

A Practical Approach for Computing the Diameter of a Point Set

Sariel Har-Peled*

March 26, 2001

Abstract

We present an approximation algorithm for computing the diameter of a point-set in d -dimensions. The new algorithm is sensitive to the “hardness” of computing the diameter of the given input, and for most inputs it is able to compute the *exact* diameter extremely fast. The new algorithm is simple, robust, has good empirical performance, and can be implemented quickly. As such, it seems to be the algorithm of choice in practice for computing/approximating the diameter.

1 Introduction

Given a set of n points $S \subseteq \mathbb{R}^d$ computing the diameter of S — the pair of points of S that are furthest away from each other — had attracted a lot of attention in recent years. While the problem is quite easy in 2-dimensions [Tou83], even in three dimensions it becomes non-trivial. Clarkson and Shor [CS89] were the first to give an $O(n \log n)$ expected time algorithm for computing the diameter in 3-dimensions. However, coming up with a similar deterministic algorithm proved to be surprisingly hard (see [Ram00] for the long and somewhat painful history of this problem), concluding with the recent result of Ramos [Ram00].

In practice, the diameter is quite useful as it provides a reliable estimate of the point-set extent, it can be used in computing a tight fitting bounding box for the point-set [BHP99], and it can be also used in visualization and data-mining [FL95]. In particular, in the graphics community PCA (principal component analysis) was used to find such tight fitting bounding boxes. Unfortunately, PCA work reasonably well for the majority of the inputs but it can be easily fooled.

While theoretically very satisfying, in practice, those algorithms are quite hard to implement requiring weeks (or even months) to implement. Furthermore, such implementations would probably suffer from serious numerical problems. At the other end of the scale, the trivial algorithm, of trying all possible pairs and picking the longest pair, is trivial and can be implemented in a few minutes. However, even for small inputs it is too slow to be used

*Department of Computer Science, University of Illinois, Urbana, IL, 61801, USA; sariel@cs.uiuc.edu; <http://www.uiuc.edu/~sariel/>

(for example, for 10,000 points in 3d the naive algorithm computes the diameter in about ten seconds on a Sun Ultra 5).

Motivated by this discrepancy between theory and practice, we present, in this paper, a simple algorithm for approximating the diameter of a point-set. For $\varepsilon > 0$, and a set of points P in \mathbb{R}^3 , it computes a pair of points $p, q \in S$, so that $|pq| \geq (1 - \varepsilon)\Delta(P)$ in $O((n + 1/\varepsilon^3) \log 1/\varepsilon)$ time (the algorithm works also in higher dimensions, see Theorem 4.15 for details), where $\Delta(P)$ is the length of the diameter of P . This contrast with the faster $O(n + 1/\varepsilon^{3/2} \log 1/\varepsilon)$ time algorithm due to Barequet and Har-Peled [BHP99] (this algorithm uses the exact algorithm as a subroutine and is thus impractical), and $O(n + 1/\varepsilon^{5/2})$ algorithm due to Chan [Cha00]¹.

The new algorithm can also be used to compute the exact diameter. Although, in the worst case, the algorithm running time is still quadratic, this running time is sensitive to the "hardness" of the input. Namely, an instance of the diameter problem is hard, if almost all the pairs of points in the input have length close to the diameter of the point-set. However, in most inputs the diameter is quite distinctive and only a small fraction of all possible pairs are serious contenders to be the diametrical pair. The running time of the new algorithm is related to this quantity and for most inputs it computes the exact diameter extremely fast.

The idea underlining our algorithm is the following: We compute a hierarchical decomposition of the given point-set maintaining implicitly all the pairs of points that might serve as the diameter. We throw away batches of pairs that are guaranteed not to contain a long pair. We repeatedly refine our hierarchical decomposition (and the associated pairs decomposition) till we reach the required approximation. In particular, the algorithm works by computing the well-separated pairs decomposition of the given point-set [CK95], throwing away all the pairs that are guaranteed to be too short to contain the diameter. The implementation of the new algorithm is available on the web [HP00].

In Section 2, we define notation and review previous known approximation algorithms for the diameter. In Section 3, we present our new algorithm. In Section 4, we provide a theoretical analysis of the new algorithm. In Section 5 we summarize our experimental results comparing our algorithm with several other algorithms. Conclusions are presented in Section 6.

2 Preliminaries

In the following, P is a set of n points in \mathbb{R}^d . The *diameter* of P , denoted by Δ , is the maximum distance between a pair of points of P . The two points $p_\Delta, q_\Delta \in P$ realizing the diameter of P are the *diametrical pair* of P .

2.1 Fair Split Tree

In our algorithm, we use the following hierarchical data-structure known as Fair Split Tree [CK95] (a similar notion was previously suggested by Vaidya [Vai89]). Generally speaking, Fair-split tree is an adaptive variant of quadtrees. (Our algorithm can be described using

¹In fact, Chan's algorithm can be speeded up to $O(n + 1/\varepsilon^2)$ using ideas from of [BHP99]

quadtrees. However, we chose to describe it using the fair-split tree as in practice it seems to perform better, resulting in a somewhat more involved analysis.)

Given a point-set P of n points in \mathbb{R}^d , a fair-split tree is a tree having the points of P stored in its leaves (a leaf might contain several points of P), where each point of P is stored only once in the tree. For a node v , let $P(v)$ denote the points stored in the subtree of v , and $\mathcal{R}(v)$ denote the minimum axis-parallel bounding box of $P(v)$. Let $c(v)$ denote the center of $\mathcal{R}(v)$, and let $r(v)$ denotes the radius of the minimum ball centered at $c(v)$ that contains $\mathcal{R}(v)$.

Let T_0 be the tree formed by a single node that corresponds to the whole set P . The tree is constructed iteratively as follows: In the i -th stage one picks a leaf u of T_{i-1} which contains more than one point of P . Compute the minimum axis parallel bounding box $\mathcal{R}(u)$, split it in the middle of its longest edge, create two children u_l, u_r of u that correspond to the two new boxes, and compute $P(u_l), P(u_r)$ from $P(u)$ and store them in u_l, u_r , respectively. We will use the naive implementation of this operation that takes $O(|P(u)|)$ time (it can be done faster with appropriate preprocessing; see [CK95]).

For a node $v \in T$, $\text{parent}(v)$ denotes the parent of v in T , and $l_{max}(v)$ denotes the length of the longest edge of $\mathcal{R}(v)$. A point $p \in P$ belongs to v if $p \in \mathcal{R}(v)$. We associate with a pair of nodes (u, v) of T , the set of pairs of points $P(u, v) = (P(u) \times P(v)) \cup (P(v) \times P(u))$. A pair of points $p, q \in P$ belongs to the pair (u, v) if $(p, q) \in P(u, v)$. For a pair $\mu = (u, v)$ of nodes of a fair-split tree T let

$$M(\mu) = M(u, v) = |c(u)c(v)| + r_u + r_v$$

denote the naive upper bound on the maximum distance of any pair of points of $P(u, v)$.

2.2 Review of Known Approximation Algorithms for computing the Diameter

In this section, we review the known approximation algorithms to the diameter. We implemented most of them and compared them to our new algorithm. Approximation algorithms for the diameter were previously suggested by Agarwal et al. [AMS92], Barequet and Har-Peled [BHP99] and Chan [Cha00]. Barequet and Har-Peled algorithm is a folklore result, although we are not aware of earlier reference.

Axis parallel bounding box Let $\mathcal{R} = \mathcal{R}(P)$ denote the axis parallel bounding box of P . Let $p, q \in P$ be the two points that lie on the two sides of \mathcal{R} furthest away from each other. Clearly, $|pq| \leq \Delta \leq \sqrt{d}|pq|$. Since B can be computed in linear time, it follows that we can compute a constant approximation to the diameter of P (i.e., p, q) in linear time.

Snapping P to a Grid We decompose $\mathcal{R} = \mathcal{R}(P)$ into a grid G of $k \times k \times \dots \times k$ cells, where $k = \lceil 1/\varepsilon \rceil$. Each cell is a shrunk copy of \mathcal{R} by a factor of k . Let P'_G be the set of the centers of the cells of G that contains points of P in their interior. Note, that P'_G might contain more than two points on an axis-parallel line ℓ . Since the pair of points that realize the diameter are pair of vertices of $\mathcal{CH}(P_G)$, it is clear that we can remove all the points of P'_G on such a line ℓ which are not extreme. Repeating this process for all the axis-parallel lines

that pass through points of P'_G , reduces the number of points from $O(1/\varepsilon^d)$ to $O(1/\varepsilon^{d-1})$, and let P_G denote the resulting set of points. We call this process *grid cleaning*. It takes $O(n + 1/\varepsilon^{d-1})$ time. Being slightly more careful about the implementation, the running time can be reduced to $O(n)$. Clearly,

$$(1 - \varepsilon)\Delta(P) \leq \Delta(P_G) \leq (1 + \varepsilon)\Delta(P).$$

Computing the diameter of P_G in a brute force manner takes $O(1/\varepsilon^{2(d-1)})$ time. As observed by Barequet and Har-Peled [BHP99], by using known bounds on the number of vertices of convex hulls of grid points, together with output-sensitive convex-hull algorithms, this result can be slightly improved. See [BHP99] for the details. For $d = 3$ case, [BHP99] present an algorithm that runs in $O(n + 1/\varepsilon^{3/2})$, which is only of theoretical interest as it uses the algorithm of Clarkson and Shor [CS89].

Searching for the Diameter Direction Let ℓ be a line, and let P_ℓ be the projection of P into ℓ . Clearly, the distances between a pair of projected points of P_ℓ is smaller than the original distance of the pair. In particular, $\Delta(P_\ell) \leq \Delta(P)$. However, if the angle α between ℓ and the diametrical pair $p_\Delta q_\Delta$ is smaller than $\sqrt{2\varepsilon}$ then $\Delta(P_\ell) \geq |p_\Delta q_\Delta| \cos(\alpha) \geq (1 - \alpha^2/2)\Delta \geq (1 - \varepsilon)\Delta$. Thus, by covering the sphere of directions in \mathbb{R}^d by $O(1/\varepsilon^{(d-1)/2})$ caps of angular radius $\leq \sqrt{2\varepsilon}$, and picking a vector inside each cap, we have a set V of $O(1/\varepsilon^{(d-1)/2})$ directions, so that $p_\Delta q_\Delta$ is an angular distance $\leq \sqrt{2\varepsilon}$ from one of those directions. In particular, projecting the points of P into each of the directions of V , finding the extreme pair in each direction, and taking the pair of maximum distance, results in an ε -approximation to Δ . The running time is $O(n/\varepsilon^{(d-1)/2})$.

By first doing grid cleaning (with $\varepsilon/2$) and then using the projection algorithm, we have an ε -approximation to the diameter that works in $O(n + 1/\varepsilon^{3(d-1)/2})$ time.

Reducing the number of projections Chan [Cha00] observed that the bottleneck in the above algorithm is the large number of projection it performs. Instead of projecting the points to lines, project the points into hyperplanes, and compute the approximate diameter of the projected point-set. One can easily find a family of $O(1/\sqrt{\varepsilon})$ hyperplanes, so that the angular distance between the diameter and one of the hyperplanes is smaller than $\sqrt{\varepsilon}$. Now compute recursively, for each projected point-set its $\varepsilon/2$ approximate diameter. The maximum diametrical pair computed in the recursive calls is an ε -approximation to the diametrical pair.

Observing that one can perform grid-cleaning before projecting the points (and thus also in the recursive calls on the $(d-1)$ -dimensional projected points) results in an algorithm with $O(n + 1/\varepsilon^{d-1/2})$ running time. By doing more aggressive grid-cleaning (using convex-hull algorithms in two and three dimensions) one can reduce the running time to $O(n + 1/\varepsilon^{d-1})$. See [Cha00].

3 The Algorithm

Let P be a point-set in \mathbb{R}^d , $\varepsilon > 0$ a prescribed approximation parameter, and T be a fair-split tree of P . We will compute parts of T as the algorithm progresses in an on-demand fashion,

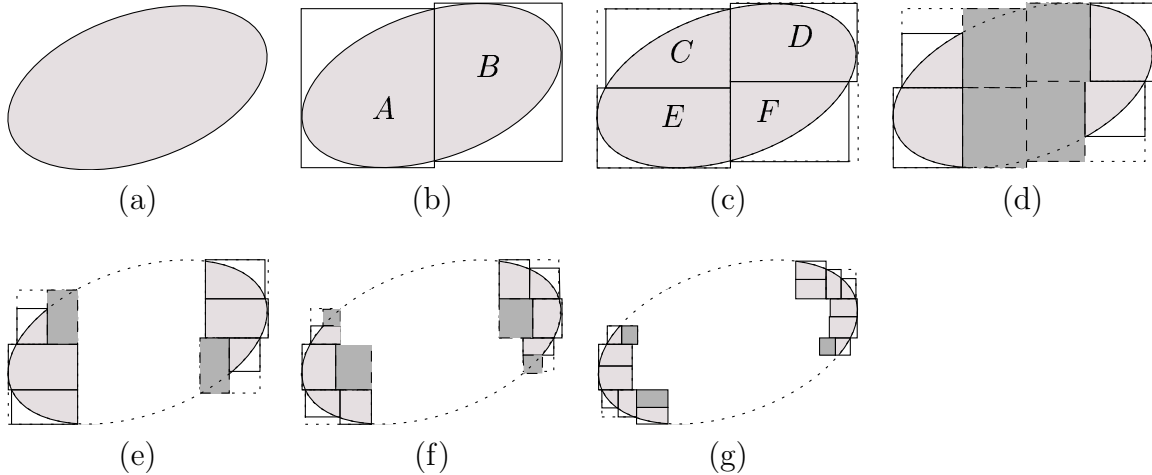


Figure 1: Illustration of how `AprxDiameter` works on a densely sampled ellipse. As the algorithm progresses the pairs-decomposition become finer, and pairs that are too short are being thrown away. The dark regions are nodes that all the pairs associated with them were thrown away. As time progresses, the pairs maintained by the algorithm converge to the real diameter. It seems that for all “real” inputs, after several iterations, only a small fraction of the input points would be contained in the active nodes maintained by the algorithm.

starting from a tree having a single node that contains all the points of P .

The algorithm works by maintaining a current approximation to the diameter, denoted by Δ_{curr} , and a set \mathcal{P}_{curr} of pairs of nodes of T , so that all the pairs of points that (substantially) might improve our approximation must belong to one of those nodes pairs. Such a node pair might couple a node with itself (i.e., in the beginning we couple the root of T with itself as all the $\Theta(n^2)$ pairs of points are candidates to be the diametrical pair).

The algorithm initializes Δ_{curr} to be the side length of $\mathcal{R}(\text{root}(T))$ (which is a \sqrt{d} approximation to the diameter). And $\mathcal{P}_{curr} = \{(\text{root}(T), \text{root}(T))\}$. The algorithm associates the value $M(u, v)$ with a pair (u, v) , and it maintains a heap of the pairs of \mathcal{P}_{curr} , sorted by a decreasing order by this value. In each iteration, the algorithm picks the maximum element in the heap (i.e., the element (u, v) in \mathcal{P}_{curr} with maximum $M(u, v)$, as intuitively, this is the pair with the highest potential to improve our current estimate of the diameter), inspect it, and either throws it away, or generates $O(1)$ pairs from it, and stores them in \mathcal{P}_{curr} (updating Δ_{curr} in the process).

In particular, a pair (u, v) can be thrown away if $P(v)$ or $P(u)$ are empty, or if

$$M(u, v) \leq (1 + \varepsilon)\Delta_{curr}, \quad (1)$$

as $M(u, v)$ is an upper bound on the distance of any pair of points in $P(v) \times P(u)$, and the above condition states that all pairs in $P(u, v)$ are within $1 + \varepsilon$ of our current estimate to the diameter.

If the algorithm decides not to throw away a pair (u, v) , it expands it by all the pairs formed by the children of u and v . Namely, (u, v) generates the four pairs $(\text{left}(u), \text{left}(v))$, $(\text{left}(u), \text{right}(v))$, $(\text{right}(u), \text{left}(v))$, $(\text{right}(u), \text{right}(v))$ (however, if $u = v$ the pairs $(\text{left}(u), \text{right}(v))$, $(\text{right}(u), \text{left}(v))$ are identical, and only three pairs should be generated in such a case),

where $\text{left}(u), \text{right}(u)$ denotes the left and right child of u respectively. Note, that if $P(u)$ (or $P(v)$) is a singleton we do not split it, splitting the pair only on one side of it. We insert the new pairs into \mathcal{P}_{curr} .

Whenever the algorithm creates a pair (u, v) , it also picks two points $p \in P(u), q \in P(v)$, computes $|pq|$, and update Δ_{curr} if $|pq| > \Delta_{curr}$. Furthermore, a new pair is being added to \mathcal{P}_{curr} only if it passes the condition of Equation (1). This guarantees that as the pair decomposition becomes finer also the current approximation improves.

As we descend in T from u to $\text{left}(u), \text{right}(u)$ in the expansion process, it might be that $\text{left}(u)$ or $\text{right}(u)$ do not exist yet in T , as the node u was not split yet. In this point in time, we split u creating $\text{left}(u), \text{right}(u)$ (splitting the set $P(u)$ into $P(\text{left}(u)), P(\text{right}(u))$). We argue below that our algorithm constructs only a small fraction of the whole fair-split tree of P .

The algorithm stop when \mathcal{P}_{curr} becomes empty and returns the value of Δ_{curr} . We call this algorithm `AprxDiameter`. We modify slightly this algorithm, as follows: When splitting a pair (u, v) instead of generating four pairs, we split only the node in the pair having the larger bounding box (i.e., for a pair (u, v) we split u if $l_{max}(u) > l_{max}(v)$), generating only two new pairs. The advantage of the new approach is that, although pairs might still be unbalanced (i.e., the sizes of their corresponding bounding boxes are of different orders of magnitude), the pair is balanced if we refer to the node's parent instead of the node itself. Let `AprxDiamWSPD` denote the new algorithm. This is how the well-separated pairs decomposition is constructed by the algorithm of [CK95].

Figure 1 illustrates how the algorithm works, and why in practice the algorithm works well: For most inputs only a small fraction of the nodes of the fair-split tree are active (i.e., participate in a pair which is stored in \mathcal{P}_{curr}) after several iterations. In particular, only small fraction of the tree is being constructed by the algorithm, and similarly, only a small fraction of all possible pairs are considered by the algorithm.

Lemma 3.1 *The algorithms `AprxDiameter`, `AprxDiamWSPD` always stop, and returns a number D , such that $(1 - \varepsilon)\Delta \leq D \leq \Delta$. Moreover, the running time of the algorithms is $O(n^2 \log n)$ for any value of ε .*

Proof: Note, that when a node is split in the fair-split tree, at least one point of P goes to each child. Thus, there are $O(n)$ total nodes in T . It is easy to verify by induction that the algorithm never generates the same pair twice. Finally, if the algorithm encounters a pair (u, v) such that $P(u), P(v)$ are a singletons, then the algorithm computes the length of the corresponding pair of points, updates Δ_{curr} , and then the pair fails the condition of Equation (1) (indeed, $M(u, v)$ is no more than the length of the corresponding pair, as $r_u = r_v = 0$), and the algorithm throws this pair away.

As for the running time, we note that the time spent on constructing the whole fair-split tree is $O(n^2)$ in the worst case. Since splitting and handling each pair takes $O(\log n)$ time (i.e., finding the maximal element in the heap of pairs), and there are $O(n^2)$ pairs, it follows that the running time of the algorithm is $O(n^2 \log n)$, as $O(n^2)$ insertions/deletions are being performed on the heap.

As for the quality of approximation: Let p_Δ, q_Δ be the diametrical pair of P , and let (u, v) be the pair constructed by the algorithm such that $(p_\Delta, q_\Delta) \in P(u, v)$ and (u, v) fails

the condition of Equation (1). Furthermore, let Δ'_{curr} denote the value of Δ_{curr} when the condition failed, and let Δ_{final} denote the final value of Δ_{curr} returned by the algorithm. Clearly,

$$(1 + \varepsilon)\Delta_{final} \geq (1 + \varepsilon)\Delta'_{curr} \geq M(u, v) \geq |p_{\Delta}q_{\Delta}| = \Delta.$$

Thus,

$$\Delta_{final} \geq \frac{\Delta}{1 + \varepsilon} \geq \frac{\Delta}{1 + \varepsilon + \varepsilon^2 + \dots} = (1 - \varepsilon)\Delta. \quad \blacksquare$$

4 Theoretical Analysis of AprxDiameter and AprxDiamWSPD

In this section, we prove that the running times of `AprxDiameter` and `AprxDiamWSPD` are $O((n + 1/\varepsilon^{2d}) \log 1/\varepsilon)$ and $O((n + 1/\varepsilon^{3(d-1)/2}) \log 1/\varepsilon)$, respectively. The less theoretically inclined reader is encouraged to skip this section.

Corollary 4.1 *Let $(u_1, v_1), (u_2, v_2), \dots, (u_m, v_m)$ the sequence of pairs handled by `AprxDiameter`. Then $M(u_1, v_1) \geq M(u_2, v_2) \geq \dots \geq M(u_m, v_m)$.*

Proof: Observe that if a pair (w, x) was expanded from (u, v) than $M(u, v) \geq M(w, x)$. Since the algorithm always pick the pair in \mathcal{P}_{curr} that maximizes the value of $M(\cdot, \cdot)$ implies, by induction, the statement. \blacksquare

We next improve the analysis of Lemma 3.1. For a pair $\mu = (u, v)$ let $l_{max}(\mu) = l_{max}(u, v) = \max(l_{max}(\mathcal{R}(u)), l_{max}(\mathcal{R}(v)))$. Let \mathcal{U} denote the set of pairs created by `AprxDiameter` during its execution. We partition \mathcal{U} into classes:

$$S_i = \left\{ (u, v) \mid (u, v) \in \mathcal{U}, \frac{\Delta}{2^{i+1}} < l_{max}(u, v) \leq \frac{\Delta}{2^i} \right\},$$

for $i \geq 1$. Let r_i denote the maximum radius of nodes of pairs of S_i :

$$r_i = \max_{(u,v) \in S_i} \max(r(u), r(v)).$$

Observation 4.2 $\Delta/2^{i+1} \leq r_i \leq \sqrt{d}\Delta/2^i$.

Lemma 4.3 *For $i > K = \lceil \log \frac{4d}{\varepsilon} \rceil$, the sets S_i are empty. This holds for the pairs generated by the algorithms `AprxDiameter` and `AprxDiamWSPD`.*

Proof: Let (u, v) be a pair of S_i , and Δ'_{curr} be the value of Δ_{curr} when (u, v) was inserted into \mathcal{P}_{curr} . Clearly, $M(u, v) \leq \Delta'_{curr} + 4r_i$, since Δ'_{curr} was updated by a pair of points $p \in P(u), q \in P(v)$ before the pair was created (i.e., $\Delta'_{curr} \geq |pq|$). On the other hand, the pair (u, v) was inserted into the set \mathcal{P}_{curr} , which implies that $M(u, v) > (1 + \varepsilon)\Delta'_{curr}$. Furthermore, $\Delta'_{curr} \geq \Delta/\sqrt{d}$ (as Δ_{curr} initial value was no smaller than this, and its value only increase during the algorithm execution). Thus,

$$r_i \leq \frac{\sqrt{d}\Delta}{2^i} \leq \frac{\varepsilon}{4d} \cdot \sqrt{d}\Delta = \varepsilon \frac{\Delta}{4\sqrt{d}} \leq \frac{\varepsilon\Delta'_{curr}}{4},$$

and

$$\Delta'_{curr} + 4r_i \geq M(u, v) > (1 + \varepsilon)\Delta'_{curr} \geq \Delta'_{curr} + 4(\varepsilon\Delta'_{curr}/4) \geq \Delta'_{curr} + 4r_i.$$

A contradiction. ■

We observe, that the children of a pair $(u, v) \in S_i$ might still be in S_i . However, after at most d splits the descendant pair must belong to a class S_j , where $j > i$. Indeed, each split shrinks by half the longest dimension of the bounding boxes on both sides of the pair. After d splits, the maximum side of the a box must have shrunk by 2, excluding it from S_i . Since the algorithm compute the minimum axis-parallel bounding box of a node during its construction, a pair generated from a pair μ might be several classes away from μ .

The pairs generated from pair (u, v) of depth l in T (i.e., the maximum depth of u and v is l) have depth at most $l + 1$. Since there are only $\lceil \log \frac{4d}{\varepsilon} \rceil$ non-empty classes, and at most d consecutive elements on a path of T might belong to the same class. We conclude that the deepest node visited by the algorithm is of depth $\leq d \lceil \log \frac{4d}{\varepsilon} \rceil$. We conclude:

Corollary 4.4 *The algorithm `AprxDiameter` constructs at most $O(1/\varepsilon^d)$ nodes of T . In particular, the running time of `AprxDiameter` is $O((n + 1/\varepsilon^{2d}) \log 1/\varepsilon)$.*

Proof: The fair-split tree T is a binary tree. By the above discussion, the deepest node constructed is of depth $\leq d \lceil \log \frac{4d}{\varepsilon} \rceil$. Thus, the overall number of nodes constructed is

$$O\left(2^{d \lceil \log \frac{4d}{\varepsilon} \rceil}\right) = O\left(\left(\frac{4d}{\varepsilon}\right)^d\right) = O\left(\frac{1}{\varepsilon^d}\right).$$

Thus, the overall number of pairs constructed by the algorithm is $O(1/\varepsilon^{2d})$. Furthermore, each operation on the heap now takes only $O(\log 1/\varepsilon)$ time. Constructing each level in T takes $O(n)$ times, and the algorithm constructs $O(\log 1/\varepsilon)$ levels. We conclude that the overall running time of the algorithm is $O((n + 1/\varepsilon^{2d}) \log 1/\varepsilon)$. ■

Unfortunately, Corollary 4.4 is almost tight. The reason being that pairs might have a node with a huge bounding box on one side, and a tiny bounding box on the other side. Although this is not a “natural” input, one can easily construct such an input for which the running time is close to the bound stated in Corollary 4.4. We thus turn our attention to `AprxDiamWSPD`.

Observation 4.5 *If a pair (u, v) computed by `AprxDiamWSPD` belongs to the class S_i , then $l_{max}(\text{parent}(u)), l_{max}(\text{parent}(v)) > \Delta/2^{i+1}$.*

Lemma 4.6 ([CK95], Lemma 4.1) *Let T be a fair split tree associated with P , C a cube, $S = \{v_1, \dots, v_l\}$ be a set of nodes in T such that $P(v_i) \cap P(v_j) = \emptyset$ for all $i \neq j$, $l_{max}(\text{parent}(v_i)) \geq l_{max}(C)/c$, $\mathcal{R}(v_i) \cap C \neq \emptyset$. Let $K(c, d)$ be the maximum number of elements of S . We have, $K(c, d) \leq (3c + 2)^d$.*

Intuitively, Lemma 4.11 states that the number of nodes of a fair-split tree that are roughly of the same size which are close to each other is limited (i.e., it is impossible to have a huge number of very thin and long cells close to each other). In the following, let N_i denote the set of all the nodes of the fair split tree that participate in pairs of S_i .

We bound the running time of `AprxDiamWSPD` by bounding the number of pairs created by the algorithm. Our proof relies on the following observations: (i) the nodes of N_i are not densely packed together (Lemma 4.7), (ii) all the pairs of S_i are of similar length (Lemma 4.9), (iii) The nodes of N_i lie “close” to the boundary of the convex-hull of P (Lemma 4.10), (iv) no two pairs of S_i can be too “far” from each other (Lemma 4.11), and (v) clustering the pairs by directions and proximity, we derive our bound on the number of pairs generated by the algorithm (Lemma 4.14).

Lemma 4.7 *Let T be a fair split tree associated with P , C a cube, $S = \{v_1, \dots, v_l\} \subseteq N_i$ be a set of nodes in T , $\mathcal{R}(v_j) \cap C \neq \emptyset$, for $j = 1, \dots, l$. Let $K(r, d, i)$ be the maximum number of elements of S , where $r = l_{\max}(C)$. We have,*

$$K(r, d, i) = O\left(\left(\frac{r2^i}{\Delta}\right)^d\right).$$

Proof: Let u, v be two nodes of S . If u is an ancestor of v in T , we claim that the distance in T between the two nodes is at most $2d + 1$. Indeed, by Observation 4.5, $l_{\max}(\text{parent}(v)) > \Delta/2^{i+1}$. In particular, let v_a be the ancestor of v in distance $2d + 1$ from it. Since each time we climb a level in T , the size of one of the coordinates double at least, and this new value is larger or equal to the previous longest dimension, it follows, that $l_{\max}(\cdot)$ at least doubles when we climb d levels. It follows that $l_{\max}(v_a) > \Delta/2^{i-1}$. However, v_a can not possibly participate in a pair of S_i (i.e., its bounding box $\mathcal{R}(v_a)$ is too large). Implying that the distance between u and v in T is at most $2d + 1$.

Let S' be the set of all nodes of S , such that none of their ancestors is in S (this is the set of “maximal” nodes in S). By the above argument, and since T is a binary tree, it follows: $|S| \leq |S'|2^{2d+1} = O(|S'|)$. Furthermore, all the nodes of S' corresponds to rectangles which are disjoint and not intersecting. Note that for $v \in S'$, we have $l_{\max}(v) \geq \Delta/2^{i+1}$, and

$$\frac{l_{\max}(C)}{l_{\max}(v)} \leq \frac{r2^{i+1}}{\Delta}.$$

In particular, by Lemma 4.6, we have

$$|S'| \leq \left(\left(\frac{3r2^{i+1}}{\Delta} + 2\right)^d\right) = O\left(\left(\frac{r2^i}{\Delta}\right)^d\right),$$

which implies the result as $|S| = O(|S'|)$. ■

Lemma 4.8 *Given a cube C of side length ℓ_0 partitioned into k^d equal size cubes by a grid of cubes G , and D be convex shape. The number of cells of G in distance at most L from any point of the boundary of D is $O(k^{d-1}(1 + (kL/\ell_0)^d))$.*

Proof: This is a standard argument about the relation between a surface area of a convex shape, and the number of grid cells that it intersects (in fact, stronger bounds can be proved). For $x = 0$ this is no more than the number of grid cells of G that intersects ∂D . Let C' be a

cell of G that intersects ∂D . Let f be the highest dimensional feature of $\partial C'$ that intersects ∂D . One can charge this intersection to the full dimensional flat that supports f . Since the number of such flats is $O(k^{d-1})$ as can be easily verified, and each such flat is charged at most twice. It follows that the number of such grid cells is $O(k^{d-1})$.

As for any value of x , we charge a grid cell C' in distance $\leq x$ from ∂D to the grid cell that contains this nearest point. Each grid cell that intersects ∂D is being charged $O((kL/\ell_0)^d)$ times, and the result follows. \blacksquare

Let S'_i be the subset of pairs of S_i that were expanded by `AprxDiamWSPD`. In the following, we charge the elements of S_i to their parents (a pair can be charged at most $O(1)$ times), and we bound the number of elements in S'_i .

Let M_i denote the maximum of $M(u, v)$ overall pairs $(u, v) \in S'_i$, and let μ_i the pair that realizes M_i . Clearly, by Corollary 4.1, μ_i is the first pair of S'_i being handled by `AprxDiamWSPD`. In particular, when μ_i is being handled the value of Δ_{curr} , denoted by Δ_{curr}^i , is at least $M_i - 4r_i$. Furthermore, for all $(u, v) \in S'_i$, we have

$$M(u, v) \geq \Delta_{curr} \geq \Delta_{curr}^i \geq M_i - 4r_i. \quad (2)$$

Note, that if any of the pairs (u, v) of S_i is such that $P(u, v)$ contains the diametrical pair of P , it follows that $M_i \geq \Delta$. Otherwise, the diametrical pair was thrown away together with a pair (u, v) . But arguing as in the proof of Lemma 3.1, it follows that $M_i \geq \Delta_{curr}^i \geq (1-\varepsilon)\Delta$. Furthermore, since μ was expanded by the algorithm, it means that condition Equation (1) failed, implying that $M_i \geq (1+\varepsilon)(1-\varepsilon)\Delta = (1-\varepsilon^2)\Delta$. Thus, $(1-\varepsilon^2)\Delta \leq M_i \leq \Delta + 4r_i$. However, by Lemma 4.3, if S'_i is not empty then $16dr_i \geq \varepsilon\Delta$. In particular, $\Delta - 16dr_i \leq M_i \leq \Delta + 4r_i$. We conclude:

Lemma 4.9 *If the set S'_i is not empty, then for any pair $(u, v) \in S'_i$ we have*

$$\Delta - (16d + 4)r_i \leq M(u, v) \leq \Delta + 4r_i,$$

and, in particular, $\Delta \leq M_i + (16d + 4)r_i$.

Lemma 4.10 *Let \mathcal{CH} be the convex hull of P , and let v be a node of N_i . The distance of $\mathcal{R}(v)$ from the boundary of \mathcal{CH} is at most $21dr_i$.*

Proof: Indeed, let (u, v) be a pair of S_i that contains v . Clearly, if the distance between $\mathcal{R}(v)$ and the boundary of $\mathcal{CH}(P)$ is larger than $21dr_i$ then

$$\Delta = \Delta(\mathcal{CH}(P)) > M(u, v) - 2r_i + 21dr_i,$$

as we can pick two points of $\mathcal{R}(u), \mathcal{R}(v)$ that lies inside $\mathcal{CH}(P)$ of distance $\geq M(u, v) - 2r_i$, and by the above condition, we can extend this segment by length $21dr_i$ while remaining inside $\mathcal{CH}(P)$.

This implies that

$$\Delta > M(u, v) + 21dr_i - 2r_i \geq M_i + 21dr_i - 6r_i \geq M_i + 18r_i \geq M_i + 16r_i + 4r_i,$$

since $d \geq 2$, and by Equation (2). Contradiction to Lemma 4.9. \blacksquare

In the following, we will bound the number of pairs generated by `AprxDiamWSPD` by a packing argument.

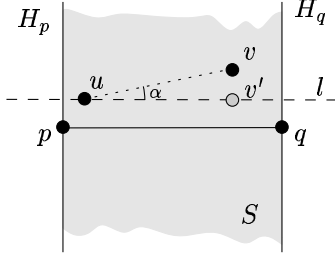


Figure 2: Illustration of the proof of Lemma 4.11

Lemma 4.11 *Let $\delta, 0 < \delta < 0.5$ be a given parameter, and u, v, p, q four points taken from a set $Q, Q \subseteq \mathbb{R}^d$. And we assume that the following holds:*

- *pq, uv are “almost” diametrical pair: $1 - \delta \leq |pq|, |uv| \leq 1 + \delta$.*
- *None of the four points are too close to each other: $|pu|, |pv|, |qu|, |qv| \geq 3\sqrt{\delta}$*
- *The angle $\alpha = \alpha(uv, pq)$ between the line supporting uv and the line supporting pq is $\leq \sqrt{\delta}/3$.*

Then, $\Delta(Q) \geq \max\{|qu|, |pv|, |pu|, |qv|\} > 1 + \delta$.

Proof: Let H_p, H_q be the two hyperplanes passing through p, q , respectively, and perpendicular to the segment pq . Let S be the strip bounded by H_p and H_q .

If $u \notin S$, and assume that u is closer to p than to q , then $|qu|$ is minimized when u lies on H_p . Thus,

$$|qu| = \sqrt{|pq|^2 + |pu|^2} \geq \sqrt{|pq|^2 + 3^2\delta} = \sqrt{(1 - \delta)^2 + 9\delta} > \sqrt{(1 + \delta)^2} = 1 + \delta,$$

since $|pu| \geq 3\sqrt{\delta}$.

The case that $v \notin S$ is handled in a similar fashion. Thus, we remain with the case $u, v \in S$. Let ℓ be the line passing through u and parallel to pq (we assume here that u is closer to pq than v). Let v' be the projection of v on ℓ . It is easy to verify that $|pv'| \leq |pv|$, $|qv'| \leq |qv|$. See Figure 2.

Furthermore,

$$|uv'| = |uv| \cos \alpha \geq \left(1 - \frac{\alpha^2}{2}\right) |uv| \geq \left(1 - \frac{\delta}{18}\right) |uv| \geq \left(1 - \frac{\delta}{18}\right) (1 - \delta) \geq 1 - 2\delta.$$

Assume that p is closer to u . It is easy to verify that in such a case, $|pv'|$ is minimized when $u \in H_p$, and $|pu| = 3\sqrt{\delta}$. Indeed, u, v' are by construction in S . Thus, $|pv'|$ is minimized, when $|pu| = 3\sqrt{\delta}$. In turn, this case is minimized when the angle $\alpha(pu, uv') = \pi/2$, which implies that $u \in H_p$. However,

$$|pv'| = \sqrt{|pu|^2 + |uv'|^2} \geq \sqrt{9\delta + (1 - 2\delta)} > 1 + \delta,$$

and the result follows. ■

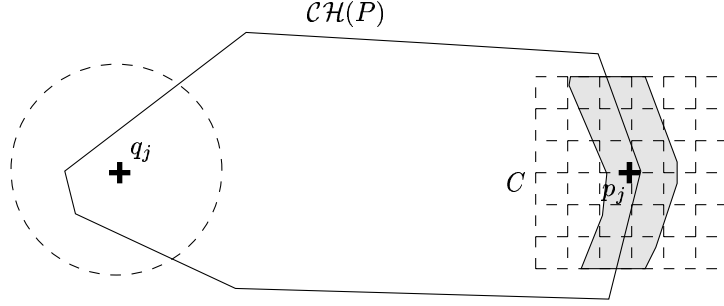


Figure 3: Illustration of the proof of Lemma 4.12. The cells that participate in a cluster U_j are centered around p_j and q_j . By Lemma 4.10 all such nodes must lie close to the surface of the convex-hull $\mathcal{CH}(P)$.

We cluster the pairs of S'_i as follows: We pick any pair $(u_1, v_1) \in S'_i$ and let U_1 be the set of all pairs (u', v') in S'_i such that, $|c(u)c(u')|, |c(v)c(v')| \leq \rho_i$, where

$$\rho_i = 100\sqrt{\Delta dr_i} + 54dr_i.$$

We continue in similar fashion, clustering $S'_i \setminus U_1$. Let U_1, \dots, U_{k_i} be the set of clusters of S'_i , where k_i denotes the number of resulting clusters. Let (u_j, v_j) denote the pair of S'_i that created U_j .

Lemma 4.12 *The number of pairs in a cluster U_j is $O(2^{i(d-1)})$, for $j = 1, \dots, k_i$.*

Proof: A cluster U_j have pairs with bounding boxes with distance $\leq \rho_i$ from the centers p_j, q_j of $\mathcal{R}(u_j), \mathcal{R}(v_j)$, respectively, where (u_j, v_j) is the pair of nodes defining the cluster U_j .

Let $W \subseteq N_i$ be the set of all the nodes of T that participate in U_j and that are in distance at most ρ_i from p_j .

Let C be a cube side length $2\rho_i$, and let G be the grid created inside C by breaking it into cubes of side length r_i . All the nodes of W must intersect C (and thus one of the cells of G), and are in distance at most $21dr_i$ from the boundary of $\mathcal{CH}(P)$, by Lemma 4.10. See Figure 3. By Lemma 4.8 the number of grid cells of G that are in such distance from $\partial\mathcal{CH}(P)$ is

$$O\left(\left(\frac{\rho_i}{r_i}\right)^{d-1}(1 + (21dr_i/r_i)^d)\right) = O\left(2^{i(d-1)/2}\right).$$

But on the other hand, each such cell grid of G , intersects at most $K(r_i, d, i) = O((r_i 2^i / \Delta)^d) = O(1)$ cells of N_i (and thus of W) by Lemma 4.7. In particular, the number of nodes in W is $O(2^{i(d-1)/2} K(r_i, d, i)) = O(2^{i(d-1)/2})$.

Using a similar argument, we can bound the number of nodes of N_i that appear in the other side of U_j . In particular, the overall number of pairs in U_j , is the product of those two quantities, which is $O((2^{i(d-1)/2})^2) = O(2^{i(d-1)})$. \blacksquare

Lemma 4.13 *For any two clusters U_j, U_m , the angle between $c(u_j)c(v_j)$ and $c(u_m)c(v_m)$ is larger than β , for $j \neq m$, where $(u_j, v_j), (u_m, v_m)$ are two pairs defining the clusters U_j, U_m , and*

$$\beta = \sqrt{\frac{3dr_i}{\Delta}}.$$

Proof: Assume that α , the angle between $c(u_j)c(v_j)$ and $c(u_m)c(v_m)$, is smaller than β . We remind the reader, that the clustering radius is $\rho_i = 100\sqrt{\Delta dr_i} + 54dr_i$. And let $\delta = 27dr_i/\Delta$. If all the lengths of all the pairs in $F = \{c(u_j), c(v_j)\} \times \{c(u_m), c(v_m)\}$ are all larger than $\rho_i/2 \geq 3\sqrt{\delta}\Delta$, then by Lemma 4.11, we know that the length of one of the pairs (u, v) of F is larger than $\Delta(1 + \delta)$. Which is impossible, because then there are two points $p \in P(u), q \in P(v)$ such that $|pq| \geq \Delta(1 + \delta) - 4r_i > \Delta + 20dr_i > \Delta$. A contradiction.

Thus, there must be a pair in F which is shorter than $\rho_i/2$, and assume without loss of generality that $|c(u_j)c(u_m)| < \rho_i/2$. However, since (u_m, v_m) is not in the cluster of (u_j, v_j) , it follows that $|c(v_j)c(v_m)| > \rho_i$. Using simple geometric arguments, it follows that

$$\frac{\alpha}{2} \geq \sin \frac{\alpha}{2} \geq \frac{\rho_i - 2\delta\Delta}{2} \geq 50\sqrt{\frac{dr_i}{\Delta}} \geq \beta.$$

A contradiction. ■

Lemma 4.14 *The number of pairs created by AprxDiamWSPD is $O(1/\varepsilon^{3(d-1)/2})$.*

Proof: By Lemma 4.13, the angle between the axes of any two clusters U_j, U_m is larger than $\beta = \sqrt{\frac{3dr_i}{\Delta}}$. This implies, by using a packing argument on the sphere of directions in \mathbb{R}^d , that the number of clusters k_i is at most

$$O\left(\frac{1}{\beta^{d-1}}\right) = O\left(\left(\frac{\Delta}{r_i}\right)^{(d-1)/2}\right) = O\left(2^{i(d-1)/2}\right),$$

by Observation 4.2.

By Lemma 4.12, the number of pairs of S'_i in each cluster is $O(2^{i(d-1)})$. We conclude, that the number of pairs in S_i is $O(2^{3i(d-1)/2})$. Summing over $i = 1, \dots, \lceil \log \frac{4d}{\varepsilon} \rceil$ (using Lemma 4.3), we conclude, that the overall number of pairs created by the algorithm AprxDiamWSPD is $O(1/\varepsilon^{3(d-1)/2})$. ■

Theorem 4.15 *The algorithm AprxDiamWSPD ε -approximates the diameter in*

$$O\left(\left(n + \frac{1}{\varepsilon^{3(d-1)/2}}\right) \log \frac{1}{\varepsilon}\right)$$

time.

5 Experimental Results

We had implemented a number of algorithms for computing the diameter. The algorithms were implemented using C++ on a Pentium III 800MhZ running Linux with 512MB memory. Below we detail the implemented algorithms, and the results themselves are shown in Tables 1, 2. Note, that the running time of the naive exact algorithm is only estimated (this was done by running the algorithm for 10,000 points and scaling the running time according to the number of points at hand).

5.1 Algorithms Implemented

The source code of the program is available in [HP00]. In particular, an independent module that implements the algorithm is provided to be used by other programs.

5.1.1 Approximation Algorithms that can Compute the Exact Solution

In the following, FS stands for Fair-Split tree and WSPD stand for Well Separated Pairs Decomposition [CK95].

- **FS Heap** - implementation of `AprxDiameter`.
- **FS Heap Ptr** - implementation of `AprxDiameter` so that the algorithm manipulates pointers to the points during the tree construction, instead of copying the points themselves. This modification results in an improvement in execution time, mainly because of a slightly more careful implementation of this module.
- **FS WSPD** - implementation of `AprxDiamWSPD`.
- **FS Levels** - A variant of FS WSPD that avoids the usage of a heap by using a FIFO (first in, first out) queue (i.e., in the i -th iteration all the pairs of depth i in the tree are split).
- **FS Levels Ptr** - implementation of FS Levels so that the algorithm manipulates pointers to the points during the tree construction, instead of copying the points themselves. This modification results in a minor improvement in execution time.

5.1.2 ϵ -Approximation Algorithms

- **Chan** - implementation of the algorithm of [Cha00].
- **Chan Mod** - implementation of the algorithm of [Cha00], where the 2d recursive call is computed using a 2d variant of `AprxDiameter`. This variant seems to be slower than the original algorithm.
- **Grid** - First do grid cleaning, and then use FS Levels on the resulting point set.
- **FS Directions** - A variant of FS Heap. When a pair (u, v) of nodes is created, so that $\nu(u, v) \leq \sqrt{\epsilon}$, we project the points in the two pairs to the axis of the pair, and find the two extreme points along the projection, where $\nu(u, v)$ is a conservative estimate to the maximum angle between $c(u)c(v)$ and any segment pq , where $p \in P(u), q \in P(v)$. Return the corresponding pair of points, as the candidate of this pair to be the diameter.
- **Grid FS Dir** - Uses grid cleaning and then apply FS Directions on the resulting set.

5.1.3 Constant factor approximation to the diameter

- **BBox** - Computes the axis parallel bounding box of the input point-set and the two furthest points that lie on opposite faces of the bounding box. This algorithm reads the points only once and as such is a good reference in comparing the running times of the other algorithms.
- **PCA** - computes the center of mass, and principal component analysis to get three orthogonal vectors that represents the input point-set. Scan the point-set to find the three extreme pairs in those directions, and chooses the longest pair as the candidate to be the diameter. Always return a $\sqrt{3}$ approximation to the diameter.

5.2 Inputs

We tried two types of inputs: (i) real graphics models - taken from various publicly-available sources (3D Cafe, and [LGM]). For *all* such models the new algorithm performed extremely well, and the results are shown in Table 1.

We also tried the algorithm on two synthetic inputs for which it performs badly. The first (denoted as sphere in Table 2) is generated by randomly sampling points on a sphere. The distribution used was the uniform distribution (one can prove that the expected running time of `AprxDiamWSPD` to compute the exact diameter for such input in 3d is about $O(n^{3/2})$).

The second type of input used, were points picked from two tiny arcs that are far away from each other and are orthogonal to each other (the supporting tangents of the two arcs lie are orthogonal). Rotated properly, this input requires $\Theta(n^2)$ from `AprxDiamWSPD` to compute the exact diameter. Even for this case, `AprxDiamWSPD` is about twice faster than the naive algorithm (because only $n^2/4$ pairs of points are being tested compared to $n^2/2$ by the naive algorithm).

5.3 Analysis of Results

The new algorithm `AprxDiameter` (FS Heap) and its variants perform well in practice. In particular, it computed the *exact* diameter for the models we tested it with, in time which is at most ten times the time to compute the axis-parallel bounding box. Similar performance gains were seen for $\varepsilon = 0.01$. Chan’s algorithm is faster only for large values of ε , namely $\varepsilon = 0.1$. Even for those values, the fastest algorithm seems to be **Grid** - probably because grid cleaning is so effective for such values of ε . Overall, it seems that the new algorithm performs especially well for “real” inputs.

For the synthetic inputs, the picture is less encouraging. Computing the exact diameter by any of the new algorithms requires a lot of work. For the sphere inputs, once the inputs are large enough, Chan’s algorithms shines being much faster than any of the alternatives. To some extent, this is the most favorable input for Chan algorithm: there are no clear candidates to be the diameter and all feasible directions are equally feasible.

For the arcs type of inputs, the picture is discouraging only if one considers the exact variants. Computing the approximate solution is quite easy, and `AprxDiameter` performs extremely well.

6 Conclusions

We presented a new algorithm for computing the diameter of a point-set in 2 and higher dimensions, concentrating on the 3d case for our experiments. The new algorithms can be easily be implemented in a few hours of work and it performed extremely well for real inputs, although its performance on specific synthetic inputs is quadratic. The new algorithm can also be used as an approximation algorithm and then concrete (near linear) guarantees on its performance are given. The new algorithm has two favorable properties: (i) if executed for long-enough time it arrives sooner or later to the exact solution (this, of course, holds when executing the algorithm with $\varepsilon = 0$), and furthermore one can give guarantees on the quality of approximation as a function of the time the algorithm was executed, and (ii) the algorithm is sensitive to the hardness of the input and is fast on “easy” inputs.

In conclusion, the problem of the diameter seems to be easy in practice. Even the most naive algorithm (i.e., axis parallel bounding-box) computes the *exact* diameter for most inputs. Although there were “real” inputs for which only `AprxDiameter` (and its variants) was able to provide the exact diameter, in most cases the error was quite small. Inputs for which the simple algorithms perform really badly compared with `AprxDiameter` are rare (i.e., error larger than 10%). However, it is quite easy to generate inputs for which both the BBox and PCA algorithms give only a $\sqrt{3}$ approximation. Thus, the new algorithm should be used in applications where guarantees on the quality of approximation are necessary, and speed is a major consideration. Otherwise, if the application can manage with a rough approximation of the diameter it seems that the axis-parallel bounding box approach might be sufficient.

As mentioned before, the source code of the program used in the experiments is available from [HP00].

The author conjectures that the currently (theoretically) fastest approximation algorithm for $d > 3$, due to Chan [Cha00], which runs in $O(n + 1/\varepsilon^{d-1})$ time, can be significantly improved. We leave that as an open question for further research.

Acknowledgments

The author wishes to thank Pankaj Agarwal, Boris Aronov, Jeff Erickson, Piotr Indyk, Lutz Kettner and Edgar Ramos for helpful discussions concerning the problem studied in this paper and related problems. Finally, the author thank the anonymous referees for a number of useful comments.

References

- [AMS92] P.K. Agarwal, J. Matoušek, and S. Suri. Farthest neighbors, maximum spanning trees and related problems in higher dimensions. *Comput. Geom. Theory Appl.*, 1(4):189–201, 1992.
- [BHP99] G. Barequet and S. Har-Peled. Efficiently approximating the minimum-volume bounding box of a point set in three dimensions. In *Proc. 10th ACM-SIAM Sympos. Discrete Algorithms*, pages 82–91, 1999.

- [Cha00] T.M. Chan. Approximating the diameter, width, smallest enclosing cylinder and minimum-width annulus. In *Proc. 16th Annu. ACM Sympos. Comput. Geom.*, pages 300–309, 2000.
- [CK95] P.B. Callahan and S.R. Kosaraju. A decomposition of multidimensional point sets with applications to k -nearest-neighbors and n -body potential fields. *J. ACM*, 42:67–90, 1995.
- [CS89] K. L. Clarkson and P. W. Shor. Applications of random sampling in computational geometry, II. *Discrete Comput. Geom.*, 4:387–421, 1989.
- [FL95] C. Faloutsos and K. I. Lin. FastMap: a fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 24(2):163–174, June 1995.
- [HP00] S. Har-Peled. Source code of program for computing and approximating the diameter of a point-set in 3d, 2000. http://www.uiuc.edu/~sariel/papers/00/diameter/diam_prog.html.
- [LGM] Large geometric models archive. Georgia Tech. http://www.cc.gatech.edu/projects/large_models/index.html.
- [Ram00] E. Ramos. Deterministic algorithms for 3-d diameter and some 2-d lower envelopes. In *Proc. 16th Annu. ACM Sympos. Comput. Geom.*, pages 290–299, 2000.
- [Tou83] G. T. Toussaint. Solving geometric problems with the rotating calipers. In *Proc. IEEE MELECON '83*, pages A10.02/1–4, 1983.
- [Vai89] P. M. Vaidya. An $O(n \log n)$ algorithm for the all-nearest-neighbors problem. *Discrete Comput. Geom.*, 4:101–115, 1989.

input points	Running time in seconds							
	Clock 30,744	Bunny 35,947	Dragon 54,831	DS9 298,517	Hand 327,323	Dragon II 437,645	Buddha 543,652	Blade 882,954
$\varepsilon = 0$								
Naive (estimate)	79.77	109.06	253.76	7521.81	9043.52	16167.00	24947.50	65805.40
FS Heap	0.09	0.08	0.10	0.94	0.62	1.08	1.10	1.58
FS Heap Ptr	0.07	0.07	0.07	0.72	0.46	0.85	0.85	1.22
FS WSPD	0.08	0.10	0.10	0.93	0.61	1.05	1.08	1.58
FS Ptr Levels	0.09	0.11	0.12	0.88	0.58	1.12	1.11	1.69
FS Levels	0.09	0.11	0.13	0.94	0.63	1.13	1.15	1.80
$\varepsilon = 0.01$								
FS Heap	0.08	0.09	0.09	0.89	0.61	1.01	1.06	1.60
FS Heap Ptr	0.07	0.06	0.07	0.68	0.45	0.79	0.83	1.22
FS WSPD	0.08	0.10	0.09	0.87	0.59	0.99	1.05	1.59
FS Ptr Levels	0.08	0.10	0.12	0.86	0.58	1.06	1.05	1.67
FS Levels	0.10	0.10	0.12	0.90	0.61	1.07	1.09	1.78
Chan	1.55	1.77	2.55	4.75	8.78	12.27	14.83	16.02
Chan Mod	1.31	1.67	2.43	4.67	8.79	12.35	14.95	16.11
Grid	0.12	0.20	0.22	0.61	0.79	1.22	1.30	1.61
Grid FS Dir	0.32	0.44	0.29	5.61	0.80	2.28	1.90	1.63
FS Directions	0.40	0.24	0.15	17.18	0.67	2.03	1.81	1.75
$\varepsilon = 0.1$								
FS Heap	0.06	0.07	0.09	0.76	0.56	0.84	0.91	1.38
FS Heap Ptr	0.04	0.06	0.07	0.55	0.41	0.67	0.71	1.04
FS WSPD	0.06	0.06	0.09	0.76	0.50	0.82	0.91	1.39
FS Ptr Levels	0.06	0.08	0.10	0.71	0.53	0.82	0.87	1.30
FS Levels	0.06	0.08	0.11	0.75	0.56	0.84	0.93	1.39
Chan	0.09	0.12	0.11	0.35	0.38	0.56	0.69	1.03
Chan Mod	0.09	0.13	0.10	0.35	0.38	0.55	0.68	1.02
Grid	0.04	0.04	0.06	0.32	0.34	0.47	0.58	0.94
Grid FS Dir	0.04	0.05	0.07	0.32	0.33	0.47	0.58	0.91
FS Directions	0.11	0.07	0.10	1.99	0.55	0.91	0.97	1.38
$\varepsilon = \frac{\sqrt{3}-1}{\sqrt{3}} \approx 0.42$								
BBox	0.01	0.01	0.01	0.05	0.05	0.07	0.09	0.14
PCA	0.01	0.01	0.03	0.10	0.11	0.14	0.18	0.29

Table 1: Results for various models

input	Running time in seconds							
	Sphere	Sphere	Sphere	Sphere	Arcs	Arcs	Arcs	Arcs
points	1,000	10,000	100,000	200,000	1,000	10,000	100,000	200,000
$\epsilon = 0$								
Naive (estimate)	0.08	8.44	844.07	3376.32	0.08	8.44	844.07	3376.32
FS Heap	0.04	1.25	55.41	223.36	0.04	3.75	365.21	1442.58
FS Heap Ptr	0.03	2.15	136.55	423.34	0.03	5.07	362.04	1292.57
FS WSPD	0.04	1.34	59.76	173.44	0.04	3.75	365.74	1444.46
FS Ptr Levels	0.04	1.13	55.80	203.32	0.04	3.55	351.15	1404.38
FS Levels	0.03	1.18	61.93	208.66	0.04	3.64	359.02	1443.97
$\epsilon = 0.01$								
FS Heap	0.06	8.20	43.46	154.39	0.06	7.93	0.07	0.14
FS Heap Ptr	0.08	7.07	123.71	358.02	0.08	7.11	0.05	0.10
FS WSPD	0.03	1.05	41.75	107.21	0.00	0.00	0.06	0.13
FS Ptr Levels	0.03	1.95	38.19	138.19	0.00	0.01	0.06	0.13
FS Levels	0.04	1.11	39.23	141.77	0.00	0.00	0.06	0.13
Chan	0.03	0.94	4.52	8.43	0.00	0.00	0.15	0.27
Chan Mod	0.04	0.98	4.96	8.95	0.00	0.01	0.15	0.26
Grid	0.21	0.64	43.71	137.76	0.06	0.06	0.12	0.23
Grid FS Dir	0.35	1.00	1205.56	> 1 hour	0.05	0.06	0.13	0.22
FS Directions	0.06	1.09	553.49	1193.06	0.02	0.03	0.06	0.13
$\epsilon = 0.1$								
FS Heap	0.06	1.65	0.40	0.78	0.02	0.03	0.06	0.13
FS Heap Ptr	0.04	1.62	4.72	6.00	0.00	0.01	0.05	0.09
FS WSPD	0.02	0.16	0.37	0.74	0.00	0.00	0.07	0.15
FS Ptr Levels	0.02	0.44	0.44	0.89	0.00	0.00	0.06	0.13
FS Levels	0.02	0.19	0.38	0.76	0.00	0.01	0.07	0.13
Chan	0.02	0.14	0.24	0.37	0.00	0.01	0.11	0.22
Chan Mod	0.02	0.15	0.24	0.36	0.00	0.00	0.10	0.21
Grid	0.01	0.08	0.30	0.42	0.01	0.02	0.10	0.22
Grid FS Dir	0.01	0.08	0.44	0.56	0.00	0.01	0.11	0.21
FS Directions	0.02	0.19	10.24	20.97	0.00	0.01	0.07	0.13
$\epsilon = \frac{\sqrt{3}-1}{\sqrt{3}} \approx 0.42$								
BBox	0.03	0.31	0.02	0.03	0.00	0.01	0.02	0.04
PCA	0.02	0.52	0.04	0.06	0.00	0.01	0.03	0.07

Table 2: Results for synthetic inputs