More Output-Sensitive Geometric Algorithms

extended abstract

Kenneth L. Clarkson
AT&T Bell Laboratories
Murray Hill, New Jersey 07974
e-mail: clarkson@research.att.com

Abstract

A simple idea for speeding up the computation of extrema of a partially ordered set turns out to have a number of interesting applications in geometric algorithms; the resulting algorithms generally replace an appearance of the input size n in the running time by an output size $A \leq n$. In particular, the A coordinate-wise minima of a set of n points in R^d can be found by an algorithm needing $O(nA)$ time. Given n points uniformly distributed in the unit square, the algorithm needs $n+O(n^{5/8})$ point comparisons on average. Given a set of n points in R^d, another algorithm can find its A extreme points in $O(nA)$ time. Thinning for nearest-neighbor classification can be done in time $O(n \log n) \sum_i A_i n_i$, finding the A_i irredundant points among n_i points for each class i, where $n = \sum_i n_i$ is the total number of input points. This sharpens a more obvious $O(n^3)$ algorithm, which is also given here. Another algorithm is given that needs $O(n)$ space to compute the convex hull of n points in $O(nA)$ time. Finally, a new randomized algorithm finds the convex hull of n points in $O(n \log A)$ expected time, under the condition that a random subset of the points of size r has expected hull complexity $O(r)$. All but the last of these algorithms has polynomial dependence on the dimension d, except possibly for linear programming.

1 Introduction

This paper describes several output-sensitive geometric algorithms, all inspired by a simple algorithm for finding minima of partially ordered sets. The minima problem: given a set S of n elements with partial order $\prec$ on its elements, find the set $F \subset S$ of minimal elements, so that no $e \in F$ has $g \prec e$ for any $g \in S$. The algorithm is this: process the elements of S in turn, maintaining a subset E of F. For each $p \in S$, check if $e \prec p$ for some $e \in E$; if so, then p is not minimal and can be discarded for further consideration in the algorithm Otherwise, although p is not necessarily minimal, it can be used to find a minimal element of S that is not in E, by looping through $S \setminus E$, maintaining $t \in S$ with $t \preceq p$ and with no $t' \in S \setminus E$ having $t' \prec t$.

The algorithm is described in C in Appendix 1.

This algorithm needs at most $2nA$ comparisons to find the A minima: each element of S is compared with the elements of E, which has size no more than A. When an element p is used to find a new member of E, n or fewer comparisons are done, but this occurs at most A times; hence no more than $2nA$ comparisons are required overall.

The algorithmic idea here is certainly trivial, and indeed there are other methods to obtain such a running time: frequently a total order $<$ on S is known that is a "topological sort" on S, so that $p \prec q$ implies $p < q$. If S is sorted with $<$, then no element q after p in the sorted order can have $q \prec p$; hence a similar algorithm can be used, but the step of finding a minimal $t \in S \setminus E$ with $t \preceq p$ can be skipped.

However, the total order $<$ may not be available, and sorting by it may be slower; it is always more complicated. In §2, the above algorithm is analyzed when S is a set of points in the plane, uniformly distributed in a square, and the partial order is coordinate-wise dominance. An earlier, more complicated algorithm of Bentley *et al.*[2] was shown by Golin[6] to need an expected $n+o(n)$ comparisons (and the comparison time dominates the running time). The new algorithm is shown here to have a comparable expected performance, in addition to the $O(nA)$ worst-case bound. The earlier algorithm can require $\Omega(n^2)$ work in the worst case, even when $A = 1$.

Moreover, the idea of the new algorithm also applies to finding the extreme points of a set S of n points in R^d. An extreme point of S is a vertex of its convex hull; such a point cannot be expressed as a convex combination of the other points of S. Moreover, an extreme point e has the property that there is a witness vector n such that $n \cdot e > n \cdot q$ for all other $q \in S$.

The algorithm here is as follows: process the points of S in turn, maintaining a set $E \subset S$ of extreme points. Given $p \in S$, it is possible in $O(|E|) = O(A)$ time, using linear programming, to either show that p is a convex combination of points of E, or find a witness vector n for p, so that $n \cdot p > n \cdot q$ for all $q \in E$. While p is not necessarily an extreme point of S, one can obviously find the point $p' \in S$ that maximizes $n \cdot p'$, in $O(n)$ time. Such a point is extremal, and can be added to E; note that it cannot already be in E. As before, each point of S requires $O(A)$ work, except for A points that yield extreme points, and these need $O(n)$ work. Hence $O(nA)$ work is required overall.

The main surprise for this algorithm is perhaps that it has apparently not been reported before. It is worth remarking, however, that the algorithm is polynomial in d, and hence likely much faster for large d than many algorithms that compute the full convex hull; sometimes only the vertices of the hull are wanted. In §3, the recursive application of this idea yields an algorithm for convex hulls that needs $O(nA)$ time for a hull with A faces (of all dimensions).

The algorithm uses $O(n)+O(d^2)$ space. This performance is roughly comparable to the algorithms of Rote or Avis and Fukuda[11, 1]; moreover, the algorithm's cost per output face can be substantially smaller than $\Omega(n)$.

By combining current technology for extreme points [9, 7] and for half-space range queries[8], it is possible to obtain an algorithm that requires $O((nA)^{1-c_d})$, where $c_d = \Omega(1/d)$.

In §4, the basic idea above is combined with a randomized incremental algorithm for convex hulls to obtain an algorithm that requires $O(n \log A)$ time to compute the convex hull of a set of n points in R^d, under the following conditions: the expected number of facets of the hull of a random subset of r points of S is $O(r)$, for all r, and the hull of S has A facets. The former condition always holds for $d \leq 3$, and frequently does for many natural distributions. The algorithm is somewhat simpler than one previously given for $d = 3$ with the same complexity.[4] (Like the earlier one, this one also can be "derandomized" for $d \leq 3$ to obtain a deterministic algorithm with the same asymptotic complexity.[3]) This result is perhaps best interpreted as a rough theoretical justification for algorithms like Quickhull[5, 10], which find extreme points on their way to computing the hull.

As further evidence of the interest of this approach, §5 gives an output-sensitive algorithm for the problem of *thinning* a point set for nearest-neighbor (NN) classification. Here we are given a set S of sites (points) in R^d, and each site is given a class number from 1 to k. A thinning problem motivated by nearest-neighbor (NN) pattern classification is to find all sites $p \in S$ whose Delaunay neighbors all have the same class as p. Such a site is redundant for purposes of NN classification, and so its removal from S can speed up classification without affecting accuracy. The algorithm given here can do this thinning in $O(n \log n) \sum_i A_i n_i$ time, finding the A_i irredundant points among n_i points for each class i. The time is polynomial in d, as is a necessity for the high-dimensional data used in NN classification.

The final section gives some concluding remarks.

2 Analysis for planar points

This section analyzes the minima algorithm of §1, assuming that the input points are uniformly distributed in a square. (The results hold for any distribution for which the x and y coordinates of the input points are independent random variables.) Here most of the work is done in finding the first minimal point, when E is empty: starting with t assigned to the first input site (point), the algorithm loops through the remaining sites p of S; if $t \prec p$, p can be discarded as a minimum. If $p \prec t$, the point t can be discarded as a minimum and p becomes the new t. This section will show that $t \prec p$ for all but $O(n^{5/8})$ such comparisons, on average, and that subsequent work is $O(n^{5/8})$, so that the total expected number of comparisons is $n + O(n^{5/8})$; this is not too far from Golin's analysis of the algorithm of Bentley and others.[6, 2]

Observe that t takes on a series of values t_k, $k \geq 0$, with $t_i \prec t_j$ for $i > j$. We can bound the expected number of wasted comparisons (with points incomparable to t) by bounding the expected number of these for each t_k, and summing these for $k \geq 0$.

When t_k is a point (α, β) with $0 \leq \alpha, \beta \leq 1$, the expected number of comparisons until some $t_{k+1} \prec t_k$ is found is $\min\{n, 1/\alpha\beta\}$. During that period, the expected number of wasted comparisons is $(\alpha + \beta - \alpha\beta)/\min\{n, 1/\alpha\beta\}$. We need a bound on the expectation of this quantity, given that $t = t_k$.

The following lemmas will be useful.

Lemma 1 *The probability density function of t_k is $f_k(\alpha, \beta) \equiv \ln(1/\alpha)^k \ln(1/\beta)^k / k!^2$.*

Proof. Omitted. □

Lemma 2

$$\int \alpha^{j-1} \ln(1/\alpha)^k = k! \frac{\alpha^j}{j^{k+1}} \sum_{0 \leq i \leq k} \frac{j^i \ln(1/\alpha)^i}{i!},$$

for $j \neq 0$, and

$$\int \frac{\ln(1/\alpha)^k}{\alpha} = \frac{\ln(1/\alpha)^{k+1}}{\alpha}.$$

Proof. Omitted. □

Note that the sum converges to $1/\alpha^j$ as $k \to \infty$, and the summand has roughly that maximum value, at $i = j\ln(1/\alpha)$,

Theorem 3 *The expected wasted work in finding the first element of E is $O(n^{5/8})$.*

Before proving this theorem, the bound on the running time that it implies:

Theorem 4 *The total expected number of comparisons of the algorithm is $n + O(n^{5/8})$.*

Proof. (Sketch) It's not hard to show that with high probability, k will get large enough that with high probability, both coordinates of the first element of E will be less than $(\log n)/\sqrt{n}$. This implies that with high probability, the number of points that survive comparison with that first element is $\sqrt{n} \log^{O(1)} n$, giving a bound of $A\sqrt{n} \log^{O(1)} n$ on the remaining work, with high probability. Since the expected value of A is $O(\log n)$, the expected work after finding the first element of E is $O(n^{5/8})$. □

Proof. (of Theorem 3) The proof is divided into two main cases, depending on the location of t_k, and the corresponding bound for the expected number of points compared to t_k.

Case 1: $\alpha\beta \geq 1/n$. Consider first the region of the unit square with $\alpha\beta \geq 1/n$, so that t_k is compared with $1/\alpha\beta$ points on average, and the expected wasted work for t_k in this region is no more than

$$\int_{1/n}^{1}\int_{1/n\alpha}^{1}\left(\frac{1}{\alpha}+\frac{1}{\beta}\right)\frac{\ln(1/\alpha)^k \ln(1/\beta)^k}{k!^2}\,d\beta\,d\alpha.$$

By symmetry we can replace $1/\alpha + 1/\beta$ by $2/\beta$, and so by using Lemma 2 obtain, within a constant factor,

$$\int_{1/n}^{1}\frac{\ln(1/\alpha)^k \ln(n\alpha)^{k+1}}{k!(k+1)!}\,d\alpha.$$

With a change of variables $\alpha = s^{\gamma-1}$, where $s \equiv \sqrt{n}$, we obtain

$$\frac{(\ln s)^{2k+1}}{sk!(k+1)!}\int_{-1}^{1}(1-\gamma)^k(1+\gamma)^{k+1}s^{\gamma}\,d\gamma.$$

Using $1+x \leq e^x$, the integral is no more than

$$2\int_{-1}^{1}\exp(k\ln(1-\gamma^2)+\gamma\ln s)\,d\gamma.$$

The integral is no more than twice the maximum value of the integrand; maximizing this for given k, and then maximizing the whole expression with respect to k, the desired bound is obtained.

Case 2: $\alpha\beta \leq 1/n$. For the portion of the unit square with α or β less than $1/n$, we may bound the expected waste by twice

$$\int_0^1\int_0^{1/n} n(\alpha+\beta)\frac{\ln(1/\alpha)^k \ln(1/\beta)^k}{k!^2}\,d\beta\,d\alpha.$$

This is n times

$$\int_0^1 \frac{\ln(1/\beta)^k}{k!}\,d\beta \int_0^{1/n}\alpha\frac{\ln(1/\alpha)^k}{k!}\,d\alpha + \int_0^1\frac{\beta\ln(1/\beta)^k}{k!}\,d\beta\int_0^{1/n}\frac{\ln(1/\alpha)^k}{k!}\,d\alpha.$$

Using Lemma 2, we seek n times

$$1\cdot\frac{(1/n)^2}{2^{k+1}}\sum_{0\leq i\leq k}\frac{(2\ln n)^i}{i!}+\frac{1}{2^{k+1}}(1/n)\sum_{0\leq i\leq k}\frac{(\ln n)^i}{i!}.$$

We wish to bound the sum of these values over all k, so by summing over $k \geq i$ and then over i, we obtain

$$\frac{1}{n}\exp(\ln n)+\exp\left((\ln n)/2\right) = 1+\sqrt{n}.$$

Now consider $\alpha\beta \le 1/n$ and $\alpha, \beta \ge 1/n$. By symmetry, it's enough to consider

$$\int_{1/n}^{1/\sqrt{n}} \int_{\alpha}^{1/n\alpha} n(\alpha+\beta) f_k(\alpha,\beta)\, d\beta\, d\alpha,$$

and again with symmetry, this is no more than

$$\int_{1/n}^{1} \int_{0}^{1/n\alpha} n\alpha f_k(\alpha,\beta)\, d\beta\, d\alpha.$$

The expected work for $t_k = (\alpha,\beta)$ with $\alpha\beta \le 1/n$ and $\alpha,\beta \ge 1/n$ is therefore within a constant factor of

$$\int_{1/n}^{1} \int_{0}^{1/n\alpha} n\alpha \frac{\ln(1/\alpha)^k \ln(1/\beta)^k}{k!^2}\, d\beta\, d\alpha.$$

Using Lemma 2, we have

$$\sum_{0\le i\le k} \int_{1/n}^{1} \frac{\ln(1/\alpha)^k (\ln(n\alpha)^i}{i!k!}\, d\alpha,$$

With a change of variables $\alpha = s^{\gamma-1}$ as in Case 1, where $s \equiv \sqrt{n}$, we obtain

$$\frac{1}{s} \sum_{0\le i\le k} \frac{(\ln s)^{i+k+1}}{i!k!} \int_{-1}^{1} (1-\gamma)^k (1+\gamma)^i s^\gamma\, d\gamma. \tag{1}$$

First consider the integral between -1 and 0: push the integral outside to obtain

$$\frac{\ln s}{s} \int_{-1}^{0} \sum_{0\le i\le k} \frac{(\ln s)^{i+k}}{i!k!} (1-\gamma)^k (1+\gamma)^i s^\gamma\, d\gamma,$$

which is no more than

$$\frac{\ln s}{s} \int_{-1}^{0} s^{1-\gamma} s^{1+\gamma} s^\gamma\, d\gamma,$$

which is less than $s = \sqrt{n}$ upon evaluating the integral.

Now for the integral between 0 and 1, and three subcases:

$k \ge 2\ln s$**.** Here we bound the sum over i by $s^{1+\gamma}$, and bound $(1-\gamma)^k$ by $e^{-\gamma k} \le s^{-2\gamma}$, so part of (1) is bounded by

$$\frac{\ln s}{s} \sum_{k\ge 2\ln s} \frac{(\ln s)^k}{k!} \int_0^1 s\, d\gamma < s.$$

$k \le \ln s$**.** Here the sum over i is bounded by k times its maximum value at the $i = k$ term, with the bound

$$\frac{1}{s} \sum_{0\le k\le \ln s} \frac{(\ln s)^{2k+2}}{k!k!} \int_0^1 (1-\gamma^2)^k s^\gamma\, d\gamma,$$

which is bounded by $O(n^{5/8})$, as in Case 1.

$\ln s \leq k \leq 2 \ln s$. Rearrange the subsum of (1) to $1/s$ times

$$\sum_{k=\ln s}^{2\ln s} \frac{(\ln s)^{k+1}}{k!} \int_0^1 (1-\gamma)^k \sum_{0 \leq i \leq k} (1+\gamma)^i \frac{(\ln s)^i}{i!} s^\gamma \, d\gamma, \tag{2}$$

and consider first the portion of the integral with $\gamma \leq \epsilon$, where $\epsilon \equiv (k/\ln s) - 1$. We bound the sum over i by $s^{1+\gamma}$, obtaining

$$\sum_{\ln s \leq k \leq 2\ln s} \frac{(\ln s)^{k+1}}{k!} \int_0^\epsilon s^{(1+\epsilon)\ln(1-\gamma)+2\gamma} \, d\gamma,$$

which is $O(1)$ times

$$\max_{0 \leq \gamma \leq \epsilon \leq 1} (\ln s)^2 s^{(1+\epsilon)-(1+\epsilon)\log(1+\epsilon)} s^{(1+\epsilon)\ln(1-\gamma)+2\gamma}.$$

For given $\epsilon \geq 1/3$, this is maximized by $\gamma = (1-\epsilon)/2$, and by $\gamma = \epsilon$ for $\epsilon \leq 1/3$. In either case, the expression is maximized at $\epsilon = 1/3$, and gives $(\ln s)^2 s^{2-(4/3)\ln 2} = O(n^{5/8})$.

Finally, for the portion of the integral in (2) when $\gamma \geq \epsilon$, we may bound the sum over i by $\ln s$ times the maximum value of the summand, at $i = k$, to obtain an expression as in the subcase $k \leq \ln s$, or Case 1, and also obtain the bound $O(n^{5/8})$. $\square$

3 Space-efficient convex hulls

The limiting resource for finding convex hulls in higher dimensions is often not time, but space. A large space requirement also affects computation time if the algorithm has little locality of reference, as may be true for many convex hull algorithms.

Avis and Fukuda gave an algorithm that computes the convex hull of a set S of n points in R^d, and needs little more space than is required to represent the input. Their algorithm needs $O(nA)$ time to output all feasible bases of the (projective) dual polytope. The number of such bases is the same as the number of vertices of the dual polytope, or facets of the hull, if the hull polytope is simplicial. However, for degenerate data, the number of bases may be much larger than the number of facets of the hull. Rote gave an algorithm that removes the dependence on the number of bases, with a time bound of $O(nA)$ for computing the facets of the hull. This section gives an algorithm with performance similar to Rote's, and with some similar characteristics, but derived in a quite different way.

Following discussion of the algorithm, there is some speculation as to the performance of the new algorithm relative to the earlier ones.

The algorithm here is based simply on the application of the basic algorithm of §1, applied recursively in the dimension. The algorithm outputs the faces of the convex hull conv S in lexicographic order, where the faces are described by a list of their incident vertices, sorted in increasing order. (Each input point is arbitrarily given a unique number between 1 and n.)

The algorithm is perhaps best described in the dual setting of the computation of the intersection $\mathcal{P}$ of a collection of halfspaces. Here an extreme point of S corresponds dually to an irredundant halfspace for $\mathcal{P}$: a halfspace for which the intersection of its boundary plane h with $\mathcal{P}$ is a polytope of dimension one lower, so that $h \cap \mathcal{P}$ is a facet of $\mathcal{P}$.

The algorithm first finds the irredundant halfspaces of $\mathcal{P}$, corresponding to the extreme points of S. These halfspaces can be found in time $O(n)$ per halfspace: for each halfspace, solve the linear programming problem of checking that there is a point on its boundary that is contained in the interior of every other halfspace.

Now the algorithm picks the lowest numbered irredundant halfspace, call it 1, and recursively computes the facet corresponding to halfspace 1. Having done so, it recursively computes the facet corresponding to the next higher-numbered halfspace, say 2. However, consider the ridge, if it exists, that is the $(d-2)$-dimensional polytope contained in the intersection of the boundaries of halfspaces 1 and 2. Since that ridge was reported recursively when halfspace 1 was processed, it need not be reported (or processed recursively) when processing halfspace 2.

In the general case, the algorithm is considering a face that is contained in the flat (affine subspace) that is the intersection of some set T of j boundary planes, and is finding the polytope $\mathcal{P}'$ that the intersection of the remaining halfspaces with that flat. Such a problem can be viewed, by a change of variables, as a lower-dimensional version of the original problem. The algorithm finds the irredundant halfspaces whose intersection with the flat gives $\mathcal{P}'$, and then recursively computes the facets corresponding to halfspaces whose numbers are higher than any of those in T, doing so in increasing order of those numbers. Thus a lexicographic order is preserved, and each j-tuple of halfspaces that yields a $(d-j)$-face is reported once.

With degenerate input, however, it could occur that a j-face, say a 0-face or vertex, can be represented by more than one distinct d-tuple of halfspaces. Here we can use lexicography to remove this redundancy: we seek to report only the lexicographically first j-tuple that yields a given face. Degeneracy will be manifested during the testing to find irredundant halfspaces: the test is given a flat w and a collection of halfspaces, and for each halfspace with boundary h, checks if there is a point in $h \cap w$ that is in the interior of every halfspace of the collection. In the degenerate situation, it may be that there is no such point, but there is a point in $h \cap w$ that is either contained in the interior or is on the boundary of each halfspace. If all the latter halfspaces have numbers higher than h's, the test of h for irredundancy will be considered "passed," but otherwise not. This should result in nonredundant reporting of faces in the degenerate case.

When the collection of halfspaces is found that is irredundant in determining a given face, $O(n)$ work is done per halfspace, using the trick of §1. While only facets of that face that are determined by higher-numbered halfspaces are recursively computed, nonetheless $O(n)$ is paid even for lower-numbered halfspaces. However, the work for a given face determined by j halfspaces is wasted in this way at most j times, so the overall work remains $O(nA)$ for A faces reported.

While this algorithm is comparable asymptotically to Rote's, it is difficult to predict with confidence its relative speed in practice. On the one hand, the algorithm reports all faces of the intersection, and not just the vertices, so there would seem to be a $d!$ penalty relative to Rote's. Also, the algorithm does linear programming at each step, which Rote's wouldn't necessarily do at all if there were no degeneracies. On the other hand, a halfspace that is redundant for a face is redundant for the facets of that face, and so there is a potential for a substantial reduction in the number of halfspaces in the test performed for a face. For example, in the Voronoi diagram of sites in R^3 uniformly distributed in a cute, a given site has about 15 Delaunay neighbors on average, each corresponding to a Voronoi polygon. The Voronoi edges and vertices for a site can be found using only those 15 neighbors, and so the work is dominated by the time to find the Delaunay neighbors. There are about $7n$ Voronoi vertices, on average, however, so the "leverage" for Rote's algorithm is 2 here and not $4! = 24$, as might be expected.

4 Output-sensitive convex hulls

This section gives an algorithm for finding the convex hull of a set S of n points in R^d. The algorithm combines a randomized incremental approach with the incremental computation of a set of extreme points of S. The randomized incremental portion allows the extreme points to be found quickly; the extreme points allow the randomized incremental algorithm to quit early when the number A of extreme points is small.

The discussion below assumes that $2d \log A < \log n$; otherwise the randomized incremental algorithm by itself gives the desired expected bound.

The algorithm can be sketched as follows: adding points at random one by one to a random subset R of S, maintain the convex hull of R, together with auxiliary *conflict graph* information, as discussed in [4]. (This auxiliary information lists, for each facet of $\operatorname{conv} R$, those sites of $S \setminus R$ that see the facet, and lists for each site the facets it sees.) Also maintain a set E of extreme points of S, starting with some initial set of fixed size. Build E incrementally, adding a point to it when R is first of size r at least $|E| \ln |E|$, so that a point in added to E, (and $|E|$ increases) about every $\log r$ steps. Mark the points of

S *active* initially. To add a point to E, pick an active point p of S, and check if p is outside the convex hull of E. If not, mark p as not active. If so, find a *witness vector* n for p, such that $n \cdot p > n \cdot q$ for all $q \in E$. (Finding such a witness, or proving that none exists because $p \in \operatorname{conv} E$, is discussed below.) Given the witness vector n, find d facets of $\operatorname{conv} R$ such that n is a positive linear combination of the outward normals of these facets. This implies that any point $p \in S$ that maximizes $n \cdot p$ must see one of these d facets, and so is in one of the conflict lists for these facets. By examining these lists, such a $p \in S$ can be found and added to E.

A witness vector, that proves that a point p is outside $\operatorname{conv} E$, can be found in $O(\log A)$ time, assuming appropriate preprocessing of E in $A^{O(1)}$ time. For $d = 3$, a planar point location data structure or Dobkin and Kirkpatrick's hierarchical decomposition can be used for this purpose. In higher dimensions, the best algorithm for this is due to Matoušek and Schwarzkopf[7].

To bound the running time of this algorithm, note that a point is tested with respect to containment in $\operatorname{conv} E$ at most once, and as just mentioned, this requires $O(\log A)$ time. To find d facets of $\operatorname{conv} R$ whose normals contain a given vector requires time at most $O(r^{\lfloor d/2 \rfloor}) = O(A^d)$, and this is done A times. The conflict lists of these d facets contain $O(n/r)\log r = O(n/|E|)$ sites, with probability $1 - 1/A^{\Omega(1)}$, using standard random sampling arguments, and so the total time to search these lists is no more than $O(n)\sum_{1 \leq j < A} 1/j = O(n \log A)$. The maintenance of $\operatorname{conv} R$ and its conflict lists, up to size $r \leq A \log A$, requires $O(n \log A)$ expected time,[4] and so $O(n \log A)$ is needed overall.

5 Thinning

In *nearest neighbor classification* a set S of n points in R^d is given, with each site $p \in S$ having a class number c_p. Given a query point q, it is classified by giving it the same class number as that of the closest site to it. It may be that some site is redundant for this purpose, because no point in R^d would change classification upon removal of that site from S. The operation of removing such redundant sites is called *thinning*. (In particular, thinning that is "decision boundary preserving.")

A site p is redundant if and all only if all its Delaunay neighbors have the same class number as p does; since the Delaunay neighbors of a given site can be found in $O(n^2)$ time using standard methods with linear programming, thinning can be done in $O(n^3)$ time, with polynomial dependence on the dimension (except perhaps for linear programming). Suppose the number of sites with the same class as p is n_p, and the number of sites of class i is n_i. Then since the appropriate linear programming problem for site p requires only n_p constraints, the time bound for thinning can be better expressed as $\sum_p O(n)n_p = O(n)\sum_i n_i^2$; when there are k equal-sized classes, this represents an improvement by a factor of k in running time over $O(n^3)$.

These observations apparently are new.[13, 12]

Suppose that there are A_p irredundant sites the same class as p, and A_i irredundant sites of class i. This section will next give a thinning algorithm needing $O(n)\sum_i A_i n_i \log n$ time. Except possibly for linear programming, the algorithm has polynomial dependence on the dimension.

The algorithm applies a similar trick to thinning as this paper has used for extreme points and minima: for each class, maintain a set E_i of irredundant sites of that class, and add sites to E_i one by one. The algorithm begins with each E_i empty, and proceeds as follows: pick a site p, and check if the current E_i proves redundant. To do this check, take every site q with $c_q \neq c_p$, and check if there is any point x in R^d that is equidistant from p and q and closer to both than to any sites in E_i. For given p and q, this can be done via linear programming with $|E_i|$ constraints. If there is no such q, then p is redundant. If there is such a q and x, the sites in E_i do not show that p is redundant. This does not mean that p is irredundant, but it does mean that p and x can be used to find an irredundant site of class i that is not in E_i. To find such a site p', consider the line segments $\overline{px}$ and $\overline{xq}$. All points on those segments are closer to p or q than to any sites in E_i. Moreover, in walking from p to q along those segments, the classifications of points on the segments will begin as c_p and end as c_q. This means there is some point x' on those segments that is equidistant from some sites p' and q', and closer to those sites than any others, and with $c_{p'} = c_p \neq c_{q'}$. The site p' is therefore irredundant, and by construction cannot be in E_i, since the distance from x' to p' must be less than the distance of x' to any site in E_i. Hence p' can be added to E_i.

A key step here is to find x'; this can be done by computing the intersection of the Voronoi diagram of S with the segments $\overline{px}$ and $\overline{xq}$. These intersections can be found in $O(n \log n)$ time, however, since the squared distance of a point x on a line ℓ to a point p can be expressed as the sum of squared distance of p to ℓ and the squared distance of x to the projection of p to ℓ. This implies that the intersection of a Voronoi diagram with a line can be computed as an additively weighted Voronoi diagram on that line.

6 Concluding remarks

A more careful analysis of the coordinate-wise dominance algorithm would be of interest. The analysis here should be generalizable to higher dimensions by someone less faint-hearted than the author.

The $O(n \log A)$ algorithm of §4 should be simpler; heuristically there is no reason to keep track of the hulls of two different sets. Perhaps a simpler but still provably good algorithm can found, particularly for $d \leq 3$.

```
for (min=0,dis=num_points-1; min<=dis; ) {
  for (pp=0; pp<min && !prec(pp,dis); pp++);
  if (pp<min) {
    dis--;
  } else {
    swap(dis,min);
    for (inc=min+1; inc<=dis; ) {
     if (prec(min,dis)) dis--;
     else if (prec(dis,min)) swap(dis--,min);
     else swap(dis,inc++);
    }
    min++;
  }
}
```

Figure 1: The minima algorithm, in C

Appendix

The triviality of the algorithm has the fortunate consequence that implementation is also trivial. The C code fragment of Figure 6 is part of an implementation of the algorithm; the elements of the set are in an array, and pairs of elements at two locations are compared with the `prec` function, and swapped by the `swap` function. The minima are kept at the beginning of the array, and the non-minimal, discarded elements are kept at the end. Note that in the usual case, where an element is discarded, no swapping is done.

References

[1] D. Avis and K. Fukuda. A pivoting algorithm for convex hulls and vertex enumeration of arrangements and polyhedra. *Discrete Comput. Geom.*, 8:295–313, 1992.

[2] J. L. Bentley, K. L. Clarkson, and D. B. Levine. Fast linear expected-time algorithms for computing maxima and convex hulls. In *Proc. 1st ACM-SIAM Sympos. Discrete Algorithms*, pages 179–187, 1990.

[3] B. Chazelle and J. Matoušek. Derandomizing an output-sensitive convex hull algorithm in three dimensions. Technical report, Dept. Comput. Sci., Princeton Univ., 1992.

[4] K. L. Clarkson and P. W. Shor. Applications of random sampling in computational geometry, II. *Discrete Comput. Geom.*, 4:387–421, 1989.

[5] W. F. Eddy. A new convex hull algorithm for planar sets. *ACM Trans. Math. Softw.*, 3:398–403 and 411–412, 1977.

[6] M. J. Golin. *Probabilistic analysis of geometric algorithms.* Ph.D. thesis, Dept. Comput. Sci., Princeton Univ., Princeton, NY, 1990.

[7] J. Matoušek and O. Schwarzkopf. Linear optimization queries. In *Proc. 8th Annu. ACM Sympos. Comput. Geom.*, pages 16–25, 1992.

[8] J. Matoušek. Reporting points in halfspaces. *Computational Geometry: Theory and Applications*, pages 169–186, 1992.

[9] J. Matoušek. Linear optimization queries. *J. Algorithms*, 14:432–448, 1993.

[10] F. P. Preparata and M. I. Shamos. *Computational Geometry.* Springer-Verlag, Berlin, 1985.

[11] G. Rote. Degenerate convex hulls in high dimensions without extra storage. In *Proc. 8th Annu. ACM Sympos. Comput. Geom.*, pages 26–32, 1992.

[12] G. T. Toussaint, B. K. Bhattacharya, and R. S. Poulsen. The application of Voronoi diagrams to nonparametric decision rules. In *Computer Science and Statistics: The Interface*, pages 97–108, 1985.

[13] G. Wilfong. Nearest neighbor problems. In *Proc. 7th Annu. ACM Sympos. Comput. Geom.*, pages 224–233, 1991.