

Enclosing Weighted Points with an Almost-Unit Ball

Celina M. H. de Figueiredo* Guilherme D. da Fonseca*

To appear on Information Processing Letters

Abstract

Given a set of n points with positive real weights in d -dimensional space, we consider an approximation to the problem of placing a unit ball, in order to maximize the sum of the weights of the points enclosed by the ball. Given an approximation parameter $\varepsilon < 1$, we present an $O(n/\varepsilon^{d-1})$ expected time algorithm that determines a ball of radius $1 + \varepsilon$ enclosing a weight at least as large as the weight of the optimal unit ball. This is the first approximate algorithm for the weighted version of the problem in d -dimensional space. We also present a matching lower bound for a certain class of algorithms for the problem.

Keywords: Computational geometry; Geometric approximation.

1 Introduction

Let P be a set of n points in \mathbb{R}^d , for constant d , and $w : P \rightarrow \mathbb{R}^+$ be a *weight* function. We define the sum of the weights of the points inside a d -dimensional ball B as the *weight* $w(B) = \sum_{p \in P \cap B} w(p)$. In this paper, we consider an approximation to the problem of placing a ball B of radius 1 maximizing $w(B)$. We call the exact version of the problem *optimal placement*, and the unit ball B^* of maximal weight the *optimal ball*. The less general *unweighted version* of the optimal placement problem is the special case when $w(p) = 1$ for all $p \in P$. Let $k = w(B^*)$ in the unweighted version.

In the *approximate optimal placement* problem, we are also given a parameter $\varepsilon < 1$, and need to find a ball B of radius $1 + \varepsilon$ such that $w(B) \geq w(B^*)$. We call such ball an *approximate optimal ball* and present an algorithm to determine an approximate optimal ball in $O(n/\varepsilon^{d-1})$ expected time. We present a matching lower bound, proving that our algorithm is optimal for $n \geq 1/\varepsilon$, under certain assumptions.

The optimal placement problem has applications in many areas including computer vision, machine learning, clustering, and pattern recognition [1, 14]. Surprisingly, we could not find explicit solutions for $d \geq 3$ in the literature, neither for the exact nor for approximate versions.

Previous results The exact version of the problem is well studied for $d = 2$. Chazelle and Lee [8] presented an $O(n^2)$ algorithm for the exact planar case. Aronov and Har-Peled [2] showed that even the unweighted version of the optimal placement problem

*COPPE, Universidade Federal do Rio de Janeiro, Rio de Janeiro, RJ, Brazil

is 3SUM-hard, that is, the problem belongs to a class of problems for which no sub-quadratic algorithms are known [13]. For arbitrary dimension d , the problem can be solved exactly by projecting the points onto a $(d + 1)$ -dimensional paraboloid and using halfspace range searching data structures [15] to compute the weight of the $O(n^d)$ unit balls defined by subsets of d points, solving the problem in roughly $O(n^{d+1/d})$ time.

The consideration that a practical solution for large data sets needs to be roughly linear in terms of n motivated the study of approximations to the problem. There are two natural ways to define an approximation to the problem: we can approximate the weight $w(B^*)$ or the radius of the ball. In the unweighted version, approximating the weight $w(B^*)$ consists of approximating the number k of points enclosed by the optimal ball. An algorithm to find a unit disk, in the planar case, containing at least $(1 - \varepsilon)k$ points in $O(n^{1+\alpha} + n/\varepsilon)$ time, where α is an arbitrarily small constant, is presented in [1]. Monte-Carlo approximate algorithms for the same problem are presented in [1, 2].

We consider an alternative approximation criterion introduced in [12]: finding a ball B of radius $1 + \varepsilon$ such that $w(B) \geq w(B^*)$. Funke, Malamatos, and Ray [12] solve the unweighted planar version of the approximate optimal placement problem in expected time

$$O\left(n \cdot \min\left(k, \frac{1}{\varepsilon}, 1 + \frac{1}{k\varepsilon^{8/3}} \log^{2/3} \frac{1}{\varepsilon}\right)\right).$$

The complexity above is obtained through three different algorithms, with expected running times

$$(i) O(nk), (ii) O\left(\frac{n}{\varepsilon} + \frac{n}{k\varepsilon^2}\right), \text{ and } (iii) O\left(n + \frac{n}{k\varepsilon^{8/3}} \log^{2/3} \frac{1}{\varepsilon}\right).$$

All three algorithms are based on a preprocessing stage which reduces an instance of the (approximate) optimal placement problem with n points to $O(n/k)$ instances of the problem such that each instance has $O(k)$ points inside a hypercube of constant diameter. This preprocessing stage only works for the unweighted version of the problem.

Algorithm (i) uses the exact algorithm for $d = 2$ [8] after the preprocessing stage. Algorithm (ii) uses grids to calculate the number of points inside each of $O(n/(k\varepsilon^2))$ balls. In the plane, algorithm (ii) can be combined with algorithm (i) to attain an expected running time of $O(n/\varepsilon)$, since algorithm (i) takes $O(n/\varepsilon)$ time for $k \leq 1/\varepsilon$ and algorithm (ii) takes $O(n/\varepsilon)$ time for $k \geq 1/\varepsilon$. In higher dimensions, an extension of algorithm (ii) takes $O(n/\varepsilon^{d-1} + n/(k\varepsilon^d))$ time. Using a recent approximate range searching data structure by Fonseca and Mount [11], the running time of algorithm (iii) is reduced to $O(n + n/(k\varepsilon^{2.5}))$ in the plane and $O(n + n/(k\varepsilon^{(3d-1)/2}))$ in d -dimensional space.

All existing algorithms for the exact and approximate versions of the optimal placement problem work by calculating $w(B)$ for each ball B in a sufficiently large set \mathcal{B} of balls, and then choosing the ball of maximum weight. We can analyze previous planar algorithms in terms of $|\mathcal{B}|$ and the amortized time T to calculate $w(B)$ for $B \in \mathcal{B}$. The exact algorithm [8] has $|\mathcal{B}| = O(n^2)$ and $T = O(1)$. Algorithm (i) uses the preprocessing stage to reduce $|\mathcal{B}|$ to $O(nk)$, keeping $T = O(1)$. Algorithm (ii) has $|\mathcal{B}| = O(n/(k\varepsilon^2))$ and $T = O(1 + k\varepsilon)$. Algorithm (iii), using recent data structures [11], has $|\mathcal{B}| = O(n/(k\varepsilon^2))$ and $T = O(1/\sqrt{\varepsilon})$.

A problem related to optimal placement consists of, given an integer k , determining the smallest ball enclosing k points. Har-Peled and Mazumdar [14] presented an algorithm to solve the exact planar version of the problem in $O(nk)$ time and an ε -approximate algorithm which extends to d -dimensional space with a running time of $O(n + n \log \frac{1}{\varepsilon} / (k\varepsilon^{2d-1}))$. Using recent data structures [11], the running time can be improved to $O(n + n \log \frac{1}{\varepsilon} / (k\varepsilon^{(3d-1)/2}))$. Chan [5] presented an exact algorithm for large values of k , using linear programming with violations. Another related problem considered by de Berg, Cabello, and Har-Peled [10] consists of determining not one, but a set of multiple balls enclosing the maximum possible weight.

Contributions The first contribution of this paper is a lower bound for the exact and approximate versions of the optimal placement problem. We present a lower bound of $\Omega(n^d)$ for the number of balls $|\mathcal{B}|$ that actually need to be inspected in the exact version. In the approximate version, we present an $\Omega(n \min(n, 1/\varepsilon)^{d-1})$ lower bound. Our lower bounds assume that the algorithm decides which operations to perform on point weights irrespective of the weights of the points. Algorithms for the unweighted version of the problem do not meet the assumption, since they use the fact that $w(p) = 1$ for all $p \in P$ when deciding which weights to add. It is an open question if there is an algorithm that beats our lower bounds by not meeting our assumption, even in the unweighted version.

The second contribution of this paper consists of an optimal approximate algorithm to solve the weighted version of the problem in arbitrary dimensions. Our algorithm takes $O(n/\varepsilon)$ expected time in the plane and, more generally, $O(n/\varepsilon^{d-1})$ expected time in d -dimensional space, matching our lower bound for $n \geq 1/\varepsilon$. Our algorithm does not make use of exact algorithms for the problem to attain this complexity. Our algorithm is randomized because it makes use of constant time hashing, but randomization is not used anywhere else in the algorithm.

We can use our algorithm to improve the running time of the approximate algorithm [14] for certain values of k , finding an ε -approximate k -enclosing ball in $O(n \log \frac{1}{\varepsilon} / \varepsilon^{d-1})$ expected time. The importance of minimizing the ε dependencies has been recognized in many recent works [3, 6, 12]. Unlike other algorithms for the problem, our algorithm makes use of subtraction of point weights. It is an open problem whether the same time complexity can be attained without subtraction.

Throughout this paper, we consider Euclidean balls, but our results hold for other convex shapes of constant complexity. We also consider $d \geq 2$ to be an asymptotic constant and the same model of computation used in [12, 14], which is the real RAM with integer division and random bits (for hashing).

In Section 2, we prove the lower bounds for the exact and approximate versions. In Section 3, we present our approximate algorithm. Concluding remarks are discussed in Section 4.

2 Lower Bounds

In this section, we show that any weight-oblivious algorithm (defined next) for the optimal placement problem takes $\Omega(n^d)$ time in the exact version and $\Omega(n \min(n, 1/\varepsilon)^{d-1})$ time in the approximate version. The lower bound is tight in the approximate version

when $n \geq 1/\varepsilon$ and is tight in the exact version for $d = 2$. For comparison purposes, we note that the lower bound construction has the number of points enclosed by the (approximate) optimal ball fixed at $k = \Theta(n)$ for the exact version and $k = \Theta(1/\varepsilon)$ for the approximate version with $n \geq 1/\varepsilon$.

We say that an algorithm to compute the weight of the (approximate) optimal ball is *weight-oblivious* if the algorithm decides which arithmetic operations to perform on point weights based solely on the coordinates of the points, irrespective of their weights. Given a set of points P , a weight-oblivious algorithm must calculate the weight of $w(B)$ for each ball B that is optimal for some weight function of the point set. Otherwise, there is a weight function for which the algorithm fails to calculate the weight of the optimal ball.

Previous algorithms [8] that work for the weighted version of the problem are weight-oblivious, and therefore agree with our lower bound. Algorithms for the unweighted version of the problem are not weight-oblivious, since they use the fact that $w(p) = 1$ for all $p \in P$ when deciding which weights to operate on. Nevertheless, existing algorithms for the unweighted version of the problem do not beat our lower bound in the worst case.

The assumption of not considering the weights when deciding which point weights to operate on is also used in the semigroup arithmetic model [4, 7] to prove lower bounds for range searching. We note that, in some rare situations such as halfspace emptiness queries, existing data structures beat the semigroup arithmetic model lower bounds by using the fact the all points have the same weight.

Exact version We say that a unit ball B is *maximal* if there is no unit ball B' with $P \cap B \subset P \cap B'$ and we say that two unit balls B, B' are *distinct* if $P \cap B \neq P \cap B'$. Our lower bound proof is based on the following lemma.

Lemma 2.1. *If \mathcal{B} is a set of pairwise distinct maximal unit balls, then an weight-oblivious algorithm for the optimal placement problem takes $\Omega(|\mathcal{B}|)$ time.*

Proof. The running time of the algorithm is at least as large as the number of balls B for which $w(B)$ is calculated. Suppose an algorithm did not calculate $w(B)$ for some maximal unit ball $B \in \mathcal{B}$. Since the algorithm is weight-oblivious, the decision of not calculating $w(B)$ is irrespective of the weights. We set $w(p) = 1$ for $p \notin P \cap B$ and $w(p) = 1 + |P \setminus B|$ for $p \in P \cap B$. The ball B is the only optimal ball for these weight assignments. Since the algorithm did not calculate $w(B)$, it cannot return its weight as required. \square

We now prove our lower bound for the exact version by using a random set of points.

Theorem 2.2. *Any weight-oblivious algorithm for the exact optimal placement problem takes $\Omega(n^d)$ time.*

Proof. We show that there exists a set \mathcal{B} of pairwise distinct maximal unit balls with $|\mathcal{B}| = \Omega(n^d)$. The theorem follows from Lemma 2.1. Instead of considering a particular pathological set of points P , we consider P to be a random set of n uniformly distributed points inside the hypercube $[-2, 2]^d$. We show that $E[|\mathcal{B}|] = \Omega(n^d)$. Therefore, there is some set P for which $|\mathcal{B}|$ is at least as large as the $\Omega(n^d)$ expectation.

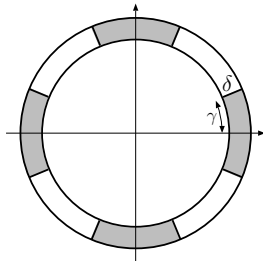


Figure 1: Set of $2d$ crust caps inside the δ -crust of a unit ball.

Let $\delta = \Theta(1/n)$ be a constant. Let \mathcal{B}' denote the set of unit balls centered at coordinates that are multiples of δ between -1 and 1 . We have $|\mathcal{B}'| = \Theta(1/\delta^d) = \Theta(n^d)$, but the balls in \mathcal{B}' are neither necessarily distinct nor maximal. Given a unit ball B , we can obtain a maximal ball $M(B)$ with $B \cap P \subseteq M(B) \cap P$ by translating B . Next, we show that the centers of B and $M(B)$ cannot be too far apart for a constant fraction of the balls $B \in \mathcal{B}'$.

Let C denote the δ -crust of B , that is, the locus of the points inside B and within distance at most δ from the boundary of B . We call *crust caps* the $2d$ disjoint regions of C that are the locus of the points within an angle (centered at the center of the ball) of at most γ from a positive or negative coordinate axis (Figure 1), where γ is a constant to be defined next. Each crust cap has a volume of $\Theta(\delta)$. Therefore, with constant probability there is at least one point inside each of the $2d$ crust caps.

We show that if there is at least one point inside each crust cap, then the distance between the centers of $M(B)$ and B is $O(\delta)$. Let v be the translation vector from $M(B)$ to B . If we set $\gamma = \arcsin(1/\sqrt{d})/2$, then there is a point in a crust cap defining an angle $\theta < \pi/2$ (centered at the center of B) with v . It follows from basic trigonometry that $\|v\| \leq \delta / \cos \theta = O(\delta)$.

Let \mathcal{B} denote a set of $M(B)$ for $B \in \mathcal{B}'$ with balls that are not pairwise distinct removed. By definition, \mathcal{B} contains only maximal pairwise distinct unit balls. Since the distance between $M(B)$ and B is $O(\delta)$ with constant probability, $E[|\mathcal{B}|] = \Theta(1/\delta^d) = \Theta(n^d)$. \square

Approximate version Recall that, in the approximate version, the algorithm needs to find a ball B' of radius $1 + \varepsilon$ such that $w(B') \geq w(B^*)$, where B^* is the optimal unit ball. To prove a lower bound for the approximate version, we first modify Lemma 2.1. We say that a set of balls \mathcal{C} covers a set of unit balls \mathcal{B} if for each unit ball $B \in \mathcal{B}$ there is a ball $C \in \mathcal{C}$ such that $B \cap P \subseteq C \cap P$.

Lemma 2.3. *If \mathcal{B} is a set of pairwise distinct maximal unit balls and \mathcal{C} is a minimum cardinality set of balls of size $1 + \varepsilon$ that covers \mathcal{B} , then an weight-oblivious approximate algorithm for the optimal placement problem takes $\Omega(|\mathcal{C}|)$ time.*

Proof. An approximate weight-oblivious algorithm must calculate $w(C)$ for a ball C of radius $1 + \varepsilon$ that contains the exact optimal ball B^* . Assume that the algorithm did not calculate $w(C)$. We can set $w(p) = 1$ for $p \notin P \cap B^*$ and $w(p) = 1 + |P \setminus B^*|$ for $p \in P \cap B^*$ in order to make B^* a subset of any approximate optimal ball C . \square

We now prove our lower bound for the approximate version.

Theorem 2.4. *Any weight-oblivious algorithm for the approximate optimal placement problem takes $\Omega(n \min(n, 1/\varepsilon)^{d-1})$ time.*

Proof. Consider the construction of \mathcal{B} for the exact version, but set $\delta = \Theta(\varepsilon + 1/n)$. It follows that \mathcal{B} has $\Theta(\min(n, 1/\varepsilon)^d)$ balls such that any pair of balls dist $\Omega(\varepsilon)$ from each other, even if we allow the balls to be translated without changing the set of points enclosed by each ball. Therefore, any cover \mathcal{C} has cardinality $|\mathcal{C}| = \Omega(\min(n, 1/\varepsilon)^d)$.

To improve the previous lower bound to $\Omega(n \min(n, 1/\varepsilon)^{d-1})$, we use the previous lower bound construction for a set of $1/\varepsilon$ points, getting a lower bound of $\Omega(1/\varepsilon^d)$. We place $n\varepsilon$ sets of $1/\varepsilon$ points, each set sufficiently far from each other, obtaining a lower bound of $\Omega(n/\varepsilon^{d-1})$ for $n \geq 1/\varepsilon$. Combining the $\Omega(n/\varepsilon^{d-1})$ bound, for $n \geq 1/\varepsilon$, with the $\Omega(\min(n, 1/\varepsilon)^d)$ bound, we get a lower bound of $\Omega(n \min(n, 1/\varepsilon)^{d-1})$. \square

3 An Approximate Algorithm

Let x_1, \dots, x_d denote the orthogonal axes. We call the x_d axis *vertical*, the hyperplane determined by the remaining axes *horizontal*, and use standard terms such as *upper* with respect to these directions. Since we can translate a ball downwards until a point $p \in P$ is on the upper boundary of the ball, we can restrict our search to balls that contain a point $p \in P$ on the upper boundary. Since the approximate optimal ball has radius $1 + \varepsilon$ while the optimal ball has radius 1, we can restrict our search to balls such that the center of the ball has all horizontal coordinates x_1, \dots, x_{d-1} as multiples of ε . There are $|\mathcal{B}| = O(n/\varepsilon^{d-1})$ balls satisfying these two conditions. The main difficulty consists of approximating the sum of the weights of the points inside each of these balls in $T = O(1)$ time per ball.

Preprocessing Even though the optimal ball has radius 1, the point set P can have arbitrarily large diameter. The first stage of our algorithm reduces the problem of finding an approximate optimal ball in a set P of n points to the problem of finding an approximate optimal ball among multiple non-empty point sets P_1, \dots, P_m with the following two properties: (i) The diameter of P_i is $O(1)$ for $1 \leq i \leq m$. (ii) The sum of the cardinalities $|P_1| + \dots + |P_m| = O(n)$. We define the cardinality of P_i as n_i .

We construct a grid of cells with side length 2 and determine the list and the count of points within each non-empty cell. This takes $O(n)$ expected time and space using hashing [9], and the list and count for a given cell can later be determined in $O(1)$ time. Alternatively, the same $O(n)$ time complexity can be achieved deterministically with unbounded space, or an $O(n \log n)$ deterministic time bound can be obtained with $O(n)$ space using balanced binary search trees [9].

Let the *neighborhood* of a cell i denote the set of 3^d cells surrounding cell i , including cell i itself. Let P_i denote the set of points in the neighborhood of a non-empty cell i . Since the optimal ball must be contained in the neighborhood of some non-empty cell, it suffices to find the approximate optimal ball among m different neighborhoods. Note that the neighborhood of a cell is contained in a hypercube of diameter $6\sqrt{d} = O(1)$. Since each point is contained in the neighborhood of at most 3^d cells, $n_1 + \dots + n_m = O(n)$.

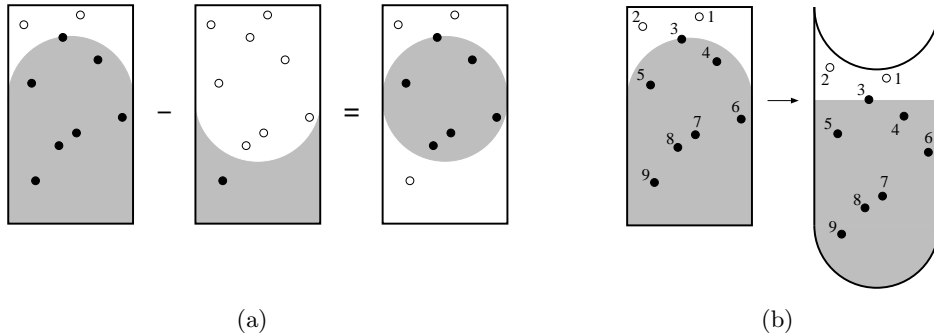


Figure 2: (a) Counting the number of points within a ball using subtraction. (b) Transformation used to reduce the below-ball relation to vertical order.

Cylinders Now, we solve the problem for the set P_i . Since P_i has $O(1)$ diameter, there are $O(1/\varepsilon^{d-1})$ horizontal coordinates that are multiples of ε . Each horizontal coordinate uniquely defines the center axis of an infinite vertical cylinder of radius $1 + \varepsilon$. Each cylinder contains $O(n_i)$ balls of radius $1 + \varepsilon$ with a point on the upper boundary of the ball. We count the points inside a ball as the difference between the number of points below the upper boundary of the ball and the number of points below the lower boundary (Figure 2(a)).

To count the points below the upper boundary of each ball, we first apply a non-linear transformation to the space. For each point within distance r from the central axis of the cylinder we subtract $\sqrt{1 - r^2}$ from the x_d coordinate of the point. This transformation converts the upper boundaries of the balls into horizontal regions (Figure 2(b)). We separately perform the reverse transformation to convert the lower boundaries of the balls into horizontal regions.

After the transformation, we ignore all coordinate values except for x_d . Since we are interested in an ε -approximate solution, we translate the point by at most $\varepsilon/2$ by rounding the x_d coordinate of the points to the nearest multiples of ε . We also scale the x_d coordinate by a factor of $1/\varepsilon$, in order to obtain an integer x_d coordinate within a range of $O(1/\varepsilon)$ possible values. We build two lists containing these $O(n_i)$ integer coordinate values, one for the upper boundaries and one for the lower boundaries.

We sort these two lists using an algorithm that we explain later. For now, assume we have the lists sorted in non-decreasing order. We can sweep the lists to find the sum of the weights of the elements that are greater than and respectively less than each element in $O(n_i)$ time. With this information, we determine the weight of each ball in $O(1)$ time per ball, and then pick the ball of maximum weight.

Sorting Since the lists contain $O(n_i)$ integers within a range of $O(1/\varepsilon)$ values, it would be easy to sort them in $O(n_i \log n_i)$ time or alternatively $O(n_i + 1/\varepsilon)$ time using counting sort [9]. These two approaches would result in total running times of $O(n \log n/\varepsilon^{d-1})$ and $O(n/\varepsilon^d)$, respectively. To attain the claimed $O(n/\varepsilon^{d-1})$ running time, we need an additional idea.

We have $O(1/\varepsilon^{d-1})$ transformed cylinders, each with two corresponding lists. Therefore, we have $O(1/\varepsilon^{d-1})$ lists with $O(n_i)$ elements in each and the elements correspond

to $O(1/\varepsilon)$ integer values. We add a link from the elements to the corresponding list, concatenate all lists, sort all lists in $O(n_i/\varepsilon^{d-1})$ time using counting sort, and finally separate the now sorted lists back.

For each set P_i we consider $O(1/\varepsilon^{d-1})$ cylinders. For each cylinder, we consider $O(n_i)$ balls, computing the weights of the balls in $O(1)$ time per ball after sorting. We sort the $O(n_i/\varepsilon^{d-1})$ values corresponding to P_i in $O(n_i/\varepsilon^{d-1})$ time. Therefore, our algorithm takes $O(n_i/\varepsilon^{d-1})$ for each set P_i . Since $n_1 + \dots + n_m = O(n)$, we have the following theorem.

Theorem 3.1. *There exists a weight-oblivious algorithm for the approximate optimal placement problem with expected running time $O(n/\varepsilon^{d-1})$.*

4 Conclusion

We presented an $O(n/\varepsilon^{d-1})$ expected time algorithm for the approximate optimal placement problem. Our model of approximation is the same one used in [12] and differs from the one used in [1, 2, 10] because we are ε -approximating the radius of the ball instead of the count or weight. Since we are approximating the radius, and not the weights, our algorithm works in the general case where the points have positive weights from an arbitrary ordered Abelian group.

In the more general ordered semigroup version of the problem, the weights of the points are positive elements from an ordered commutative semigroup. Since semigroup elements do not necessarily have an inverse, weights cannot be subtracted. The use of subtraction seems to be crucial for our algorithm. It would be interesting to obtain an $O(n/\varepsilon^{d-1})$ time algorithm for the ordered semigroup version of the problem, or perhaps prove a lower bound.

A related problem consists of determining the ball B^* of minimum weight with the center of B^* constrained to a given constant complexity region R of the space. An algorithm to approximate the weight of B^* is presented in [10]. It is easy to adapt our algorithm to approximate the ball of minimum weight, that is, determine a ball B of radius $1 - \varepsilon$ with center within distance at most ε from R and weight $w(B) \leq w(B^*)$, in $O(n/\varepsilon^{d-1})$ time.

We presented lower bounds of $\Omega(n^d)$ and $\Omega(n \min(n, 1/\varepsilon)^{d-1})$ for the exact and approximate problems. Our lower bounds are restricted to weight-oblivious algorithms. It is an open question if there is a non-weight-oblivious algorithm that beats our lower bounds, even in the unweighted version.

Our lower bounds do not consider the time spent computing the weight of each ball. Spherical range searching is known to be a hard problem in the exact version [4]. In the approximate version, the problem is considerably easier [3, 4]. Our algorithm approximates the weight of each ball in $O(1)$ amortized time. It would be elucidating to understand which sets of balls \mathcal{B} allow $w(B)$ for all $B \in \mathcal{B}$ to be computed in $O(1)$ amortized time.

Acknowledgments The authors would like to thank Theocharis Malamatos, David Mount Jorge Stolfi and the reviewers for their valuable comments.

References

- [1] P.K. Agarwal, T. Hagerup, R. Ray, M. Sharir, M.H.M. Smid, and E. Welzl. Translating a planar object to maximize point containment. In *10th European Symp. on Algorithms (ESA '02)*, pages 42–53, 2002.
- [2] B. Aronov and S. Har-Peled. On approximating the depth and related problems. *SIAM J. Comput.*, 38(3):899–921, 2008.
- [3] S. Arya, G.D. da Fonseca, and D.M. Mount. Tradeoffs in approximate range searching made simpler. In *SIBGRAPI 2008*, pages 237–244, 2008.
- [4] S. Arya, T. Malamatos, and D. M. Mount. On the importance of idempotence. In *38th ACM Symp. on Theory of Computing (STOC'06)*, pages 564–573, 2006.
- [5] T.M. Chan. Low-dimensional linear programming with violations. *SIAM J. Comput.*, 34(4):879–893, 2005.
- [6] T.M. Chan. Faster core-set constructions and data-stream algorithms in fixed dimensions. *Comput. Geom.*, 35(1):20–35, 2006.
- [7] B. Chazelle. Lower bounds on the complexity of polytope range searching. *J. Amer. Math. Soc.*, 2:637–666, 1989.
- [8] B. Chazelle and D.T. Lee. On a circle placement problem. *Computing*, 36(1-2):1–16, 1986.
- [9] T.H. Cormen, C. Stein, R.L. Rivest, and C.E. Leiserson. *Introduction to Algorithms*. McGraw-Hill Higher Education, 2001.
- [10] M. de Berg, S. Cabello, and S. Har-Peled. Covering many or few points with unit disks. *Theory Comput. Syst.*, To appear. DOI 10.1007/s00224-008-9135-9.
- [11] G.D. da Fonseca and D.M. Mount. Approximate range searching: The absolute model. *Comput. Geom.*, To appear. doi:10.1016/j.comgeo.2008.09.009.
- [12] S. Funke, T. Malamatos, and R. Ray. Finding planar regions in a terrain: in practice and with a guarantee. *Internat. J. Comput. Geom. Appl.*, 15(4):379–401, 2005.
- [13] A. Gajentaan and M.H. Overmars. On a class of $O(n^2)$ problems in computational geometry. *Comput. Geom.*, 5(3):165–185, 1995.
- [14] S. Har-Peled and S. Mazumdar. Fast algorithms for computing the smallest k -enclosing circle. *Algorithmica*, 41(3):147–157, 2005.
- [15] J. Matoušek. Range searching with efficient hierarchical cutting. *Discrete Comput. Geom.*, 10:157–182, 1993.