APPEARED IN "New Results and New Trends in Computer Science", (H. Maurer, Ed.),
Lecture Notes in Computer Science **555** (1991) 359-370.

Smallest enclosing disks (balls and ellipsoids)

EMO WELZL*
Institut für Informatik, Freie Universität Berlin
Arnimallee 2-6, W 1000 Berlin 33, Germany
e-mail: emo@tcs.fu-berlin.de

Abstract

A simple randomized algorithm is developed which computes the smallest enclosing disk of a finite set of points in the plane in expected linear time. The algorithm is based on Seidel's recent Linear Programming algorithm, and it can be generalized to computing smallest enclosing balls or ellipsoids of point sets in higher dimensions in a straightforward way. Experimental results of an implementation are presented.

1 Introduction

During the recent years randomized algorithms have been developed for a host of problems in computational geometry. Many of these algorithms are not only attractive because of their efficiency, but also because of their appealing simplicity. This feature makes them easier to access for non-experts in the field, and for actual implementation. One of these simple algorithms is Seidel's Linear Programming algorithm, [Sei1], which solves a Linear Program with n constraints and d variables in expected $O(n)$ time, provided d is constant; see also [DyF] and [Cla] for randomized LP algorithms, where Clarkson [Cla] offers the best constant in dependence on d. The expectation is not dependent on the input distribution; it averages over random choices ('coin flips') made by the algorithm. In particular, there is no input which may force the algorithm to perform badly (like a sorted sequence causes an implementation of Quicksort to take quadratic time).

The goal of this paper is to show that the basic idea of Seidel's LP algorithm can be applied to a broader class of optimization problems, including the computation of smallest volume enclosing balls (or ellipsoids) of point sets in d-space in expected linear time for fixed d. The dependence of the constant in d is $O(\delta\,\delta!)$, where $\delta = d+1$ in the case of balls, and $\delta = (d+3)d/2$ in the case of ellipsoids.

A deterministic linear time algorithm for computing smallest enclosing balls has already been presented by Megiddo in [Meg]. However his method is not nearly as easy to describe and to implement, and the dependence of the constant in d falls far behind the one achieved by our method. In the plane, a simple $O(n \log n)$ algorithm can be found in [Sky]. There

*Work partially supported by the ESPRIT II Basic Research Action Program of the EC under contract no. 3075 (project ALCOM)

are several other methods described in the literature, mostly without time analysis; see e.g. [DöF], where three approaches are compared.

The best previous method, due to Post [Pos], for computing smallest volume enclosing ellipsoids has running time $O(n^2)$; see also [ST], [Tit].

The presentation in Section 2 will concentrate on the case of smallest enclosing disks in the plane, but the extensions will be obvious as soon as the principle is revealed. In Section 3 we describe a few variations; perhaps most important, we provide a heuristic which leads to a significant improvement of the performance of the procedure and allows to compute the smallest enclosing ball for a set of 5000 points in 10-space, say, which is out of reach for the original method. Experimental results are presented.

2 The algorithm

Given a set P of n points in the plane, let $\mathrm{md}(P)$ denote the closed disk of smallest radius containing all points in P. We allow also $P = \emptyset$, when $\mathrm{md}(P) = \emptyset$, and $P = \{p\}$, when $\mathrm{md}(P) = p$.

It is easy to see that such a disk is unique: Suppose D_1 and D_2 are smallest enclosing disks of equal radius r with centers z_1 and z_2, respectively. If $P \subset D_1$ and $P \subset D_2$, then $P \subset D_1 \cap D_2$, and $D_1 \cap D_2$ is contained in the disk D with center $\frac{1}{2}(z_1 + z_2)$ and radius $\sqrt{r^2 - a^2}$, where a is half the distance between z_1 and z_2. Hence $a = 0$, since otherwise D has a radius smaller than r contradicting the fact that D_1 and D_2 are smallest disks. Consequently, D_1 and D_2 coincide.

We will need also the fact that $\mathrm{md}(P)$ is already determined by at most three points in P which lie on the boundary of $\mathrm{md}(P)$. That is, there is a subset S of P on the boundary of $\mathrm{md}(P)$ such that $|S| \leq 3$ and $\mathrm{md}(P) = \mathrm{md}(S)$; so if $p \notin S$, then $\mathrm{md}(P - \{p\}) = \mathrm{md}(P)$, or equivalently, if $\mathrm{md}(P - \{p\}) \neq \mathrm{md}(P)$ then $p \in S$ and p lies on the boundary of $\mathrm{md}(P)$. These are known facts; a somewhat more general version of these claims as we need it here will be proved below.

For a set P of n points, we compute $\mathrm{md}(P)$ in an incremental fashion, starting with the empty set and adding the points in P one after another while maintaining the smallest enclosing disk of the points considered so far. Let $P = \{p_1, p_2, \ldots, p_n\}$ and suppose we have already computed $D = \mathrm{md}(\{p_1, p_2, \ldots, p_i\})$ for some i, $1 \leq i < n$. If $p_{i+1} \in D$, then D is also the smallest enclosing disk of the first $i+1$ points, and we can proceed to the next point. Otherwise we use the fact that p_{i+1} has to lie on the boundary of $D' = \mathrm{md}(\{p_1, p_2, \ldots, p_{i+1}\})$ (as claimed above), and we compute D' by a call to a procedure `b_minidisk`(A, p) which computes the smallest disk enclosing $A = \{p_1, p_2, \ldots, p_i\}$ with $p = p_{i+1}$ on its boundary.

The intuition is that the problem becomes easier as we fix a point p to be on the boundary of the disk, since this can be seen as a decrease in the degrees of freedom we have. So, for the time being, let us assume that `b_minidisk` exists already. Then the above algorithm can be formulated as a recursive procedure as follows.

```
function procedure minidisk(P);  comment:  returns md(P)
    if P = ∅ then
        D := ∅
    else
        choose p ∈ P;
        D := minidisk(P − {p});
        if p ∉ D then
            D := b_minidisk(P − {p}, p);
    return D;
```

Before we provide a description of `b_minidisk`(A, p), let us assume that it needs $c|A|$ steps to compute its output. What is the time complexity of our algorithm? We choose $p \in P$ randomly, each point in P with equal probability $1/|P|$. Let $t(n)$ be the expected number of steps taken by `minidisk`(P) for $|P| = n$. Then t obeys

$$t(n) \leq 1 + t(n-1) + \mathrm{Prob}(p \notin \mathrm{md}(P - \{p\})) \cdot c\,(n-1),$$

where '1' accounts for the constant work required, and the two other terms refer to the expected work caused by the calls to `minidisk` and `b_minidisk`, respectively. There are at most three points p in P such that $\mathrm{md}(P) \neq \mathrm{md}(P - \{p\})$; for all other points we have that $p \in \mathrm{md}(P - \{p\})$ and so $\mathrm{Prob}(p \notin \mathrm{md}(P - \{p\})) \leq \frac{3}{n}$. We conclude that $t(n) \leq (1 + 3c)n$.

The algorithm for `b_minidisk`(A, p) follows roughly the same lines as `minidisk` provided above, but now we need as a subroutine a procedure that computes the smallest disk enclosing a set of points with two specified points on its boundary, and so on. Before we give a full description of these procedures, we introduce a notion and prove a basic lemma.

For P and R finite sets of points in the plane, let $\mathrm{b\_md}(P, R)$ be the closed disk of smallest radius which contains all points in P with all points in R on its boundary. Obviously, $\mathrm{b\_md}(P, \emptyset) = \mathrm{md}(P)$, and $\mathrm{b\_md}(P, R)$ may be undefined as soon as R is nonempty.

Lemma 1 *Let P and R be finite point sets in the plane, P nonempty, and let p be a point in P.*
(i) *If there exists a disk containing P with R on its boundary, then $\mathrm{b\_md}(P, R)$ is well-defined (unique).*
(ii) *If $p \notin \mathrm{b\_md}(P - \{p\}, R)$ then p lies on the boundary of $\mathrm{b\_md}(P, R)$, provided it exists, i.e., $\mathrm{b\_md}(P, R) = \mathrm{b\_md}(P - \{p\}, R \cup \{p\})$.*
(iii) *If $\mathrm{b\_md}(P, R)$ exists, there is a set S of at most $\max\{0, 3 - |R|\}$ points in P such that $\mathrm{b\_md}(P, R) = \mathrm{b\_md}(S, R)$.*

Proof. We prepare the proof with a definition of a convex combination of two disks.

A closed disk with center z and radius $r > 0$ can be written as the set of points x satisfying $f(x) \leq 1$ for $f(x) = \frac{1}{r^2}\|x - z\|^2$; the points on the boundary are those which satisfy equality. Let D_0 and D_1 be disks with defining functions f_0 and f_1, respectively. For each λ, $0 \leq \lambda \leq 1$, the set of points x satisfying

$$f_\lambda(x) = (1 - \lambda) f_0(x) + \lambda f_1(x) \leq 1$$

is again a disk D_λ; if D_0 and D_1 are distinct, then, for $0 < \lambda < 1$, the radius of D_λ is smaller than the maximum of the radii of D_0 and D_1. These facts can be checked by elementary calculations. Moreover, it can be directly read off the definition of f_λ that $D_\lambda \supseteq D_0 \cap D_1$ and that the boundary of D_λ contains the intersection of the boundaries of D_0 and D_1.

For a proof of (i), let us first observe that the infimum of all radii which allow *closed* disks containing P with R on the boundary can actually be realized by a closed disk.

Suppose now that there are two distinct disks D_0 and D_1 which attain this minimum. Then D_λ, defined for $\lambda = \frac{1}{2}$ as above, is also a disk containing P with R on its boundary, but with a smaller radius; contradiction.

The proof of (ii) assumes first that $p \notin D_0 = \mathrm{b\_md}(P - \{p\}, R)$ and p does not lie on the boundary of $D_1 = \mathrm{b\_md}(P, R)$. As D_λ continuously deforms D_0 into D_1 as λ goes from 0 to 1, there is a value $\lambda' < 1$ for which p lies on the boundary of $D_{\lambda'}$. This disk covers P, has p on its boundary (in addition to R), and it has a radius smaller than the one of D_1; contradiction.

Finally let us settle (iii). If $|R| \geq 3$, then $\mathrm{b\_md}(P, R) = \mathrm{b\_md}(\emptyset, R)$, provided $\mathrm{b\_md}(P, R)$ exists. So let us assume that R contains two points at most. Note that (ii) shows already, that $\mathrm{b\_md}(P, R) = \mathrm{b\_md}(S, R)$, for S the set of points in P which lie on the boundary of $\mathrm{b\_md}(P, R)$. So if the points in $P \cup R$ are in general position, i.e., no four cocircular, then we are done. Otherwise perform an infinitesimal perturbation on the points in P, so that for the resulting point set P', we have $P' \cup R$ in general position (this is possible, since $|R| \leq 2$). Then the preimages of the at most $3 - |R|$ points S' on the boundary of $\mathrm{b\_md}(P', R)$ provide the set S as required (details omitted). $\square$

Note that in general the set S is not unique. The important implication of (iii) is that there are at most $\max\{0, 3 - |R|\}$ points in P for which $p \notin \mathrm{b\_md}(P - \{p\}, R)$. The reader may find a shorter proof of the lemma (perhaps less 'notational'), but the intention was to allow an almost verbatim generalization to balls in higher dimensions (and even to ellipsoids).

Point (ii) of the lemma suggests how to compute $\mathrm{b\_md}(P, R)$. If $P = \emptyset$, the problem is easy, and we compute $\mathrm{b\_md}(\emptyset, R)$ directly. Otherwise we choose a random $p \in P$ and compute $D = \mathrm{b\_md}(P - \{p\}, R)$. If $p \in D$, then $\mathrm{b\_md}(P, R) = D$; otherwise, $\mathrm{b\_md}(P, R) = \mathrm{b\_md}(P - \{p\}, R \cup \{p\})$. In a first reading of the following procedure, the reader is supposed to neglect the bracketed part '`or` $|R| = 3$'; but 'D `defined and`' should be observed.

```
function procedure B_MINIDISK(P, R); comment:  returns b_md(P, R)
    if P = ∅ [or |R| = 3] then
        D := b_md(∅, R)
    else
        choose random p ∈ P;
        D := B_MINIDISK(P − {p}, R);
        if [D defined and] p ∉ D then
            D := B_MINIDISK(P − {p}, R ∪ {p});
    return D;
```

Our original problem of computing $\mathrm{md}(P)$ can be solved by:

```
function procedure MINIDISK(P); comment:  returns md(P)
    return B_MINIDISK(P, ∅);
```

Note that since $\mathrm{b\_md}(P, \emptyset)$ is always defined, in the whole computation initiated by `MINIDISK` no call to `B_MINIDISK` returns an undefined result. What does that mean for the case when we call B_MINIDISK with a boundary set R of cardinality 3? The disk is already determined by these points, so the only thing that remains to be done is to check whether all points to be covered are in this disk. If the test fails for a point p, we make a call to B_MINIDISK with boundary set $R \cup \{p\}$ — four points which are not cocircular — and the returned value will be undefined. This cannot happen in a computation of $\mathrm{b\_md}(P, \emptyset)$. So there was no reason to check!

The conclusion is that if we use procedure B_MINIDISK as a subroutine of MINIDISK only, then we can speed it up by inserting the bracketed part '`or` $|R| = 3$', and leave out the test for 'D `defined`'.

It remains to analyze the procedure MINIDISK considering the version of B_MINIDISK using the shortcut for $|R| = 3$. The analysis is basically identical to the one given in [Sei1], and it is an instance of *backwards analysis* of which many examples can be found in [Sei2]. We want to count the expected number how often we execute the test '$p \notin D$'. The actual number of steps will be a constant multiple of this value (as long as P is nonempty). To this end, for $0 \leq j \leq 3$, let $t_j(n)$ be this number for a call B_MINIDISK(P, R) with $|P| = n$ and $|R| = 3 - j$. Then observe that $t_0(n) = 0$; this might be irritating, but the constant amount of work needed in this case is accounted for by the test which lead to the respective call. Obviously, also $t_j(0) = 0$.

For $j > 0$ and $n > 0$, we do one call B_MINIDISK($P - \{p\}, R$) — p a random point in P —, one test '$p \notin D$', and one call B_MINIDISK($P - \{p\}, R \cup \{p\}$), provided $\mathrm{b\_md}(P, R) \neq \mathrm{b\_md}(P - \{p\}, R)$. The probability that the latter happens is at most $\frac{j}{n}$ as it follows from Lemma 1(iii): there is a set S of at most $3 - |R| = j$ points in P with $\mathrm{b\_md}(P, R) = \mathrm{b\_md}(S, R)$; so out of the n choices we have for p, at most j will cause this second subroutine call. Note that not even all points p in such a set S need to satisfy $\mathrm{b\_md}(P, R) \neq \mathrm{b\_md}(P - \{p\}, R)$, and actually this probability may be as small as 0 (even when R is empty), e.g., if the points are the vertices of a regular 6-gon.

We obtain the recursion

$$t_j(n) \leq t_j(n-1) + 1 + \frac{j}{n} t_{j-1}(n-1), \tag{1}$$

and so $t_1(n) \leq n$, $t_2(n) \leq 3n$, and $t_3(n) \leq 10n$. '10' is the constant observed for point sets uniformly distributed in the unit disk (see experimental results in Section 3).

Theorem 2 MINIDISK *computes the smallest enclosing disk of a set of n points in the plane in expected $O(n)$ time.* □

3 Variations, extensions

As indicated in the Introduction, we now can easily extend the algorithm to problems also in higher dimensions, as smallest enclosing balls and ellipsoids.

Balls. The algorithm of the previous section may be used for computing the ball of smallest radius enclosing a set P of n points in $\mathbb{R}^d$. The only difference (except for perhaps renaming the procedure to MINIBALL) is that the constant 3 is replaced by $d+1$, since a sphere in $\mathbb{R}^d$ is determined by $d+1$ points. Of course, we have to redefine also $\mathrm{b\_md}(P, R)$ in the obvious way. Lemma 1 generalizes with '3' replaced by '$d+1$'; actually, its proof can be taken over almost verbatim, see also [Jun].

The recursion (1) is still valid, with the difference that the running time is now determined by $t_\delta(n)$ (instead of $t_3(n)$) for $\delta = d+1$. Note that a test for a point in a ball takes now $O(\delta)$ arithmetic operations. A simple proof by induction demonstrates

$$t_j(n) \leq \left(\sum_{k=1}^{j} \frac{1}{k!}\right) j!\, n = \lfloor (e-1)j! \rfloor n$$

for $j \geq 1$. The bound can be shown to be tight (up to low order terms), e.g. for sets of n points in $I\!R^d$, with buckets of $n/(d+1)$ points clustered around the $d+1$ vertices of a regular simplex.

Leaving the verification of our claims to the reader (perhaps via a second reading of Section 2), we state our result.

Theorem 3 *The smallest enclosing ball of a set of n points in d-space can be computed in expected time $O(\delta\delta!\, n)$, $\delta = d+1$, by a randomized algorithm.* □

Ellipsoids. An ellipsoid is the affine image of the unit ball centered at the origin. So we can define such an ellipsoid in $I\!R^d$ as the set of points $x \in I\!R^d$ satisfying $f(x) \leq 1$, $f(x) = (x-z)^T A(x-z)$, where z is the center of the ellipsoid, and A is a positive definite matrix. Note that f does not change, if we vary a pair of elements symmetric along the main diagonal, as long as its sum remains the same. Consequently, we may as well assume that A is symmetric. This leads to the known fact that an ellipsoid is determined by $(d+3)d/2$ points on its boundary.

For a point set P in $I\!R^d$ we define $\mathrm{me}(P)$ as the ellipsoid in the affine hull of P which contains all points in P, and has smallest volume. Unicity of $\mathrm{me}(P)$ was proven in [Beh], [DLL]. Again the algorithm from the previous section can be adapted to compute the smallest ellipsoid enclosing a point set in a straightforward way. Now the 'parameter 3' has to be replaced by $\delta = (d+3)d/2$. Lemma 1 also generalizes, although a few facts in its proof are somewhat more tedious to verify (see also [Pos], [Juh]).

Theorem 4 *The smallest volume enclosing ellipsoid of a set of n points in d-space can be computed in expected time $O(\delta\delta!\, n)$, $\delta = (d+3)d/2$, by a randomized algorithm.* □

It is perhaps worthwhile to mention here that the smallest volume enclosing ellipsoid of a convex polytope $\mathcal{P}$ — also called *Löwner–John ellipsoid* — has the property, that if it is scaled by a factor $\frac{1}{d}$ about its center, then it is contained in $\mathcal{P}$, [Joh], [Lei]. This shows that the Löwner–John ellipsoid approximates a polytope $\mathcal{P}$ with some guaranteed quality, which makes it attractive for a bounding 'box'–heuristic, e.g. in motion planning. Smallest enclosing ellipsoids are also used in statistics for peeling off outliers in multidimensional data, [Bar].

There are a number of other problems which can be solved with the method, as e.g. computing the largest ball or ellipsoid in the intersection of halfspaces bounded by hyperplanes, or convex programming in general. The main ingredients needed are a notion corresponding to 'lies on the boundary', and an analogue of Lemma 1.

Expensive operations. When it comes to actually implementing the algorithm, most of the effort goes to the solution of the *basic case*, i.e., the realization of the statement '$D := \mathrm{b\_md}(\emptyset, R)$'; in particular, in higher dimensions, and already for ellipses in the plane, this is a nontrivial task (see, e.g. [Tit], [ST]). Since the actual execution of this statement is much more costly than a simple containment test, we are interested how often we have to go through this basic step. Let us consider the case of computing the smallest enclosing ball in d-space. We denote by $s_j(n)$ the expected number of executions of '$D :=$ b_md$(\emptyset, R)$' for a call where there are n points to cover and $\delta - j$ points are forced on the boundary, $\delta = d+1$. Then $s_0(n) = 1$, $s_j(0) = 1$, and for all $j > 0$ and $n > 0$, we have

$$s_j(n) \leq s_j(n-1) + \frac{j}{n} s_{j-1}(n-1). \tag{2}$$

We claim that

$$s_j(n) \leq (1 + H_n)^j, \quad H_n = 1 + \frac{1}{2} + \cdots + \frac{1}{n}.$$

This is obviously true for $j = 0$ and for $n = 0$ (with the convention $H_0 = 0$). For $j > 0$ and $n > 0$ the claim follows from the induction step

$$\begin{aligned} s_j(n) &\leq (1 + H_{n-1})^j + \frac{j}{n}(1 + H_{n-1})^{j-1} \\ &\leq \sum_{k=0}^{j} \binom{j}{k} (1 + H_{n-1})^{j-k} (\frac{1}{n})^k = (1 + H_{n-1} + \frac{1}{n})^j = (1 + H_n)^j \ ; \end{aligned}$$

the first inequality uses (2) and the induction hypothesis, and the second inequality holds since the left hand side represents the first two summands of the sum on the right hand side. Since $H_n \leq 1 + \ln n$ for $n \geq 1$, we conclude that the expected number of executions of the basic case is bounded by $(2 + \ln n)^\delta$.

One permutation suffices. We investigate how many random choices our algorithm needs and show that it suffices to choose one random permutation of $1 \ldots n$ in the beginning of the computation. The following considerations — although technical — will further simplify the algorithm and lead us to a heuristic which considerably speeds up the algorithm in practice — in particular, in higher dimensions.

Let us, in a first step, change the view of our procedure `B_MINIDISK`(P, R) by assuming that the argument P is actually a (randomly) *ordered sequence* of points; the '`else`'–part of procedure `B_MINIDISK` is reformulated as follows.

```
      ⋮
choose last p in P;
D := B_MINIDISK(P − {p}, R);
if p ∉ D then
    choose a random permutation π of 1...(|P| − 1);
    D := B_MINIDISK(π(P − {p}), R ∪ {p});
      ⋮
```

The operation '$P - \{p\}$' removes the element p in the sequence P and leaves the order of the remaining elements unchanged. We let `MINIDISK` also choose a random permutation of the points before it calls `B_MINIDISK`:

```
function procedure MINIDISK(P); comment:  returns md(P)
    choose a random permutation π of 1...|P|;
    return B_MINIDISK(π(P), ∅);
```

It is clear that nothing changes in the expected running time: choosing the last element in a random order of the elements in a set is the same as choosing a random element in the set. We want to argue that if the only random permutation is the one chosen in `MINIDISK`, still nothing changes in the expected running time.

For a sequence P and a set R of points in the plane, $|R| \le 3$, let $T(P,R)$ be the expected running time of the call `B_MINIDISK`(P,R) in the formulation above. Then the expected running time of `MINIDISK`(P) is

$$t(P) = \frac{1}{n!}\sum_{\pi \in S_n} T(\pi(P), \emptyset),$$

for $n = |P|$ and S_n the set of all permutations of $1 \ldots n$.

If P is nonempty and p is the last point in $\pi(P)$, then

$$\begin{aligned} T(\pi(P), R) &= T(\pi(P) - \{p\}, R) + 1 \\ &\quad + \frac{1}{(n-1)!}\sum_{\rho \in S_{n-1}} \chi(p \notin \mathrm{b\_md}(P - \{p\}, R))\, T(\rho(P - \{p\}), R \cup \{p\}), \end{aligned}$$

where $\chi(\cdot)$ is 1, if its argument is true, and 0, otherwise. We get

$$\begin{aligned} t(P) &= \tfrac{1}{n!}\textstyle\sum_{p \in P} \displaystyle\sum_{\substack{\pi \in S_n \\ p \text{ last in } \pi(P)}} T(\pi(P), \emptyset) \\ &= \tfrac{1}{n!}\textstyle\sum_{p \in P} \displaystyle\sum_{\sigma \in S_{n-1}} \Big[T(\sigma(P - \{p\}), \emptyset) + 1 \\ &\qquad + \frac{1}{(n-1)!}\sum_{\rho \in S_{n-1}} \chi(p \notin \mathrm{b\_md}(P - \{p\}, \emptyset))\, T(\rho(P - \{p\}), \{p\})\Big] \\ &= \tfrac{1}{n!}\textstyle\sum_{p \in P} \displaystyle\Big[\sum_{\sigma \in S_{n-1}} T(\sigma(P - \{p\}), \emptyset) \\ &\qquad + (n-1)! \\ &\qquad + \sum_{\rho \in S_{n-1}} \chi(p \notin \mathrm{b\_md}(P - \{p\}, \emptyset))\, T(\rho(P - \{p\}), \{p\})\Big] \\ &= \tfrac{1}{n!}\textstyle\sum_{p \in P} \displaystyle\sum_{\tau \in S_{n-1}} \Big[T(\tau(P - \{p\}), \emptyset) + 1 \\ &\qquad + \chi(p \notin \mathrm{b\_md}(P - \{p\}, \emptyset))\, T(\tau(P - \{p\}), \{p\})\Big]. \end{aligned}$$

The last expression in this derivation represents the expected time in case `B_MINIDISK`(P,R) does not choose a new permutation for its calls if $R = \emptyset$, and similar transformations show the fact for R arbitrary.

Consequently, the revised version of `MINIDISK` has the same expected running time in terms of the number of containment queries, if it uses `B_MINIDISK` with the following '`else`'–part.

```
⋮
choose last p in P;
D := B_MINIDISK(P − {p}, R);
if p ∉ D then
    D := B_MINIDISK(P − {p}, R ∪ {p});
⋮
```

The actual running time of the procedure decreases, of course, since we save the generation of random numbers except for those needed for the first permutation.

A move-to-front heuristic. Considering the just developed one-permutation-version, what would be a good permutation to begin with? Of course, if the first elements are those which determine the solution, then we have the optimal situation, and the algorithm will not make more than $n + O(1)$ containment queries. Somewhat less ambitious, we would like to have points early in the sequence which determine a disk with few points outside. Although we are not given such a sequence, we can gradually update the sequence during the computation by moving points to the front of the sequence which we consider important. Intuitively, these are the points p which satisfy the test '$p \notin D$'. This leads us to the final iteration of our algorithm with the *move-to-front* heuristic implemented. Here we assume that the point set is stored in a global sequence, preferably in a linked list which enables us to move points to the front in constant time.

```
function procedure MTFDISK(P); comment: returns md(P)
    choose a random permutation π of 1...|P|;
    return B_MTFDISK(π(P), ∅);

function procedure B_MTFDISK(P, R); comment: returns b_md(P, R)
    if P = ∅ or |R| = 3 then
        D := b_md(∅, R)
    else
        choose last p in P;
        D := B_MTFDISK(P − {p}, R);
        if p ∉ D then
            D := B_MTFDISK(P − {p}, R ∪ {p});
            move p to the first position;
    return D;
```

At this point we do not know how to analyze MTFDISK (see Discussion), but the improvement in the performance in experiments is striking.

Experimental results. The one-permutation- and the move-to-front-versions of the algorithm have been implemented for balls in arbitrary dimensions, and for ellipses in the plane. Table 1 displays the number of containment queries divided by n (number of points), both the average and the maximum over 100 runs (* = only 40 runs) for points randomly chosen in the unit ball (indicated by '◯') and in the unit cube (indicated by '□'). The MINIBALL procedure reaches its limits in 5 dimensions, while MTFBALL allows to compute smallest enclosing balls for 5000 points in 10 dimension. Another aspect we want to point out is the high variance of MINIBALL compared to MTFBALL which we observed in

Smallest enclosing ball for n points in d dimensions –
number of containment queries divided by n, average and maximum.

d	$\lfloor(e-1)(d+1)!\rfloor$	n	$\bigcirc$MINIBALL		$\bigcirc$MTFBALL		$\square$MINIBALL		$\square$MTFBALL	
			av.	max.	av.	max.	av.	max.	av.	max.
2	10	1000	9.8	30	4.2	8.1	7.5	18	3.7	6.8
		5000	10.5	26	4.1	8.7	7.7	18	3.5	8.0
3	41	1000	33	101	5.6	13	22	114	4.6	10.2
		5000	36	123	5.6	12	21	82	4.9	11
5	1237	1000	627	2748	9.5	17	265	1226	7.7	16
		5000	*944	*2367	8.9	16	360	2277	7.0	14
10	$6.7 \cdot 10^7$	1000	–	–	59	85	–	–	19	39
		5000	–	–	*30	*37	–	–	15	24

Smallest enclosing ellipse for n points in the plane –
number of containment queries divided by n, average and maximum.

$\lfloor(e-1)5!\rfloor$	n	$\bigcirc$MINIELL		$\bigcirc$MTFELL		$\square$MINIELL		$\square$MTFELL	
		av.	max.	av.	max.	av.	max.	av.	max.
206	1000	145	441	7.2	13	71	306	5.4	9.4
	5000	193	708	6.8	13	70	304	5.0	9.0
	10000	*172	*586	7.0	13	*75	*255	5.2	9.1

Table 1: Number of containment queries for one-permutation-version (`MINIBALL` and `MINIELL`) and move-to-front-version (`MTFBALL` and `MTFELL`).

our experiments.

Although little attempts have been made to tune the performance of the program, we provide in Table 2 the average runtime on a personal computer (80386, 20MHz). It is interesting to observe the 'sublinear' behaviour of the runtime for ellipses, which can be explained by the fact that much of the time is spent for the basic case, which grows only polylogarithmically in n, as we have shown. For example, for `MTFELL`, the average number of executions of the basic case is 514, 725, and 816 for $n = 1000$, 5000, and 10000 random points, respectively, in the unit disk. For `MINIELL` the corresponding numbers are 11700, 37000, and 50000, respectively.

4 Discussion

We have described algorithms for the computation of smallest enclosing balls and ellipsoids: a simple algorithm with provably linear running time, and a heuristic which appears to run fast in practice. The move-to-front heuristic has been developed in an interplay between analyzing several features of the original algorithm on the one hand, and phenomena observed in experiments on the other hand.

Clarkson's algorithm for Linear Programming, [Cla], can also be turned in an algorithm for computing smallest enclosing balls and ellipsoids, and the dependence of the constant

Smallest enclosing ball for 5000 points in d dimensions – average runtime				
d	○MINIBALL	○MTFBALL	□MINIBALL	□MTFBALL
2	3.3 sec	1.3 sec	2.4 sec	1.1 sec
3	23 sec	2.4 sec	14 sec	2.1 sec
5	35 min	18 sec	8 min	10 sec
10	–	*20 min	–	4 min

Smallest enclosing ellipses for n points in the plane – average runtime				
n	○MINIELL	○MTFELL	□MINIELL	□MTFELL
1000	5 min	7.8 sec	2 min	3 sec
5000	15 min	19 sec	5 min	10 s
10000	*22 min	31 sec	*8.5 min	17 s

Table 2: Average runtime on a PC (80386, 20 MHz).

in the dimension is better than the one for Seidel's method we have used here; however, it is not as simple. Nevertheless, it would be interesting to compare implementations of several methods including Clarkson's or the ones described in [DöF], and we plan to pursue this line in the future.

An interesting open problem is the analysis of the move-to-front heuristic. There are first steps in this direction in [SW], where we show that a variant closely related to move-to-front has expected running time $O(\delta 2^{\delta} n)$, with $\delta = d + 1$ for balls; actually, this yields a new 'combinatorial bound' also for Linear Programming for certain values of d and n. In addition, the paper offers also a formal framework for the class of problems which can be solved by these methods.

Acknowledgements. The author thanks Raimund Seidel for several discussions on the subject. Special thanks also to Bernd Gärtner for implementing the algorithms, and for accepting my ongoing requests for new variants of the algorithm and new test data.

References

[Bar] V. Barnett, The ordering of multivariate data, *J. Roy. Statist. Soc. Ser. A* **139** (176) 318–354

[Beh] F. Behrend, Über die kleinste umbeschriebene und die größte einbeschriebene Ellipse eines konvexen Bereiches, *Math. Ann.* **115** (1938) 379–411

[Cla] K. L. Clarkson, Las Vegas algorithms for linear and integer programming when the dimension is small, manuscript (1989)

[DLL] L. Danzer, D. Laugwitz and H. Lenz, Über das Löwnersche Ellipsoid und sein Analogon unter den einem Eikörper eingeschriebenen Ellipsoiden, *Arch. Math.* **8** (1957) 214–219

[DyF] M. E. Dyer and A. M. Frieze, A randomized algorithm for fixed-dimensional linear programming, manuscript (1987)

[DöF] J. Dörflinger and W. Forst, Approximation durch Kreise: Verfahren zur Berechnung der Hüllkugel, manuscript (1991)

[Joh] F. John, Extremum problems with inequalities as subsidiary conditions, *in* Courant Anniversary Volume (1948) 187–204, New York

[Juh] F. Juhnke, Löwner ellipsoids via semiinfinite optimization and (quasi-) convexity theory, Technische Universität Magdeburg, Sektion Mathematik, Report 4/90 (1990)

[Jun] H. Jung, Über die kleinste Kugel, die eine räumliche Figur einschließt, *J. Reine Angew. Math.* **123** (1901) 241–257

[Lei] K. Leichtweiß, Über die affine Exzentrizität konvexer Körper, *Arch. Math.* **10** (1959) 187–199

[Meg] N. Megiddo, Linear-time algorithms for linear programming in $\mathbb{R}^3$ and related problems, *SIAM J. Comput.* **12** (1983) 759–776

[Pos] M. J. Post, Minimum spanning ellipsoids, *in* "Proc. 16th Annual ACM Symposium on Theory of Computing" (1984) 108–116

[Sei1] R. Seidel, Linear programming and convex hulls made easy, *in* "Proc. 6th Annual ACM Symposium on Computational Geometry" (1990) 211–215

[Sei2] R. Seidel, Backwards analysis of randomized algorithms, manuscript (1991)

[ST] B. W. Silverman and D. M. Titterington, Minimum covering ellipses, *SIAM J. Sci. Stat. Comput.* **1** (1980) 401 – 409

[SW] M. Sharir and E. Welzl, A new combinatorial bound for linear programming and related problems, in preparation (1991)

[Sky] S. Skyum, A simple algorithm for computing the smallest circle, Aarhus University, Report DAIMI PB-314

[Tit] D. M. Titterington, Estimation of correlation coefficients by ellipsoidal trimming, *Appl. Statist.* **27** (1978) 227–234