A branch-and-bound algorithm for the minimum radius k—enclosing ball problem

Marta Cavaleiro* Farid Alizadeh†

July 11, 2017

Abstract

The minimum k-enclosing ball problem seeks the ball with smallest radius that contains at least k of m given points in a general n-dimensional Euclidean space. This problem is NP-hard. We present a branch-and-bound algorithm on the tree of the subsets of k points to solve this problem. The nodes on the tree are ordered in a suitable way, which, complemented with a last-in-first-out search strategy, allows for only a small fraction of nodes to be explored. Additionally, an efficient dual algorithm to solve the subproblems at each node is employed.

Keywords minimum covering ball, smallest enclosing ball, 1-center with outliers, branch-and-bound

1 Introduction

For a set $\mathcal{P} := \{p_1, \ldots, p_m\}$ of m points in a general n-dimensional Euclidean space, define a k-enclosing ball of $\mathcal{P}$ as an Euclidean hypersphere that covers at least k points of $\mathcal{P}$. The *minimum* k-*enclosing ball* problem, MkEB in short, seeks the k-enclosing ball of $\mathcal{P}$ with smallest radius. It can be formulated as

$$\mathsf{MkEB}(\mathcal{P}) := \begin{array}{ll} \min\limits_{r,x} & r \\ \text{s.t.} & |\{p_j \in \mathcal{P} : \|x - p_j\| \leq r\}| \geq k, \end{array} \tag{1}$$

with $|.|$ denoting the cardinality of a set and $\|.\|$ denoting the Euclidean norm. When $k = m$, the problem becomes the well known *minimum enclosing ball* (MEB) problem, also called the 1-center problem:

$$\mathsf{MEB}(\mathcal{P}) := \begin{array}{ll} \min\limits_{r,x} & r \\ \text{s.t.} & \|x - p_j\| \leq r, \quad p_j \in \mathcal{P}. \end{array} \tag{2}$$

The MkEB problem can then be seen as the robust version of the MEB problem. In many applications, data is usually noisy, and the presence of outliers can affect the solution of the MEB problem dramatically, justifying the need for a robust solution. Both are fundamental problems in computational geometry having many applications in areas like data mining, statistics, image processing, machine learning, etc.

The initial approaches to solve the MkEB problem consisted on constructing higher order Voronoi diagrams and then performing a search in all or some of the Voronoi cells (see Aggarwal et al. in [2]).

*Rutgers University, MSIS Department & RUTCOR, 100 Rockafellar Rd, Piscataway, NJ 08854. marta.cavaleiro@rutgers.edu

†Rutgers University, MSIS Department & RUTCOR, 100 Rockafellar Rd, Piscataway, NJ 08854. farid.alizadeh@rutgers.edu

A variety of methods were developed next: Efrat et al. [9] developed an algorithm to solve the problem using the parametric search technique (due to Megiddo [19]); Eppstein and Erickson [10] started by computing the $\mathcal{O}(k)$ nearest neighbors for each point, reducing the initial problem to $\mathcal{O}(m/k)$ smaller subproblems (see also [7] for an improvement on their algorithm); and Matoušek developed a simple randomized search algorithm in [18]. These methods solved the problem exactly, and though they were targeting the planar case, their extension to higher dimensions is possible.

Approximation algorithms started to be developed more recently. Har-Peled and Mazumdar [15] focused on the planar case and presented a linear 2-approximation algorithm by dividing the plane in *grids*, based on which they developed both a randomized exact algorithm and a ϵ-approximation algorithm for the MkEB problem. The same ideas were further explored in [16] where an ϵ-algorithm with expected $\mathcal{O}(m/\epsilon^n)$ time was shown as an application of a more general framework to solve several computational geometry problems. Other approximation algorithms have been developed based on the computation of *(robust) core sets* [17, 1].

Recently, Shenmaier [21] proved that the minimum k-enclosing ball problem is NP hard in the strong sense, by reducing in polynomial time the NP-complete k-clique problem in a regular graph to the MkEB problem. In the same paper, a polynomial-time ϵ-approximation algorithm was also proposed with complexity $\mathcal{O}(m^{1+1/\epsilon^2}n)$.

The minimum enclosing ball problem (2) can be easily formulated as a second-order-cone program (SOCP), more specifically, as a quadratic program (QP) [12], and thus solved in polynomial time. For a convex quadratic program, that is a problem with a convex quadratic objective and polyhedral constraints, simplex-type methods have been known for decades, see for instance [23, 22, 13]. However, due to its particular geometric structure, many special purpose simplex-type methods have been developed to solve the MEB problem. Among them there are both primal and dual iterative algorithms that have an analogous mechanics to the simplex method for linear programming [11, 8].

In this paper, we present a branch-and-bound (B&B) algorithm that builds the tree of all k-subsets of $\mathcal{P}$, that is, subsets of size k, and that uses the algorithm presented in [5], an improvement on [8], to solve the minimum enclosing ball problem in each node. Branch-and-bound methods using the same search tree have also been considered in [4] to solve the minimum k-enclosing ellipsoid, and also in [20, 24, 6] to address the feature selection problem.

The paper is organized as follows. In section 2, we introduce the basics of the B&B algorithm. Then, in section 3, we describe how the B&B can be modified to further improve the performance and provide a detailed description of the revised algorithm. In section 4 we cover the algorithm chosen to solve the subproblem at each node of the tree. In section 5, we give experimental results, and finally, draw conclusions in section 6.

2 The branch-and-bound algorithm

Consider the minimum k-enclosing ball problem (1). The search space, that is the set of all k-subsets of $\mathcal{P}$, can be represented as a tree, as shown in Figure 1 for the case of $m = 5$ and $k = 3$.

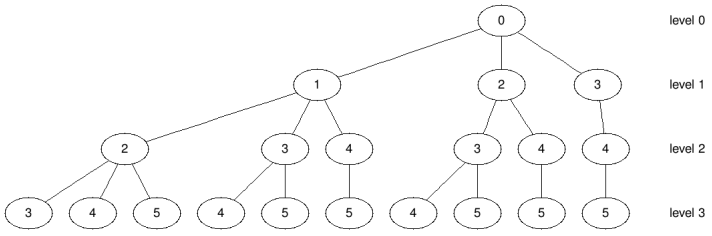


Figure 1: Search tree for $m = 5$ and $k = 3$.

Each node on the tree corresponds to one point of $\mathcal{P}$ (except the root, which represents the empty set). The set of nodes along any path from the root to a certain node represents a different subset of $\mathcal{P}$ with as many points as the tree level number. Thus, each path to a leaf represents a different k-subset of $\mathcal{P}$.

The branch-and-bound algorithm constructs the tree in a top-down manner, as it searches for the minimum radius k-enclosing ball of $\mathcal{P}$. At each node N of the tree, the algorithm solves the following subproblem: find the minimum enclosing ball that encloses the points corresponding to the nodes on the path from the root to N. The algorithm does not need necessarily to reach a leaf to find a ball that encloses at least k points, since, at any node, there may be other points of $\mathcal{P}$ that are enclosed besides those on the respective path. Information about the radius, r^*, and center, x^*, of the currently smallest k-enclosing ball found so far is kept at all times. We call r^* the *bound*.

The search begins at level 0, when the root node is created and added to the pool of *live* nodes $\mathcal{L}$. A *live* node is a node that has been explored, that is, whose corresponding MEB problem has been solved, but whose immediate successors (child nodes) have not been generated yet. At each iteration, and following some selection rule, a node is removed from $\mathcal{L}$ to be branched. Let r and x be the solution to the node's corresponding MEB problem. If, for that node, $r \geq r^*$, then its successor nodes do not need to be analyzed, since the radius can never decrease as we proceed to higher levels down the tree, since a new point is being added at each level. On the other hand, if $r < r^*$, each child node is created and the associated MEB problem solved. If the radius r corresponding to a child node is such that $r \geq r^*$, then that node can be immediately disregarded. Otherwise, if the child's MEB encloses k points of $\mathcal{P}$, then its radius becomes the new bound. If neither of these situations occur, the child node is simply added to $\mathcal{L}$ to be branched later. The algorithm terminates when $\mathcal{L}$ is empty, meaning that all nodes were either analyzed or cut off from the tree. The solution of (1) is the radius r^* and center x^* as defined at termination.

3 Features of the B&B

3.1 Search tree design

It is evident that, given a search tree topology, different point-node assignments may be defined. Our B&B algorithm aims at positioning "bad" points, points that will likely give a higher bound, on the denser part of the tree (left side), and, "good" points on the less dense part of the tree (right side). Nodes on the denser part are therefore more likely to be cut off.

We introduce a node-ordering method that works as follows. When a node N is selected for branching, first, the distances between the center of its MEB and each of the points corresponding to points on node N's subtree are calculated and sorted in descending order. The j-th child node of N corresponds to the point with the j-th largest distance. This in-level node-ordering heuristic is effective because, for a general set $\mathcal{Q}$ and a point q, the distance from q to the center of $\mathrm{MEB}(\mathcal{Q})$ is a good predictor of the radius of the $\mathrm{MEB}(\mathcal{Q} \cup \{q\})$. Following this heuristic, nodes with higher radius will tend to be on the denser side of the tree, resulting in a better chance of larger-scale pruning. As expected, when compared to simply ordering the nodes in lexicographic order, or randomly, this heuristic caused the number of evaluated nodes to decrease considerably.

One could also think of picking a node's children as explained before, and sorting them, after solving the MEB problem for each, by the corresponding radius. While this approach could result in a (small) decrease on the number of explored nodes, such benefit is outweighed by the increase in the running time due to an additional sorting at each node.

Children of the root node: The $m - k + 1$ child nodes of the root are special and the heuristic above cannot be applied to them. An approach that proved to give good results in practice consists of first calculating the minimum enclosing ball of all points in $\mathcal{P}$, and then selecting the $m - k + 1$ farthest points from its center as the children of the root.

The minimum solution tree[1]*:* Consider Figure 1. The rightmost nodes of any subtree are all degree-one nodes and therefore simply constitute a path. Instead of evaluating each node on such paths one at a time, we can jump directly to the leaf and solve the corresponding MEB problem. The path below such one-degree nodes can therefore be merged into a *special* single node, yielding a modified topology of the search tree called the *minimum solution tree*, as shown in Figure 2.

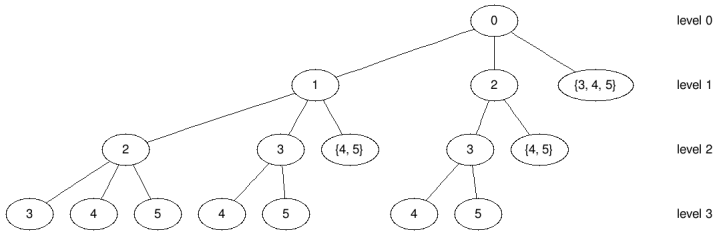


Figure 2: Minimum solution search tree for $m = 5$ and $k = 3$.

The computational work of solving the MEB problem at a *special* node can only be less than the one of solving separately the MEB for each one of the nodes it contains. However, this improvement is not expected to be significant due to the specifics of the algorithm chosen to solve the MEB problem at each node (see Section 4). Nevertheless, the use of the minimum solution tree leads to time savings in our implementation since it allows to avoid the addition and removal from $\mathcal{L}$ of nodes from the aforementioned paths.

3.2 Search strategy

Assume that, when branching on a node, its child nodes are created and added to $\mathcal{L}$ in a left-to-right order. $\mathcal{L}$ is maintained as a stack, that is, the last node added is the first one to be removed. This last-in-first-out (LIFO) strategy offers several advantages. First, a stack is an easy data structure to maintain. Second, given our node-ordering heuristics, this LIFO strategy implies that the less dense part of the tree, that is the most promising part, is searched first. Moreover, unless a leaf is reached or a node is cut off (because it leads to a radius larger than the current bound), at every iteration the B&B descends one level. This underlying depth-first-search allows reaching leaves fast. Finally, it is possible to calculate the maximum size of $\mathcal{L}$, as Theorem 1 states, allowing the required memory to be allocated before hand.

Theorem 1 (Maximum length of $\mathcal{L}$) *When LIFO is used as node-selection rule and the minimum solution tree is adopted, the maximum length of $\mathcal{L}$ is $m - k$. This upper bound is tight.*

Proof. Let $l(N)$ be the level of node N and $N_0, N_1, \ldots, N_{l(N)}$ be the nodes on the path from the root N_0 to $N \equiv N_{l(N)}$. Let us define the index of N, $i(N)$, as the index of the point that N would correspond to if the nodes were to be organized in lexicographical order. By definition $i(N_0) = 0$. Together, $i(N)$ and $l(N)$ uniquely define the shape of N's subtree. It is easy to see that the number of immediate successors of any node N is $m-k+l(N)-i(N)+1$ and that $i(N) \leq m-k+l(N)$ always holds.

Suppose the worst case scenario: no subtree is ever cut-off, that is, every node will be added to $\mathcal{L}$ at some point, with the exception of the leaves (which are disregarded after being evaluated).

Consider any stage of the algorithm when a node N was removed from $\mathcal{L}$ to be branched. Note that N cannot be a leaf, therefore $l(N) \leq k-1$. If $l(N) = 0$, then N is the root and $\mathcal{L}$ is empty. The

[1]The concept of minimum solution tree was first introduced by Yu and Yuan in [24] for a branch-and-bound algorithm for solving feature selection problems.

root has $m-k+1$ immediate successors, each of which will be added to $\mathcal{L}$ with the exception of the last one, which is a leaf since the minimum solution tree is being adopted. After this process $\mathcal{L}$ has size $m-k$.

Consider now the case $l(N) \geq 1$. Since N was removed from $\mathcal{L}$, all nodes on the path from the root to N have also been removed from $\mathcal{L}$. Moreover, $\mathcal{L}$ cannot contain any nodes of higher level than $l(N)$. Let j be any level between 1 and $l(N)$. The indexes of the nodes of level j that are still in $\mathcal{L}$ are $i(N_{j-1})+1, \ldots, i(N_j)-1$. Thus, the number of nodes in $\mathcal{L}$ is:

$$\sum_{j=1}^{l(N)} (i(N_j) - i(N_{j-1}) - 1) = i(N) - l(N).$$

Once N is branched, its $m-k+l(N)-i(N)+1$ immediate successor nodes are created and evaluated, and with the exception of the last one, they are all added to $\mathcal{L}$. The size of $\mathcal{L}$ then becomes $m-k$. ■

We experimented with other search strategies but the LIFO showed to be the most efficient. Breadth-first-search and best-first-search strategies tend to keep a very large number of nodes in $\mathcal{L}$, which leads to substantial memory space usage, and, in the case of best-first-search, more computational effort maintaining the priority queue. Note that, as a consequence of our node-ordering method, the LIFO strategy can be seen as a combination of depth-first-search and best-first-search.

3.3 Getting an initial solution

A good initial solution, a ball that covers at least k points and with a radius close to the optimal one, is expected to result in more aggressive pruning. There are many approaches to get a good initial k-enclosing ball. For instance, we could randomly select k points of $\mathcal{P}$ and find their minimum radius enclosing ball. Or, we could pick a random point in $\mathcal{P}$ and find the $k-1$ closest points to it, and then find the minimum radius ball of the resulting set. However, in general, these methods do not produce a ball with an enough small radius. Other, usually more successful, approaches are described next. They were also studied in [3] for the more general problem of the minimum volume ellipsoid with partial inclusion:

Spherical ordering: First, find the minimum enclosing ball of $\mathcal{P}$. Next, pick the k closest points to the center of the ball. Finally, find the minimum enclosing ball of those k points.

Spherical peeling: Find the minimum enclosing ball of $\mathcal{P}$, discard one of the points from the boundary, and find the minimum enclosing ball of the remaining points. Repeat this until only k points remain (note that the final ball may still cover more than k points).

If the points have a "spherical" shape and either their density is uniform (e.g. points sampled uniformly from a ball) or there is a high concentration of points closer to the center of the sphere (e.g. points from a standard normal distribution), then spherical ordering naturally gives a better approximation for the minimum k-enclosing ball independently of the value of k. On the other hand, when the density of points is low near the center, but the data is still "spherical" (e.g. points sampled uniformly from the surface of a sphere) the best approach depends on k. When k is large then spherical ordering performs well, but when k is small, spherical ordering will likely return a ball centered close to the center of the data and that covers a very sparse set of points, resulting in a bad initial solution. In this situation, randomly selecting a point and finding the minimum enclosing ball of its $k-1$ closest points is likely to provide a better initial solution.

When the data does not have a spherical shape, spherical peeling is likely to give better approximations, since, by iteratively removing points from the boundary of the minimum enclosing ball of the remaining data, the final ball will likely cover a dense subset of k points.

The choice of the method to obtain an initial solution should therefore be based on the arrangement of the points in $\mathcal{P}$ and on k, whenever possible.

The spherical ordering and spherical peeling methods may instead be implemented by starting with the minimum enclosing ball of any subset of $\mathcal{P}$ with k points (picked randomly, for instance). In practice, we observed that this random approach provides just as good initial solutions as the methods which start with all of $\mathcal{P}$.

In Section 5.3 we present some studies on the maximum improvement a good initial solution can provide for different datasets and values of k.

3.4 Lower bounds

Obtaining a lower bound for the radius of the MkEB is useful for cases where we are satisfied with suboptimal solutions, or cases where there is a limit on the number of iterations, or the running time of the algorithm. Having a good lower bound will indicate approximately how good our current best solution is.

A lower bound can be easily obtained by observing that the diameter of the MEB of any set of points $\mathcal{Q}$ (twice the radius of the ball) is larger than or equal to $\text{diam}(\mathcal{Q})$, the diameter of set $\mathcal{Q}$ (the maximum Euclidean distance between any two points in $\mathcal{Q}$). Suppose the solution of (1) is a ball which encloses the subset $\mathcal{Q} \subseteq \mathcal{P}$ and has radius r^* (notice that $|\mathcal{Q}| \geq k$). We then have

$$\begin{aligned} 2r^* &\geq \text{diam}(\mathcal{Q}) \\ &= \left(\tfrac{|\mathcal{Q}|(|\mathcal{Q}|-1)}{2}\right)\text{-th smallest pairwise distance of } \mathcal{Q} \\ &\geq \left(\tfrac{|\mathcal{Q}|(|\mathcal{Q}|-1)}{2}\right)\text{-th smallest pairwise distance of } \mathcal{P} \\ &\geq \left(\tfrac{k(k-1)}{2}\right)\text{-th smallest pairwise distance of } \mathcal{P}. \end{aligned} \tag{3}$$

A lower bound for r^* is therefore half the $(k(k-1)/2)$-smallest pairwise distance between any two points of $\mathcal{P}$.

An alternative way to obtain a lower bound for the optimal radius is to solve a relaxation of (1). It is possible to formulate (1) as a mixed-integer-second-order-cone problem as follows:

$$\begin{aligned} \min_{r,x,\beta_j} \quad & r \\ \text{s.t.} \quad & \|\beta_j p_j + (1-\beta_j)v_j - x\| \leq r, \quad p_j \in \mathcal{P} \\ & \sum_{j=1}^{m} \beta_j \geq k \\ & \beta_j \in \{0,1\},\ j = 1,...,m, \end{aligned} \tag{4}$$

where v_j is any point that is guaranteed to be covered by the optimal k-enclosing ball of $\mathcal{P}$. If we relax the integrality constraints of (4) to $0 \leq \beta_j \leq 1$, and solve the resulting SOCP $m-k+1$ times, each time with v_j being a different point of $\mathcal{P}$, we obtain a lower bound for r^* by selecting the minimum radius among all the obtained solutions. The relaxation of (4) is not known to be equivalent to a quadratic programming problem, so as a result we are not aware of a simplex-type algorithm to solve it. Instead, interior point methods seem to be the only algorithms that can solve it effectively. Consequently, setting up a branch-and-bound algorithm centered on this relaxation does not seem to be efficient, since using interior point methods at each node is excessively time consuming.

Problem (1) can also be formulated as a mixed-integer-quadratic problem as follows:

$$\begin{aligned} \min_{r,x,\beta_j} \quad & r^2 \\ \text{s.t.} \quad & \|p_j - x\|^2 \leq r^2 + (1-\beta_j)M, \quad p_j \in \mathcal{P} \\ & \sum_{j=1}^{m} \beta_j \geq k \\ & \beta_j \in \{0,1\},\ j = 1,...,m. \end{aligned} \tag{5}$$

Relaxing the integrality constraints yields a lower bound for r^*. However, the value of M needs to be very large so as to not jeopardize feasibility. Without any knowledge about the location of the minimum k-enclosing ball in $\mathcal{P}$, all we can conclude is that M is smaller than the square of $\text{diam}(\mathcal{P})$, which could be quite large. As a result, the relaxation (5) tends not to be very useful, except for when k is very close to the number of points in $\mathcal{P}$.

The lower bounds for the solution of problem (1) presented above can be modified to calculate, at each node N of the search tree, a lower bound for the best solution that can be found on the nodes of its subtree. When that lower bound is higher than the smallest radius of a k-enclosing ball found so far, node N can be cut off. Consider node N of the search tree. Let $\mathcal{St}(N)$ be the points of $\mathcal{P}$ corresponding to the nodes on N's subtree, and $\mathcal{P}(N)$ be the points of $\mathcal{P}$ corresponding to nodes on the path from the root node to N (included). The smallest radius of a k-enclosing ball that can be found in any of N's subtree nodes is given by the problem of finding the minimum radius k-enclosing ball that covers $\mathcal{P}(N)$ and at least $k - l(N)$ points of $\mathcal{St}(N)$. That problem can be formulated based on (4):

$$\begin{aligned} \min_{r,x,\beta_j} \quad & r \\ \text{s.t.} \quad & \|p_i - x\| \le r, && p_i \in \mathcal{P}(N) \\ & \|\beta_j q_j + (1-\beta_j)v_j - x\| \le r, && q_j \in \mathcal{St}(N) \\ & \sum_{j=1}^{|\mathcal{St}(N)|} \beta_j \ge k - l(N) \\ & \beta_j \in \{0,1\}, && j = 1, ..., |\mathcal{St}(N)|. \end{aligned} \tag{6}$$

where v_j is a point that is guaranteed to be covered by the smallest k-enclosing ball found on N's subtree. Relaxing the integrality condition on the multipliers β_j yields a SOCP that provides a lower bound. Picking v_j such that the distance between v_j and q_j is small gives the best bound, although finding such v_j causes an increase in the running time. A better alternative is choosing v_j as the closest point of $\mathcal{P}(N)$ to q_j.

A formulation based on (5) is obtained in a similar way:

$$\begin{aligned} \min_{r,x,\beta_j} \quad & r^2 \\ \text{s.t.} \quad & \|p_i - x\|^2 \le r^2, && p_i \in \mathcal{P}(N) \\ & \|q_j - x\|^2 \le r^2 + (1-\beta_j)M, && q_j \in \mathcal{St}(N) \\ & \sum_{j=1}^{|\mathcal{St}(N)|} \beta_j \ge k - l(N) \\ & \beta_j \in \{0,1\}, && j = 1, ..., |\mathcal{St}(N)|. \end{aligned} \tag{7}$$

As usual, the relaxation of (7) can be solved to obtain a lower bound.

A lower bound on the best solution of a node's subtree can be obtained based on (3). However, this lower bound is not advantageous given the node-ordering method we adopt. The points of $\mathcal{P}(N)$ are among the ones with higher pairwise distances of the set $\mathcal{P}(N) \cup \mathcal{St}(N)$, given the way they were selected. Therefore, in most cases, half the $(k(k-1)/2)$-smallest distance between the points in $\mathcal{P}(N) \cup \mathcal{St}(N)$ will be smaller than the radius of the MEB of $\mathcal{P}(N)$.

Finally, we point out that these lower bounds on the best solution of a node's subtree are only useful when the number of points in $\mathcal{St}(N)$ is large, so it is not efficient to calculate such lower bounds at all nodes of the tree, but only on those on the upper levels. In Section 5.2 we show how applying these lower bounds in some of the nodes of the search tree affects the performance of the branch-and-bound algorithm.

3.5 Pseudo - algorithm

We now introduce/review some notation:

N_0	root node
$\mathcal{L}$	set of live nodes (maintained as a stack)
r^*, x^*	radius and center of the best solution found at any stage
$\mathcal{P}(N)$	set of points corresponding to nodes on the path from N_0 to N (included)
$\mathcal{S}t(N)$	set of points corresponding to nodes on the subtree of N (excluded)
$r(N)$, $x(N)$	radius and center of $MEB(N)$, respectively
$b(N)$	number of branches/child nodes of N
$C(N)$	list of child nodes of N

We present the pseudo-code of the branch-and-bound algorithm in Algorithm 1. Calculating the lower bound on lines 7-10 is optional.

Algorithm 1 A branch-and-bound algorithm for the MkEB problem

Input: $\mathcal{P}$, and an initial solution x^*, r^*
Output: x^*, r^*

1: $\mathcal{L} = \{N_0\}$.
2: **while** $\mathcal{L} \neq \emptyset$ **do**
3: Select the top node N from $\mathcal{L}$.
4: **if** $r(N) \geq r^*$ **then**
5: prune N; **break**
6: **end if**
7: Get $\bar{r}$, by solving the relaxation of (6) or (7).
8: **if** $\bar{r} \geq r^*$ **then**
9: prune N. **break**.
10: **end if**
11: Branch N: for each $p \in \mathcal{S}t(N)$, calculate $\|p - x(N)\|$, and select the $b(N)$ points of $\mathcal{S}t(N)$ with the highest distances, sorting them in decreasing order, to get $C(N) = \{C_1, C_2 \ldots, C_{b(N)}\}$, where C_j corresponds to the point of $\mathcal{S}t(N)$ with the j-th highest distance.
12: **for each** node C_j in $C(N)$ **do**
13: Get $r(C_j)$ and $x(C_j)$ by solving $MEB(\mathcal{P}(C_j))$ (see Algorithm 2).
14: **if** C_j was not pruned **then**
15: Add C_j to the top of $\mathcal{L}$.
16: **end if**
17: **end for**
18: **end while**

4 Solving the MEB problem at each node

Suppose node N was selected to be branched and $C_1, ..., C_{b(N)}$ are its child nodes. We know the solution of the subproblem corresponding to N, $MEB(\mathcal{P}(N))$, and now, for each node C_j, we need to solve $MEB(\mathcal{P}(C_j))$. The latter subproblem differs from $MEB(\mathcal{P}(N))$ by an additional point in the set to enclose. An algorithm for the MEB problem that is able to take advantage of that fact and compute with small additional work the solution of each successor's subproblem is crucial. A dual algorithm is consequently a natural choice, since the solution of $MEB(\mathcal{P}(N))$ is dual feasible for the problem $MEB(\mathcal{P}(C_j))$.

Dearing and Zeck proposed in [8] a dual algorithm based on the fact that the minimum enclosing ball of $\mathcal{A}$, a set of points in the Euclidean space, is determined by an affinely independent subset $\mathcal{S}$ of $\mathcal{A}$, called the *support set*, whose points are on the boundary of the optimal ball. At each iteration of the algorithm, the current ball is optimal (namely it has minimum radius) with respect to all points it contains. However, it may not be feasible since it may not include all the points in $\mathcal{A}$. The algorithm is initialized with a support set $\mathcal{S}$ of two points and the trivial solution, (x, r), to $\mathsf{MEB}(\mathcal{S})$. Then, it iterates between solving $\mathsf{MEB}(\mathcal{S})$ and updating $\mathcal{S}$ until feasibility is achieved. At each iteration, if (x, r) is not optimal (feasible) for $\mathsf{MEB}(\mathcal{A})$, a point $p \in \mathcal{A}$, that is not yet enclosed, is chosen to enter $\mathcal{S}$. $\mathcal{S}$ is then updated by either the addition of p to $\mathcal{S}$ or by the replacement of an existing point in $\mathcal{S}$ by p. In either case the points in $\mathcal{S}$ remain affinely independent and the size of $\mathcal{S}$ never exceeds $n + 1$. Given the updated set $\mathcal{S}$, $\mathsf{MEB}(\mathcal{S})$ is then solved and p becomes enclosed (while some of the previously enclosed points may not be anymore). During the solution of $\mathsf{MEB}(\mathcal{S})$ some points may be dropped from $\mathcal{S}$. Since $\mathsf{MEB}(\mathcal{S})$ is a relaxation of $\mathsf{MEB}(\mathcal{A})$, the radius of $\mathsf{MEB}(\mathcal{S})$ is a lower bound on the optimal radius of $\mathsf{MEB}(\mathcal{A})$. The radius strictly increases at each iteration, so the algorithm terminates in a finite number of iterations.

In [5] the authors present a modification of Dearing and Zeck's algorithm that makes it more efficient by reducing the computational cost of each iteration from $\mathcal{O}(n^3)$ to $\mathcal{O}(n^2)$.

At each node of the search tree, we use Dearing and Zeck's dual algorithm to solve the MEB problem associated to that node. To see why this dual algorithm is most suitable, suppose node N is selected to be branched. When N is analyzed, $\mathsf{MEB}(\mathcal{P}(N))$ is solved, and the corresponding radius, center, and support set information kept in the stack along with the node itself. Once N is branched, problem $\mathsf{MEB}(\mathcal{P}(C_j))$ needs to be solved for each child node C_j, where $\mathcal{P}(C_j) = \mathcal{P}(N) \cup \{q_j\}$ and q_j is the point C_j corresponds to. If q_j is covered by the ball of $\mathsf{MEB}(\mathcal{P}(N))$, then the solution of $\mathsf{MEB}(\mathcal{P}(C_j))$ is the same as of $\mathsf{MEB}(\mathcal{P}(N))$. Otherwise, we apply Dearing and Zeck's algorithm starting with the radius, center, and support set of $\mathsf{MEB}(\mathcal{P}(N))$, and q_j as the point to enter the support set. Typically, solving $\mathsf{MEB}(\mathcal{P}(N) \cup \{q_j\})$ requires a modest number of iterations. Moreover, since at each iteration of the algorithm the radius increases, if it ever becomes larger than the best bound we do not need to run the algorithm until optimality; the node q_j and its successors can be pruned. Finally, the improvement proposed in [5] has a substantial impact in the context of high dimensional MkEB instances, making the solution of $\mathsf{MEB}(N)$ much faster for each node N.

4.1 Lower Bound on the radius of a node

Before solving problem $\mathsf{MEB}(\mathcal{P}(C_j))$, we can obtain a lower bound on its optimal radius, and, if it is worse than the best solution found, use it to prune C_j. If q_j, the point corresponding to C_j, is not covered by $\mathsf{MEB}(\mathcal{P}(N))$, then it is always going to be part of the optimal support set of $\mathsf{MEB}(\mathcal{P}(N) \cup \{q_j\})$ [8], and therefore will be on the boundary of the optimal ball. Using the triangle inequality, we conclude that

$$\|q_j - p\| \leq \|p - x(C_j)\| + \|q_j - x(C_j)\| \leq 2r(C_j) \quad \forall p \in \mathcal{P}(N),$$

where $x(C_j)$ and $r(C_j)$ are the center and radius of $\mathsf{MEB}(\mathcal{P}(N) \cup \{q_j\})$, respectively. Thus, we obtain the following lower bound on $r(C_j)$

$$r(C_j) \geq \frac{1}{2} \max_{p \in \mathcal{P}(N)} \|q_j - p\|. \tag{8}$$

The burden of finding lower bound (8) can be avoided by calculating an easy upper bound on $r(C_j)$. This upper bound is again a consequence of the observation that, if not covered already, q_j will be on the boundary of $\mathsf{MEB}(\mathcal{P}(N) \cup \{q_j\})$. Therefore the diameter of the optimal ball will always be smaller than the distance between q_j and the farthest point on the boundary of $\mathsf{MEB}(\mathcal{P}(N))$. That distance is $\|q_j - x(N)\| + r(N)$. As a consequence, we obtain the following upper bound on $r(C_j)$

$$r(C_j) \leq \frac{1}{2}\left(\|q_j - x(N)\| + r(N)\right). \tag{9}$$

Algorithm 2 summarizes the procedure to solve the subproblem at each node.

Algorithm 2 Solving $\mathrm{MEB}(\mathcal{P}(C_j))$ for each child node of N

Input: $x(N), r(N), q_j, r^*$
Output: $r(C_j), x(C_j)$

Calculate $d_j = \|q_j - x(N)\|$.
if $d_j \leq r(N)$ **then**
    q_j is already covered, so $r(C_j) = r(N)$ and $x(C_j) = x(N)$.
else if $\frac{1}{2}(d_j + r(N)) > r^*$ **then**
    if $\max_{p \in \mathcal{P}(N)} \frac{1}{2}\|q_j - p\| \geq r^*$ **then**
        Prune C_j.
    else
        Find $r(C_j)$ and $x(C_j)$ by solving $\mathrm{MEB}(\mathcal{P}(N) \cup \{q_j\})$ using the solution of $\mathrm{MEB}(\mathcal{P}(N))$.
    end if
end if

5 Experimental results

In this section we examine the effectiveness of our algorithm by evaluating its performance on various synthetic datasets. The algorithm was implemented using MATLAB 2014a. The tests were run on a PC with an Intel Core i5 2.30GHz processor, with 4GB RAM, running Windows 7.

We generated our datasets with the following approaches:

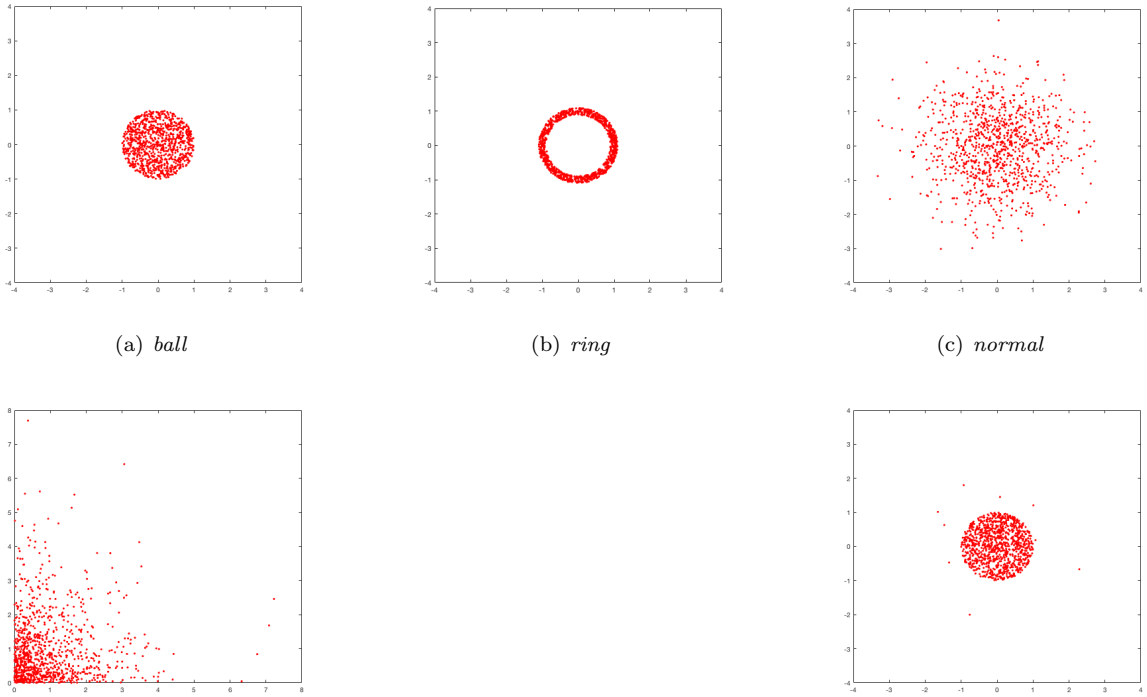
1. *ball*: points uniformly sampled on a unit ball;
2. *ring*: points uniformly sampled on a ring with inner radius 0.8 and outer radius 1.2;
3. *normal*: points sampled from a multivariate standard normal distribution;
4. *exponential*: each coordinate of the points independently sampled from an exponential distribution with mean equal to 1;
5. b-*outliers*: points uniformly sampled on a unit ball and b artificial outliers uniformly sampled from the ring with same center and radius between 1 and 3.

Figure 3 shows 2-dimensional instances for each of these datasets.

5.1 Discussion on the B&B algorithm performance

We ran the B&B algorithm on different datasets, varying the dimension and the number of points. For each dataset, we generated 10 random instances and solved the MkEB problem for different values of k. In these tests we did not take advantage of lower bounds to further prune the tree (so, relaxations (6) or (7) were not computed). For the reasons discussed in Section 3.3, to obtain an initial solution for the datasets *normal* and *ball*, we used the Spherical Ordering method; for the *ring* dataset we selected a random point and the $k-1$ closest points to it, and found the MEB of that set; and finally, for the *exponential* dataset we used Spherical Peeling.

The following performance measures for the algorithm are reported:



(a) *ball* (b) *ring* (c) *normal*

(d) *exponential* (e) 10-*outliers*

Figure 3: Examples of the five different types of datasets with 1000 points and dimension 2.

- **Explored nodes (EN)**: the number of explored nodes, that is, nodes where a MEB problem was solved;
- **% EN optimum is found**: the percentage of explored nodes after which the optimal solution was found;
- **Dual iter. per node**: the ratio between the number of dual MEB iterations and number of explored nodes;
- **Time**: the total elapsed CPU time in seconds.

Tables 1, 2, and 3 report the average values of the above performance measures for different experiments.

The most challenging instances in terms of number of explored nodes are those that contain many k-subsets whose MEB radius is close to the radius of the optimal k-enclosing ball. In these instances, a large portion of the tree has to be scanned before finding the optimal solution and prove its optimality. The *ball* instances for all values of k and the *ring* instances for large enough k are such examples. On the other hand, when the data has many outlying points, like in the *normal* or the *exponential* case, the algorithm tends to explore a smaller number of nodes, because the B&B will place many of the outlying points on the left side of the tree at the early stages of the algorithm. The nodes corresponding to those points are quickly realized that they can be pruned, and so are large parts of the tree. The same behavior can be observed in the *ring* instances when k is small, since many k-subsets will have points at opposite sides of the ring. We point out that the number of

MEB iterations per explored node is close to 1. This validates the choice of Dearing and Zeck's dual algorithm for solving MEB subproblems for each node. As we will see, while the number of MEB iterations tends to increase with the dimension, it generally remains rather small.

Small increases in the dimension cause a substantial increase in the difficulty of the problem due to the "curse of dimensionality". Consider when a node is branched and its minimum ball is inflated in order to cover a point corresponding to one of its child nodes. When the dimension is small this inflated ball has a good chance of covering many other points in the vicinity. This is true especially if the point to cover is far from the current center. When the dimension is large, however, the inflated ball has less chance to cover many other points unless the number of points also increases (exponentially) with the dimension. In other words, by increasing the dimension we have a lot more room to disperse the points and so the chance of an inflating ball to capture many other points decreases.

Another issue with increasing the dimension is that the number of dual iterations per node increases. When a ball is inflated and translated to enclose a new point, as the dimension increases, the chance that a previously covered point becomes uncovered also increases. For the same reason, this behavior tends to become more frequent for larger k.

Table 2 reports the performance results for the *normal* case, with $n = 10$ and $k = 50$, $k = 900$, and $k = 990$. These can be compared with the analogous results reported in Table 1 to show how the dimension increases the complexity of the problem. The number of explored nodes in dimension 10 increases significantly with respect to dimension 2, and for larger values of k, the average number of dual iterations per node also increases modestly.

We compared the results of Table 3 to those of the general-purpose MIQP-solver of Gurobi [14] (version 6.5.2) using its MATLAB interface. All Gurobi parameters were kept at their default values[2], with only the following exception: the dual simplex method was the algorithm chosen for solving the relaxation at each node. Moreover, the same initial solutions given in the experiments of Table 3 were given to Gurobi as warm start. For each dataset we compared the number of Gurobi's dual simplex iterations and the number of Dearing and Zeck's dual algorithm iterations in the B&B (Fig. 4 (a)); we also compared the CPU time of Gurobi and the B&B (Fig. 4 (b)). The maximum time allowed was 10^5 seconds for both algorithms.

We observe that our method was faster for all the datasets studied. It is of course not surprising that a dedicated algorithm is superior to a general-purpose code. Still, the comparison is necessary in order to argue that off-the-shelf methods cannot successfully compete with our approach. Our comparison is based on the 100-point instances, since Gurobi was not able to solve the instances with 1000 points within the allocated time.

In order to show that larger instances can also be solved in useful time, for certain datasets, we run some experiments for 10000-point instances of b-*outliers* datasets in dimension 10 and 100, for different values of b. Table 4 shows the average results of 10 runs.

[2]We realize that by fine-tuning these parameters Gurobi may have a much better performance, but the idea here is to show that our special purpose algorithm works better than a generic and default branch-and-bound method using the dual QP Simplex algorithm.

Dataset	k	Explored nodes (EN)	% EN optimum is found	Dual iter. per node	Time
normal	50	231243	11%	1.00	150.12
normal	100	315868	19%	1.00	220.39
normal	250	717512	42%	1.01	580.14
normal	750	228192	40%	1.01	232.68
normal	900	24439	34%	1.02	30.60
normal	990	94	44%	1.05	0.15
ball	50	307411	43%	1.00	228.58
ball	100	797306	69%	1.01	638.79
ball	250	8295548	74%	1.00	7401.77
ball	750	3933869	72%	1.00	4875.08
ball	900	543560	57%	1.01	518.71
ball	990	772	65%	1.11	0.87
ring	50	185739	91%	1.00	160.55
ring	100	347188	80%	1.00	295.75
ring	250	1239892	87%	1.00	1152.42
ring	750	25015147	64%	1.00	19527.62
ring	900	1591675	70%	1.01	1204.24
ring	990	1082	46%	1.12	0.90
exponential	50	275855	98%	1.00	188.83
exponential	100	400178	99%	1.00	267.75
exponential	250	532264	94%	1.00	356.61
exponential	750	568182	95%	1.00	439.69
exponential	900	47783	97%	1.02	4 16
exponential	990	130	93%	1.16	0.16

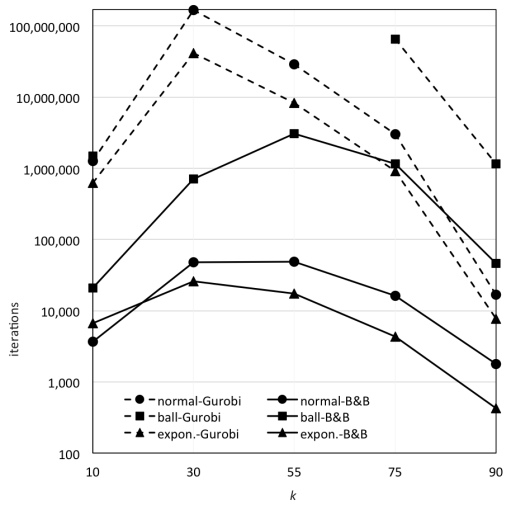
Table 1: Performance of the B&B for different 2-dimensional datasets with 1000 points

Dataset	k	Explored nodes (EN)	% EN optimum is found	Dual iter. per node	Time
normal	50	33906317	37%	1.03	36503.59
normal	900	450714331	63%	1.06	575461.26
normal	990	3929	55%	1.36	9.89

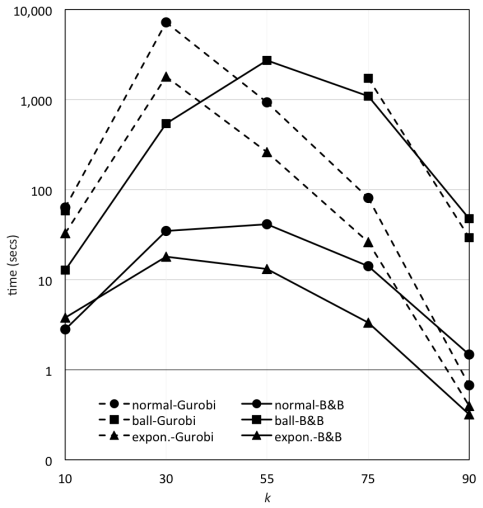
Table 2: Performance of the B&B for 10-dimensional *normal* datasets with 1000 points.

Dataset	k	Explored nodes (EN)	% EN optimum is found	Dual iter. per node	Time
normal	10	3426	38%	1.06	2.81
normal	30	42847	60%	1.11	34.66
normal	55	43179	43%	1.13	41.43
normal	75	13519	48%	1.19	14.22
normal	90	1315	57%	1.35	1.47
ball	10	19973	66%	1.05	12.76
ball	30	662078	71%	1.08	544.00
ball	55	2775139	88%	1.11	2713.10
ball	75	973051	60%	1.19	1094.62
ball	90	32722	63%	1.41	47.97
exponential	10	6396	66%	1.04	3.80
exponential	30	23390	79%	1.10	18.04
exponential	55	15183	80%	1.16	13.09
exponential	75	3583	69%	1.20	3.35
exponential	90	298	53%	1.42	0.32

Table 3: Performance of the B&B for different 10-dimensional datasets with 100 points.



(a) Average (dual) iterations as a function of k



(b) Average CPU time in seconds as a function of k

Figure 4: Gurobi vs. B&B for different 10-dimensional *normal*, *ball*, and *exponential* datasets with 100 points. The data for the *ball* dataset for $k = 30$ and $k = 55$ is missing since Gurobi was not able to solve these problems to optimality in the allowed time.

Dataset	n	b	k	Explored nodes (EN)	% EN optimum is found	Dual iter. per node	Time
b-*outliers*	10	10 (0.1%)	9990	69	84%	3.46	0.72
b-*outliers*	10	50 (0.5%)	9950	3593	96%	2.74	37.40
b-*outliers*	10	100 (1%)	9900	43012	98%	2.43	459.19
b-*outliers*	100	10 (0.1%)	9990	58	57%	15.29	17.22
b-*outliers*	100	50 (0.5%)	9950	814	52%	3.03	46.62
b-*outliers*	100	100 (1%)	9990	9711	55%	1.71	245.22

Table 4: Performance of the B&B for 10000-point b-*outliers* datasets for different values of b and dimension n.

5.2 On the impact of using lower bounds

In Section 3.4 we introduced two relaxations of the problem of finding the best k-enclosing ball on the subtree of a node: (6) is an SOCP and (7) is a QP. These methods can be applied on some or all nodes of the search tree in order to prune parts of the tree more aggressively.

We carried out some experiments to understand how these relaxations help us reduce the number of explored nodes. Both relaxations were applied separately for each instance. We did not compute these lower bounds on every single node, but only when all the following were met: the level of the node was greater than 1; the number of points involved in the relaxation was larger than k (otherwise the relaxation is simply an MEB problem); and the number of points on the subtree of a node (that is the set $\mathcal{S}t(N)$) was at least $0.1N$. Note than when the number of points in $\mathcal{S}t(N)$ is small, it is likely that the radius returned by the relaxation is not going to be much larger than that of $\mathsf{MEB}(\mathcal{P}(N))$. Of course, the relaxation was never solved on leaf nodes.

The experiments were conducted under the same conditions as the experiments of Tables 1 and 3. The results are presented in Tables 5 and 6, with the following measures

- **EN SOCP (%)**: percent ratio of the number of explored nodes when the SOCP relaxation (6) was used, over the number of explored nodes when no lower bound was used;
- **EN QP (%)**: percent ratio of the number of explored nodes when the QP relaxation (7) was used, over the number of explored nodes when no lower bound was used.

We observe that the QP relaxation had almost no impact on decreasing the number of explored nodes. This phenomenon has to do with the fact that the value of M is usually very large. This, in turn, affects the optimal relaxed solution and causes it to be not much larger than the radius of $\mathsf{MEB}(\mathcal{P}(N))$. A small reduction was however observed when k is very large, which is explained by the fact that, since the value of most of the β_j will be close to 1, the effect of M is neutralized for most of the constraints. The SOCP relaxation performs better. However, as mentioned earlier, this relaxation is full-fledged SOCP and at this point no effective algorithm other than interior point methods are known to solve it. As a result, it becomes exceedingly time-consuming to apply it to every node.

In general terms, it is not surprising that these relaxations have less impact when k is small. Note that the constraint $\sum \beta_j \geq k - l(N)$, when k is small, causes the solution of relaxations to be too small (compared to the value of $\mathsf{MEB}(\mathcal{P}(N))$) to produce useful values for pruning.

As a final comment, the (usually low) impact of using these relaxations cannot be only justified by formulation features. We must also consider the fact that due to the node/point attribution rule at each node, the radius of the MEB of $\mathcal{P}(N)$ is already a very good lower bound and blunts the effect of these relaxations. In fact, these relaxations, when applied to a branch-and-bound based on the lexicographic node/point attribution rule, performed much better and resulted in much more aggressive pruning.

Dataset	k	EN SOCP (%)	EN QP (%)
normal	50	96%	100%
normal	100	85%	100%
normal	250	84%	99%
normal	750	89%	100%
normal	900	85%	99%
normal	990	91%	96%
exponential	50	80%	99%
exponential	100	61%	97%
exponential	250	49%	98%
exponential	750	74%	99%
exponential	900	86%	99%
exponential	990	93%	95%

Table 5: Effect of relaxations (6) and (7) on the number of explored nodes, for 2-dimensional datasets with 1000 points.

Dataset	k	EN SOCP (%)	EN QP (%)
normal	10	99%	99%
normal	30	91%	99%
normal	55	89%	99%
normal	75	90%	99%
normal	90	93%	99%
exponential	10	95%	97%
exponential	30	87%	98%
exponential	55	85%	98%
exponential	75	86%	98%
exponential	90	89%	96%

Table 6: Effect of relaxations (6) and (7) on the number of explored nodes, for 10-dimensional datasets with 100 points.

5.3 On the impact of an initial solution

Starting the B&B algorithm with the knowledge of a good upper bound on the optimal solution can be potentially advantageous. In Section 3.3 we presented different methods to obtain such initial solutions. But how much effort should be put in finding a good initial solution? We performed some experiments to see how much of an impact good initial solutions can have on the performance of the algorithm. In particular, we wished to see whether or not a good initial solution is essential. We conducted two diametrically opposite experiments: In one we ran our B&B algorithm without any initial solution. In the second we used the optimal solution (obtained in an earlier run) as the initial solution. Both methods were tested on the same datasets. We used the following performance measures:

- **EN optimum given as initial sol. (%)**: percent ratio of the number of explored nodes when the optimum was given as initial solution, over the number of explored nodes, when no initial solution was provided;
- **EN optimum found with no initial sol. (%)**: the percentage of the number of explored nodes after which the optimal solution was found, over the total, when no initial solution was provided to the algorithm.

Tables 7 and 8 contain the averages of the measures above for different 10 samples of each dataset.

Two factors explain why, in some cases, the number of explored nodes when the optimum is provided as initial solution is close to the number of explored nodes when no initial solution is given: (a) either a good upper bound or the optimum is found early, when no initial solution is given, or (b) after finding a good upper bound, or even the optimal solution, the algorithm still needs to scan a large part of the search tree to prove it has found the optimum. It is also possible that both these factors are at work in some cases. For example, in the *normal* datasets, when no initial solution is given, the optimal solution is found earlier for smaller k. However, the reduction in the number of explored nodes is comparable for different values of k. So factor (a) seems to have more weight for smaller k while factor (b) has more weight for larger k.

On the other hand, if the number of explored nodes when the optimum is provided as initial solution is significantly smaller than when no initial solution is given, then, when no initial solution is given, a good upper bound is found in a later stage. In this case the knowledge of a good initial solution makes a considerable difference. This can be observed in the *exponential* dataset with $n = 2$ and $m = 1000$.

The main conclusion, therefore, is that although an initial solution can sometimes reduce the number of explored nodes considerably, in many cases it does not. Therefore, spending too much effort at the outset of the algorithm to find a good upper bound can often be wasteful.

6 Conclusions

In this paper we propose a branch-and-bound algorithm to solve the NP-hard problem of finding the ball of minimum radius that encloses at least k points from a given set. Our algorithm makes a correspondence between the subsets of points and the tree nodes. It uses a heuristic that aims at placing subsets of points enclosed in larger balls at the root of large subtrees. As a result, it has the ability to prune very large subsets of points from the search tree early on. This heuristic, combined with a Last-In-First-Out strategy for selecting the next node to explore, results in a combination of depth-first-search and best-first-search techniques. The algorithm retains the advantages of both these methods without the typical disadvantages. In fact, an important consequence of our LIFO approach is that the memory required to hold the active set of nodes is bounded by $m - k$.

In general, the algorithm does quite well on datasets with the presence of outliers, and the most difficult datasets for the algorithm seem to be those with an uniform distribution of the points drawn from a ball. Our experimental results also show that in almost all cases a small fraction of the nodes

Dataset	k	EN optimum given as initial sol. (%)	EN optimum found with no initial sol. (%)
normal	50	96%	14%
normal	100	94%	22%
normal	250	90%	37%
normal	750	94%	40%
normal	900	94%	44%
normal	990	93%	50%
ball	50	90%	64%
ball	100	81%	72%
ball	250	86%	61%
ball	750	94%	72%
ball	900	96%	59%
ball	990	94%	46%
exponential	50	64%	97%
exponential	100	50%	96%
exponential	250	71%	95%
exponential	750	35%	94%
exponential	900	34%	98%
exponential	990	66%	93%

Table 7: Maximum impact of an initial solution, for 2-dimensional datasets with 1000 points.

Dataset	k	EN optimum given as initial sol. (%)	EN optimum found with no initial sol. (%)
normal	10	86%	33%
normal	30	88%	37%
normal	55	85%	46%
normal	75	83%	51%
normal	90	86%	52%
ball	10	69%	82%
ball	30	67%	75%
ball	55	68%	76%
ball	75	76%	92%
ball	90	78%	84%
exponential	10	68%	67%
exponential	30	58%	82%
exponential	55	64%	76%
exponential	75	77%	61%
exponential	90	89%	44%

Table 8: Maximum impact of an initial solution, for 10-dimensional datasets with 100 points.

of the search tree need to be explored. They also support our choice of the Dearing and Zeck dual algorithm to solve the subproblems at each node, since in general we need a very small number of iterations per node. The fact that each iteration has a computational work of $\mathcal{O}(n^2)$ further speeds up the search process.

We also studied two relaxations of the MkEB problem in order to find better lower bounds. One was based on a SOCP relaxation while another was base on a QP. Our study revealed that these lower bounds are not very effective given our node/point attribution rule, but they can be very useful if the sequences of points in the tree are to be organized in arbitrary fashion (for example in lexicographic order).

Finally, we observed that the use of an initial good solution is not very consequential in reducing the number of explored nodes.

References

[1] P. K. Agarwal, S. Har-peled, and H. Yu, *Robust shape fitting via peeling and grating coresets*, in Proc. 17th Annual ACM-SIAM Symp. on Discrete Algorithms, SODA '06, Philadelphia, PA, USA, 2006, SIAM, pp. 182–191.

[2] A. Aggarwal, H. Imai, N. Katoh, and S. Suri, *Finding k points with minimum diameter and related problems*, J. Algorithms, 12 (1991), pp. 38 – 56.

[3] S. D. Ahipaşaoğlu, *Fast algorithms for the minimum volume estimator*, J. Global Optim., 62 (2015), pp. 351–370.

[4] J. A. Candela, *Exact iterative computation of the multivariate minimum volume ellipsoid estimator with a branch and bound algorithm*, in COMPSTAT: Proc. in Computational Statistics 12th Symp., Heidelberg, 1996, Physica-Verlag HD, pp. 175–180.

[5] M. Cavaleiro and F. Alizadeh, *A faster dual algorithm for the Euclidean minimum covering ball problem*, arXiv:1706.10256v2 [math.OC], (2017).

[6] X. W. Chen, *An improved branch and bound algorithm for feature selection*, Pattern Recognit. Lett., 24 (2003), pp. 1925 – 1933.

[7] A. Datta, H. Lenhof, C. Schwarz, and M. Smid, *Static and dynamic algorithms for k-point clustering problems*, J. Algorithms, 19 (1995), pp. 474 – 503.

[8] P. Dearing and C. R. Zeck, *A dual algorithm for the minimum covering ball problem in* $\mathbb{R}^n$, Oper. Res. Lett., 37 (2009), pp. 171–175.

[9] A. Efrat, M. Sharir, and A. Ziv, *Computing the smallest k-enclosing circle and related problems*, in Algorithms and Data Structures: 3rd Workshop Proc., WADS '93, Springer Berlin Heidelberg, 1993, pp. 325–336.

[10] D. Eppstein and J. G. Erickson, *Iterated nearest neighbors and finding minimal polytopes*, in Proc. 4th ACM-SIAM Symp. on Discrete Algorithms, SODA '13, Philadelphia, PA, USA, 1993, SIAM, pp. 64–73.

[11] K. Fischer, B. Gärtner, and M. Kutz, *Fast smallest-enclosing-ball computation in high dimensions*, in Proc. Algorithms - ESA 2003: 11th Annual European Symp., Berlin, Heidelberg, 2003, Springer Berlin Heidelberg, pp. 630–641.

[12] B. Gärtner and S. Schönherr, *An efficient, exact, and generic quadratic programming solver for geometric optimization*, in Proc. 16th Annual Symp. on Computational Geometry, SCG '00, New York, NY, USA, 2000, ACM, pp. 110–118.

[13] D. Goldfarb and A. Idnani, *A numerically stable dual method for solving strictly convex quadratic programs*, Math. Program., 27 (1983), pp. 1–33.

[14] I. Gurobi Optimization, *Gurobi optimizer reference manual*, 2016.

[15] S. Har-Peled and S. Mazumdar, *Fast algorithms for computing the smallest k-enclosing circle*, Algorithmica, 41 (2005), pp. 147–157.

[16] S. Har-Peled and B. Raichel, *Net and prune: A linear time algorithm for euclidean distance problems*, J. ACM, 62 (2015), pp. 44:1–44:35.

[17] S. Har-Peled and Y. Wang, *Shape fitting with outliers*, SIAM J. Comput., 33 (2004), pp. 269–285.

[18] J. Matoušek, *On enclosing k points by a circle*, Inform. Process. Lett., 53 (1995), pp. 217 – 221.

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

[20] P. M. Narendra and K. Fukunaga, *A branch and bound algorithm for feature subset selection*, IEEE Trans. Comput., 26 (1977), pp. 917–922.

[21] V. V. Shenmaier, *The problem of a minimal ball enclosing k points*, J. Appl. Ind. Math., 7 (2013), pp. 444–448.

[22] C. Van de Panne and A. Whinston, *Simplicial methods for quadratic programming*, Naval Res. Logist. Quart., 11 (1964), pp. 273–302.

[23] P. Wolfe, *The simplex method for quadratic programming*, Econometrica, 27 (1959), pp. 382–398.

[24] B. Yu and B. Yuan, *A more efficient branch and bound algorithm for feature selection*, Pattern Recognit., 26 (1993), pp. 883 – 889.