Dynamic Coresets

Timothy M. Chan*

January 2, 2009

Abstract

We give a dynamic data structure that can maintain an ε-coreset of n points, with respect to the extent measure, in $O(\log n)$ time per update for any constant $\varepsilon > 0$ and any constant dimension. The previous method by Agarwal, Har-Peled, and Varadarajan requires polylogarithmic update time. For points with integer coordinates bounded by U, we alternatively get $O(\log\log U)$ time. Numerous applications follow, for example, on dynamically approximating the width, smallest enclosing cylinder, minimum bounding box, or minimum-width annulus. We can also use the same approach to maintain approximate k-centers in time $O(\log n)$ (or $O(\log\log U)$ if the spread is bounded by U) for any constant k and any constant dimension.

For the smallest enclosing cylinder problem, we also show that a constant-factor approximation can be maintained in $O(1)$ randomized amortized time on the word RAM.

1 Introduction

The topic of this paper lies at the intersection of two often-studied subareas in computational geometry: approximation algorithms and dynamic data structures.

Geometric approximation algorithms, via coresets. It is a well-known fact that many geometric optimization problems can be solved more efficiently if approximate solutions are acceptable. For example, the fastest exact algorithm known to compute the width of n points requires $O(n^{3/2+\delta})$ randomized time [8] in dimension 3 (for any fixed $\delta > 0$) and $O(n^{\lceil d/2 \rceil})$ time [16] in a fixed dimension $d \geq 4$, but linear-time approximation algorithms [27, 16] guaranteeing an approximation factor $1+\varepsilon$ exist for any arbitrarily small constant $\varepsilon > 0$ in all fixed dimensions.

Agarwal, Har-Peled, and Varadarajan [2] have proposed a unified approach to developing linear-time approximation algorithms for many basic geometric optimization problems, including the computation of the width, smallest enclosing cylinder, and minimum-volume enclosing box of a point set P. The approach involves finding a subset S of P of constant size so that a solution for S yields an approximate solution for P. Such a subset is now commonly referred to as a *coreset*.

Agarwal, Har-Peled, and Varadarajan went on to show that computing coresets for many problems can be reduced to computing coresets for one particular problem: the extent. The *extent* of a point set P along a given direction is the width of the minimum slab orthogonal to the direction that encloses P. We want a subset S of P such that the extent of S is at least $1-\varepsilon$ times the extent of P

*School of Computer Science, University of Waterloo, Waterloo, Ontario N2L 3G1, Canada (tmchan@uwaterloo.ca).
This work has been supported by NSERC. A preliminary version of this paper has appeared in *Proc. 24th ACM Sympos. Comput. Geom.*, 2008.

along *every* direction. (Such a coreset is specifically called an ε-*kernel* by some authors.) Agarwal *et al.* showed that for the extent problem, a coreset of constant size ($O(\varepsilon^{-(d-1)/2})$) always exists and can be found in linear time.

Coresets with respect to extent enjoy many nice properties. For example, it is possible (though not entirely trivial) to prove that a $(1+O(\varepsilon))$-factor scaled and translated copy of the convex hull of the coreset S contains the convex hull of P, which in turn contains the convex hull of S. The notion is thus extremely fundamental and corresponds to an approximate version of convex hulls in the strongest (containment-wise) sense. (Before, proposed definitions of "approximate convex hulls" from the literature were weaker and less clean.) It follows immediately that we can get coresets for problems like width and minimum-volume bounding box, since these problems "depend" only on the convex hull. The approach can also yield coresets for some other problems of a "non-convex" nature, like minimum-width spherical shells (annuli in the 2-d case) and minimum-width cylindrical shells, by first applying a lifting transformation, also shown by Agarwal, Har-Peled, and Varadarajan.

Subsequently, a paper by the author [16] has examined improvements of the ε-dependence in the running time, and a paper by Yu *et al.* [49] has explored implementations. (See [3] for a recent survey.) Numerous recent papers have also studied other types of coresets, for clustering problems like k-centers, k-medians, k-means, k-line-centers, etc. (e.g., [36, 35]), problems dealing with outliers (e.g., [4, 38]), problems in high, nonconstant dimensions (e.g., [11, 10, 37]), and combinations of the above.

Dynamic geometric data structures. Designing dynamic data structures constitute another fundamental topic in computational geometry. For example, the dynamic 2-d convex hull problem has been studied since Overmars and van Leeuwen's seminal work in the early 1980s [44], with breakthroughs found two decades later [15, 14]. It is now known that we can maintain a 2-d point set P under insertions and deletions of points in $O(\log n)$ amortized time, so that we can find an extreme point on the convex hull of P along any query direction in $O(\log n)$ time [14]. With $O(\log^{3/2} n)$ amortized update time, we can also find the intersection of the convex hull of P with any query line (i.e., perform linear programming queries in the dual) in $O(\log^{3/2} n)$ time [15]. There has also been recent progress on the dynamic 3-d convex hull problem [19]: with $O(\log^6 n)$ amortized expected update time, similar types of queries can be answered in $o(\log^6 n)$ time. For higher-dimensional convex hulls, the update or query time deteriorates rapidly, however [5].

Unfortunately, for many geometric optimization problems, maintaining an exact solution under insertions and deletions can be difficult. Even for the 2-d width problem, the best update time bound known for a fully dynamic data structure is $O(\sqrt{n}\,\mathrm{polylog}\, n)$ [17]. For 2-d minimum-area bounding box, a fully dynamic algorithm with sublinear update time has yet to be found (if it exists at all); in the insertion-only or offline special case, the current best update bound is $O(n^{5/6+\delta})$ [18]. The situation gets even worse in higher dimensions.

Dynamization meets approximation. It thus makes perfect sense to turn to dynamic *approximation* algorithms for more reasonable update times. Some early work in this direction [40, 46] had considered the 2-d width problem and showed that a $(1+\varepsilon)$-factor approximation can be maintained in $O(\log^2 n)$ time per insertion and deletion, using the known dynamic convex hull results at the time; with current results [14], it is actually not difficult to find an algorithm with $O(\log n)$ amortized update time.

Surprisingly, determining the best update bound for the next most basic problem, 3-d width, has remained open. With known dynamic 3-d convex hull results (or 3-d linear programming) [19], it is possible to maintain a $(1+\varepsilon)$-factor approximation in $O(\log^6 n)$ amortized update time, by adapting a known static algorithm [27]. But is there a more direct dynamic approximation algorithm that eliminates some or all of these logarithmic factors? And what about width in dimensions beyond 3, where polylogarithmic dynamic convex hull results are not available at all?

Agarwal, Har-Peled, and Varadarajan's remarkable paper [2] also touched on these questions. They gave a simple data structure that can maintain a constant-size coreset, with respect to extent, in $O(\log^d n)$ time per insertion and deletion (without any resort to dynamic exact convex hull results). This result automatically leads to polylogarithmic dynamic algorithms for approximate width, minimum-volume enclosing box, minimum-width spherical shell, etc. in *all* fixed dimensions.

The author's improved coreset algorithm [20] can trim the update time slightly to $O(\log^{d-1} n \log\log n)$, but it is unclear how to get a bound that has a constant number of logarithmic factors independent of d. In particular, for 3-d width, we so far do not have an $O(\log n)$ algorithm.

Better update time is known for one special case: the insertion-only case. Surprisingly, the author [20] has shown how to maintain a coreset (and thus a $(1+\varepsilon)$-factor approximation for width, minimum-volume enclosing box, etc.) in *constant* time and *with constant space* for any constant $\varepsilon > 0$. Insertion-only algorithms that use small amounts of space are better known as *streaming* algorithms. Subsequent papers [9, 50] have looked into refining the ε-dependences of the space bound.

The literature on streaming-related models has other relevant results. For example, approximation algorithms for diameter and 2-d width have been proposed under the *sliding-window* model [29, 24]—here, both insertions and deletions are allowed, but points must be deleted in the same order as they are inserted. Approximation algorithms for other problems, such as k-median clustering and Euclidean minimum spanning tree, have also been given under the general dynamic streaming model [39, 31, 30]. All these results in the sliding-window and dynamic streaming models require space and time that have several logarithmic factors at least; worse, these logarithmic factors are not in n but in the universe size[1] U, under the assumption that the input coordinates are integers in the range $\{0,\ldots,U\}$ (where $U = \Omega(n^{1/d})$). For these models, the reliance on U is unavoidable. So, these streaming-related results do not help resolve our question of finding the best dynamic data structures, where $O(n)$ space is acceptable.

We mention that a lower bound of $\Omega(\log n)$ update time can be obtained in the decision tree model, for example, for the problem of maintaining a factor-c approximate width under insertions and deletions in 2-d, for any constant c. We can simply reduce from the 1-d priority queue problem over a set S of positive integers: fix two anchor points at $(-M,0)$ and $(M,0)$ for a sufficiently large M, and create a point $(0,(c+1)^k)$ for each $k \in S$; then the current width is equal to $(c+1)^m$ where m is the maximum of S. (If we could solve the priority queue problem over S in $o(\log n)$ time, then we would be able to sort any permutation of S in $o(n\log n)$ time, a contradiction.)

Our main result. In this paper, we present a new data structure that can maintain a constant-size coreset, with respect to extent, in $O(\log n)$ time[2] for all constant dimensions. The space usage is $O(n)$. Automatically, the result implies dynamic $(1+\varepsilon)$-factor approximation algorithms with

[1] In some papers, the universe size is denoted by Δ.

[2] We have omitted writing out dependences of the constant factors on ε in the introduction, to keep the readers focused on the main parameter n. In the body of the paper, we will write these ε-dependences fully.

logarithmic update time for the long list of problems mentioned (width, minimum-volume bounding box, ...). Thus, we have successfully eliminated all the extra logarithmic factors and essentially settled the complexity of these dynamic approximation problems for the first time.

Our method is conceptually simple and nice, at least in the randomized version, as described in Section 3. The basic idea involves an interesting, dynamic variation of prune-and-search: we try to identify a fraction of the input point set for which maintaining a coreset is easy, and recurse on the rest. A derandomization is theoretically possible, as shown in Section 4, using (curiously) approximate centerpoints.

Applications. Just as dynamic data structures have applications to the design of algorithms, our data structures can be applied to speed up existing approximation algorithms. We mention two examples:

- Agarwal, Har-Peled, and Yu [4] have recently given a simple algorithm to solve variants of geometric optimization problems (width, minimum-volume bounding box, ...) that allow for k outliers. Their "peeling" algorithm works by computing a coreset, then deleting all of its points, and repeating for $O(k)$ iterations. Their running time can be reduced from $O(nk)$ to $O(n + k \log n)$ by using our dynamic coreset method. (The situation is loosely analogous to how dynamic convex-hull data structures are used to compute convex layers [25].)

- Har-Peled [34] gave an algorithm for an approximate decision version of the "2-plane-center" problem (covering a point set by 2 slabs of minimum maximum width) in 3-d. His algorithm uses dynamic approximate 3-d width as a subroutine. His $O(n \log^3 n)$ time bound can now be reduced to $O(n \log n)$. Improvements for the corresponding optimization problem also follow. (The situation is loosely analogous to how dynamic width data structures are used to solve the exact 2-line-center problem [7].)

More results: approximation meets word RAM. As we have noted, $O(\log n)$ update time is optimal in the decision tree model, but actually is not the end of the story. The first reason is theoretical: for the analogous 1-d priority queue problem, sublogarithmic data structures are known on the word RAM model [48, 47]; see [22, 23] for more on word-RAM algorithms in computational geometry. The second reason is more practically significant: in order to reduce the standard priority queue problem to dynamic approximate width, for example, one needs to construct point sets whose spread is at least exponential in n, which are uncommon.

Geometric approximation algorithms frequently use the floor function to round input to grid points with small integer coordinates, and on occasion even incorporate RAM techniques like hashing. Thus, an argument could be made that the word RAM model is more appropriate than the real RAM for approximation algorithms (e.g., see [41]); the word RAM is actually more "realistic", since words cannot hold infinite-precision real numbers in practice. Also, many approximation algorithms in the past have specifically made use of the (reasonable) assumption that the input has bounded spread (e.g., see [21]), or that the input coordinates are bounded integers. Regarding the latter assumption, we have already mentioned known $O(\text{polylog}\, U)$-type results in dynamic streaming and sliding window models; in more traditional settings, we can also cite Edwards and Varadarajan's work [28] on coresets for the k-hyperplane-center problem (covering a point set by k slabs of minimum maximum width), where coresets of small size are known *not* to exist without the bounded integer input assumption [34], but coresets of $O(\text{polylog}\, U)$ size are possible.

In Section 5, we show that an update time much smaller than $O(\log n)$ is possible for points with coordinates from $\{0, \ldots, U\}$ for reasonable values of U. Specifically, we get $O(\log\log U)$ for coresets with respect to extent and the other problems. The new method is obtained by combining our $O(\log n)$ method with additional ideas. It does not involve van Emde Boas trees (contrary to the reader's first impression perhaps) or fancier RAM data structures.

In Section 7, we show that $O(\log\log U)$ might not be the end of the story either, at least in one special case. Specifically, we show that for smallest enclosing cylinder, a constant-factor approximation can be maintained in *constant* randomized amortized time under both insertions and deletions. Note that in the 2-d case, the smallest enclosing cylinder problem is equivalent to width.

More results: k-centers. In Section 6, we venture beyond extent-related problems and show that our approach can be adapted to at least one clustering problem: the k-center problem (finding k congruent balls of minimum radius covering a point set). Coresets for k-centers have been studied before (e.g., see [3, 33, 34]), but no satisfactory dynamic algorithms have been given to the best of the author's knowledge. We show that a $(1+\varepsilon)$-factor approximation can be maintained in $O(\log n)$ or $O(\log\log U)$ randomized amortized time for any constant ε, constant k, and constant dimension. Our algorithm for k-center is uncannily similar to our algorithm for extent.

It should be stated that all our randomized results are Las Vegas and only require the assumption that the update sequence is oblivious to the random choices made by our algorithms.

2 Preliminaries

We assume that the dimension d is constant; big-O notation may hide constant factors that depend on d (possibly exponentially).

We begin by stating the particular definition of coresets that we will work with in the next three sections.

Definition 2.1 Given a unit vector x, let $w(P, x) = \max_{p,q\in P}(p-q)\cdot x$ (the "extent" along direction x) and let $f(P, x) = \max_{p\in P} p \cdot x$. Call a subset $S \subseteq Q$ an *ε-coreset of Q relative to P* if $f(S, x) \geq f(Q, x) - \varepsilon w(P, x)$ for every unit vector $x \in \mathbb{R}^d$.

We will describe how to maintain an ε-coreset S of a point set P relative to P itself. Since $w(S, x) = f(S, x) + f(S, -x)$, it follows that $w(S, x) \geq (1 - 2\varepsilon)w(P, x)$, so S is indeed a coreset with respect to extent (after readjusting ε by a constant factor).

Trivially, an ε-coreset of Q relative to any subset of P is an ε-coreset of Q relative to P. An ε-coreset of an ε'-coreset of Q is an $(\varepsilon + \varepsilon')$-coreset of Q (relative to the same set P). Moreover, if S_i is an ε-coreset of Q_i, then $\bigcup_i S_i$ is an ε-coreset of $\bigcup_i Q_i$ (relative to the same set P); thus, coresets are "decomposable". These properties alone are sufficient to imply an efficient polylogarithmic dynamic algorithm, by taking coresets of unions of coresets recursively in the form of a tree, as described by Agarwal, Har-Peled, and Varadarajan [2]. The trouble is: we lose precision at each of the $\Theta(\log n)$ layers of recursion, so we need to reset ε to $\Theta(\varepsilon/\log n)$ at the bottom layer, resulting in numerous extra logarithmic factors.

Instead, we attempt to modify directly a static coreset algorithm, to identify a special case where dynamization is easy. The following lemma is based on the static algorithm by Agarwal, Har-Peled, and Varadarajan, which in turn builds on a simple constant-factor algorithm by Barequet and Har-Peled [12]. A quick proof (essentially a reinterpretation of known ideas) is included for the sake of completeness.

Let $\operatorname{aff} S$ denote the affine hull (smallest enclosing flat) of S, and $\mathrm{d}(A, B)$ denote the minimum Euclidean distance between sets A and B.

Lemma 2.2 *Fix $d+1$ "anchor" points $p_0, \ldots, p_d$. Consider a "special" point set Q satisfying*

$$\mathrm{d}(p, \operatorname{aff}\{p_0, \ldots, p_{j-1}\}) \ \leq \ c\, \mathrm{d}(p_j, \operatorname{aff}\{p_0, \ldots, p_{j-1}\}) \qquad \forall j = 1, \ldots, d, \ \forall p \in Q \cup \{p_0, \ldots, p_d\},$$

for some constant $c \geq 1$. We can maintain an ε-coreset of Q relative to $\{p_0, \ldots, p_d\}$ of size $O(\varepsilon^{-d})$, in $O(1)$ time under insertions and deletions of points in Q; in particular, the preprocessing time is $O(|Q|)$.

Alternatively, the size bound can be reduced to $O(\varepsilon^{-(d-1)/2})$ but the update time is increased to $O(\varepsilon^{-(d-1)/2} \log |Q|)$ and preprocessing time to $O(\varepsilon^{-(d-1)/2}|Q|)$.

Proof: Let $e_0 = (0, \ldots, 0)$, $e_1 = (1, 0, \ldots, 0)$, $\ldots$, $e_d = (0, \ldots, 0, 1)$. We describe an affine transformation to make $p_0 = e_0$, $\ldots$, $p_d = e_d$ while preserving the above condition for some (larger) constant: Let f_{j-1} denote the flat $\operatorname{aff}\{p_0, \ldots, p_{j-1}\}$. Assume inductively that $p_0 = e_0$, $\ldots$, $p_{j-1} = e_{j-1}$. Say $p_j = (\xi_1, \ldots, \xi_d)$. By rotating around f_{j-1}, we can make $\xi_{j+1} = \cdots = \xi_d = 0$. For every point $p = (x_1, \ldots, x_d) \in Q \cup \{p_0, \ldots, p_d\}$, the stated condition implies that $|x_j| \leq c|\xi_j|$; furthermore, $|x_1|, \ldots, |x_d|, |\xi_1|, \ldots, |\xi_j| \leq c$, since p_0 is the origin and $\mathrm{d}(p, p_0), \mathrm{d}(p_j, p_0) \leq c\mathrm{d}(p_1, p_0) = c$. (We may assume $\xi_j \neq 0$, because otherwise the j-th coordinates would all be 0 and the dimension d can be decreased.) We apply a shear-like mapping $(x_1, \ldots, x_d) \mapsto (x_1 - \frac{\xi_1}{\xi_j} x_j, \ldots, x_{j-1} - \frac{\xi_{j-1}}{\xi_j} x_j, \frac{1}{\xi_j} x_j, x_{j+1}, \ldots, x_d)$. This sends p_j to e_j without changing $p_0, \ldots, p_{j-1}$. The coordinates of any point $p \in Q \cup \{p_0, \ldots, p_d\}$ are still bounded by a constant; hence, so are the distances of p to $f_0, \ldots, f_{j-1}$. The distances of p to $f_j, \ldots, f_{d-1}$ are unchanged. Thus, the stated condition is still true after readjusting c. After d inductive steps, the final c remains a constant. (Note that in implementation, the transformation can be found directly by computing a matrix inverse.)

We are justified to take the transformation because of the following property: if S is an ε-coreset of Q relative to P and M is an affine transformation, then $M(S)$ is an ε-coreset of $M(Q)$ relative to $M(P)$.

After transformation, we can use a simple method to construct a coreset S: build a uniform grid with side length ε and keep only one point per grid cell in S. Then for any unit vector x, $f(S, x)$ and $f(Q, x)$ differ by at most an additive error of $O(\varepsilon)$. This is at most $O(\varepsilon) w(\{p_0, \ldots, p_d\}, x)$, since the width of $\{e_0, \ldots, e_d\}$ is a constant $1/\sqrt{d}$ by straightforward calculations. (We can readjust ε by a constant factor.)

Since Q is contained in $[-c, c]^d$, the maximum size of the coreset, i.e., the number of grid cells, is $O(\varepsilon^{-d})$. Insertion/deletion is easily doable in $O(1)$ time, by looking up an appropriate grid cell via the floor function, as the transformation does not change (the anchor points are fixed).

Alternatively, the coreset size can be reduced to $O(\varepsilon^{-(d-1)/2})$ by directly applying a refined method by the author [20] or Yu *et al.* [49] to the transformed point set. This method requires finding the nearest neighbors to a set of $O(\varepsilon^{-(d-1)/2})$ grid points. Using a priority queue for each such point, these nearest neighbors can be maintained in $O(\varepsilon^{-(d-1)/2} \log |Q|)$ (or actually $O(\varepsilon^{-(d-1)/2} \log(1/\varepsilon))$ time per update. (Both the stated preprocessing and update bounds can be further improved with more effort, though such improvements will not matter at the end.) $\square$

We will let Q.insert-special() and Q.delete-special() denote the insertion and deletion procedure in the above lemma, and Q.preprocess-special$(p_0, \ldots, p_d)$ denote the preprocessing procedure, which takes linear time by n insertions.

The lemma immediately implies a linear-time static algorithm for computing an ε-coreset of Q relative to Q for any point set Q. For each $j = 1, \ldots, d$, we can set p_j to be the point of Q farthest from aff$\{p_0, \ldots, p_{j-1}\}$, computable in linear time. The condition in the lemma is met, with $c = 1$. We would like to dynamize this method.

Generally speaking, deletions tend to be more challenging than insertions in designing dynamic data structures. (Indeed, because of decomposability of coresets, we can adapt known techniques to handle insertions [13].) The main difficulty arises when we delete one of the anchor points $p_0, \ldots, p_d$. For worst-case update sequences, such bad events could happen on every update. (Indeed, in the application to coresets with outliers, Agarwal, Har-Peled, and Yu's peeling algorithm [4] tends to select extreme points as the next points to delete.)

To avoid this difficulty, one has to somehow select anchor points that will last (i.e., not be deleted) for a large number of updates. It seems there is not enough freedom in finding anchor points that serve the entire point set. Our idea is to find anchor points that serve a subset Q containing a *fraction* of the given points, and let recursion handle the remaining points R (since coresets are decomposable). The number of special subsets Q maintained at any time would then be logarithmic.

3 A Simple Randomized Method with $O(\log n)$ Update Time

To execute this idea, our first solution will use randomization to select anchor points. The main innovation lies in the preprocessing strategy, which we describe neatly in the pseudocode below.

Fix a positive constant $\alpha \ll 1/d$. Let P be the given point set.

P.preprocess():
0. if $|P|$ is below a constant then return
1. $Q \leftarrow P$, $p_0 \leftarrow$ a random point of P
2. for $j = 1, \ldots, d$ do
3. $\quad A_j \leftarrow$ the $\alpha|P|$ farthest points of Q from aff$\{p_0, \ldots, p_{j-1}\}$
4. $\quad Q \leftarrow Q - A_j$
5. $\quad p_j \leftarrow$ a random point of A_j
6. $R \leftarrow P - Q$
7. Q.preprocess-special$(p_0, \ldots, p_d)$
8. R.preprocess()

The overall coreset is defined as the union of the coresets of all the special subsets kept.

Observe that the condition in Lemma 2.2 is met, with $c = 1$, so indeed an ε-coreset of Q relative to $\{p_0, \ldots, p_d\}$ (and thus relative to P) can be maintained directly. We opt for the second alternative of Lemma 2.2 here. The union of the ε-coresets of Q and R clearly yields an ε-coreset of P relative to P.

After the for loop in lines 2–5, $|R| \leq d\alpha|P|$.

The preprocessing time for a point set P of size n then satisfies the recurrence $T(n) \leq T(d\alpha n) + O(\varepsilon^{-(d-1)/2} n)$, yielding $T(n) = O(\varepsilon^{-(d-1)/2} n)$.

We now describe the update procedures. Fix a sufficiently small positive constant δ. The idea is to rebuild the data structure after every $\delta|P|$ updates, or when an anchor point is deleted. Insertions are applied to R recursively, and not Q (so that the condition in Lemma 2.2 remains true).

```
P.delete(p):
 0. if |P| is below a constant then delete directly and return
 1. P ← P − {p},  P.counter ← P.counter − 1
 2. if P.counter ≤ 0 or p ∈ {p_0, ..., p_d} then
 3.    P.counter ← δ|P|,  P.preprocess(), and return
 4. if p ∈ Q then Q.delete-special(p) else R.delete(p)

P.insert(p):
 0. if |P| is below a constant then insert directly and return
 1. P ← P ∪ {p},  P.counter ← P.counter − 1
 2. if P.counter ≤ 0 then
 3.    P.counter ← δ|P|,  P.preprocess(), and return
 4. R.insert(p)
```

At any time, $|R| \leq (d\alpha + \delta)|P|$. Let $\widetilde{n}$ be the size of P during the last rebuild; the current size $|P| = n$ is always $\Theta(\widetilde{n})$.

The rebuilding cost in line 3 is $O(\varepsilon^{-(d-1)/2}n)$. If the rebuilding is caused by P's counter reaching 0, we can cover this cost by charging $O(\varepsilon^{-(d-1)/2})$ units to each of the $\delta\widetilde{n}$ updates since the last rebuild. If instead we have just deleted one of the anchor points $\{p_0, \ldots, p_d\}$, then we charge $O(\varepsilon^{-(d-1)/2}n)$ units to the current update but the probability of this occurring is only $O(1/n)$, as we observe below. This observation is perhaps intuitively clear (because each anchor point p_j is chosen at random from a set A_j of size $\Omega(n)$), but the formal justification requires some care (because of dependence of A_j on $p_0, \ldots, p_{j-1}$):

Lemma 3.1 *Let p be the point being inserted/deleted at the current time. Then* $\Pr\{p \in \{p_0, \ldots, p_d\}\} = O(1/n)$ *for a sufficiently small constant $\delta > 0$.*

Proof: Fix a time t before the current time. Let $P(t)$ denote P at time t. Let $A_j(t)$ and $p_j(t)$ denote A_j and p_j at time t if a rebuild occurs at time t (they are undefined otherwise), whereas p_j refers to p_j at current time. Set $A_0 = P$ by default. Let $D(t)$ be the points deleted between t and current time. (Note that $D(t)$ and p are fixed and not random variables, by the obliviousness assumption.)

Let $G(t)$ be the event that the last rebuild occurs at time t. Let $E(t)$ be the event that a rebuild occurs at time t, and $F(t)$ be the event that no rebuild occurs in the period after time t and before the current time; then $G(t) = E(t) \wedge F(t)$. For any fixed j, $\Pr\{p_j = p \wedge F(t) \mid E(t)\} \leq \frac{1}{|A_j(t)|} = \frac{1}{\alpha|P(t)|}$. So, $\Pr\{p \in \{p_0(t), \ldots, p_d(t)\} \wedge F(t) \mid E(t)\} \leq \frac{d+1}{\alpha|P(t)|}$. Now, $\Pr\{F(t) \mid E(t)\} \geq 1 - \frac{(d+1)\delta}{\alpha}$, since we have $F(t)$ iff $p_j(t) \notin D(t)$ for all j, and $\Pr\{p_j(t) \in D(t)\} \leq \frac{|D(t)|}{|A_j(t)|} \leq \frac{\delta|P(t)|}{\alpha|P(t)|}$. This assumes that the number of updates between t and the current time is less than $\delta|P(t)|$; otherwise, $F(t)$ would be trivially false. It follows that

$$\Pr\{p \in \{p_0, \ldots, p_d\} \mid G(t)\} \;=\; \frac{\Pr\{p \in \{p_0, \ldots, p_d\} \wedge F(t) \wedge E(t)\}}{\Pr\{F(t) \wedge E(t)\}}$$

$$
\begin{aligned}
&= \frac{\Pr\{p \in \{p_0, \ldots, p_d\} \wedge F(t) \mid E(t)\}}{\Pr\{F(t) \mid E(t)\}} \\
&\le \frac{(d+1)/(\alpha|P(t))}{1-(d+1)\delta/\alpha} = \frac{1}{(\alpha/(d+1)-\delta)\widetilde{n}}.
\end{aligned}
$$

Since $G(t)$ is true for precisely one t, the $O(1/n)$ probability bound holds unconditionally. $\square$

We conclude that the expected charge per update is $O(\varepsilon^{-(d-1)/2})$ for line 3. The overall expected amortized time per update satisfies the recurrence $U(n) \le \max\{U((d\alpha+\delta)n) + O(\varepsilon^{-(d-1)/2}), O(\varepsilon^{-(d-1)/2}\log n)\}$, yielding $U(n) = O(\varepsilon^{-(d-1)/2}\log n)$, as long as δ is sufficiently small so that $d\alpha + \delta \ll 1$ and $\delta \ll \alpha/(d+1)$.

The size of the overall coreset satisfies the recurrence $N(n) \le N((d\alpha+\delta)n) + O(\varepsilon^{-(d-1)/2})$, yielding $N(n) = O(\varepsilon^{-(d-1)/2}\log n)$. We can reduce the size to $O(\varepsilon^{-(d-1)/2})$ by computing an ε-coreset of the ε-coreset after every update. (We can readjust ε by a constant factor.) The additional time per update is $O(N(n) + \varepsilon^{-(d-3/2)}) = O(\varepsilon^{-(d-1)/2}\log n + \varepsilon^{-(d-3/2)})$ by a static algorithm of the author [20].

4 Derandomization

We now present a deterministic variant of our data structure. To guarantee that anchor points will last, the new idea is to permit anchor points that are not necessarily members of the maintained point set. The key tool is approximate centerpoints:

Definition 4.1 Given a point set $P \subset \mathbb{R}^d$, a β-*centerpoint* is a point $q \in \mathbb{R}^d$ with the property that any halfspace containing q contains at least $\beta|P|$ points of P. Equivalently, we want the point q to lie inside the convex hull of P even after the removal of any subset of less than $\beta|P|$ points from P.

It is well known that a $1/(d+1)$-centerpoint exists for any point set [45]. Linear-time algorithms are also known for computing β-centerpoints for constants $\beta < 1/(d+1)$. For example, one can take a random sample of constant size and return a $1/(d+1)$-centerpoint of the sample by brute force. (Clarkson *et al.* [26] described even faster randomized algorithms.) Deterministically, one can replace the sample with ε-*approximations* [42], constructible in linear time.

Extra complications arise in adapting the preprocessing algorithm. We present the new pseudocode below:

P.preprocess():

0. if $|P|$ is below a constant then return
1. $Q \leftarrow P$, $p_0 \leftarrow$ a β-centerpoint of P
2. for $j = 1, \ldots, d$ do
3. form 2^{d-j+1} "orthants" by drawing $d - j + 1$ pairwise orthogonal hyperplanes through $\operatorname{aff}\{p_0, \ldots, p_{j-1}\}$
4. $Q \leftarrow$ the points of Q in an orthant with $\geq |Q|/2^{d-j+1}$ points of Q
5. $A_j \leftarrow$ the $\alpha|Q|$ farthest points of Q from $\operatorname{aff}\{p_0, \ldots, p_{j-1}\}$
6. $Q \leftarrow Q - A_j$
7. $p_j \leftarrow$ a β-centerpoint of A_j
8. $R \leftarrow P - Q$
9. Q.preprocess-special$(p_0, \ldots, p_d)$
10. R.preprocess()

For the analysis, the following fact is useful:

Lemma 4.2 *Let f be a j-flat and O^+ be one of its orthants. The distance of* $\operatorname{conv}\{p \in O^+ : \mathrm{d}(p, f) \geq r\}$ *to f is $r/\sqrt{d-j}$.*

Proof: By transformation, we can take $r = 1$ and $f = \{(x_1, \ldots, x_d) : x_1 = \cdots = x_{d-j} = 0\}$. The lemma is equivalent to the statement that the distance of $\operatorname{conv}\{(x_1, \ldots, x_{d-j}) : x_1^2 + \cdots + x_{d-j}^2 \geq 1,\ x_1, \ldots, x_{d-j} \geq 0\})$ to the origin is $1/\sqrt{d-j}$. This follows from straightforward calculations. $\square$

Let $f_{j-1} = \operatorname{aff}\{p_0, \ldots, p_{j-1}\}$. By line 5, all points $p \in A_j$ satisfy $\mathrm{d}(p, f_{j-1}) \geq \max_{q \in Q \cup A_{j+1} \cup \cdots \cup A_d} \mathrm{d}(q, f_{j-1}) \geq \max_{q \in Q \cup \{p_{j+1}, \ldots, p_d\}} \mathrm{d}(q, f_{j-1})$, since $p_i \in \operatorname{conv} A_i$. Since $p_j \in \operatorname{conv} A_j$ and A_j is contained in an orthant of f_{j-1}, Lemma 4.2 then implies that $\mathrm{d}(p_j, f_{j-1}) \geq \max_{q \in Q \cup \{p_{j+1}, \ldots, p_d\}} \mathrm{d}(q, f_{j-1})/\sqrt{d}$. So, the condition in Lemma 2.2 is met, with $c = \sqrt{d}$.

After the for loop in lines 2–7, $|Q| \geq [2^{-d}(1-\alpha)]^d |P|$ and so $|R| \leq \left(1 - [2^{-d}(1-\alpha)]^d\right) |P|$. Each A_j has size at least $\alpha[2^{-d}(1-\alpha)]^d |P|$.

We use exactly the same pseudocode for P.insert() and P.delete(), with one exception: the clause $p \in \{p_0, \ldots, p_d\}$ in the if statement can be dropped.

Observe that p_j lies inside $\operatorname{conv} A_j$ even after up to $\beta|A_j|$ points have been deleted. Thus, p_j lies inside $\operatorname{conv} P$ at all times as long as we set $\delta < \beta\alpha[2^{-d}(1-\alpha)]^d$. Therefore, even though the anchor points $p_0, \ldots, p_d$ are not members of P, we know that any ε-coreset of Q relative to $\{p_0, \ldots, p_d\}$ is an ε-coreset relative to P. So, the union of the ε-coresets of Q and R is still an ε-coreset of P relative to P.

At any time, $|R| \leq \left(1 - [2^{-d}(1-\alpha)]^d + \delta\right) |P|$.

The cost of the last rebuild in line 3 is $O(\varepsilon^{-(d-1)/2} n)$. We can cover this cost by charging $O(\varepsilon^{-(d-1)/2})$ units to each of the $\delta\tilde{n}$ updates since the last rebuild.

The amortized update time satisfies the recurrence $U(n) \leq \max\{U(\left(1 - [2^{-d}(1-\alpha)]^d + \delta\right) n) + O(\varepsilon^{-(d-1)/2}),\ O(\varepsilon^{-(d-1)/2} \log n)\}$, yielding $U(n) = O(\varepsilon^{-(d-1)/2} \log n)$, as long as δ is sufficiently small.

We can also make the amortized bound worst-case by applying a standard deamortization technique [43] in which the rebuilding work is spread out over multiple updates.

Theorem 4.3 *We can maintain an ε-coreset of an n-point set in $\mathbb{R}^d$ w.r.t. extent of size $O(\varepsilon^{-(d-1)/2})$ in $O(\varepsilon^{-(d-1)/2} \log n + \varepsilon^{-(d-3/2)})$ time per update.*

5 A Method with $O(\log\log U)$ Update Time

We now describe a data structure with sublogarithmic update time for integer input where coordinates lie in $\{0,\ldots,U\}$. The approach involves combining a slower U-sensitive method with our earlier $O(\log n)$-update-time method.

We first present the slower method, which maintains an ε-coreset of size $O(\varepsilon^{-d}\log^d U)$ but has the advantage of having $O(1)$ amortized update time. The method is again based on Lemma 2.2, but we use a more common idea. Namely, we partition P into a small number of subsets where points having similar distances, to within a constant factor, are grouped into the same subset (this idea has been used before, e.g., in [24, 28]). Then Lemma 2.2 is automatically applicable to each subset, regardless of which anchor points we choose.

More precisely, the preprocessing pseudocode is given below. Initially, we set $j=1$ and $P_{i_0}=P$. We again use randomization to select anchor points.

$P_{i_0\cdots i_{j-1}}$.preprocess():
1. $p_{i_0\cdots i_{j-1}} \leftarrow$ a random point of $P_{i_0\cdots i_{j-1}}$
2. if $j=d+1$ then
3. $\quad P_{i_0\cdots i_d}$.preprocess-special$(p_{i_0}, p_{i_0 i_1},\ldots,p_{i_0\cdots i_d})$ and return
4. for each i_j do
5. $\quad P_{i_0\cdots i_j} \leftarrow \{p\in P_{i_0\cdots i_{j-1}} : \lfloor \log_2 \mathrm{d}(p, \mathrm{aff}\{p_{i_0}, p_{i_0 i_1},\ldots,p_{i_0\cdots i_{j-1}}\})\rfloor = i_j\}$
6. $\quad P_{i_0\cdots i_j}$.preprocess()

By line 5, distances of all points in $P_{i_0\cdots i_j}$ to $\mathrm{aff}\{p_{i_0}, p_{i_0 i_1},\ldots,p_{i_0\cdots i_{j-1}}\}$ are within a factor of 2 from each other. So, the condition of Lemma 2.2 is met for each subset $P_{i_0\cdots i_d}$, with $c=2$. This time, we opt for the first alternative of Lemma 2.2.

In line 4, only indices i_j for which $P_{i_0\cdots i_j}$ is nonempty are considered. Because coordinates are integers in the range $[0,U]$, distances of points to flats can lie in the range $[\Omega(1/U^{d-1}), O(U)]\cup\{0\}$. (Calculations show that squared distances are ratios of integers with denominators bounded by $O(U^{2(d-1)})$, and must thus be $\Omega(1/U^{2(d-1)})$ if nonzero.) The number of choices of each index i_j (positive or negative) is thus $O(\log U)$. In particular, the number of subsets $P_{i_0\cdots i_d}$ is $O(\log^d U)$, and the union of the ε-coresets of these subsets, plus all the anchor points $p_{i_0\cdots i_j}$, yields a coreset of P of size $O(\varepsilon^{-d}\log^d U)$.

The preprocessing time satisfies the recurrence $T_{j-1}(n)\le T_j(n)+O(n)$ with $T_d(n)=O(n)$, yielding $T_0(n)=O(n)$.

The update procedures are straightforward and are described by the rough pseudocode below:

$P_{i_0\cdots i_{j-1}}$.delete(p):
1. $P_{i_0\cdots i_{j-1}} \leftarrow P_{i_0\cdots i_{j-1}} - \{p\}$
2. if $p = p_{i_0\cdots i_{j-1}}$ then $P_{i_0\cdots i_{j-1}}$.preprocess() and return
3. if $j = d+1$ then $P_{i_0\cdots i_d}$.delete-special(p) and return
4. $i_j \leftarrow \lfloor \log_2 \mathrm{d}(p, \mathrm{aff}\{p_{i_0}, p_{i_0 i_1}, \ldots, p_{i_0\cdots i_{j-1}}\}) \rfloor$
5. $P_{i_0\cdots i_j}$.delete(p)

$P_{i_0\cdots i_{j-1}}$.insert(p):
1. $P_{i_0\cdots i_{j-1}} \leftarrow P_{i_0\cdots i_{j-1}} \cup \{p\}$
2. if $|P_{i_0\cdots i_{j-1}}| = 1$ then $P_{i_0\cdots i_{j-1}}$.preprocess() and return
3. if $j = d+1$ then $P_{i_0\cdots i_d}$.insert-special(p) and return
4. $i_j \leftarrow \lfloor \log_2 \mathrm{d}(p, \mathrm{aff}\{p_{i_0}, p_{i_0 i_1}, \ldots, p_{i_0\cdots i_{j-1}}\}) \rfloor$
5. $P_{i_0\cdots i_j}$.insert(p)

For the analysis, fix j and a subset $P_{i_0\cdots i_{j-1}}$, let $\widetilde{n}$ be its size during its last rebuild and u be the number of updates to the subset since the last rebuild. Fix t and let $G(t)$ be the event that the last rebuild of this subset occurs at time t. Let $P_{i_0\cdots i_{j-1}}(t)$ denote the subset at time t, and let $D(t)$ be the points deleted between t and current time.

The rebuilding cost in line 1 of $P_{i_0\cdots i_{j-1}}$.delete() is $O(|P_{i_0\cdots i_{j-1}}|) = O(\widetilde{n} + u)$. We can cover the cost by charging $O(1)$ units to each of the u updates since the last rebuild, plus $O(\max\{\widetilde{n} - u, 1\})$ units to the current update if we have just deleted the anchor point $p_{i_0\cdots i_{j-1}}$. Conditioned on $G(t)$, the probability of this occurring, i.e., that $p_{i_0\cdots i_{j-1}} = p$, is at most $1/\max\{\widetilde{n} - u, 1\}$, since $p_{i_0\cdots i_{j-1}}$ is equally likely to be any member of $P_{i_0\cdots i_{j-1}}(t) - D(t)$.

Therefore, the expected charge per update is $O(1)$ for each fixed j, unconditionally. The overall expected amortized update time satisfies the recurrence $U_{j-1}(n) \leq U_j(n) + O(1)$ with $U_d(n) = O(1)$, yielding $U_0(n) = O(1)$.

We can derandomize the above method in the same way as in Section 4, using approximate centerpoints and subdividing into orthants.

Finally, we can reduce the size of the overall coreset from $N(n) = O(\varepsilon^{-d} \log^d U)$ all the way to $O(\varepsilon^{-(d-1)/2})$, by maintaining a coreset S' of the coreset S using the method from Section 3 or 4. Since each update causes only $O(1)$ (amortized or worst-case) number of updates to S, the update time becomes $O(\varepsilon^{-(d-1)/2} \log N(n) + \varepsilon^{-(d-3/2)}) = O(\varepsilon^{-(d-1)/2} \log\log U + \varepsilon^{-(d-3/2)})$.

Theorem 5.1 *We can maintain an ε-coreset of a point set in $\{0, \ldots, U\}^d$ w.r.t. extent of size $O(\varepsilon^{-(d-1)/2})$ in $O(\varepsilon^{-(d-1)/2} \log\log U + \varepsilon^{-(d-3/2)})$ time per update.*

6 k-Centers

In this section, we adapt the randomized approach in Section 3 to solve the approximate k-center problem for small k in a constant dimension d. The k-center problem can be formulated as follows: given a set P of n points and a number k, we want to find a set of k points $X \subset \mathbb{R}^d$ and a radius r, such that $\mathrm{d}(p, X) \leq r$ for every $p \in P$, minimizing r; in other words, we want X to minimize $\max_{p\in P} \mathrm{d}(p, X)$. We thus switch to the following definition of coresets:[3]

[3]The coresets in our definition has been called "additive" coresets in the literature, somewhat confusingly. "Multiplicative" coresets for k-centers satisfy a stronger condition; see [3, 34].

Definition 6.1 Given a k-point set X, define $g(P, X) = \max_{p \in P} \mathrm{d}(p, X)$. Call a subset $S \subseteq Q$ an *ε-coreset of Q* if $g(S, X) \geq (1 - \varepsilon)g(Q, X)$ for every k-point set $X \subset \mathbb{R}^d$.

A factor-$(1+\varepsilon)$ solution to the k-center problem is a k-point set $\hat{X}$ such that $g(p, \hat{X}) \leq (1 + \varepsilon)g(p, X)$ for every k-point set X. If S is an ε-coreset of P and $\hat{X}$ is a factor-$(1+\varepsilon')$ solution for S, then $\hat{X}$ is a factor-$(1+O(\varepsilon+\varepsilon'))$ solution for P, since $g(P, \hat{X}) \leq g(S, \hat{X})/(1-\varepsilon) \leq g(S, X)(1+\varepsilon')/(1-\varepsilon) \leq g(P, X)(1+\varepsilon')/(1-\varepsilon)$ for every k-point set X.

This definition of coresets satisfies properties similar to before: An ε-coreset of an ε'-coreset is an $(\varepsilon + \varepsilon')$-coreset. If S_i is an ε-coreset of Q, then $\bigcup_i S_i$ is an ε-coreset of $\bigcup_i Q_i$.

We use the following analog of Lemma 2.2, which is based on a well-known greedy approximation algorithm for the k-center problem, attributed to Gonzalez [32]:

Lemma 6.2 *Fix $k+1$ "anchor" points $p_0, \ldots, p_k$. Consider a "special" point set Q satisfying*

$$\mathrm{d}(p, \{p_0, \ldots, p_{j-1}\}) \;\leq\; c\,\mathrm{d}(p_j, \{p_0, \ldots, p_{j-1}\}) \qquad \forall j = 1, \ldots, k,\ \forall p \in Q \cup \{p_0, \ldots, p_k\}.$$

We can maintain an ε-coreset of $Q \cup \{p_0, \ldots, p_k\}$ of size $O(\varepsilon^{-d}k)$ in $O(1)$ time under insertions and deletions of points in Q; in particular, the preprocessing time is $O(|Q|)$.

Proof: Let $r = \mathrm{d}(p_k, \{p_0, \ldots, p_{k-1}\})$.

For any k-point set $X \subset \mathbb{R}^d$, we prove that $g(\{p_0, \ldots, p_k\}, X) \geq \frac{1}{2c}r$: Let $r^* = g(\{p_0, \ldots, p_k\}, X)$. The k balls of radius r^* centered at the points of X cover all points in $\{p_0, \ldots, p_k\}$. By the pigeonhole principle, one of the balls must contain at least two points p_i and p_j for some $i < j$. Then

$$r \;\leq\; \mathrm{d}(p_k, \{p_0, \ldots, p_{j-1}\}) \;\leq\; c\,\mathrm{d}(p_j, \{p_0, \ldots, p_{j-1}\}) \;\leq\; c\,\mathrm{d}(p_j, p_i) \;\leq\; 2cr^*.$$

The method to construct the coreset S is simple: build a uniform grid of side length εr and keep only one point per grid cell in S. Then for any k-point set X, $g(S, X)$ and $g(Q, X)$ differ by at most an additive error of $O(\varepsilon r) = O(\varepsilon)g(\{p_0, \ldots, p_k\}, X)$.

Since Q can be covered by k balls of radius cr, the maximum size of this coreset, i.e., the number of nonempty grid cells, is $O(k\varepsilon^{-d})$. Insertion/deletion is easily doable in $O(1)$ time, by looking up an appropriate grid cell. □

Statically, for each $j = 1, \ldots, k$, we can set p_j to be the point of Q with largest distance to $\{p_0, \ldots, p_{j-1}\}$, computable in linear time.

Dynamically, we face a similar difficulty: what if an anchor point gets deleted? In the worst case, this may happen often. (Indeed, in clustering applications, one is particularly interested in times when clusters undergo important changes.) Fortunately, we can use exactly the same idea as in Section 3. In the pseudocode of P.preprocess(), we only have to replace a few d's with k's and modify line 3 to:

3. $\quad A_j \leftarrow$ the $\alpha|P|$ points $p \in Q$ with the largest $\mathrm{d}(p, \{p_0, \ldots, p_{j-1}\})$

We choose parameters so that $k\alpha + \delta \ll 1$ and $\delta \ll \alpha/(k+1)$, for example, $\alpha = 1/(2k)$ and $\delta = 1/(4k^2)$. The probability that $p \in \{p_0, \ldots, p_d\}$ is now $O(k/n)$. The recurrence for the amortized expected update time becomes $U(n) \leq U((k\alpha + \delta)n) + O(k^{O(1)})$, which solves to $U(n) = O(k^{O(1)} \log n)$. The size of the overall coreset satisfies the recurrence $N(n) \leq N((k\alpha+\delta)n) + O(\varepsilon^{-d}k)$, which gives $N(n) = O(\varepsilon^{-d} k \log n)$. We can compute a factor-$(1+\varepsilon)$ solution for the coreset after every update. (We can readjust ε by a constant factor.) The additional time per update is $O(N(n)\log k + (k/\varepsilon)^{O(k^{1-1/d})}) = O(\varepsilon^{-d} k \log k \log n + (k/\varepsilon)^{O(k^{1-1/d})})$ by a static approximate k-center algorithm of Agarwal and Procopiuc [6].

The randomized approach in Section 5 is also applicable with no major changes. In fact, the integer-input assumption is not necessary, as long as the spread is bounded by U. Consequently, we get:

Theorem 6.3 *We can maintain a factor-$(1+\varepsilon)$ solution to the k-center problem for an n-point set in $\mathbb{R}^d$ in expected amortized update time $O(\varepsilon^{-d} k^{O(1)} \min\{\log n, \log\log U\} + (k/\varepsilon)^{O(k^{1-1/d})})$, where U is an upper bound on the spread.*

Remarks. The approach also works in the high-dimensional case where d may not be a constant, or more generally, the metric-space setting, if a larger approximation factor is tolerable. In Lemma 6.2, $\{p_0, \ldots, p_k\}$ itself is a constant-factor coreset of $Q \cup \{p_0, \ldots, p_k\}$ of size $O(k)$ (since $g(Q \cup \{p_0, \ldots, p_k\}, X) \leq g(\{p_0, \ldots, p_k\}, X) + cr \leq O(1) g(\{p_0, \ldots, p_k\}, X)$ from the proof of the lemma). The ε^{-d} factors disappear in the bound for $N(n)$. Instead of Agarwal and Procopiuc's algorithm, we can use the static factor-2 algorithm of Gonzalez [32], which runs in $O(N(n)k)$ time. The overall approximation factor is $O(1)$, and the number of operations (e.g., distance computations) is $O(k^{O(1)} \min\{\log n, \log\log U\})$ per update.

We leave open the question of whether our k-center result can be derandomized.

7 Constant Update Time?

In this last section, we consider the possibility of beating $O(\log\log U)$ in the integer input case. Specifically, we address the problem of approximating the smallest enclosing cylinder of a point set.

Let $\mathrm{Rad}\, P$ denote the radius of the smallest cylinder enclosing P. For a simple $O(1)$-factor, static approximation algorithm [1], one can do the following: pick an anchor point $s \in P$, find its farthest point $t \in P$, and return $w = \max_{p \in P} \mathrm{Rad}\,\{s, p, t\}$. Unfortunately, an insertion of a new point may require changing t and recomputing w from scratch. We are similarly in trouble if either anchor point s or t is deleted.

In a previous paper [20], the author proposes a simple streaming algorithm that takes care of insertions using a "doubling" trick. This will be the basis of our dynamic algorithm.

Lemma 7.1 *The following algorithm computes a factor-18 approximation w to $\mathrm{Rad}\, P$ for any sequence P of points:*

1. $w \leftarrow 0$, $s, t \leftarrow$ first two points of P
2. for each remaining point $p \in P$ do
3. $\quad w \leftarrow \max\{w, \mathrm{Rad}\,\{s, p, t\}\}$
4. $\quad$ if $\lfloor \log_2 \mathrm{d}(s,p) \rfloor > \lfloor \log_2 \mathrm{d}(s,t) \rfloor$ then $t = p$

Proof: The version of the algorithm from [20, Theorem 3.1] uses the condition $\mathrm{d}(s,p) > 2\mathrm{d}(s,t)$ instead in line 4, but the same analysis follows through. $\square$

The key observation behind our new algorithm is the following recasting of Lemma 7.1, in which we partition the point set into groups like in Section 5:

Corollary 7.2 *Fix a point $s \in P$. Let $P_i = \{p \in P : \lfloor \log_2 \mathrm{d}(s,p) \rfloor = i\}$ and pick a point $p_i \in P_i$. Let i^+ (resp. i^-) denote the successor (resp. predecessor) of i in the set $I = \{i : P_i \neq \emptyset\}$, i.e., the smallest element greater than (resp. the largest element less than) i in I. Then*

$$w = \max\left\{ \max_{i \in I,\, p \in P_i} \mathrm{Rad}\,\{s, p, p_i\},\ \max_{i \in I} \mathrm{Rad}\,\{s, p_i, p_{i^+}\} \right\}$$

is a factor-18 approximation to $\mathrm{Rad}\, P$.

Proof: Imagine running the algorithm in Lemma 7.1 where the points of P are ordered so that p_i is the first point of P_i, and P_i precedes P_{i^+} for each i. The returned value is given precisely by the expression above. $\square$

Once we have Corollary 7.2, the pseudocode to maintain the value of w under deletions and insertions is straightforward to write out. The terms in the maximum are stored in a set Q. When s is deleted, we rebuild the data structure from scratch; when p_i is deleted, we rebuild only information related to the group P_i. Again, we use randomization to select these anchor points s and p_i.

delete(p):
if $p = s$ then
    $P \leftarrow$ all current points except p
    $s \leftarrow$ a random point of P
    for each i do
        $P_i \leftarrow \{p \in P : \lfloor \log_2 \mathrm{d}(s,p) \rfloor = i\}$
        $p_i \leftarrow$ a random point of P_i
    $I \leftarrow \{i : P_i \neq \emptyset\}$
    $Q \leftarrow \bigcup_{i \in I} \{\mathrm{Rad}\,\{s,p,p_i\} : p \in P_i\} \ \cup \{\mathrm{Rad}\,\{s, p_i, p_{i^+}\}\}$
    return
$i \leftarrow \lfloor \log_2 \mathrm{d}(s,p) \rfloor$, $P_i \leftarrow P_i - \{p\}$
if $P_i = \emptyset$ then
    $Q \leftarrow Q - \{\mathrm{Rad}\,\{s, p_{i^-}, p_i\}, \mathrm{Rad}\,\{s, p_i, p_{i^+}\}\} \ \cup \{\mathrm{Rad}\,\{s, p_{i^-}, p_{i^+}\}\}$
    $I \leftarrow I - \{i\}$
else if $p = p_i$ then
    $Q \leftarrow Q - \{\mathrm{Rad}\,\{s,p,p_i\} : p \in P_i\} \ - \{\mathrm{Rad}\,\{s, p_{i^-}, p_i\}, \mathrm{Rad}\,\{s, p_i, p_{i^+}\}\}$
    $p_i \leftarrow$ a random point of P_i
    $Q \leftarrow Q \cup \{\mathrm{Rad}\,\{s,p,p_i\} : p \in P_i\} \ \cup \{\mathrm{Rad}\,\{s, p_{i^-}, p_i\}, \mathrm{Rad}\,\{s, p_i, p_{i^+}\}\}$
else $Q \leftarrow Q - \{\mathrm{Rad}\,\{s,p,p_i\}\}$

insert(p):
$i \leftarrow \lfloor \log_2 \mathrm{d}(s,p) \rfloor$, $P_i \leftarrow P_i \cup \{p\}$
if $|P_i| = 1$ then
    $p_i \leftarrow p$, $I \leftarrow I \cup \{i\}$
    $Q \leftarrow Q \cup \{\mathrm{Rad}\,\{s, p_{i^-}, p_i\}, \mathrm{Rad}\,\{s, p_i, p_{i^+}\}\} \ - \{\mathrm{Rad}\,\{s, p_{i^-}, p_{i^+}\}\}$
else $Q \leftarrow Q \cup \{\mathrm{Rad}\,\{s,p,p_i\}\}$

Because coordinates are integers in the range $[0, U]$, distances and Rad values can lie in the range $[\Omega(1/U), O(U)] \cup \{0\}$. The elements in I are thus integers bounded by $O(\log U)$. For a factor-$18(1+\varepsilon)$ approximation to Rad P, it suffices to maintain a factor-$(1+\varepsilon)$ approximation of $w = \max Q$, which can be obtained from $\max Q'$ where $Q' = \{\lfloor \log_{1+\varepsilon} z \rfloor : z \in Q\}$ is stored in a priority queue. The elements in Q' are integers bounded by $O((1/\varepsilon)\log U)$ (in absolute value). We can implement both the priority queue Q' and the ordered set I by the following data structure:

Lemma 7.3 *We can maintain a multiset of small integers bounded by $O(\log U)$ so that insertions, deletions, and successor/predecessor searches (in particular, finding maximum/minimum) take $O(1)$ time each on the word RAM with word size $\Omega(\log U)$.*

Proof: Maintain an array of $O(\log U)$ linked lists, where list $L[j]$ holds all elements of the same value j. Maintain a bit vector V, where $V[j]$ is 1 if $L[j]$ is nonempty. The vector V can be stored in $O(1)$ words. A search reduces to $O(1)$ standard word operations on V (bitwise-and/or and finding most/least significant 1-bit). $\square$

For the analysis of delete(p), let $\tilde{n}$ (resp. $\tilde{n}_i$) be the size of P (resp. P_i) during the last rebuild and let u (resp. u_i) be the number of updates to P (resp. P_i) since the last rebuild.

The rebuilding cost in lines 2–8 involves $O(|P|) = O(\tilde{n} + u)$ operations on Q and I. We can cover the cost by charging $O(1)$ units to each of the u updates since the last rebuild, plus $O(\max\{\tilde{n} - u, 1\})$ units to the current update if $p = s$. The probability of this occurring is at most $1/\max\{\tilde{n} - u, 1\}$, conditioned to a fixed time value for the last rebuild (as in Section 5).

The cost in lines 15–17 involves $O(|P_i|) = O(\tilde{n}_i + u_i)$ operations on Q and I. Like before, we can cover the cost by charging $O(1)$ units to every update, plus $O(\max\{n_i - u_i, 1\})$ units to the current update if $p = p_i$. The probability of this occurring is at most $1/\max\{n_i - u_i, 1\}$, conditioned to a fixed time value for the last rebuild of P_i.

Therefore, the expected amortized cost for insert() and delete() is $O(1)$.

Theorem 7.4 *We can maintain a factor-$O(1)$ approximation to the smallest enclosing cylinder for a point set in $\{0, \ldots, U\}^d$ in $O(1)$ expected amortized time per update, on the word RAM with word size $\Omega(\log U)$.*

Remarks. This method works well in high dimensions, with polynomial dependence on d, since Rad $\{s, p, t\}$ can be computed in $O(d)$ time for each triple $\{s, p, t\}$.

The factor $18(1+\varepsilon)$ could perhaps be reduced, but we do not know how to obtain factor $1+\varepsilon$, nor extend the method to approximate width in dimensions beyond 2.

Acknowledgements

I thank the anonymous referees for their valuable comments.

References

[1] P. K. Agarwal, B. Aronov, and M. Sharir. Line transversals of balls and smallest enclosing cylinders in three dimensions. *Discrete Comput. Geom.*, 21:373–388, 1999.

[2] P. K. Agarwal, S. Har-Peled, and K. R. Varadarajan. Approximating extent measures of points. *J. ACM*, 51:606–635, 2004.

[3] P. K. Agarwal, S. Har-Peled, and K. R. Varadarajan. Geometric approximation via coresets. In *Current Trends in Combinatorial and Computational Geometry* (E. Welzl, ed.), Cambridge University Press, New York, to appear.

[4] P. K. Agarwal, S. Har-Peled, and H. Yu. Robust shape fitting via peeling and grating coresets. *Discrete Comput. Geom.*, 29:38–58, 2008.

[5] P. K. Agarwal and J. Matoušek. Dynamic half-space range reporting and its applications. *Algorithmica*, 13:325–345, 1995.

[6] P. K. Agarwal and C. M. Procopiuc. Approximation algorithms for projective clustering. *J. Algorithms*, 46:115–139, 2003.

[7] P. K. Agarwal and M. Sharir. Off-line dynamic maintenance of the width of a planar point set. *Comput. Geom. Theory Appl.*, 1:65–78, 1991.

[8] P. K. Agarwal and M. Sharir. Efficient randomized algorithms for some geometric optimization problems. *Discrete Comput. Geom.*, 16:317–337, 1996.

[9] P. K. Agarwal and H. Yu. A space-optimal data-stream algorithm for coresets in the plane. In *Proc. 23rd ACM Sympos. Comput. Geom.*, pages 1–10, 2007.

[10] M. Bădoiu and K. L. Clarkson. Optimal core-sets for balls. Manuscript, 2003. Preliminary version in *Proc. 14th ACM-SIAM Sympos. Discrete Algorithms*, pages 801–802, 2003.

[11] M. Bădoiu, S. Har-Peled, and P. Indyk. Approximate clustering via core-sets. In *Proc. 34th ACM Sympos. Theory Comput.*, pages 250–257, 2002.

[12] G. Barequet and S. Har-Peled. Efficiently approximating the minimum-volume bounding box of a point set in three dimensions. *J. Algorithms*, 38:91–109, 2001.

[13] J. Bentley and J. Saxe. Decomposable searching problems I: static-to-dynamic transformation. *J. Algorithms*, 1:301–358, 1980.

[14] G. S. Brodal and R. Jacob. Dynamic planar convex hull. In *Proc. 43rd IEEE Sympos. Found. Comput. Sci.*, pages 617–626, 2002.

[15] T. M. Chan. Dynamic planar convex hull operations in near-logarithmic amortized time. *J. ACM*, 48:1–12, 2001.

[16] T. M. Chan. Approximating the diameter, width, smallest enclosing cylinder, and minimum-width annulus. *Int. J. Comput. Geom. Appl.*, 12:67–85, 2002.

[17] T. M. Chan. A fully dynamic algorithm for planar width. *Discrete Comput. Geom.*, 30:17–24, 2003.

[18] T. M. Chan. Semi-online maintenance of geometric optima and measures. *SIAM J. Comput.*, 32:700-716, 2003.

[19] T. M. Chan. A dynamic data structure for 3-d convex hulls and 2-d nearest neighbor queries. In *Proc. 17th ACM-SIAM Sympos. Discrete Algorithms*, pages 1196–1202, 2006.

[20] T. M. Chan. Faster core-set constructions and data stream algorithms in fixed dimensions. *Comput. Geom. Theory Appl.*, 35:20–35, 2006.

[21] T. M. Chan. Well-separated pair decomposition in linear time? *Inform. Process. Lett.*, 107:138–141, 2008.

[22] T. M. Chan and M. Pătraşcu. Transdichotomous results in computational geometry, I: Point location in sublogarithmic time. *SIAM J. Comput.*, to appear. Preliminary versions in *Proc. 47th IEEE Sympos. Found. Comput. Sci.*, pages 325–332 and 333–342, 2006.

[23] T. M. Chan and M. Pătraşcu. Transdichotomous results in computational geometry, II: Offline search. In preparation. Preliminary version appeared in *Proc. 39th ACM Sympos. Theory Comput.*, pages 31–39, 2007.

[24] T. M. Chan and B. S. Sadjad. Geometric optimization problems over sliding windows. *Int. J. Comput. Geom. Appl.*, 16:145–157, 2006.

[25] B. Chazelle. On the convex layers of a planar set. *IEEE Trans. Inform. Theory*, IT-31:509–517, 1985.

[26] K. L. Clarkson, D. Eppstein, G. L. Miller, C. Sturtivant, and S.-H. Teng. Approximating center points with iterative Radon points. *Int. J. Comput. Geom. Appl.*, 6:357–377, 1996.

[27] C. A. Duncan, M. T. Goodrich, and E. A. Ramos. Efficient approximation and optimization algorithms for computational metrology. In *Proc. 8th ACM-SIAM Sympos. Discrete Algorithms*, pages 121–130, 1997.

[28] M. Edwards and K. R. Varadarajan. No coreset, no cry: II. In *Proc. 25th Int. Conf. Found. Soft. Tech. Theoret. Comput. Sci.*, Lect. Notes Comput. Sci., vol. 3821, Springer-Verlag, pages 107–115, 2005.

[29] J. Feigenbaum, S. Kannan, and J. Zhang. Computing diameter in the streaming and sliding-window models. *Algorithmica*, 41:25–41, 2004.

[30] G. Frahling, P. Indyk, and C. Sohler. Sampling in dynamic data streams and applications. *Int. J. Comput. Geom. Appl.*, 18:3–28, 2008.

[31] G. Frahling and C. Sohler. Coresets in dynamic geometric data streams. In *Proc. 37th ACM Sympos. Theory Comput.*, pages 209–217, 2005.

[32] T. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoret. Comput. Sci.*, 38:293–306, 1985.

[33] S. Har-Peled. Clustering motion. *Discrete Comput. Geom.*, 31:545–565, 2004.

[34] S. Har-Peled. No coreset, no cry. In *Proc. 24th Int. Conf. Found. Soft. Tech. Theoret. Comput. Sci.*, Lect. Notes Comput. Sci., vol. 3328, Springer-Verlag, pages 324–335, 2004.

[35] S. Har-Peled and A. Kushal. Smaller coresets for k-median and k-means clustering. *Discrete Comput. Geom.*, 37:3–19, 2007.

[36] S. Har-Peled and S. Mazumdar. Coresets for k-means and k-median clustering and their applications. In *Proc. 36th ACM Sympos. Theory Comput.*, pages 291–300, 2004.

[37] S. Har-Peled and K. R. Varadarajan. Projective clustering in high dimensions using core-sets. In *Proc. 18th ACM Sympos. Comput. Geom.*, pages 312–318, 2002.

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

[39] P. Indyk. Algorithms for dynamic geometric problems over data streams. In *Proc. 36th ACM Sympos. Theory Comput.*, pages 373–380, 2004.

[40] R. Janardan. On maintaining the width and diameter of a planar point-set online. *Int. J. Comput. Geom. Appl.*, 3:331–344, 1993.

[41] Y. Matias, J. S. Vitter, and N. E. Young. Approximate data structures with applications. In *Proc. 5th ACM-SIAM Sympos. Discrete Algorithms*, pages 187–194, 1994.

[42] J. Matoušek. Derandomization in computational geometry. In *Handbook of Computational Geometry* (J. Urrutia and J. Sack, ed.), North-Holland, pages 559–595, 2000.

[43] M. H. Overmars. *The Design of Dynamic Data Structures.* Lect. Notes in Comput. Sci., vol. 156, Springer-Verlag, 1983.

[44] M. H. Overmars and J. van Leeuwen. Maintenance of configurations in the plane. *J. Comput. Sys. Sci.*, 23:166–204, 1981.

[45] J. Pach and P. K. Agarwal. *Combinatorial Geometry.* Wiley-Interscience, New York, 1995.

[46] G. Rote, C. Schwarz, and J. Snoeyink. Maintaining the approximate width of a set of points in the plane. In *Proc. 5th Canad. Conf. Comput. Geom.*, pages 258–263, 1993.

[47] M. Thorup. Equivalence between priority queues and sorting. *J. ACM*, 54:6, 2007.

[48] P. van Emde Boas. Preserving order in a forest in less than logarithmic time and linear space. *Inform. Process. Lett.*, 6:80–82, 1977.

[49] H. Yu, P. K. Agarwal, R. Poreddy, and K. R. Varadarajan. Practical methods for shape fitting and kinetic data structures using core sets. *Algorithmica*, 52:378–402, 2008.

[50] H. Zarrabi-Zadeh. An almost space-optimal streaming algorithm for coresets in fixed dimensions. In *Proc. 16th European Sympos. Algorithms*, Lect. Notes Comput. Sci., vol. 5193, Springer-Verlag, pages 817–829, 2008.