Approximate Minimum Enclosing Balls in High Dimensions Using Core-Sets

Piyush Kumar [1]
Joseph S. B. Mitchell[2]
E. Alper Yıldırım[3]

We study the minimum enclosing ball (MEB) problem for sets of points or balls in high dimensions. Using techniques of second-order cone programming and "core-sets", we have developed $(1+\epsilon)$-approximation algorithms that perform well in practice, especially for very high dimensions, in addition to having provable guarantees. We prove the existence of core-sets of size $O(1/\epsilon)$, improving the previous bound of $O(1/\epsilon^2)$, and we study empirically how the core-set size grows with dimension. We show that our algorithm, which is simple to implement, results in fast computation of nearly optimal solutions for point sets in much higher dimension than previously computable using exact techniques.

Categories and Subject Descriptors: F.2.0 [**Analysis of algorithms and problem complexity**]: General—*Geometrical Problems and Computations*

General Terms: Core-Sets

Additional Key Words and Phrases: Minimum enclosing ball, second-order cone programming, approximation algorithms

1. INTRODUCTION

We study the *minimum enclosing ball* (MEB) problem: Compute a ball of minimum radius enclosing a given set of objects (points, balls, etc) in $\mathbb{R}^d$. The MEB problem arises in a number of important applications, often requiring that it be solved in relatively high dimensions. Applications of MEB computation include gap tolerant classifiers [8] in Machine Learning, tuning Support Vector Machine parameters [10], Support Vector Clustering [4; 3], preprocessing for fast farthest neighbor query approximation [19], k-center clustering [5], testing of clustering [2], solving the approximate 1-cylinder problem [5], computation of spatial hierarchies (e.g., sphere trees [20]), and other applications [13].

[1]Department of Computer Science, Stony Brook University, Stony Brook, NY 11794-4400. Email: piyush@acm.org
[2]Department of Applied Mathematics and Statistics, Stony Brook University, Stony Brook, NY 11794-3600, USA. Email: jsbm@ams.sunysb.edu
[3]Department of Applied Mathematics and Statistics, Stony Brook University, Stony Brook, NY 11794-3600, USA. Email: yildirim@ams.sunysb.edu

The software associated with this paper can be downloaded from http://www.compgeom.com/meb/.
P. Kumar and J. Mitchell are partially supported by a grant from the National Science Foundation (CCR-0098172). J. Mitchell is also partially supported by grants from the Honda Fundamental Research Labs, Metron Aviation, NASA Ames Research (NAG2-1325), and the US-Israel Binational Science Foundation. E. A. Yıldırım is partially supported by an NSF CAREER award (DMI-0237415).

In this paper, we give improved time bounds for approximation algorithms for the MEB problem in which the given set of objects consists of points or balls in high dimensions. We prove a time bound of $O(\frac{nd}{\epsilon} + \frac{1}{\epsilon^{4.5}} \log \frac{1}{\epsilon})$, which is based on an improved bound of $O(1/\epsilon)$ on the size of "core-sets" as well as the use of second-order cone programming (SOCP) for solving subproblems. We have performed an experimental investigation to determine how the core-set size tends to behave in practice for a variety of input distributions. We show that substantially larger instances, both in terms of the number n of input points and the dimension d, of the MEB problem can be solved $(1+\epsilon)$-approximately, with very small values of $\epsilon > 0$, compared with the best known implementations of exact solvers. (We note that, since the original appearance of this paper, Fischer, Gärtner and Kutz [16] have announced significantly improved results with a new exact solver.) We also demonstrate that the sizes of the core-sets tend to be much smaller than the worst-case theoretical upper bounds.

Preliminaries. Throughout this paper, S will be either a set of points in $\mathbb{R}^d$ or a set of balls. We let $n = |S|$.

We let $B_{c,r}$ denote a ball of radius r centered at point $c \in \mathbb{R}^d$. Given an input set $S = \{p_1, \ldots, p_n\}$ of n objects in $\mathbb{R}^d$, the *minimum enclosing ball MEB*(S) *of* S is the unique minimum-radius ball containing S. (Uniqueness follows from results of [14; 37]: if B_1 and B_2 are distinct minimum enclosing balls for S, then one can construct a smaller ball containing $B_1 \cap B_2$ and therefore containing S.) The center, c^*, of MEB(S) is often called the *1-center* of S, since it is the point of $\mathbb{R}^d$ that minimizes the maximum distance to points in S. We let r^* denote the radius of MEB(S). A ball $B_{c,(1+\epsilon)r}$ is said to be $(1+\epsilon)$-*approximation of MEB*(S) if $r \leq r^*$ and $S \subset B_{c,(1+\epsilon)r}$.

Given $\epsilon > 0$, a subset, $X \subseteq S$, is said to be an ϵ-*core-set* (or *core-set*) of S if $B_{c,(1+\epsilon)r} \supset S$, where $B_{c,r} = \text{MEB}(X)$; in other words, X is a core-set if an expansion by factor $(1+\epsilon)$ of its MEB contains S. Since $X \subseteq S$, $r \leq r^*$; thus, the ball $B_{c,(1+\epsilon)r}$ is a $(1+\epsilon)$-approximation of MEB(S).

Related work. For small (fixed) dimension d, the MEB problem can be solved in $O(n)$ time for n points using the fact that it is an LP-type problem [14; 23]. One of the best implementable solutions to compute the MEB exactly in moderately high dimensions is given by Gärtner and Schönherr [18]; the largest instance they solve is in dimension $d = 300$ for $n = 10,000$ points (in about 20 minutes on their platform). In comparison, the largest instance we solve[4] $(1+\epsilon)$-approximately is in dimension $d = 1500$ for $n = 100,000$ points, with $\epsilon = 10^{-3}$. Other implementations of exact solvers to which we compare our method include the algorithm of Gärtner [17] and the algorithm of the CGAL[5] library (based on the algorithm of Welzl [34]). For large dimensions, our approximation algorithm is found to be much faster than these exact solvers. We are not aware of other implementations of polynomial-time approximation schemes for the MEB problem.

Very recently, in a paper that appeared after the conference publication of our paper, Fischer, Gärtner and Kutz [16] gave a very fast algorithm to compute exact minimum enclosing balls of point sets in high dimensions. Their method is

[4] This instance took less than 17 minutes to solve.
[5] `http://www.cgal.org` (Version 2.4)

similar to the simplex method for solving the MEB problem [18; 16]. The current version of our implementation (improved after conference publication) seems to be competitive with their implementation for moderately small values of ϵ. (See Figure 3.)

Bădoiu et al. [5] introduced the notion of core-sets and their use in approximation algorithms for high-dimensional clustering problems. In particular, they give an $O\left(\frac{nd}{\epsilon^2} + \frac{1}{\epsilon^{10}} \log \frac{1}{\epsilon}\right)$-time $(1+\epsilon)$-approximation algorithm based on their upper bound of $O(1/\epsilon^2)$ on the size of core-sets; the upper bound on the core-set size is remarkable in that it does not depend on d. In comparison, our time bound (Theorem 1) is $O\left(\frac{nd}{\epsilon} + \frac{1}{\epsilon^{4.5}} \log \frac{1}{\epsilon}\right)$.

Independent of our work, the MEB problem in high dimensions has been recently studied by Zhou et al. [37]. The authors consider two approaches, one based on reformulation as an unconstrained convex optimization problem and another based on a second-order cone programming (SOCP) formulation. Xu et al. [35] perform a comparison of four algorithms (including a randomized algorithm) for the computation of the minimum enclosing circle of circles in the plane ($d = 2$). Both studies reveal that solving the MEB problem using a direct SOCP formulation suffers from memory problems as the dimension, d, and the number of points, n, increase. Our approach in this paper is to apply core-sets, in combination with SOCP, to design practical approximation algorithms for the MEB problem.

In parallel with our work, Bădoiu and Clarkson [6] independently obtained an $O(1/\epsilon)$ bound on the size of core-sets. Most recently, Bădoiu and Clarkson [7] have obtained an upper bound of $\lceil 1/\epsilon \rceil$ and shown that it is worst-case tight. The proof of the tightness employs a lower bound construction, placing $d + 1$ points at the vertices of a regular simplex in dimension $d = \lfloor 1/\epsilon \rfloor$. Thus, the lower bound is based on having $d \geq \lfloor 1/\epsilon \rfloor$. In the experiments reported here, however, almost always the dimension d satisfies $d < \frac{1}{\epsilon}$, and we find that, on a wide variety of input sets, the core-set size is smaller than $\min\{1/\epsilon, d+1\}$; see Figures 4 and 6. (Note that core-sets of size at most $d + 1$ always exist for any $\epsilon \geq 0$, since $d + 1$ points suffice to determine a ball in $\mathbb{R}^d$.)

Outline of paper. We first show in Section 2 how to use second-order cone programming to solve the MEB problem in $O(\sqrt{n}d^2(n + d)\log(n/\epsilon))$ arithmetic operations. In Section 3 we describe the main $(1 + \epsilon)$-approximation algorithm, then in Section 4 we analyze the algorithm and prove the new bound of $O(1/\epsilon)$ on the size of core-sets. Section 5 is devoted to discussion of the experimental results obtained with our implementation.

2. SOCP FORMULATION

The minimum enclosing ball (MEB) problem can be formulated as a second-order cone programming (SOCP) problem. SOCP is a class of convex optimization problems in which a linear function is minimized over an affine subset of products of second-order cones (also known as "Lorenz cones", or "quadratic cones"), defined as

$$K = \{(\sigma, x) \in \mathbb{R}^{1+d} : \|x\| \leq \sigma\}.$$

SOCP therefore can be viewed as an extension of linear programming in which the nonnegative orthant is replaced by the product of second-order cones. Linear programming is a special case of SOCP since each non-negativity constraint is equivalent to a one-dimensional second-order cone constraint. As with the non-negative orthant, K is a full-dimensional, convex cone in $\mathbb{R}^{d+1}$; however, K is not polyhedral for $d \geq 2$.

Recently, SOCP has received a lot of attention from the optimization community due to its applications in a wide variety of areas (see, e.g., [22; 1]) and due also to the existence of very efficient interior-point algorithms to solve this class of optimization problems.

The MEB problem for an input set $S = \{B_{c_i,r_i},\ i = 1,\ldots,n\}$ of n balls can be formulated as an SOCP problem as

$$\min_{c,r}\ r, \quad \text{s.t.} \quad \|c - c_i\| + r_i \leq r,\ i = 1,\ldots,n, \tag{1}$$

where $c \in \mathbb{R}^d$ and $r \in \mathbb{R}$ are the decision variables corresponding to the center and the radius of the MEB, respectively. Note that the formulation reduces to the usual MEB problem for point sets if $r_i = 0$ for $i = 1,\ldots,n$.

When applied to the nonlinear convex optimization problem (1), interior-point algorithms generate a sequence of interior feasible solutions (c^k, r^k), $k = 0, 1, 2, \ldots$ that converges to (c^*, r^*) in the limit, where $B_{c^*,r^*} := \text{MEB}(S)$. Note that an interior feasible solution (c^k, r^k) for (1) – i.e., a feasible solution that strictly satisfies all the inequalities – geometrically corresponds to a ball B_{c^k,r^k} that strictly contains all the balls in S. Consequently, interior-point algorithms converge to MEB(S) through a sequence of strictly enclosing balls.

Given any relative error $\gamma > 0$, interior-point algorithms compute an interior feasible solution (c^k, r^k) such that

$$r^k - r^* \leq \gamma(r^0 - r^*) \tag{2}$$

in $O(\sqrt{n}\log(1/\gamma))$ iterations, where r^0 is the radius of the initial strictly enclosing ball from which the algorithm is initiated [24; 26; 28]. In the context of the MEB problem, one can easily find an initial enclosing ball with $r^0 = O(r^*)$, as the following lemma shows. This result will then be used to establish that a particular choice of γ yields a solution that is a $(1+\delta)$-approximation of MEB(S).

Lemma 1. *Let $S = \{B_{c_i,r_i},\ i = 1,\ldots,n\}$ be a given set of n balls. One can compute a $\frac{1}{\sqrt{3}}$-approximation to the diameter of S in $O(nd)$ time.*

Proof. If S is viewed as an infinite collection of points in $\mathbb{R}^d$, the statement simply follows from the algorithm of Eğecioğlu and Kalantari [11] for the case of a finite point set. Pick any $p \in S$; find a point $q \in S$ that is furthest from p; find a point $q' \in S$ that is furthest from q; output the pair (q, q'). It is easy to see that the same method applies to the case in which S is a set of balls, yielding again a $\frac{1}{\sqrt{3}}$-approximation. (Principal component analysis can be used to obtain the same approximation ratio for points but does not readily generalize to the case of balls.) Note that the furthest point in each ball $B_{c_i,r_i} \in S$ from a given point $p \in S$ can be computed in $O(d)$ time, yielding an overall time complexity of $O(nd)$. □

We now use Lemma 1 to construct an initial ball enclosing S. Let Δ denote the diameter of S and $(q, q') \in S$ denote the two points that yield the diameter approximation. Let $D := \|q - q'\|$. The following inequalities easily follow.

$$\frac{1}{\sqrt{3}} r^* \leq \frac{1}{\sqrt{3}} \Delta \leq D \leq \Delta \leq 2r^*, \tag{3}$$

where r^* is the radius of MEB(S). It follows from (3) that

$$r^* \leq \sqrt{3} D \leq 2\sqrt{3} r^*. \tag{4}$$

Consequently, the ball centered at q (or q') with radius $r_0 := 2D$ (or any radius strictly greater than D) strictly encloses S, and $r_0 \leq 4r^*$. In conjunction with (2), it follows that interior-point methods compute an enclosing ball B_{c^k, r^k} in polynomial time with the property that

$$r^k \leq r^*(1 + 3\gamma). \tag{5}$$

For any given $\delta > 0$, if we set $\gamma := \delta/3$, it follows that we get a $(1+\delta)$-approximation of MEB(S) in $O(\sqrt{n} \log(1/\delta))$ iterations.

The major work at each iteration of interior-point algorithms is the solution of a linear system involving a $(d + 1) \times (d + 1)$ symmetric and positive definite matrix (see, e.g., [1]). For the MEB problem, the matrix in question can be computed using $O(nd^2)$ basic arithmetic operations (flops), and its Cholesky factorization can be carried out in $O(d^3)$ flops. Therefore, the overall complexity of computing a $(1 + \delta)$-approximation of MEB(S) with an interior-point method is $O(\sqrt{n} d^2 (n + d) \log(1/\delta))$. In practice, we stress that the number of iterations seems to be $O(1)$ or very weakly dependent on n (see, for instance, the computational results with SDPT3 in [33]).

The worst-case complexity estimate reveals that the direct application of interior-point algorithms is not computationally feasible for large-scale instances of the MEB problem due to excessive memory requirements. In [37], the largest instance solved by an interior-point solver consists of 1000 points in 2000 dimensions and requires over 13 hours on their platform. However, large-scale instances can still be handled by an interior-point algorithm if the number of points n can somehow be decreased. This can be achieved by a filtering approach in which one eliminates points that are guaranteed to be in the interior of the MEB or by selecting a subset of points and solving a smaller problem and iterating until the computed MEB contains all the points. The latter approach is simply an extension of the well-known cutting plane approach initially developed for solving large-scale linear programs that have much fewer variables than constraints. The MEB problem formulated as in (1) above precisely fits in this framework since $n \gg d$ for instances of interest in this paper. Due to the nonlinearity of the inequalities in (1), the boundary of each constraint is actually a nonlinear surface as opposed to a hyperplane. We therefore use the term "cutting plane" loosely in this paper.

We use the cutting plane approach to be able to solve large-scale MEB instances, discovering a carefully selected subset of the constraints, corresponding to a core-set. The success of such an approach depends on the following factors:

—*Initialization:* The quality of the initial core-set is crucial, since a good approximation leads to fewer updates. Furthermore, a small core-set with a good

approximation yields MEB instances with relatively few points, which can efficiently be solved by an interior-point algorithm.

—*Subproblems:* The performance of a cutting plane approach is closely related to the efficiency with which each subproblem can be solved. We use the state-of-the-art interior-point solver SDPT3 [32] in our implementation.

—*Core-set Updates:* An effective approach should update the core-set in a way that seeks to minimize the number of subsequent updates.

In the following section, we describe our algorithm in more detail.

3. THE ALGORITHM

Algorithm 1 Outputs a $(1+\epsilon)$-approximation of MEB(S) and an $O(1/\epsilon)$-size core-set

Require: Input set $S \subset \mathbb{R}^d$ of points/balls, parameter $\epsilon \in (0, 1)$

1: $X \leftarrow \{q, q'\}$, where $q, q' \in S$ are given by the diameter approximation (Lemma 1)
2: $\delta \leftarrow \epsilon^2/163$
3: **loop**
4: Let $B_{c',r'}$ denote the $(1+\delta)$-approximation to MEB(X) returned by SOCP.
5: **if** $S \subseteq B_{c',(1+\epsilon/2)r'}$ **then**
6: Return $B_{c',(1+\epsilon/2)r'}$, X
7: **else**
8: $p \leftarrow \arg\max_{x \in S} \|c' - x\|$
9: **end if**
10: $X \leftarrow X \cup \{p\}$
11: **end loop**

Given a set S of n points or balls, our algorithm for approximating MEB(S) begins with computing an approximate minimum enclosing ball of a carefully chosen subset $X \subseteq S$. For our purposes, it suffices to obtain any constant factor approximation of the diameter Δ, so we choose to use the two points given by Lemma 1 as our initial core-set X (Step 1, Algorithm 1).

Step 2 of our algorithm sets the parameter δ so that the final output is guaranteed to be a $(1+\epsilon)$-approximation of MEB(S), as we prove in Section 4.

The main loop (steps 3 to 11) first computes the approximate MEB of the current subset $X \subseteq S$, using the SOCP solver. Step 5 checks if a $(1+\epsilon/2)$-expansion of this ball contains S. If this is the case, then the algorithm returns this expanded ball and the current core-set as the solution; otherwise, the algorithm picks the furthest point in S from the center of the approximate minimum enclosing ball of X, adds it to X, and repeats the loop. (The rationale for using $\epsilon/2$ in the expansion will be given in the next section.)

We establish the following results about our algorithm.

THEOREM 1. *Algorithm 1 returns a $(1+\epsilon)$-approximation to the MEB of a set of n balls in d dimensions in time $O\left(\frac{nd}{\epsilon} + \frac{d^2}{\epsilon^{3/2}}\left(\frac{1}{\epsilon} + d\right)\log\frac{1}{\epsilon}\right)$, which is $O\left(\frac{nd}{\epsilon} + \frac{1}{\epsilon^{4.5}}\log\frac{1}{\epsilon}\right)$, if $d = O(1/\epsilon)$ (as in [5]).*

THEOREM 2. *Upon termination of Algorithm 1, X is a core-set of S and has size $O(1/\epsilon)$.*

4. ANALYSIS OF THE ALGORITHM

In this section, we present a detailed analysis of Algorithm 1. We begin with a basic lemma which holds for both sets of points and balls. The proof for the case of points (given in [5; 19]) requires only minor modification to apply to balls; we include it for completeness.

LEMMA 2. *Let $B_{c,r}$ be the MEB of a set $S \subset \mathbb{R}^d$ of balls. Then any closed halfspace containing c contains at least one point in B_i, for some $i \in \{1, \ldots, n\}$, at distance r from c.*

PROOF. We can assume that $B_i \neq B_{c,r}$ for all $i = 1 \ldots n$, since otherwise the proof is trivial. Let P_r be the set of points of $\cup_i B_i$ that are at distance exactly r from c; by our assumption, P_r is a discrete set of at most n points.

Suppose, to the contrary, that H is a closed halfspace containing c that contains no point of P_r. Let $\delta > 0$ be the minimum distance between P_r and H, and let $\rho < r$ be the maximum distance from c to a point of $S \cap H$. Pick a positive number $\epsilon < \min\{\delta, r - \rho\}$. Then, if we translate the ball $B_{c,r}$ by a distance ϵ in the direction of the outward normal of H, the new ball, $B_{c',r}$, will still contain $\cup_i B_i$ and none of the points of balls of $\cup_i B_i$ lie on the boundary of $B_{c',r}$. (For $p \in \cup_i B_i \setminus H$, $\|p - c'\| < \|p - c\|$, and for $p \in \cup_i B_i \cap H$, $\|p - c'\| < \|p - c\| + \epsilon < r$.) Thus, B' can be shrunk, contradicting the optimality of $B_{c,r}$. □

The following lemma, which is a modification of the result proved in [5], establishes that each update of the core-set strictly increases the radius of the corresponding MEB.

LEMMA 3. *Let $B_{c,r}$ be the MEB of a set $X \subset \mathbb{R}^d$ of balls. Let $q \in \mathbb{R}^d$ be such that $q \notin B_{c,(1+\epsilon/3)r}$, for some $\epsilon \in (0, 1)$. Then, the radius of MEB($X \cup \{q\}$) is at least $\left(1 + \frac{\epsilon^2}{33}\right) r$.*

PROOF. Let $B_{c',r'} = \text{MEB}(X \cup \{q\})$. If $\|c' - c\| < (\epsilon/4)r$ then, by the triangle inequality, we have $\|q - c'\| \geq \|q - c\| - \|c' - c\| \geq (1 + \epsilon/3)r - (\epsilon/4)r = (1 + \epsilon/12)r$; thus, the radius, r', of MEB($X \cup \{q\}$) must be at least $(1 + \epsilon/12)r \geq (1 + \epsilon^2/33)r$, for $\epsilon \in (0, 1)$. If $\|c' - c\| \geq (\epsilon/4)r$, then let H be the halfspace whose bounding hyperplane passes through c and is orthogonal to $\overrightarrow{cc'}$, with $c' \notin H$. By Lemma 2, there is a point $p \in H$ in some ball of X, with $\|p - c\| = r$. Thus,

$$r' \geq \|p - c'\| = \sqrt{r^2 + \|c' - c\|^2 - 2r\|c' - c\| \cos\theta} \geq \sqrt{r^2 + \left(\frac{r\epsilon}{4}\right)^2} \geq \left(1 + \frac{\epsilon^2}{33}\right) r,$$

where the equality follows from the law of cosines with $\theta = \angle c'cp \geq \pi/2$, and the last inequality uses the assumption that $\epsilon < 1$. Thus, in this case too we get that $r' \geq (1 + \epsilon^2/33)r$, completing the proof. □

Next, we show that the fact that the interior-point algorithm of Section 2 returns only an approximation of the MEB does not jeopardize the asymptotic performance of our algorithm. We begin with the following lemma, which allows us to constrain any approximate MEB within a small shell that lies between a slightly smaller copy of the (exact) MEB and a slightly larger copy of the (exact) MEB.

LEMMA 4. *Let $B_{c,r}$ be the MEB of a set $X \subset \mathbb{R}^d$ of balls. Let $B_{c',r'}$ be a $(1 + \delta)$-approximation to the MEB. Then, $\|c' - c\| \leq r\sqrt{\delta(\delta + 2)}$ and*

$$B_{c,r-\|c'-c\|} \subseteq B_{c',r'} \subseteq B_{c,(1+\delta)r+\|c'-c\|}.$$

Proof. Let H be the halfspace whose bounding hyperplane passes through c and is orthogonal to $\overrightarrow{cc'}$, with $c' \notin H$. By Lemma 2, there is a point $p \in H$ in some ball of X, with $\|p - c\| = r$. Since p is in some ball of X, and $X \subseteq B_{c',r'}$, we know that $\|c' - p\| \leq r' \leq (1 + \delta)r$. Using the law of cosines,

$$\|c' - c\|^2 = \|c' - p\|^2 - r^2 + 2r\|c' - c\| \cos\theta \leq r'^2 - r^2 \leq \delta(\delta + 2)r^2,$$

where $\theta = \angle c'cp \geq \pi/2$. Thus, $\|c' - c\| \leq r\sqrt{\delta(\delta + 2)}$, as claimed.

Now, for any $x \in B_{c,r-\|c'-c\|}$, we know that $\|x - c\| \leq r - \|c' - c\|$, so, by the triangle inequality and the fact that $r \leq r'$ we get that

$$\|x - c'\| \leq \|x - c\| + \|c' - c\| \leq r \leq r',$$

implying that $x \in B_{c',r'}$. Thus, $B_{c,r-\|c'-c\|} \subseteq B_{c',r'}$.

Similarly, for any $x \in B_{c',r'}$, we know that $\|x - c'\| \leq r'$, so, by the triangle inequality,

$$\|x - c\| \leq \|x - c'\| + \|c' - c\| \leq (1 + \delta)r + \|c' - c\|,$$

implying that $x \in B_{c,(1+\delta)r+\|c'-c\|}$. Thus, $B_{c',r'} \subseteq B_{c,(1+\delta)r+\|c'-c\|}$. □

The following lemma establishes that a point outside of an expanded approximate MEB is guaranteed to be outside of the appropriately expanded exact MEB.

Lemma 5. *Let $B_{c,r}$ be the MEB of a set $X \subset \mathbb{R}^d$ of balls. Let $B_{c',r'}$ be a $(1 + \delta)$-approximation of MEB(X). If $q \notin B_{c',(1+\epsilon/2)r'}$, then $q \notin B_{c,(1+\epsilon/3)r}$, provided that δ is chosen so that $\delta \leq \epsilon^2/163$.*

Proof. By Lemma 4, $B_{c,(1+\epsilon/2)(r-\|c'-c\|)} \subseteq B_{c',(1+\epsilon/2)r'}$ (since $B_{c,r-\|c'-c\|} \subseteq B_{c',r'}$). For any point $q \notin B_{c',(1+\epsilon/2)r'}$, then, we have

$$\|q - c\| \geq (1 + \epsilon/2)(r - \|c' - c\|) \geq (1 + \epsilon/2)\left(r - r\sqrt{\delta(2 + \delta)}\right),$$

using the fact, from Lemma 4, that $\|c' - c\| \leq r\sqrt{\delta(\delta + 2)}$. Thus, $q \notin B_{c,(1+\epsilon/3)r}$ if

$$(1 + \epsilon/2)\left(r - r\sqrt{\delta(2 + \delta)}\right) \geq (1 + \epsilon/3)r,$$

i.e., provided that

$$\sqrt{\delta(2 + \delta)} \leq \frac{\epsilon}{6 + 3\epsilon},$$

or $\delta \leq \sqrt{1 + \left(\frac{\epsilon}{6+3\epsilon}\right)^2} - 1$. For $\epsilon < 1$, it is readily checked that $\epsilon^2/163 \leq \sqrt{1 + \left(\frac{\epsilon}{6+3\epsilon}\right)^2} - 1$; thus, it suffices to choose $\delta \leq \epsilon^2/163$, as claimed. □

We next show that Algorithm 1 correctly returns a $(1 + \epsilon)$-approximation of MEB(S).

Lemma 6. *The ball returned by Algorithm 1 is a $(1 + \epsilon)$-approximation of MEB(S).*

Proof. Algorithm 1 returns the ball $B_{c',(1+\epsilon/2)r'}$, where $B_{c',r'}$ is a $(1+\delta)$-approximation of $B_{c,r} = \text{MEB}(X)$ returned by the SOCP algorithm for the set $X \subseteq S$.

We know by step 5 that $S \subseteq B_{c',(1+\epsilon/2)r'}$; thus, to show that $B_{c',(1+\epsilon/2)r'}$ is a $(1 + \epsilon)$-approximation of MEB(S), we have to show that $(1 + \epsilon/2)r'$ is at most $(1 + \epsilon)$ times

the radius of MEB(S). Since $X \subseteq S$, the radius of MEB(S) is at least r, the radius of MEB(X); thus, it suffices to show that $(1+\epsilon/2)r' \leq (1+\epsilon)r$.

By Lemma 4, $B_{c',r'} \subseteq B_{c,(1+\delta)r+\|c'-c\|}$, so

$$r' \leq (1+\delta)r + \|c'-c\|.$$

Thus, it suffices to show that

$$(1+\epsilon/2)(1+\delta)r + (1+\epsilon/2)\|c'-c\| \leq (1+\epsilon)r,$$

or,

$$\|c'-c\| \leq \left(\frac{\epsilon}{2+\epsilon} - \delta\right) r.$$

From Lemma 4, we know that $\|c'-c\| \leq r\sqrt{\delta(\delta+2)}$. Then, using the assumption that $\epsilon < 1$, we see that it suffices to show that

$$\sqrt{\delta(\delta+2)} \leq \frac{\epsilon}{2+1} - \delta = \frac{\epsilon}{3} - \delta.$$

This last inequality holds if $\delta \leq \epsilon^2/(18+6\epsilon)$, which certainly holds with our choice of $\delta = \epsilon^2/163$ in Algorithm 1. □

Lemma 7. *The set X returned by Algorithm 1 is an ϵ-core-set of S.*

Proof. Let $B_{c,r} = \text{MEB}(X)$ be the MEB of the set X returned by the algorithm. We know, by step 5, that at the conclusion of the algorithm, $S \subseteq B_{c',(1+\epsilon/2)r'}$, where $B_{c',r'}$ is a $(1+\delta)$-approximation of $B_{c,r} = \text{MEB}(X)$.

In order to prove that X is an ϵ-core-set of S, we must show that $S \subseteq B_{c,(1+\epsilon)r}$. Since $S \subseteq B_{c',(1+\epsilon/2)r'}$, it suffices to show that $B_{c',(1+\epsilon/2)r'} \subseteq B_{c,(1+\epsilon)r}$.

Now, by Lemma 4, $B_{c',(1+\epsilon/2)r'} \subseteq B_{c,(1+\epsilon/2)((1+\delta)r+\|c'-c\|)}$. Thus, it suffices to show that

$$(1+\epsilon/2)\left((1+\delta)r + \|c'-c\|\right) \leq (1+\epsilon)r.$$

Since, by Lemma 4, $\|c'-c\| \leq r\sqrt{\delta(\delta+2)}$, and $\delta \leq \sqrt{\delta(\delta+2)}$, it suffices that

$$(1+\epsilon/2)\left(1 + 2\sqrt{\delta(\delta+2)}\right) \leq 1+\epsilon,$$

or,

$$\sqrt{\delta(\delta+2)} \leq \frac{\epsilon}{2(2+\epsilon)}.$$

Since $\epsilon < 1$, it suffices if δ is chosen so that $\sqrt{\delta(\delta+2)} \leq \epsilon/6$, which holds if $\delta \leq \epsilon^2/73$, and therefore holds for our choice of $\delta = \epsilon^2/163$ in Algorithm 1. □

We are now ready to prove Theorems 1 and 2 stated at the end of Section 3. First, we note that an upper bound of $O(1/\epsilon^2)$ on the size of a core-set is straightforward: By Lemma 1, the radius of MEB(X) at the first iteration of the algorithm (when $X = \{q, q'\}$) is at least $\Delta/(2\sqrt{3})$, where Δ is the diameter of S. By Lemmas 3 and 5, each point added to X increases the radius of the corresponding minimum enclosing ball by at least $\frac{\Delta}{2\sqrt{3}}\frac{\epsilon^2}{O(1)}$. Since the radius of MEB(S) is less than the diameter Δ, the loop will be executed $O(1/\epsilon^2)$ times. We now show that a more careful analysis yields the improved bound of Theorem 2.

Proof of Theorem 2. Our proof is based on a careful analysis of the number of times the loop (steps 3-11) is executed in Algorithm 1. Without loss of generality, we assume that $\epsilon = 1/2^m$. We will obtain an upper bound on the number of points added to X in order to obtain a $(1+\epsilon_i)$-approximation of MEB(S), where $\epsilon_i := 1/2^i$, $i = 1, \ldots, m$.

Note that, since each iteration adds a point to X, the radius of MEB(X) monotonically increases; thus, once the set X becomes an ϵ_i-core-set of S, it remains an ϵ_i-core-set. We consider the iterations of the algorithm to be partitioned into m *rounds*. We consider round i to begin when X first becomes an ϵ_{i-1}-core-set of S and to end when X first becomes an ϵ_i-core-set of S. Note that a round may start and end at the same instant, since it may be that when X first becomes an ϵ_{i-1}-core-set, it also may become an ϵ_i-core-set (indeed, it may be that MEB(X) is now equal to MEB(S)). Also note that our algorithm does not determine exactly when the transitions occur between rounds, since we compute $(1+\delta)$-approximations of the MEB, not the exact MEB; however, the decomposition into rounds as defined above is useful in the analysis of the algorithm.

Let X_i denote the set X at the conclusion of round i (and thus at the start of round $i+1$), let τ_i denote the number of points of S added to X_{i-1} during round i (i.e., τ_i is the number of iterations in round i), and let r_i denote the radius of MEB(X_i).

Consider an iteration of the algorithm during round i. Let X be the current value of the subset of S, and let $B_{c,r} = \text{MEB}(X)$. The algorithm does not compute $B_{c,r}$ but does compute $B_{c',r'}$, a $(1+\delta)$-approximation of $B_{c,r}$. The algorithm then selects a point $p \in S$ to be added to X that maximizes the distance from the center, c', of $B_{c',r'}$. Since we know, by definition of rounds, that X is *not* an ϵ_i-core-set until the *end* of round i, it must be that p lies outside of the ball $B_{c',(1+\epsilon_i/2)r'}$. (Otherwise, by Lemma 7, the current set X is an ϵ_i-core-set, since $\delta \leq \epsilon^2/163 \leq \epsilon_i^2/163$, meaning that the round is over.) Thus, by Lemma 5, we know that p must lie outside of $B_{c,(1+\epsilon_i/3)r}$. Then, by Lemma 3, we know that the radius of MEB(X) goes up by at least $r\epsilon_i^2/33$ with the addition of p to X. The radius r goes up with each iteration of the round i; thus, at each iteration r is bounded below by r_{i-1}. Since the round starts with a set X_{i-1} whose MEB radius is r_{i-1} and ends with a set X_i whose MEB radius is r_i, with each iteration increasing the radius by at least $r_{i-1}\epsilon_i^2/33$, we know that the total number τ_i of iterations during round i obeys

$$\tau_i \leq \frac{r_i - r_{i-1}}{r_{i-1}\epsilon_i^2/33} = 33\left(\frac{r_i}{r_{i-1}} - 1\right)2^i \leq 33\left(\frac{\Delta}{\Delta/2\sqrt{3}} - 1\right)2^i = 33\left(2\sqrt{3} - 1\right)2^i,$$

where Δ is the diameter of the set S, and we have used the facts that $r_i \leq \Delta$ and $r_{i-1} \geq \Delta/2\sqrt{3}$. Finally, this implies that the total number of iterations over all rounds is

$$|X_m| = 2 + \sum_{i=1}^{m} \tau_i = O(2^m) = O(\tfrac{1}{\epsilon}).$$

□

Proof of Theorem 1. Since the size of the core-set is $O(1/\epsilon)$, each call to our SOCP solver takes time $O(\frac{d^2}{\sqrt{\epsilon}}\left(\frac{1}{\epsilon} + d\right)\log\frac{1}{\epsilon})$. We parse through the input $O(1/\epsilon)$ times. At

each iteration, it takes $O(nd)$ time to identify the furthest point. Therefore, the total running time is $O(\frac{nd}{\epsilon} + \frac{d^2}{\epsilon^{3/2}}\left(\frac{1}{\epsilon} + d\right)\log\frac{1}{\epsilon})$. Putting $d = O(1/\epsilon)$, as in [5], we get a total time bound of $O(\frac{nd}{\epsilon} + \frac{1}{\epsilon^{4.5}}\log\frac{1}{\epsilon})$. □

Remark 1: The improved core-set bound of Theorem 2 gives, as an immediate consequence, also improved time bounds over those of [5] for 2-center clustering (improving $2^{O(1/\epsilon^2)}dn$ to $2^{O(1/\epsilon)}dn$) and for k-center clustering (improving $2^{O((k/\epsilon^2)\log k)}dn$ to $2^{O((k/\epsilon)\log k)}dn$).

Remark 2: The time bound of Theorem 1 can be further reduced to $O\left(\frac{nd}{\epsilon} + \frac{1}{\epsilon^4}\log^2\frac{1}{\epsilon}\right)$ by using a recent algorithm due to Har-Peled [29], which can compute a $(1+\epsilon)$-approximation to the minimum enclosing ball of n points in d dimensions in $O(\frac{nd}{\epsilon}\log^2\frac{1}{\epsilon})$ time. This is slightly better than our running time and does not use SOCP.

Remark 3: The conference version of this paper had all the experiments done with δ set to $O(\epsilon)$ instead of $O(\epsilon^2)$. For the purposes of experimentation, this is not really an issue, since in most cases setting δ anywhere below ϵ results in the same radius and core-set, as we have found experimentally. However, from a theoretical perspective, there was an oversight in [30], in that our analysis was based on the assumption that the SOCP solver returned an exact MEB. We have addressed this issue here. We note that a similar oversight apparently occurs in the first core-set paper [5], in which the ellipsoid algorithm is called with δ set to $O(\epsilon)$ instead of $O(\epsilon^2)$. A more careful analysis, such as the approach we present here, is needed in order to guarantee that the algorithm for k-center clustering of [5] indeed yields a $(1+\epsilon)$-approximate solution, given that the convex programming techniques give an inexact solution.

5. IMPLEMENTATION AND EXPERIMENTS

We have implemented Algorithm 1 and report below results of experimentation with it. For comparison, we have also implemented a second algorithm, based on a variant of Algorithm 1, which we devised in an attempt to improve the running time of Algorithm 1 in practice. We refer to the (original) implementation of Algorithm 1 as the *pure implementation* and refer to the variant as the *fast implementation*.

The fast implementation attempts to address the main bottleneck in Algorithm 1, which we found to be the time spent in calls to the SOCP solver that computes the (approximate) minimum enclosing ball of the set X after each new point (the furthest outlier) is added. With each iteration, this computation is performed from scratch. In an attempt to compute the "easy" core-set points more quickly, and reduce the number of calls to the SOCP solver, we developed our fast implementation based on a hybrid algorithm that combines our Algorithm 1 with some ideas of Bădoiu and Clarkson [6]. In their simple gradient-descent method, at each iteration the current center is shifted towards the furthest outlier, resulting in a sequence of centers that converge to the center of the minimum enclosing ball. Bădoiu and Clarkson establish that a simple updating scheme returns a $(1+\epsilon)$-approximation in $O(nd/\epsilon^2)$ iterations. As ϵ decreases, the running time deteriorates (in practice), with the $1/\epsilon^2$ term becoming quite significant. On the other hand, we find that our Algorithm 1 performs well for small ϵ, even better than theoretical worst-case

analysis suggests. Thus, in order to maintain the advantages of both algorithms, in our fast implementation we first apply the algorithm of Bădoiu and Clarkson for a prespecified number of iterations and record the furthest outliers at each iteration; our experiments show that $d/3$ is a good choice for the number of these iterations. We then apply our Algorithm 1 as a second phase, using the initial choice of X to be the set of points that show up as furthest outliers in the $d/3$ iterations of the first phase. We note that some of the points may appear as furthest outliers in more than one iteration of the first phase. This often means that the initial size of X is smaller than the number of iterations in the first phase; e.g., $|X| = 30$ after the first phase of 85 iterations on the USPS data set (see Figure 7). The fast implementation has the potential advantage of obtaining quickly a fairly good approximation to the core set using a simple algorithm (not based on an SOCP solver); then, only a few more iterations of Algorithm 1 (using an SOCP solver) are usually needed to complete the core-set computation.

Most of our code is written in Matlab. However, in order to enhance the performance, some of the subroutines (e.g., computing the furthest outlier) were written in C and linked to the Matlab code using mex files. Our software is fairly compact and is available on the web[6]. The current implementation takes only point sets as input; extending it to input sets of balls should be relatively straightforward.

For the SOCP component of the algorithm, we considered two leading SOCP solvers that are freely available: SeDuMi [31] and SDPT3 [32]. Experimentation showed SDPT3 to be superior to SeDuMi for use in our application, so our results here are reported using SDPT3. We refer the reader to the web site[7] maintained by Hans Mittelmann for an independent benchmarking of a variety of optimization codes.

In an attempt to minimize the size of the core-set, our implementations find the furthest outlier at each iteration. We should emphasize, however, that the running times of our algorithms can be reduced by introducing random sampling at the outlier detection stage (see, e.g., Pellegrini [27]), at the expense of slightly larger core-sets for large-scale problems.

Another desirable property of our implementation is that it is I/O-efficient if we assume that we can solve $O(1/\epsilon)$-size subproblems in internal memory. (This was always the case for our experiments, since the size of the core-set did not even approach $1/\epsilon$ in practice.) With this assumption, the current implementation in the I/O model does at most $O(nd/B\epsilon)$ I/Os, where B denotes the disk block size, and the same bound also generalizes to the cache-oblivious model [12]. We believe that with an efficient implementation (e.g., in C++) of our algorithm, very large problems ($n \approx 10^7, d \approx 10^4, \epsilon \approx 10^{-5}$) are tractable to solve in practice on current state-of-the-art systems with sufficient memory and hard disk space.

Platform. All of the experimental results reported in this paper were done on two platforms. The main platform we used to do the experiments was a dual-processor Intel(R) Xeon(TM) 2.66GHz system with 2GB RAM, running Windows XP/Matlab 6.5.0 Release 13. Unfortunately, Matlab is a single-threaded application, so was using only 1 CPU. Figures 3–9 correspond to experiments on this platform.

[6] http://www.compgeom.com/meb/

[7] http://plato.asu.edu/bench.html

Figures 10 and 11 were generated from experiments done on a Pentium III 1Ghz, 512MB notebook computer, running Windows 2000. All of the experiments in the conference version of this paper [30] were conducted also on this platform. Our new implementation, on the new platform (Xeon 2.66GHz), resulted in significantly different results than were reported in [30]; thus, all of the results reported here (except Figures 10 and 11) are new, using the Xeon 2.66GHz platform. Figures 10 and 11 report comparison results of our algorithm with two others; due to some software availability issues, we have not yet been able to conduct the same comparison on the new platform, but we expect the relative performances to be comparable.

Datasets. Most of our experiments were conducted on randomly generated point data, according to various distributions. We also experimented with the USPS data,[8] which is a dataset of feature vectors extracted from handwritten characters, made available by the US Postal service. The USPS data contains 7291 points in 256 dimensions and is a standard data set used in the clustering and machine learning literature. For generating random point data, we used Matlab to generate random matrices, with `rand` for uniformly distributed data, `randn` for normally distributed data, and `random` for other specific distributions. Specifically, we considered the following four classes of point data:

—uniformly distributed within a unit cube;

—uniformly distributed on the vertices of a unit cube;

—normally distributed in space, with each coordinate chosen independently according to a normal distribution with mean 0 and variance 1;

—Poisson distributed, with each coordinate drawn from a Poisson distribution with parameter $\lambda = 1$.

Methods for comparison. Bernd Gärtner [17] provides a code on his web site that we used for comparison. We also used the CGAL 2.4 implementation (based on Welzl's algorithm with move-to-front for small instances and a heuristic for large instances). We were not able to compile code available from David White's web page[9]. We were unable to replicate the timings reported in the paper of Gärtner and Schönherr [18], since the version of the implementation of their algorithm in CGAL 2.4 was not robust. While preparing this paper, a recently updated version of the implementation became available in the latest release of CGAL; in future work, we will be conducting experiments for comparison with it.

Fischer, Gärtner and Kutz [16] recently presented a very fast exact method to solve the MEB problem for points. A direct comparison of running times of their method to our method does not seem appropriate, since we compute approximate solutions while they compute exact solutions. Our implementation is in Matlab (which prevents us from doing experiments on very large data sets in very high dimension), while their implementation is in C++. A theoretical drawback of their simplex approach for solving MEBs is that a polynomial running time cannot be

[8] `http://www.kernel-machines.org/data/ups.mat.gz`, 29MB
[9] `http://vision.ucsd.edu/~dwhite`

guaranteed, although in practice they show that the method is very fast, apparently much faster than the earlier implementation of our algorithm reported in [30].

Experimental results. We begin with a comparison of the fast implementation of Algorithm 1 with the pure implementation of Algorithm 1. In Figures 1 and 2 we show how the running times and the core-set sizes vary with the dimension, for $n = 10^4$ points that are normally distributed (with mean $\mu = 0$ and variance $\sigma = 1$) and $\epsilon = 0.001$. Note that the fast implementation generates core-sets of essentially the same size as the pure implementation, but does so much more quickly.

In Figure 3 we show how the running time of our fast implementation of Algorithm 1 varies with dimension, for n points that are normally distributed (with mean $\mu = 0$ and variance $\sigma = 1$) and $\epsilon = 0.001$. The plot shows two choices of n: $n = 10^4$ and 10^5. Corresponding to the same experiment, also with the fast implementation, Figure 4 shows how the core-set size varies with dimension.

In Figures 5 and 6 we show how the running time and the core-set size varies with dimension for each of the four distributions of $n = 10^4$ input points, with $\epsilon = 0.001$. Notable are the timings for points randomly chosen from the vertices of a hypercube; this distribution of cospherical points represented the most time-consuming instances for the algorithm. While we do not fully understand the non-monotone behavior with respect to dimension in this case, it seems to be related to the similar phenomenon observed and discussed (briefly) in [16].

We note that, while the core-set size is seen to increase somewhat with dimension, the observed size of the core-set in all of our experiments is substantially less than the worst-case (dimension-independent) theoretical upper bound of $O(1/\epsilon)$. The upper bound, $\lceil 1/\epsilon \rceil$, of [7] is 1000 (for our choice of $\epsilon = 0.001$), while the core-set sizes experimentally are observed to be in the range 30-170.

Figures 7 and 8 plot the running times and core-set sizes, as a function of $\log_2(1/\epsilon)$, for points that correspond to feature vectors extracted from handwritten characters, provided by the USPS. For comparison, we also plot the results for normally distributed points of the same dimension, $d = 256$. On the USPS data, the core-set size increases approximately logarithmically in $1/\epsilon$, as compared with the theoretical linear upper bound $O(1/\epsilon)$. Note that the fast implementation always returns a core-set of size 30, even though it runs for 85 iterations. We also noted that the fast implementation did not add any points to the core-set in the second phase; all 30 points were inserted during the first phase.

In Figure 9 we show timing results for low dimensions ($d = 2, 3$), as a function of $\log_{10} n$, for n normally distributed points. Plots are shown with two choices of ϵ ($\epsilon = 10^{-3}$ and $\epsilon = 10^{-6}$); however, the running times are essentially independent of the choice of ϵ. In all of these experiments, the core-set size was always less than 10. This suggests that approximate 2-center clusterings may be computed in time $O(2^{10}n)$ for low dimensions. It would be interesting to see if this result can lead to a truly practical method for approximate 2-center clustering in low dimensions; such a method may have applications in constructing effective hierarchies of bounding spheres.

Figure 10 shows a timing comparison between our algorithm, the CGAL 2.4 implementation, and Bernd Gärtner's code available from his website. (The experiments are done on a Pentium III 1Ghz, 512MB notebook computer, running

Windows 2000.) Both of these codes assume that the dimension of the input point set is fixed, and each has a threshold dimension beyond which the computation time is seen to increase sharply.

In Figure 11 we compare the running times, as a function of dimension, of our pure implementation (using $\epsilon = 0.001$) with the simple method of Bădoiu and Clarkson [6] (based on Claim 3.1 in that paper), using three choices of ϵ ($\epsilon = 0.1, 0.05, 0.03$). These experiments are done on a set of $n = 1000$ points that are normally distributed ($\mu = 0$, $\sigma = 1$). (The experiments are done on a Pentium III 1Ghz, 512MB notebook computer, running Windows 2000.) Note that for $\epsilon = 0.03$, the Bădoiu-Clarkson algorithm is already very slow compared to Algorithm 1. We have not yet implemented the main algorithm proposed in [6], which has a slightly lower running time ($O(nd/\epsilon + (1/\epsilon)^5)$) than our Algorithm 1, but it seems that when ϵ is small, its running time may suffer because of the base case solver (the simple method, based on Claim 3.1, which we tested). We suspect that the improved algorithm suggested by Har-Peled [29] (Remark 2, Section 4) is a better candidate for implementation.

Finally, we remark that in all of our experiments, we set $\delta = \epsilon^2$, ignoring the constant that we derived in the theoretical analysis. In justification of this choice, we verified first experimentally that varying δ had little or no effect: Running times varied only slightly, and core-set sizes and MEB radii did not change at all. For instance, for $n = 5000$ points in dimension $d = 500$, with $\epsilon = 0.001$, we varied δ from 10^{-3} to 10^{-9} for two different distributions of input points. For a set of points generated from a normal distribution, for all choices of δ, the core-set size was 75, the radius was 24.094 and the running time varied from 81.328 seconds to 81.922 seconds. For a set of points generated randomly from the vertices of a hypercube, the core-set size and the radius were again constant for all choices of δ, while the running time varied from 691.7 to 669.64 seconds. (Note that actually it took less time to compute using $\delta = 10^{-9}$ than using $\delta = 10^{-3}$.) In fact, we have not yet found a point set on which setting $\delta = \epsilon^2$ instead of $\delta = \epsilon$ made any change in either the core-set size or the MEB radius. This suggests that our theoretical analysis justifying the choice of δ to guarantee a $(1 + \epsilon)$-approximation is in fact overly conservative.

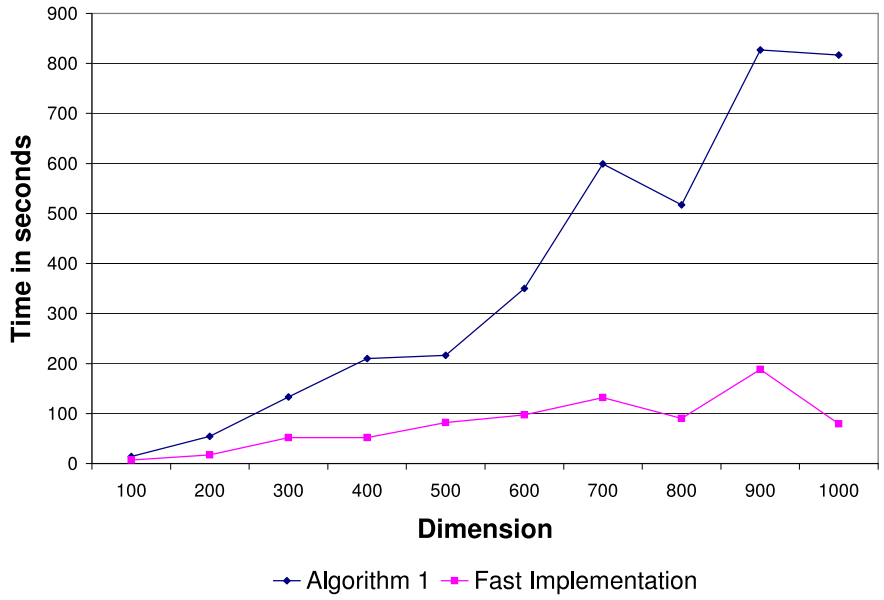

Fig. 1. Running time (in seconds) as a function of dimension, for $n = 10^4$ input points that are normally distributed ($\mu = 0, \sigma = 1$). For comparison, we plot both the pure implementation ("Algorithm 1") and the fast implementation. Here, $\epsilon = 0.001$.

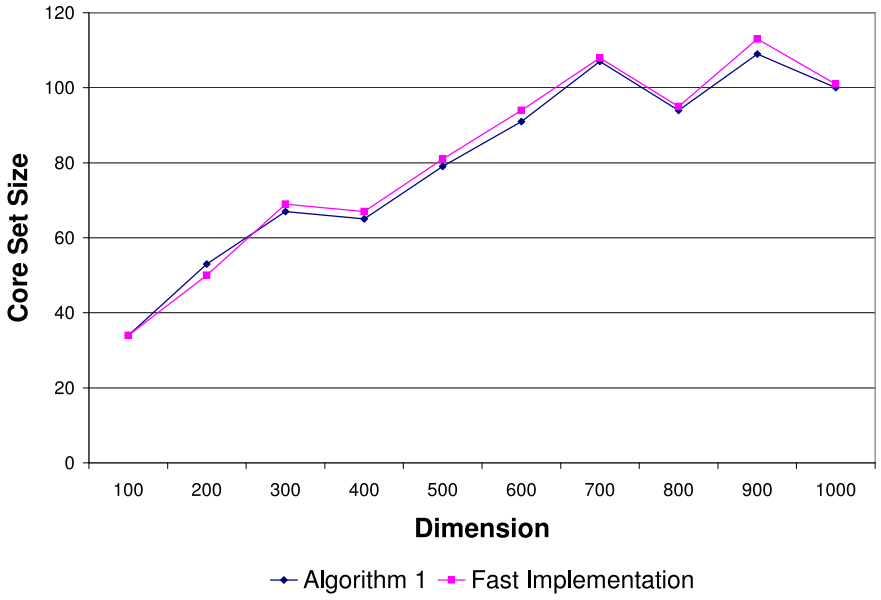

Fig. 2. Core-set size as a function of dimension, for $n = 10^4$ input points that are normally distributed ($\mu = 0, \sigma = 1$). For comparison, we plot both the pure implementation ("Algorithm 1") and the fast implementation. Here, $\epsilon = 0.001$.

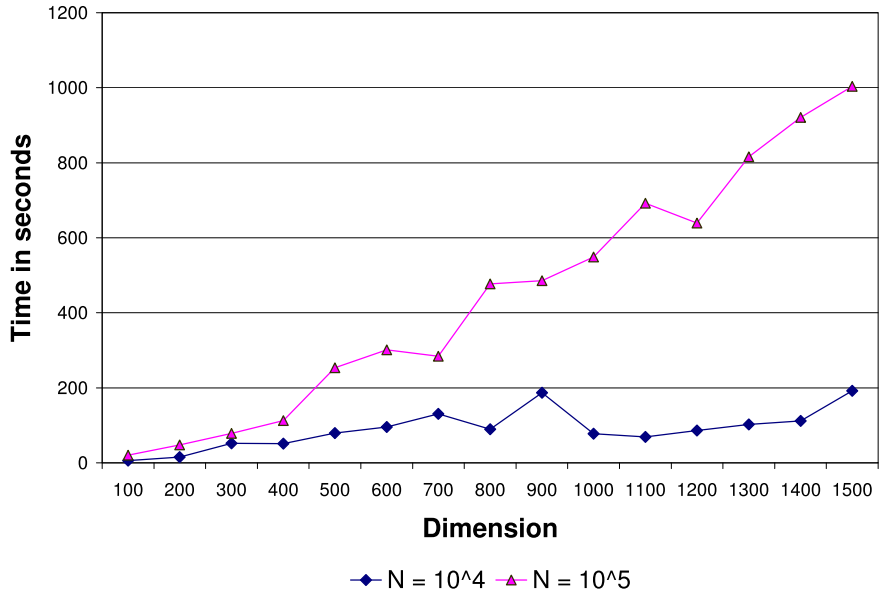

Fig. 3. Running time (in seconds) of the fast implementation of Algorithm 1 as a function of dimension, for two choices of n ($n = 10^4$, $n = 10^5$). Here, $\epsilon = 0.001$, and the input points are normally distributed ($\mu = 0, \sigma = 1$).

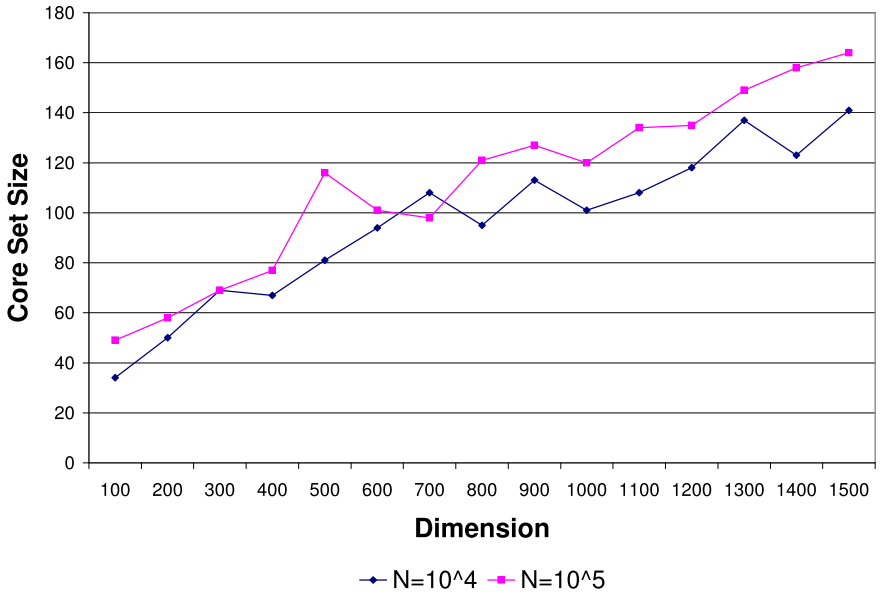

Fig. 4. Core-set size as a function of dimension, for two choices of n ($n = 10^4$, $n = 10^5$). Here, $\epsilon = 0.001$, and the input points are normally distributed ($\mu = 0, \sigma = 1$). The fast implementation of Algorithm 1 was used in this experiment.

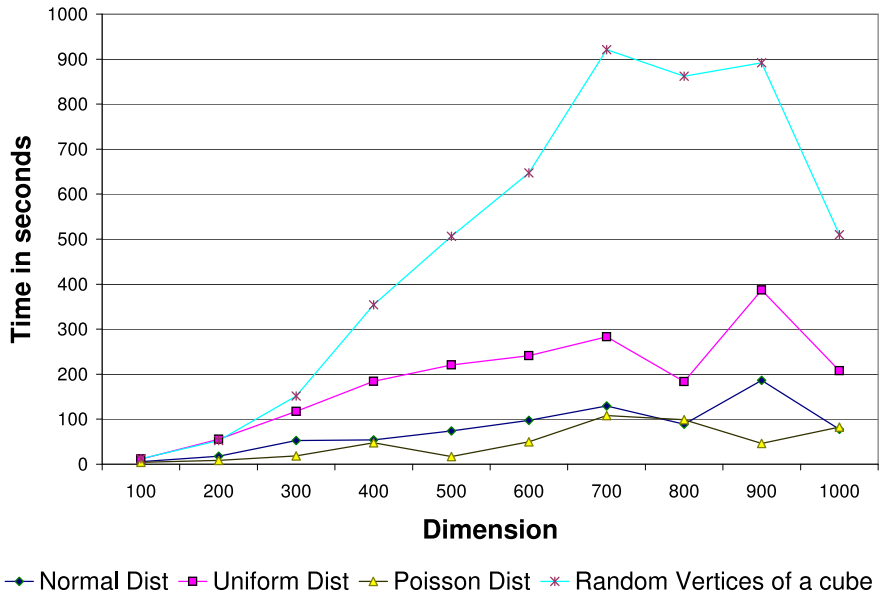

Fig. 5. Running time (in seconds) of the fast implementation of Algorithm 1 as a function of dimension, for $n = 10^4$ input points from each of four distributions: uniform, normal, Poisson, and random vertices of a cube. Here, $\epsilon = 0.001$.

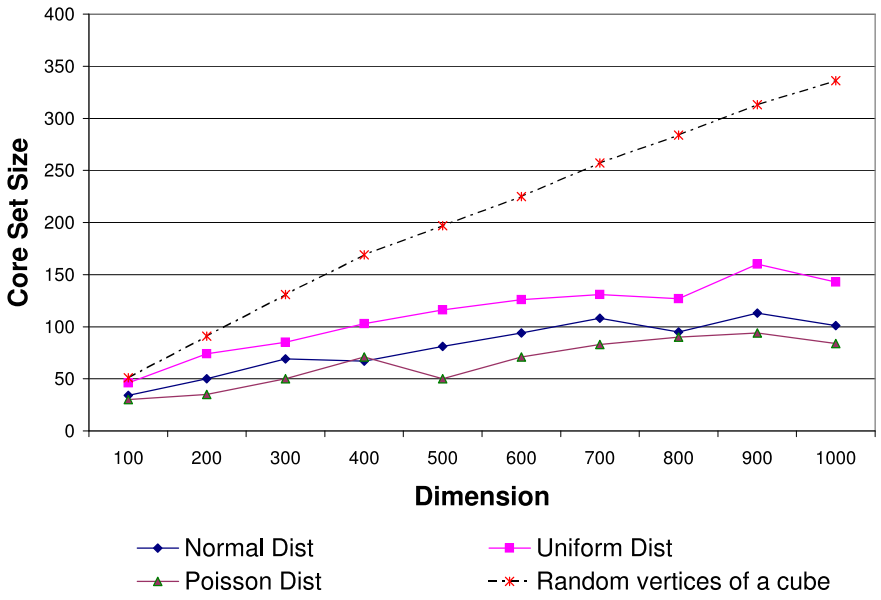

Fig. 6. Core-set size for the fast implementation of Algorithm 1 as a function of dimension, for $n = 10^4$ input points from each of four distributions: uniform, normal, Poisson, and random vertices of a cube. Here, $\epsilon = 0.001$.

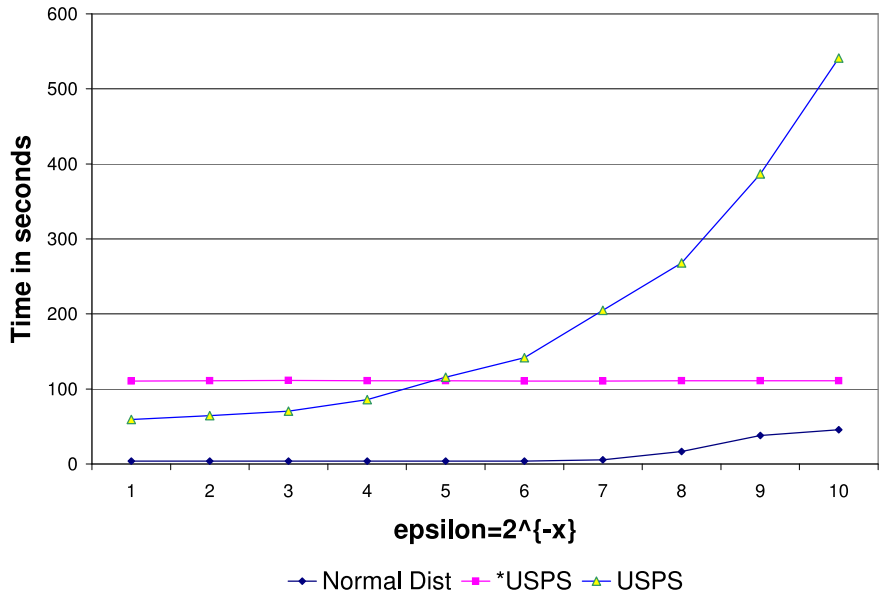

Fig. 7. Running time (in seconds) of the fast implementation of Algorithm 1 as a function of $\log_2(1/\epsilon)$, for input points that are normally distributed ($\mu = 0, \sigma = 1$) in dimension $d = 256$ and for input points from the USPS. For the normally distributed points, the fast implementation is used. For the USPS data, we plot results both for the pure implementation (indicated by "USPS") and for the fast implementation (indicated by "*USPS"). The USPS data contains 7291 points in 256 dimensions and is a standard data set, based on digitized hand-written characters, used in the clustering and machine learning literature.

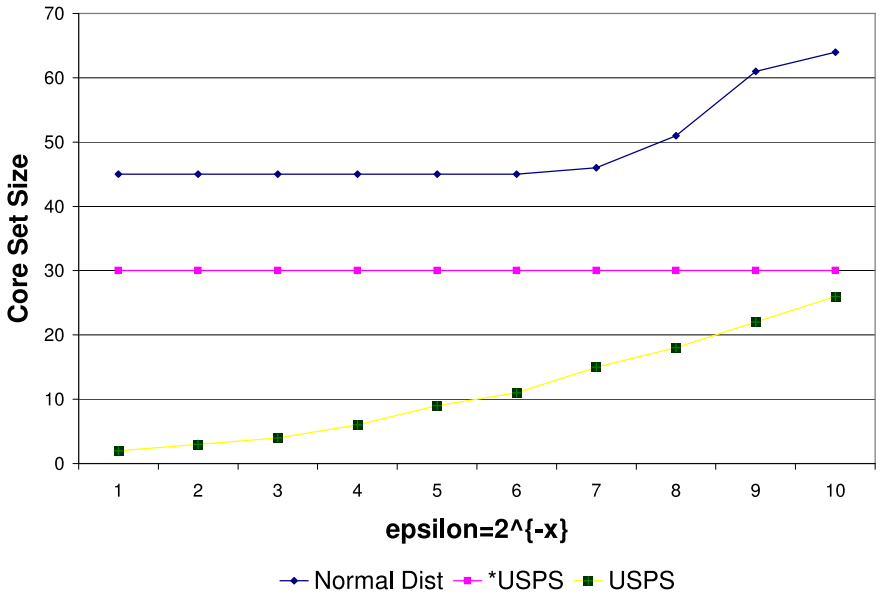

Fig. 8. Core-set size as a function of $\log_2(1/\epsilon)$, for input points that are normally distributed ($\mu = 0, \sigma = 1$) in dimension $d = 256$ and for input points from the USPS. For the normally distributed points, the fast implementation is used. For the USPS data, we plot results both for the pure implementation (indicated by "USPS") and for the fast implementation (indicated by "*USPS").

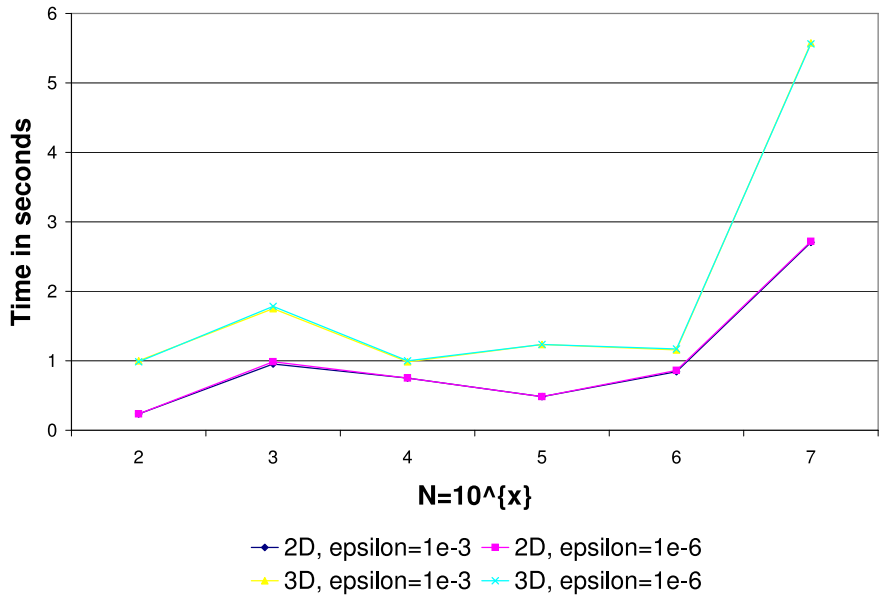

Fig. 9. Running time (in seconds) of the fast implementation of Algorithm 1 as a function of $\log_{10} n$ for n input points that are normally distributed ($\mu = 0, \sigma = 1$) in dimensions $d = 2$ and $d = 3$. For each choice of d, plots are shown for two choices of ϵ ($\epsilon = 10^{-3}$ and $\epsilon = 10^{-6}$), but they are essentially identical, with no discernible difference. In every case, the core-set size was less than 10.

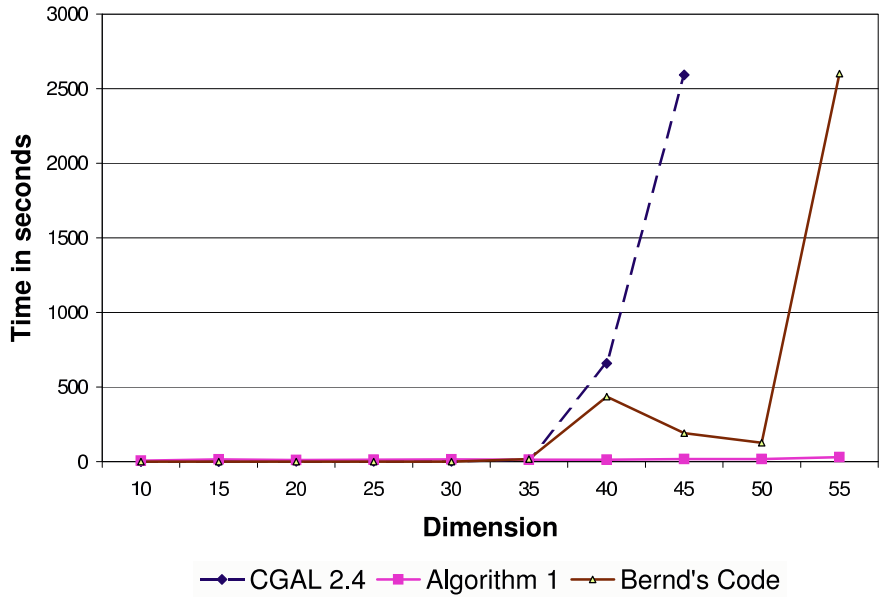

Fig. 10. Timing comparison, as a function of dimension, for $n = 1000$ points that are normally distributed ($\mu = 0, \sigma = 1$). We compare the *pure implementation* of Algorithm 1 with CGAL 2.4 and with Bernd Gärtner's code. Here, $\epsilon = 10^{-6}$. These experiments were done on a Pentium III 1Ghz, 512MB notebook computer, running Windows 2000.

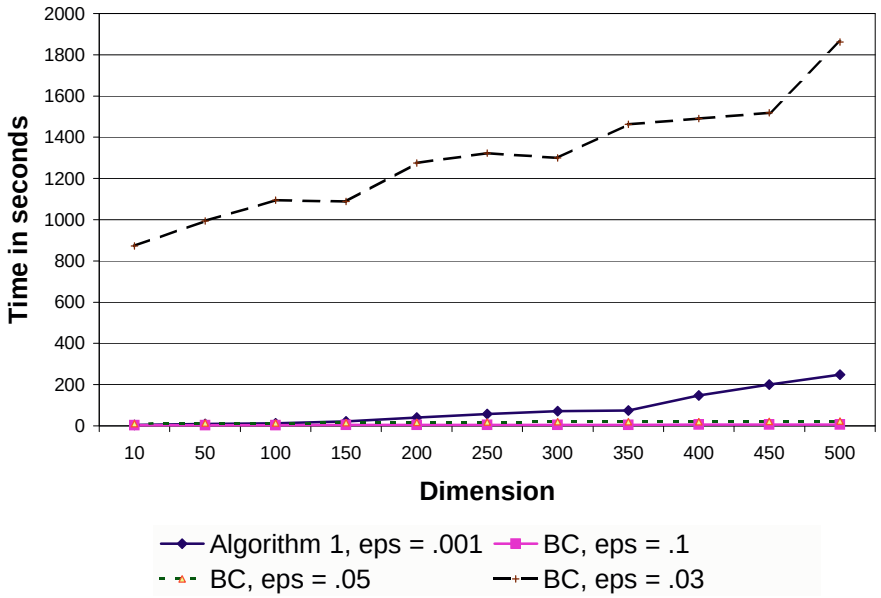

Fig. 11. Timing comparison, as a function of dimension, for $n = 1000$ points that are normally distributed ($\mu = 0, \sigma = 1$). We compare the *pure implementation* of Algorithm 1 (using $\epsilon = 0.001$) with the simple method of Bădoiu and Clarkson (BC), using three choices of ϵ ($\epsilon = 0.1, 0.05, 0.03$). These experiments were done on a Pentium III 1Ghz, 512MB notebook computer, running Windows 2000. As ϵ approaches zero, the performance of the BC algorithm degrades substantially.

6. OPEN PROBLEMS

There are many interesting theoretical and practical problems that are motivated by our investigations.

(1) Several problems deserve experimental study. Are there practical methods for computing an MEB with outliers? Can one practically compute approximate solutions for 2-center and k-center problems in high dimensions? How efficiently can one compute approximate minimum-volume enclosing ellipsoids? Are there core-sets for ellipsoids of size less than $\Theta(d^2)$? Does dimension reduction [21] help us to solve high-dimensional problems in practice? Can one use "warm start" strategies [36] to improve running times by giving a good starting point at every iteration? How does the improved algorithm, with running time $O\left(\frac{nd}{\epsilon} + \frac{1}{\epsilon^4}\log^2\frac{1}{\epsilon}\right)$, suggested by Har-Peled [29] in Remark 2 of Section 4, compare with our implementations based on Algorithm 1?

(2) For which LP-type problems can one prove the existence of dimension-independent core-sets? What are the tightest core-set bounds one can prove for various distributions of points, with $d < 1/\epsilon$?

ACKNOWLEDGMENTS

We thank the referees for a careful reading of the paper and for many suggestions that improved the presentation. We thank Sariel Har-Peled and Edgar Ramos for several helpful discussions. We thank Sachin Jambawalikar for help with importing the USPS data set in the code.

REFERENCES

[1] F. Alizadeh and D. Goldfarb. Second-order cone programming. *Mathematical Programming, Series B* 95/1: 3–51, 2003.

[2] N. Alon, S. Dar, M. Parnas and D. Ron. Testing of clustering. In *Proc. 41st Annual Symposium on Foundations of Computer Science*, pages 240–250. IEEE Computer Society Press, Los Alamitos, CA, 2000.

[3] Y. Bulatov, S. Jambawalikar, P. Kumar and S. Sethia. Hand recognition using geometric classifiers. Manuscript, 2002.

[4] A. Ben-Hur, D. Horn, H. T. Siegelmann and V. Vapnik. Support vector clustering. In *Journal of Machine Learning*, 2002.

[5] M. Bădoiu, S. Har-Peled and P. Indyk. Approximate clustering via core-sets. *Proceedings of 34th Annual ACM Symposium on Theory of Computing*, pages 250–257, 2002.

[6] M. Bădoiu and K. L. Clarkson. Smaller core-sets for balls. In *Proceedings of 14th ACM-SIAM Symposium on Discrete Algorithms*, pages 801–802, 2003.

[7] M. Bădoiu and K. L. Clarkson. Optimal core-sets for balls. Manuscript, 2002.

[8] C. J. C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2(2):121–167, 1998.

[9] T. M. Chan. Approximating the diameter, width, smallest enclosing cylinder, and minimum-width annulus. In *Proceedings of 16th Annual ACM Symposium on Computational Geometry*, pages 300–309, 2000.

[10] O. Chapelle, V. Vapnik, O. Bousquet and S. Mukherjee. Choosing multiple parameters for support vector machines. *Machine Learning*, 46(1/3):131, 2002.

[11] O. Eğecioğlu and B. Kalantari. Approximating the diameter of a set of points in the Euclidean space. *Information Processing Letters*, 32:205-211, 1989.

[12] M. Frigo, C. E. Lieserson, H. Prokop and S. Ramachandran. Cache oblivious algorithms. *Proceedings of 40th Annual Symposium on Foundations of Computer Science*, 1999.

[13] D. J. Elzinga and D. W. Hearn. The minimum covering sphere problem. *Management Science*, 19(1):96–104, 1972.

[14] K. Fischer. Smallest enclosing ball of balls. Diploma thesis, Institute of Theoretical Computer Science, ETH Zurich, 2001.

[15] K. Fischer and B. Gärtner The smallest enclosing ball of balls: Combinatorial structure and algorithms. In *Proceedings of 19th Annual ACM Symposium on Computational Geometry*, pages 292–301, 2003.

[16] K. Fischer, B. Gärtner and M. Kutz Fast smallest-enclosing-ball computation in high dimensions In *Proceedings of the 11th Annual European Symposium on Algorithms (ESA)*, pages 630–641, LNCS 2832, Springer-Verlag, 2003.

[17] B. Gärtner. Fast and robust smallest enclosing balls. In *Proceedings of 7th Annual European Symposium on Algorithms (ESA)*, pages 325–338, LNCS 1643, Springer-Verlag, 1999.

[18] B. Gärtner and S. Schönherr. An efficient, exact, and generic quadratic programming solver for geometric optimization. In *Proceedings of 16th Annual ACM Symposium on Computational Geometry*, pages 110–118, 2000.

[19] A. Goel, P. Indyk and K. R. Varadarajan. Reductions among high dimensional proximity problems. In *Poceedings of 13th ACM-SIAM Symposium on Discrete Algorithms*, pages 769–778, 2001.

[20] P. M. Hubbard. Approximating polyhedra with spheres for time-critical collision detection. *ACM Transactions on Graphics*, 15(3):179–210, 1996.

[21] W. Johnson and J. Lindenstrauss Extensions of Lipschitz maps into a Hilbert space. *Contemp. Math.* 26, pages 189–206, 1984.

[22] M. S. Lobo, L. Vandenberghe, S. Boyd and H. Lebret. Applications of second-order cone programming. *Linear Algebra and Its Applications*, 248:193–228, 1998.

[23] J. Matoušek, M. Sharir and E. Welzl. A subexponential bound for linear programming. In *Proceedings of 8th Annual ACM Symposium on Computational Geometry*, pages 1–8, 1992.

[24] Y. E. Nesterov and A. S. Nemirovskii. *Interior Point Polynomial Methods in Convex Programming*. SIAM Publications, Philadelphia, 1994.

[25] Y. E. Nesterov and M. J. Todd. Self-scaled barriers and interior-point methods for convex programming. *Mathematics of Operations Research*, 22:1–42, 1997.

[26] Y. E. Nesterov and M. J. Todd. Primal-dual interior-point methods for self-scaled cones. *SIAM Journal on Optimization*, 8:324–362, 1998.

[27] M. Pellegrini. Randomized combinatorial algorithms for linear programming when the dimension is moderately high. In *Proceedings of 13th ACM-SIAM Symposium on Discrete Algorithms*, 2001.

[28] J. Renegar. A Mathematical View of Interior-Point Methods in Convex Optimization. *MPS/SIAM Series on Optimization 3*. SIAM Publications, Philadelphia, 2001.

[29] S. Har-Peled. Manuscript, 2002. Personal communication.

[30] P. Kumar, J. S. B. Mitchell and A. Yıldırım. Computing Core-Sets and Approximate Smallest Enclosing HyperSpheres in High Dimensions. In *Proc. Fifth Workshop on Algorithm Engineering and Experiments*, Richard E. Ladner, ed., pages 45–55, SIAM, 2003.

[31] J. F. Sturm. Using SeDuMi 1.02, a MATLAB toolbox for optimization over symmetric cones. *Optimization Methods and Software*, 11/12:625–653, 1999.

[32] K. C. Toh, M. J. Todd and R. H. Tütüncü. SDPT3 — a Matlab software package for semidefinite programming. *Optimization Methods and Software*, 11:545–581, 1999. `http://www.math.nus.edu.sg/~mattohkc/sdpt3.html`.

[33] R. H. Tütüncü, K. C. Toh and M. J. Todd. Solving semidefinite-quadratic-linear programs using SDPT3. *Mathematical Programming, Series B*, 95/2:189–217, 2003.

[34] E. Welzl. Smallest enclosing disks (balls and ellipsoids). In Maurer, H., ed.: New Results and New Trends in Computer Science, pages 359–370, LNCS 555, Springer-Verlag, 1991.

[35] S. Xu, R. Freund and J. Sun. Solution methodologies for the smallest enclosing circle problem. Technical report, Singapore-MIT Alliance, National University of Singapore, Singapore, 2001.

[36] E. A. Yıldırım and S. J. Wright Warm-Start Strategies in Interior-Point Methods for Linear Programming. *SIAM Journal on Optimization* 12/3, pages 782–810, 2002.

[37] G. Zhou, J. Sun and K.-C. Toh. Efficient algorithms for the smallest enclosing ball problem in high dimensional space. Technical report, 2002. To appear, Proceedings of Fields Institute of Mathematics.

Received March 2003; accepted ??????