Computing Core-Sets and Approximate Smallest Enclosing HyperSpheres in High Dimensions*

Piyush Kumar† Joseph S. B. Mitchell‡ E. Alper Yıldırım§

Abstract

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.

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], doing fast farthest neighbor query approximation [17], k-center clustering [5], testing of radius clustering for $k = 1$ [2], approximate 1-cylinder problem [5], computation of spatial hierarchies (e.g., sphere trees [18]), and other applications [13].

In this paper, we give improved time bounds for approximation algorithms for the MEB problem, applicable to an input set of points or balls in high dimensions. We prove a time bound of $O\left(\frac{nd}{\epsilon} + \frac{1}{\epsilon^{4.5}} \log \frac{1}{\epsilon}\right)$, 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 also demonstrate that the sizes of the core-sets tend to be much smaller than the worst-case theoretical upper bounds.

Preliminaries. 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, 33]; if B_1 and B_2 are two different smallest 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 $S \subset B_{c,(1+\epsilon)r}$.

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

Given $\epsilon > 0$, a subset, $X \subseteq S$, is said to be a *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 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 [21, 14]. One of the best implementable solutions to compute MEB exactly in moderately high dimensions is given by Gärtner and Schönherr [16]; the largest instance of MEB they solve is $d = 300$, $n = 10000$ (in about 20 minutes on their platform). In comparison, the largest instance we solve $(1 + \epsilon)$-approximately is $d = 1000$, $n = 100000$, $\epsilon = 10^{-3}$;

*The code associated with this paper can be downloaded from `http://www.compgeom.com/meb/`. This research was partially supported by a DARPA subcontract from HRL Laboratories and grants from Honda Fundamental Research Labs, NASA Ames Research (NAG2-1325), NSF (CCR-9732220, CCR-0098172), and Sandia National Labs.

†Stony Brook University, `piyush@acm.org`. Part of this work was done while the author was visiting MPI-Saarbrücken.

‡Stony Brook University, `jsbm@ams.sunysb.edu`.

§Stony Brook University, `yildirim@ams.sunysb.edu`.

in this case the virtual memory was running low on the system[1]. Another implementation of an exact solver is based on the algorithm of Gärtner [15]; this code is part of the CGAL[2] library. For large dimensions, our approximation algorithm is found to be much faster than this exact solver.

We are not aware of other implementations of polynomial-time approximation schemes for the MEB problem.

Independently from our work, the MEB problem in high dimensions was also studied in [33]. 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. Similarly, four algorithms (including a randomized algorithm) are compared in [31] for the computation of the minimum enclosing circle of circles on the plane. Both studies reveal that solving MEB using a direct SOCP formulation suffers from memory problems as the dimension, d, and the number of points, n, increase. This is why we have worked to combine SOCP with core-sets in designing a practical MEB method.

In a forthcoming paper of Bădoiu and Clarkson [6], the authors have independently also obtained an upper bound of $O(1/\epsilon)$ on the size of core-sets and have, most recently [7], proved a worst-case tight upper bound of $\lceil 1/\epsilon \rceil$. Note that the worst case upper bound does not apply to our experiments since in almost all our experiments, the dimension d satisfies $d < \frac{1}{\epsilon}$. The worst case upper bound of [6, 7] only applies to the case when $d \geq \frac{1}{\epsilon}$. Our experimental results on a wide variety of input sets show that the core set size is smaller than $\min(\frac{1}{\epsilon}, d+1)$ (See Figure 2).

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 $O\left(\frac{dn}{\epsilon^2} + \frac{1}{\epsilon^{10}} \log \frac{1}{\epsilon}\right)$ 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 3.2) is $O\left(\frac{nd}{\epsilon} + \frac{1}{\epsilon^{4.5}} \log \frac{1}{\epsilon}\right)$.

Outline of paper. We first show in Section 2 how to use second-order cone programming to solve the MEB problem in $O(\sqrt{n}d^2(n+d)\log(1/\epsilon))$ arithmetic operations. This algorithm is specially suited for problems in which n is small and d is large; thus, we study algorithms to compute core-sets in Section 3 in an effort to select a small subset X, a core-set, that is sufficient for approximation purposes. This section includes our proof of the new upper bound of $O(1/\epsilon)$ on the size of core-sets. Section 4 is devoted to discussion of the experiments and of the 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 can be viewed as an extension of linear programming in which the nonnegative orthant is replaced by the *second-order cone* (also called the "Lorenz cone," or the "quadratic cone"), defined as

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

Therefore, SOCP is essentially linear programming over an affine subset of products of second-order cones. 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., [20, 1]) and due also to the existence of very efficient algorithms to solve this class of optimization problems. In particular, any SOCP problem involving n second-order cones can be solved within any specified additive error $\epsilon > 0$ in $O(\sqrt{n}\log(1/\epsilon))$ iterations by interior-point algorithms [22, 26].

The MEB problem can be formulated as an SOCP problem as

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

where $c_1, \ldots, c_n$ and $r_1, \ldots, r_n$ constitute the centers and the radii of the input set $S \subset \mathbb{R}^d$, respectively, c and r are 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 $i = 1, \ldots, n$). By introducing slack variables

$$(\gamma_i, s_i) \in K, \; i = 1, \ldots, n,$$

the MEB problem can be reformulated in (dual) standard form as

$$\max_{c,r,\gamma,s_1,\ldots,s_n} -r, \quad \text{s.t.} \quad -\begin{bmatrix} r \\ c \end{bmatrix} + \begin{bmatrix} \gamma_i \\ s_i \end{bmatrix} = -\begin{bmatrix} r_i \\ c_i \end{bmatrix},$$

along with the constraints $(\gamma_i, s_i) \in K$, $i = 1, \ldots, n$, where γ denotes the n-dimensional vector whose components are given by $\gamma_1, \ldots, \gamma_n$. The Lagrangian dual is given in (primal) standard form by

$$\begin{aligned} \min_{\sigma,x_1,\ldots,x_n} \quad & -\textstyle\sum_{i=1}^n r_i\sigma_i - \sum_{i=1}^n c_i^T x_i, \\ \text{s.t.} \quad & -\textstyle\sum_{i=1}^n \begin{bmatrix} \sigma_i \\ x_i \end{bmatrix} = \begin{bmatrix} -1 \\ 0 \end{bmatrix}, \\ & (\sigma_i, x_i) \in K, \qquad i = 1, \ldots, n, \end{aligned}$$

1. This instance took approximately 3 hours to solve.
2. `http://www.cgal.org`

where $\sigma := (\sigma_1, \ldots, \sigma_n)^T$.

The most popular and effective interior-point methods are the primal-dual path-following algorithms (see, e.g., Nesterov and Todd [23, 24]). Such algorithms generate interior-points for the primal and dual problems that follow the so-called central path, which converges to a primal-dual optimal solution in the limit. The major work per iteration 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 an approximation, with additive error at most ϵ, to the MEB problem with an interior-point method is $O(\sqrt{n}d^2(n+d)\log(1/\epsilon))$. 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 [30]).

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 [33], 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 column generation approach initially developed for solving large-scale linear programs that have much fewer constraints than variables. The MEB problem formulated in the primal standard form as above precisely satisfies this property since $n \gg d$ for instances of interest in this paper.

We use the column generation approach to be able to solve large-scale MEB instances. 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 would lead to fewer updates. Furthermore, a small core set with a good approximation would yield MEB instances with relatively few points that can efficiently be solved by an interior-point algorithm.

- ▷ *Subproblems:* The performance of a column generation approach is closely related to the efficiency with which each subproblem can be solved. We use state-of-the-art interior-point solver SDPT3 [29] in our implementation.

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

In the following sections, we describe our approach in more detail in light of these three factors.

3 Using Core-Sets for Approximating the MEB

We consider now the problem of computing a MEB of a set $S = \{B_1, B_2, ..., B_n\}$ of n balls in $\mathbb{R}^d$. One can consider the MEB of points to be the special case in which the radius of each ball is zero.

We note that computing the MEB of balls is an LP-type problem [21, 14]; thus, for *fixed* d, it can be computed in $O(n)$ time, where the constant of proportionality depends exponentially on d.

Our goal is to establish the existence of small core-sets for MEB of balls and then to use this fact, in conjunction with SOCP, to compute an approximate MEB of balls quickly, both in theory and in practice.

We begin with a lemma that generalizes a similar result known for MEB of points [5]:

Lemma 3.1. *Let $B_{c,r}$ be the MEB of the set of balls $S = \{B_1, B_2, ..., B_n\}$ in $\mathbb{R}^d$ where $n \geq d+1$. Then any closed halfspace passing through 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 each ball of S touches $\partial B_{c,r}$; any ball strictly interior to $B_{c,r}$ can be deleted without changing its optimality. Further, it is easy to see that there exists a subset $S' \subseteq S$, with $|S'| \leq d+1$, such that MEB(S') = MEB(S), and that c must lie inside the convex hull, Q, of the centers of the balls S'; see Fischer [14]. Consider a halfspace, H, defined by a hyperplane through c. Since $c \in Q$, the halfspace H must contain a vertex of Q, say c', the center of a ball $B_{c',r'}$. Let $p = \overrightarrow{cc'} \cap \partial B_{c,r}$ be the point where the ray $\overrightarrow{cc'}$ exits $B_{c,r}$ and let $q = \overrightarrow{cc'} \cap \partial B_{c',r'}$ be the point where $\overrightarrow{cc'}$ exits $B_{c',r'}$. Then, $\|cp\| = r$. By the triangle inequality, all points of $B_{c',r'} \setminus \{q\}$ are at distance from c at most that of q; thus, since $B_{c',r'}$ touches $\partial B_{c,r}$, we know that $p = q$. ⊡

Our algorithm for computing MEB(S) for a set S of n points or balls begins with an enclosing ball of S based on an approximate diameter of S. If S is a set of points, one can compute a $(1-\epsilon)$-approximation of the diameter, δ, yielding a pair of points at distance at least $(1-\epsilon)\delta$; however, the dependence on dimension d is exponential [9]. For our purposes, it suffices to obtain any constant factor approximation of the diameter δ,

so we choose to use the following simple $O(dn)$ -time method, shown by Eğecioğlu and Kalantari [11] to yield a $\frac{1}{\sqrt{3}}$-approximate diameter of a set S of points: 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.)

Algorithm 1 Outputs a $(1+\epsilon)$-approximation of MEB(S) and an $O(1/\epsilon^2)$ -size core-set

Require: Input set of points $S \in \mathbb{R}^d$, parameter $\epsilon > 0$, subset $X_0 \subset S$

```
X ← X_0
loop
    Compute B_{c,r} = MEB(X).
    if S ⊂ B_{c,(1+ε)r} then
        Return B_{c,r}, X
    else
        p ← point q ∈ S maximizing ||cq||
    end if
    X ← X ∪ {p}
end loop
```

If Algorithm 1 is applied to input data, with $X_0 = \{q, q'\}$ given by the simple $\frac{1}{\sqrt{3}}$-approximation algorithm for diameter, then it will yield an output set, X, that is of size $O(1/\epsilon^2)$, as shown by Bădoiu et al. [5] Their same proof, using Lemma 3.1 to address the case of balls, yields the following:

Lemma 3.2. *[5] For any set S of balls in $\mathbb{R}^d$ and any $0 < \epsilon < 1$ there exists a subset $X \subseteq S$, with $|X| = O(1/\epsilon^2)$, such that the radius of MEB(S) is at most $(1+\epsilon)$ times the radius of MEB(X). In other words, there exists a core-set X of size $O(1/\epsilon^2)$.*

In fact, the proof of Lemma 3.2 is based on showing that each iteration of Algorithm 1 results in an increase of the radius of the current ball, MEB(X), by a factor of at least $(1 + \epsilon^2/16)$; this in turns implies that there can be at most $O(1/\epsilon^2)$ iterations in going from the initial ball of radius at least $\frac{\delta}{\sqrt{3}}$ to the final ball (whose radius is at most δ).

We bootstrap Lemma 3.2 to give a $O(1/\epsilon)$ -size core-set, as shown in Algorithm 2.

Lemma 3.3. *The number of points added to X in round $i+1$ is at most 2^{i+6}.*

Proof. Round i gave as output the set X_i, of radius r_i, which serves as the input core-set to round $(i+1)$. Thus, we know that $(1 + 2^{-i})r_i \geq r^* \geq r_i$. For round $i + 1$, $\epsilon = 2^{-(i+1)}$, so in each iteration of Algorithm 1, the radius goes up by factor $(1 + \epsilon^2/16) = (1 + 2^{-2i-6})$; thus, each iteration increases the radius by at least $2^{-2i-6}r_i$. If in round $i + 1$ there are k_{i+1} points added, the ball at the end of the round now has radius $r_{i+1} \geq r_i + k_{i+1} \cdot 2^{-2i-6}r_i$. Since we know that $(1 + 2^{-i})r_i \geq r^*$ and that $r^* \geq r_{i+1}$, we get that $(1 + 2^{-i})r_i \geq (1 + k_{i+1} \cdot 2^{-2i-6})r_i$, implying that $k_{i+1} \leq 2^{i+6}$ as claimed. ⊞

Algorithm 2 Outputs a $(1+\epsilon)$-approximation of MEB(S) and an $O(1/\epsilon)$ -size core-set

Require: Input set of points $S \in \mathbb{R}^d$, parameter $\epsilon = 2^{-m}$, subset $X_0 \subset S$

```
for i = 1 to m do
    Call Algorithm 1 with input S, ε = 2^{-i}, X_{i-1}
    X_i ← the output core-set
end for
Return MEB(X_m), X_m
```

Theorem 3.1. *The core-set output by Algorithm 2 has size $O(1/\epsilon)$.*

Proof. The size of $|X_m|$ is equal to the sum of the number of points added in each round, which is, by Lemma 3.3, at most $\sum_{i=1}^{m} 2^{i+6} = O(2^m) = O(1/\epsilon)$. ⊞

Theorem 3.2. *A $(1 + \epsilon)$-approximation to the MEB of a set of n balls in d dimensions can be computed in time $O\left(\frac{nd}{\epsilon} + \frac{1}{\epsilon^{4.5}} \log \frac{1}{\epsilon}\right)$.*

Proof. Since the size of the basis (core-set) is $O(\frac{1}{\epsilon})$, each call to our SOCP solver incurs a cost of $O\left(\frac{d^2}{\sqrt{\epsilon}}\left(\frac{1}{\epsilon} + d\right)\log \frac{1}{\epsilon}\right)$. We parse through the input $O(\frac{1}{\epsilon})$ times, so the total cost is $O\left(\frac{nd}{\epsilon} + \frac{d^2}{\epsilon^{3/2}}\left(\frac{1}{\epsilon} + d\right)\log \frac{1}{\epsilon}\right)$. Putting $d = O(1/\epsilon)$, as in [5], we get a total bound of $O\left(\frac{nd}{\epsilon} + \frac{1}{\epsilon^{4.5}} \log \frac{1}{\epsilon}\right)$. ⊞

Remark. The above theorem also improves the best known time bounds for approximation results independent of d on the 2-center clustering ($2^{O(\frac{1}{\epsilon})}dn$) problem and the k-center clustering ($2^{O(\frac{k \log k}{\epsilon})}dn$) problem [5].

4 Implementation and Experiments

We implemented our algorithm in Matlab. The total code is less than 200 lines; it is available at `http://www.compgeom.com/meb/`. No particular attention was given to optimizing the code; our goal was to demonstrate the practicality of the algorithm, even with a fairly straightforward implementation. The current implementation takes only point sets as input; extending it to input sets of balls should be relatively

straightforward. We implemented Algorithm 1 and 2. We also implemented the new gradient descent type algorithm of [6] exactly the way it has been presented in Claim 3.1 of the paper. The problem with this algorithm is that as soon as ϵ decreases, it starts taking too much time. As is predicted by theory, it takes exactly $O(dn/\epsilon^2)$ time. Our algorithm's analysis is not as tight, so it performs much better in practice than predicted by theory as ϵ decreases. Very recently, the running time of this algorithm (Claim 3.1 [6]) was improved to $O(\frac{dn}{\epsilon}\log^2\frac{1}{\epsilon})$ by another algorithm that also avoids quadratic programming [27]. Using this algorithm as the base case solver in Theorem 3.2, the running time can be reduced to $O\left(\frac{nd}{\epsilon}+\frac{1}{\epsilon^4}\log^2\frac{1}{\epsilon}\right)$. This is slightly better than our running time and does not use SOCP. The improved algorithm might be a good competitor to our algorithm in practice.

For the SOCP component of the algorithm, we considered two leading SOCP solvers: SeDuMi [28] and SDPT3 [29]. Experimentation showed SDPT3 to be superior to SeDuMi for use in our application, so our results here are reported using SDPT3.

We found it advantageous to introduce random sampling in the step of the algorithm that searches for the existence of a point that is substantially outside the current candidate ball. This can speed the detection of a violator; a similar approach has been used recently by Pellegrini [25] in solving linear programming problems in moderately high dimension. In particular, we sample a subset, S', of the points, of size $L\sqrt{n}$, where L is initialized to be 1 and is incremented by 1 at each iteration. Only if a violator is not found in S' do we begin scanning the full set of points in search of a violator. We implemented random sampling only for the implementation of Algorithm 1. For Algorithm 2, we always found the violator farthest from the current center in order to make the core set size as small as possible. We recommend using Algorithm 2 only when the user wants to minimize the size of the core set.

Another desirable property of the implementation is that it is I/O efficient if we assume that we can solve $O(\frac{1}{\epsilon})$ size subproblems in internal memory (This was always the case for our experiments, since the size of the core-set did not even approach $\frac{1}{\epsilon}$ in practice). If this is true then the current implementation in the I/O model does at most $O(\frac{nd}{B\epsilon^2})$ I/Os[3] and the same bound also generalizes to the cache-oblivious model [12]. The implementation of Algorithm 2 has an I/O bound of $O(\frac{nd}{B\epsilon})$. Some of the large problems we report results for here did actually use more memory while running than we had installed on the system. For instance in Figure 1, the sudden increase around dimension $d = 400$ for $n = 10^5$ points is due to effects of paging from disk, as this is the largest input size that allowed subproblems to fit within main memory. We believe that with our algorithm and an efficient implementation(C++), really large problems ($n \approx 10^7, d \approx 10^4, \epsilon \approx 10^{-4}$) would become tractable 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 a Pentium III 1Ghz, 512MB notebook computer, running Windows 2000. The hard disk used was a 4200rpm/20GB hard disk drive (Fujitsu MHM2200AT).

Datasets. Most of our experiments were conducted on randomly generated point data, according to various distributions. We also experimented with the USPS data[4] which is a dataset of handwritten characters created by the US Postal service. We used Matlab to generate random matrices. For generating uniform data we used `rand`, for generating specific distributions we used `random` and for generating normally distributed random numbers we used `randn`.

Specifically, we considered the following four classes of point data:

① uniformly distributed within a unit cube;

② uniformly distributed on the vertices of a unit cube;

③ normally distributed in space, with each coordinate chosen independently according to a normal distribution with mean 0 and variance 1;

④ point coordinates that are Poisson random variables, with parameter $\lambda = 1$.

Methods for comparison. Bernd Gärtner [15] provides a code on his website that we used. We also used the CGAL 2.4 implementation (using Welzl's move-to-front heuristic, together with Gärtner's method [15]). We were not able to compile code available from David White's web page[5]. We could not replicate the timings reported in the paper by Gärtner and Schönherr [16]. In the near future, a recent implementation of [16] is going to appear in CGAL. We decided not to include the comparison with this version because the implementation is not robust enough yet and sometimes gives wrong results.

Experimental results. In Figure 1 we show how the running time of Algorithm 1 varies with dimension, for each of three sizes of inputs ($n = 10^3$, 10^4, and 10^5) for points that are normally distributed (with mean $\mu = 0$ and variance $\sigma = 1$). Here, $\epsilon = 0.001$. Corresponding to

3. Here B denotes the disk block size.
4. `http://www.kernel-machines.org/data/ups.mat.gz` , 29MB
5. `http://vision.ucsd.edu/~dwhite`

the same experiment, Figure 2 shows how the number of iterations in Algorithm 1 varies with dimension. Recall that the number of iterations is simply the size of the core-set that we compute. Note that, while the core-set size is seen to increase with dimension, it is no where near the worst case predicted by theory (of $O(1/\epsilon)$). Also notable is the fact that as the dimension grows, the timings in Figure 1 do not seem to be linearly increasing (as predicted by theory). This seems to stem from the fact that the core set size is not constant with ϵ fixed, and the SOCP solver takes more time as the core set becomes bigger. These experiments also point to the fact that the theoretical bounds, both for the core set size and the running time are not tight, at least for normally distributed points.

In Figures 3 and 4 we show how the running time and the core-set size varies with dimension for each of the four distributions of input points, and $n = 10,000$, $\epsilon = 0.001$. Note again the correlation between running times and core set sizes.

Figure 5 shows a timing comparison between our algorithm, the CGAL 2.4 implementation, and Bernd Gärtner's code available from his website. Both these codes assume that the dimension of the input point set is fixed and have a threshold dimension beyond which the computation time blows up.

Figures 6, 7, 8 and 9 compare the implementations of Algorithm 1 and 2. For all the these experiments the points were picked from a normal distribution with $\mu = 0, \sigma = 1, n = 10000$. Figure 6 compares the timings of Algorithm 1 and 2 for $\epsilon = 2^{-10}$. Note that Algorithm 2 does not implement random sampling and hence is quite slow in comparison to Algorithm 1. Recall that Algorithm 2 was designed to keep the core set size as small as possible. Also the implementation of Algorithm 2 is not well optimized. Figure 7 and 8 compare the radius computed by both algorithms on the input point set. Figure 8 is a plot of the difference of the radii computed by the two algorithms on the same input. Note that the radii computed by both algorithms is always inside the window of variability they are allowed. For instance, at dimension $d = 1400$, the radii output from Algorithm 1 and 2 could vary by at most $\epsilon r \leq 2^{-10} \times 39.2 \leq 0.0009$ whereas actually it differs by 0.000282.

Figure 9 shows the difference between the core set sizes computed by Algorithm 1 and 2. Its surprising that the theoretical improvement suggested by the worst case core set sizes does not show up at all in practice. This unexpected phenomenon might be the result of Step 7 (Algorithm 1), which chooses the farthest violator and hence maximizes the expansion of the current ball. The slight difference between the core set sizes of Algorithm 1 and 2 seems to be there because Algorithm 1 does not find the exact farthest violator but uses random sampling to find a good one as explained in section 4. It seems that Algorithm 1 is as good as Algorithm 2 in practice as far as core set sizes are concerned, but it remains open to prove a tight bound on the core set size of Algorithm 1.

Figures 10 and 11 are an attempt to show the running times and core set sizes of Algorithm 1 on real world data sets. Note that these graphs suggest that core set sizes are linear compared to $\log\left(\frac{1}{\epsilon}\right)$!

In Figures 12 and 13 we show results for low dimensions ($d = 2, 3$), as the number n of points increases, for two choices of ϵ. Note the logarithmic scale. In both these experiments the core set size was less than 10 for all runs. This means that the time to do 2-center clustering on these data sets would take at most $O(2^{10}n)$ time. It remains an open problem to determine if 2-center clustering is really practical in 2, 3-dimensions.

Finally in Figure 14 we compare our implementation with the implementation of Claim 3.1 of [6] for different values of ϵ. The number of points n for this experiment is 1000. Note that for $\epsilon = 0.03$, this algorithm is already very slow compared to Algorithm 1. We have not implemented the improved version of [6] which has a slightly slower running time than our algorithm $\left(O\left(\frac{nd}{\epsilon} + \frac{1}{\epsilon^5}\right)\right)$, but it seems that when ϵ is small, the running time of the improved algorithm might suffer because of the base case solver (Claim 3.1 [6]). But certainly, the improved algorithm suggested in section 4, using [27] seems to be a better candidate for implementation than [6].

5 Open Problems

There are interesting theoretical and practical problems that this research opens up.

- ▷ **In Practice :** Can one do MEB with outliers in practice? 1-cylinder, 2-center and k-center approximations? Is computing minimum enclosing ellipsoid approximately feasible in higher dimensions? Are there core sets for ellipsoids of size less than $\Theta(d^2)$? Does dimension reduction help us to solve large dimensional problems in practice [19]? Can one use warm start strategies to improve running times by giving a good starting point at every iteration [32]? How does the improved algorithm with running time $O\left(\frac{nd}{\epsilon} + \frac{1}{\epsilon^4}\log^2\frac{1}{\epsilon}\right)$ suggested in section 4 using the new base case algorithm of [27] with running time $O(\frac{dn}{\epsilon}\log^2\frac{1}{\epsilon})$ compare with the implementation of Algorithm 1?

▷ **In Theory :** Of particular theoretical interest is the question about the optimal core-set size of the MEB problem. Are there similar dimension independent core-sets for other LP-Type problems? It remains an open question to tighten the core-set bounds for different distributions when $d < \frac{1}{\epsilon}$. From our experiments, at least for normal distributions it seems that the core set size is less than $O(\frac{1}{\epsilon})$.

Acknowledgements

The first author would like to thank Sariel Har-Peled and Edgar Ramos for helpful discussions. The authors thank Sachin Jambawalikar for help with importing the USPS data set in their code.

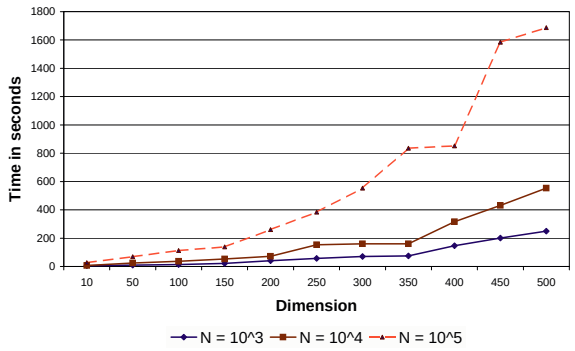

Figure 1: Running time in seconds of algorithm 1's implementation vs. dimension for $n = 10^3, 10^4, 10^5$ and $\epsilon = 0.001$ for inputs of normally distributed points($\mu = 0, \sigma = 1$).

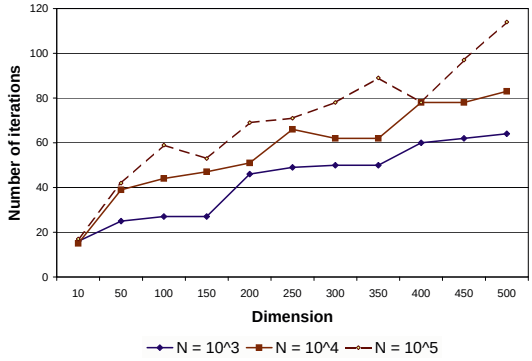

Figure 2: The number of iterations(Core set size) vs. dimension for $n = 10^3, 10^4, 10^5$ and $\epsilon = 0.001$ for inputs of normally distributed points($\mu = 0, \sigma = 1$).

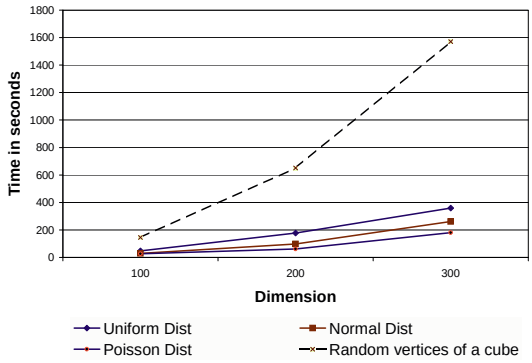

Figure 3: Running times for four distributions: uniform within a unit cube, normally distributed, Poisson distribution, and random vertices of a cube. Here, $n = 10000, \epsilon = 0.001$.

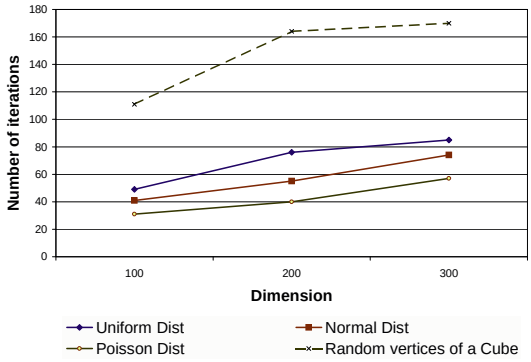

Figure 4: Number of iterations (core-set size), as a function of d, for four distributions. Here, $n = 10000$, $\epsilon = 0.001$.

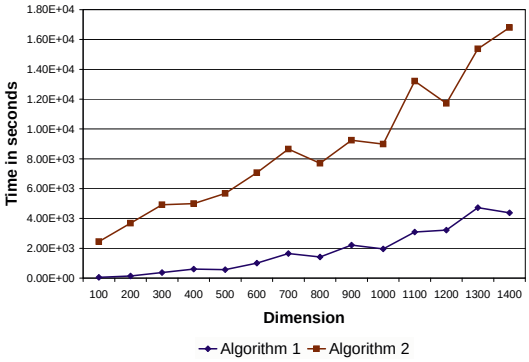

Figure 6: Timing comparison between Algorithm 1 and 2. ($n = 10000, \epsilon = 2^{-10}$, input from normal distribution, $\mu = 0, \sigma = 1$)

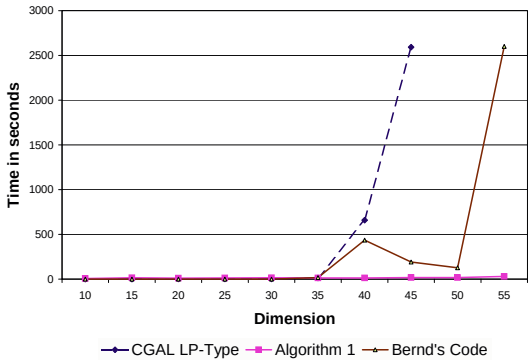

Figure 5: Timing comparison with CGAL and Bernd Gärtner's code [15], $\epsilon = 10^{-6}, n = 1000$, normally distributed points($\mu = 0, \sigma = 1$).

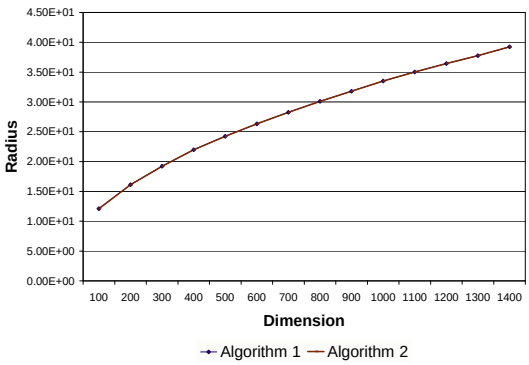

Figure 7: Radius comparison of Algorithm 1 and 2. ($n = 10000, \epsilon = 2^{-10}$, input from normal distribution, $\mu = 0, \sigma = 1$)

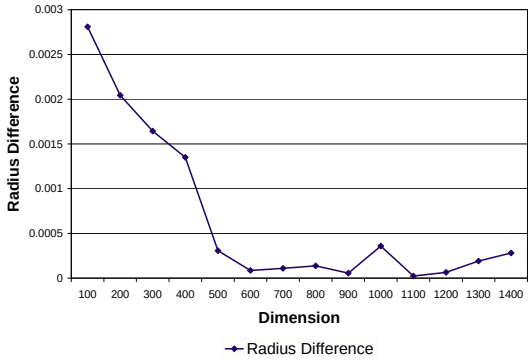

Figure 8: Radius difference plot of Algorithm 1 and 2 on the same input data. ($n = 10000, \epsilon = 2^{-10}$, input from normal distribution, $\mu = 0, \sigma = 1$)

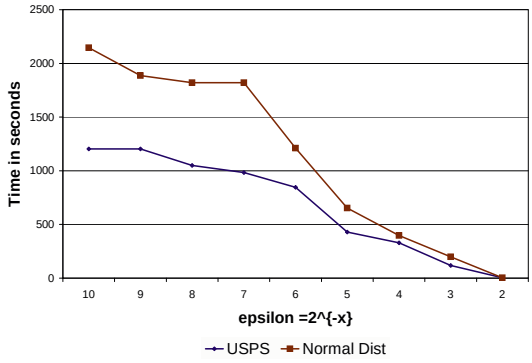

Figure 10: USPS data timing comparison with Normally distributed data($\mu = 0, \sigma = 1$). The data contains 7291 points in 256 dimensions and is a standard data set used in clustering and machine learning literature of digitized hand written characters.

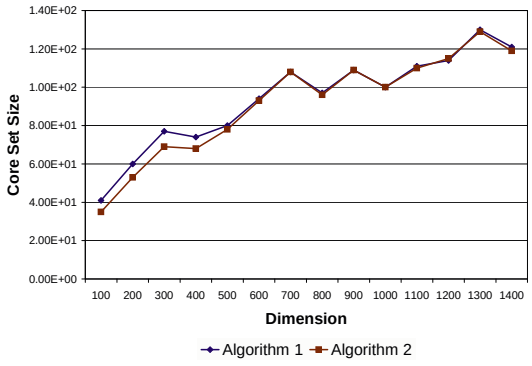

Figure 9: Core Set Size comparison between Algorithm 1 and 2. Note that Algorithm 2 does not implement random sampling. ($n = 10000, \epsilon = 2^{-10}$, input from normal distribution, $\mu = 0, \sigma = 1$)

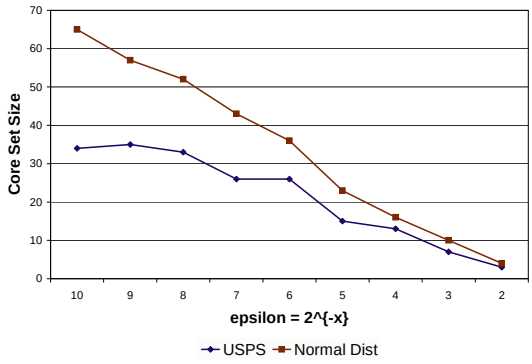

Figure 11: USPS data core set size comparison with Normally distributed data($\mu = 0, \sigma = 1$).

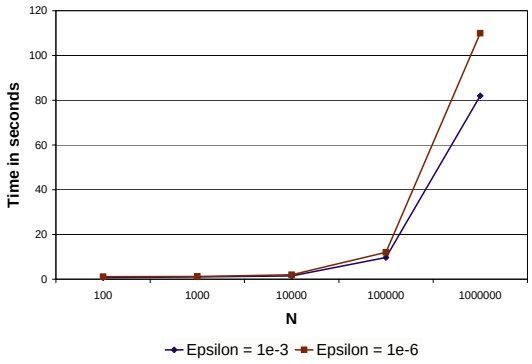

Figure 12: Experiments in 2D with different ϵ. All core set sizes for this experiment were less than 10 in size. Input from normal distribution($\mu = 0, \sigma = 1$).

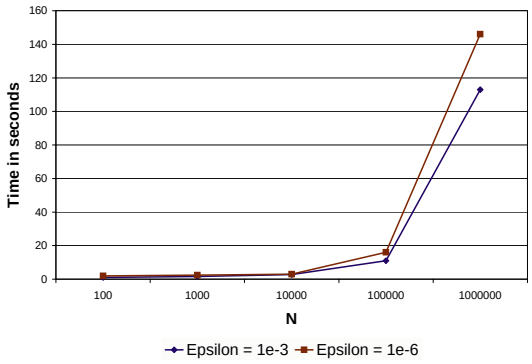

Figure 13: Experiments in 3D with different ϵ. Input from normal distribution($\mu = 0, \sigma = 1$). The core set sizes in this experiment were all less than 10. This shows that 2-center approximation in 2D/3D is practical. This could have applications in computation of spatial hierarchies based on balls [18].

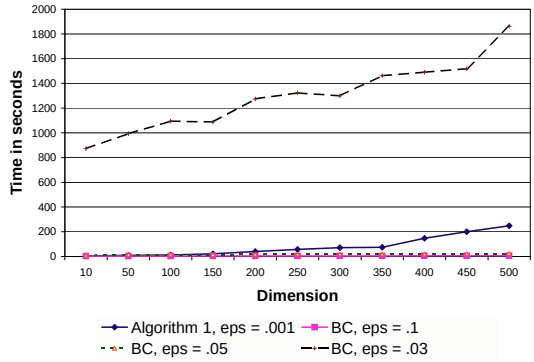

Figure 14: Our algorithm is compared with BC [6] for different epsilon values. As is evident, as soon as epsilon becomes small, their algorithm performance drastically suffers. (Is expected because of $1/\epsilon^2$ iterations on the point set).

References

[1] F. Alizadeh and D. Goldfarb. Second-order Cone Programming. *Technical Report* RRR 51, Rutgers University, Piscataway, NJ 08854. 2001

[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.

[4] A. Ben-Hur, D. Horn, H. T. Siegelmann and V. Vapnik. Support vector clustering. In *Journal of Machine Learning. revised version Jan 2002*, 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*, to appear, 2003.

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

[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. *Magangement Science*, 19(1):96–104, Sept. 1972.

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

[15] B. Gärtner. Fast and robust smallest enclosing balls[6]. In *Proceedings of 7th Annual European Symposium on Algorithms (ESA)*. Springer-Verlag, 1999.

[16] 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.

[17] 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.

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

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

[20] 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.

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

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

[23] 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.

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

[25] 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.

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

[27] Sariel Har-Peled. Personal Communications.

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

[29] K. C. Toh, M. J. Todd and R. H. Tütüncü. SDPT3 — a Matlab software package[7] for semidefinite programming. *Optimization Methods and Software*, 11:545–581, 1999.

[30] R. H. Tütüncü, K. C. Toh and M. J. Todd. Solving semidefinite-quadratic-linear programs using SDPT3. Technical report, Cornell University, 2001. To appear in *Mathematical Programming*.

[31] 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.

[32] 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.

[33] 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 in Procedings of Fields Institute of Mathematics.

6. `http://www.inf.ethz.ch/personal/gaertner`

7. `http://www.math.nus.edu.sg/~mattohkc/sdpt3.html`