if the halfspace h_p is redundant or not as we observed above.

In order to do so we make use of linear programming. Let the outer normal of the bounding hyperplane of h_p be denoted by n_p (note that $n_p = p$ if p is considered a vector). Now we consider the linear program (H_P, f_p) where $f_p(x) = <n_p, x>$ is the objective function. The problem is to determine the maximum value of the objective function on $\Pi(H_P)$ and also the set of d hyperplanes which determine the maximum value. We denote the value of a linear program with the objective function f_p and the constraints H_P by $f_p(H_P)$. We have the following lemma (see also [7]).

Lemma 2.1 *The halfspace h_p is non-redundant if and only if*

$$f_p(H_P) = f_p(H_{\{p\}}).$$

Proof: First note that $f_p(H_P) \leq f_p(H_{\{p\}})$ since $\Pi(H_P)$ is contained in h_p. To prove the "only if"-direction we assume that h_p is non-redundant. Hence, there is a least one point q_0 on the bounding hyperplane of h_p that belongs to $\Pi(H_P)$. Note that if q is a point on the bounding hyperplane of h_p, then $<n_p, q> = f_p(H_{\{p\}})$. This implies that $f_p(H_P) \geq <n_p, q_0> = f_p(H_{\{p\}})$ which proves one direction of the claim.

To see the other direction note that the equation $<n_p, x> = f_p(H_{\{p\}}) = f_p(H_P)$ defines a hyperplane which coincides with the bounding hyperplane of h_p. Hence, there is at least one point of $\Pi(H_P)$ on the bounding hyperplane of h_p. This implies that h_p is non-redundant. $\square$

In our algorithm we do not only test a halfspace for redundancy against H_P but also against a suitably chosen subset H_Q of H_P which has a lesser cardinality than H_P.

Corollary 2.2 *If h_p is a non-redundant halfspace of H_Q, then h_p is a non-redundant halfspace of H_P if and only if $f_p(H_P) = f_p(H_Q)$.*

Proof: The halfspace h_p is a non-redundant halfspace of H_P if and only if $f_p(H_P) = f_p(H_{\{p\}})$ by Lemma 2.1. Since h_p is a non-redundant halfspace of H_Q, the same lemma implies that $f_p(H_Q) = f_p(H_{\{p\}})$. Combining the two equations yields the lemma. $\square$

Hence, if h_p a non-redundant halfspace of H_Q but redundant in H_P, then $f_p(H_P) < f_p(H_Q) = f_p(H_{\{p\}})$.

All of the linear programming algorithms we mentioned before also return a *witness* v^* which is a point of $\Pi(H_P)$ with $f_p(H_P) = f_p(v^*)$ if they are applied to the linear program where the constraints are H_P and the objective function is f_p. This witness is crucial for our algorithm.

Lemma 2.3 *Let h_p be a halfspace that is non-redundant in H_Q and redundant in H_P. If v^* is a witness of the linear program (H_P, f_p) and $H_R \subseteq H_P$ is the set of halfspaces whose bounding hyperplane contains v^*, then*

- *there is at least one halfspace of H_R that is not contained in H_Q,*

- *all the halfspaces of* H_R *are non-redundant in* H_P*, and*
- $|H_R| \leq d$.

Proof: To see the first claim note that since h_p is non-redundant in H_Q but redundant in H_P, $f_p(H_P) < f_p(H_Q)$ by Corollary 2.2. Furthermore, since v^* is contained in the interior of the halfspaces $H_P \setminus H_R$, $f_p(H_P) = f_p(H_R)$. Hence, there is at least one halfspace of H_R that is not contained in H_Q.

To see the second claim just note that v^* is a point on the boundary of $\Pi(H_P)$ that is contained in boundary of each of the halfspaces in H_R. Hence, $\Pi(H_P)$ is not contained in the interior of the halfspaces $h_r \in H_R$.

The third claim of the lemma follows from the general position assumption on P. If more than d hyperplanes intersect in v^*, then there are more than d points of P contained in the hyperplane that is dual to v^*. $\square$

We make use of Lemma 2.3 to enumerate the extreme points of P in d-dimension. The main idea is to avoid testing the extremality of each point of P against the whole set P by using the set E of the extreme points that have been found so far as a filter. This reduces the complexity of the algorithm from $O(n^2)$ to $O(nm)$.

2.2 The Algorithm

We now present our method of enumerating extreme points of P. We choose $d+1$ linearly independent points such that each point has a maximum coordinate in one of the $2d$ axes-parallel directions. Initially the set E of extreme points of P is created from these $d+1$ points. Let us denote the set $P \setminus E$ by P'. We have to test for every point in P' whether it is an extreme point of P or not. Obviously, a point p in P' is not an extreme point of P if it is not an extreme point of $Q = E \cup \{p\}$, since Q is a subset of P. Therefore, it is sufficient to test the extremality of a point $p \in P'$ in P only if it is an extreme point of Q. In order to do this we first solve the linear program (H_Q, f_p). If $f_p(H_Q) = f_p(H_{\{p\}})$, then h_p is non-redundant in H_Q and p is not contained in $conv(E)$. Only then do we solve the larger linear program (H_P, f_p) which leads to the following two cases.

Case 1: $f_p(H_P) = f_p(H_{\{p\}})$ and p is an extreme point of P. In this case we can update E by adding p to E.

Case 2: $f_p(H_P) < f_p(H_{\{p\}})$ and p is not an extreme point of P. From the algorithm solving the linear program (H_P, f_p) we obtain a witness v^*. Let H_R be the set of halfspaces that contain v^* in their bounding hyperplane. By Lemma 2.3 $H_R \setminus H_E$ is a subset of the non-redundant halfspaces of H_P. Hence, if we set $E := E \cup R$, then we add only extreme points to E.

More precisely, our algorithm can be described as follows.

Algorithm *Enumeration of extreme points*
    $E := \{p_1, \ldots, p_{d+1}\}$;
    $P' := P \setminus E$;
    repeat
        let p be the next point in P';
        $P' := P' \setminus \{p\}$;
        $Q := E \cup \{p\}$;
(1)   solve the linear program (H_Q, f_p);
        if $f_p(H_Q) = f_p(H_{\{p\}})$
(2)   **then** (* p is an extreme point of $E \cup \{p\}$ *)
                solve the linear program (H_P, f_p);
                let v^* be a witness of the solution;
                if $f_p(H_P) = f_p(H_{\{p\}})$
                    then (* p is an extreme point of P *)
                        $E := E \cup \{p\}$;
                    else (* p is not an extreme point of P *)
                        let H_R be the subset of halfspaces of H_P
                        whose bounding hyperplane contains v^*;
                        $E := E \cup R$;
                        $P' := P' \setminus R$;
                end if
        end if
    until $P' = \emptyset$;

The correctness of the algorithm follows immediately from the case analysis above.

Now, we analyze the time complexity of our algorithm. Clearly, the execution time of the algorithm is dominated by the time to solve the linear programming problems in Steps (1) and (2). First consider Step (1). The linear program (H_Q, f_p) is solved for each point p in P. The maximum cardinality of H_Q is $|Q| = |E| + 1 \leq m + 1$ if m is the number of extreme points of P. Hence, the total time used for Step (1) is bounded by $O(nm)$.

Now consider Step (2). Step (2) is called for each point p of P' that is an extreme point of E. We claim that each time Step (2) is executed the cardinality of E increases at least by one. If p is an extreme point of P, then p is added to E and the claim obviously holds. If p is not an extreme point of E and v^* a witness of the value of the linear program (H_P, f_p), then there is at least one halfspace of H_P that contains v^* in its bounding hyperplane and that is not contained in H_E by Lemma 2.3. Clearly, all the halfspaces that contain v^* in its bounding hyperplane can be found in time $O(dn)$. Since all the points that correspond to halfspaces whose bounding hyperplane contains v^* are added to E, the cardinality of E increases by at least one. Hence, a linear program of the form (H_P, f_p) is solved at most $|E| = m$ times which again leads to a time bound of $O(mn)$. Hence, we have shown the following theorem.

Theorem 2.4 *Given a set of n points in d dimensions, the extreme points can be computed in $O(nm)$ time where m is the number of extreme points of P.*

3 Convex layers

We show in this section that we can efficiently compute the depth of each point in a finite set of n points in d dimensions. The *depth* of a point $p \in P$ is 1 if p is an extreme point. Otherwise, the depth of p in P is one plus the depth of p in P minus the set of extreme points. Let E_1 be the set of extreme points of P. It determines the first convex layer of P. The set E_2 of extreme points of $P \setminus E_1$ is the second convex layer of P. In general E_i, the i the convex layer, is the set of extreme points of $P \setminus (E_1 \cup E_2 \cup \ldots \cup E_{i-1})$.

After the discussion of the algorithm of enumerating the extreme points in the previous section, the procedure for constructing all the convex layers and hence finding the depth of each point of the given set P of finite number of points in d-dimensions is now straightforward.

The first convex layer E_1 of the given set P of n points can be determined in $O(nm_1)$ time where m_1 is the number of points of E_1. The ith convex layer E_i can be obtained in time $O(nm_i)$ after removing points from the first $i-1$ layers of P, where $|E_i| = m_i$. In each iteration we determine only the extreme points of the set which is obtained from discarding all the extreme points which have been found in the previous iterations. We continue the procedure until we obtain all the convex layers. This procedure for computing all the convex layers of P is known as "shelling" or "peeling ".

Theorem 3.1 *One can determine the convex layers of a finite set of n points in d dimensions by shelling in $O(n^2)$ time.*

Proof: Determining E_i takes $O(nm_i)$ time where $|E_i| = m_i$. Hence, the total time complexity for determining all the layers is $\sum m_i(n-(m_1+m_2+\cdots+m_{i-1})) \leq \sum m_i n \leq n \sum m_i$. Since each point is contained in exactly one layer, $\sum m_i = n$ and the sum is bounded by $O(n^2)$. $\square$

Corollary 3.2 *The depth of all points in a set of n points in d-dimensions can be computed in time $O(n^2)$.*

4 Conclusions

We have presented a new algorithm for computing the extreme points of a finite set of n points in d-dimensions in $O(n)$ space and $O(nm)$ time where m is the number of extreme points of the set. Using this result, we have achieved an improvement for computing the depth of all the points of a finite set of points in dimensions $d \geq 4$. The method takes $O(n^2)$ time which improves the $O(n^3)$ complexity of the previouly best deterministic algorithm. It also improves the best known randomized algorithm with expected running time $O(n^{3-\frac{2}{1+\lfloor d/2 \rfloor}+\delta})$ (for any fixed $\delta > 0$).

One of the main open questions is whether there are faster algorithms than $O(n^2)$ to compute the depth of all points.

References

[1] P. Agarwal and J. Matousek. Ray shooting and parametric search. In *Proc. 21st ACM Symp. on Theory of Computing*, pages 517–526, 1992.

[2] S. G. Akl and G. T. Toussaint. A fast convex hull algorithm. *Information Processing Letters*, 7(5):219–222, 1978.

[3] B. K. Bhattacharya and G. T. Toussaint. Time- and storage-efficient implementation of an optimal convex hull algorithm. *Image Vision Computing*, 1:140–144, 1983.

[4] K. H. Borgwardt, N. Gaffke, M. Jünger, and G. Reinelt. Computing the convex hull in the euclidean plane in linear expected time. In *Applied Geometry and Discrete Mathematics THE VICTOR KLEE FESTSCHRIFT*, pages 91–107. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Volume 4, 1991.

[5] B. Chazelle. An optimal convex hull algorithm and new results on cuttings. In *Proc. 32nd IEEE Symp. on Foundations of Computer Science*, pages 29–38, 1991.

[6] B. Chazelle. Optimal algorithms for computing depths and layers. In *Proc. 21st Allerton Conference on Communication, Control and Computation*, pages 427–436, 1983.

[7] V. Chvátal. *Linear Programming.* W. H. Freeman, New York, NY, 1983.

[8] K. L. Clarkson. A Las Vegas algorithm for linear programming when the dimension is small. In *Proc. 29th IEEE Symp. on Foundations of Computer Science*, pages 452–456, 1988.

[9] D. P. Dobkin and S. P. Reiss. The complexity of linear programming. *Theoretical Computer Science*, 11:1-18, 1980.

[10] M. E. Dyer. Linear time algorithms for two- and three-variable linear programs. *SIAM Journal of Computing*, 13:31–45, 1984.

[11] H. Edelsbrunner and W. Shi. An $O(n \log^2 h)$ time algorithm for the three-dimensional convex hull problem. *SIAM Journal of Computing*, 20:259–277, 1991.

[12] H. Edelsbrunner. *Algorithms in Combinatorial Geometry.* Springer Verlag, EATCS Monographs on Theoretical Computer Science Edition, 1987.

[13] R. L. Graham. An efficient algorithm for determining the convex hull of a finite planar set. *Information Processing Letters*, 1:132–133, 1972.

[14] G. H. Johansen and C. Gram. A simple algorithm for building the 3-d convex hull. *BIT*, 23:146–160, 1983.

[15] M. Kallay. Convex hull made easy. *Information Processing Letters*, 22:161-163, 1986.

[16] D. G. Kirkpatrick and R. Seidel. The ultimate planar convex hull algorithm? *SIAM Journal of Computing*, 15:287–299, 1986.

[17] J. Matousek and O. Schwarzkopf. Linear optimization queries. In *Proc. 8th ACM Symp. on Computational Geometry*, pages 16–25, 1992.

[18] N. Megiddo. Linear programming in linear time when the dimension is fixed. *Journal of the ACM*, 31:114–127, 1984.

[19] M. Overmars and J. van Leeuwen. Maintenance of configurations in the plane. *Journal of Computer and System Sciences*, 23:166–204, 1981.

[20] E. Welzl and R. Seidel. referred to in [17], p. 25.

[21] A. Rényi and R. Sulanke. Über die konvexe Hülle von n zufällig gewählten Punkten. *Zeitschrift für Wahrscheinlichkeitstheorie und Verwandte Gebiete*, 2:75–84, 1963.

[22] R. Seidel. Small-dimensional linear programming and convex hulls made easy. *Discrete & Computational Geometry*, 6:423–434, 1991.

[23] M. Sharir and E. Welzl. A combinatorial bound for linear programming and related problems. In *Proc. 9th Symp. on Theoretical Aspects Computer Science*, LNCS 577, pages 569–579. Springer-Verlag, 1992.