Dynamic Streaming Algorithms for ε-Kernels

Timothy M. Chan*

Cheriton School of Computer Science, University of Waterloo, Waterloo, Canada
tmchan@uwaterloo.ca

Abstract

Introduced by Agarwal, Har-Peled, and Varadarajan [J. ACM, 2004], an *ε-kernel* of a point set is a coreset that can be used to approximate the width, minimum enclosing cylinder, minimum bounding box, and solve various related geometric optimization problems. Such coresets form one of the most important tools in the design of linear-time approximation algorithms in computational geometry, as well as efficient insertion-only streaming algorithms and dynamic (non-streaming) data structures. In this paper, we continue the theme and explore dynamic streaming algorithms (in the so-called turnstile model).

Andoni and Nguyen [SODA 2012] described a dynamic streaming algorithm for maintaining a $(1+\varepsilon)$-approximation of the width using $O(\text{polylog}\, U)$ space and update time for a point set in $[U]^d$ for any constant dimension d and any constant $\varepsilon > 0$. Their sketch, based on a *polynomial method*, does not explicitly maintain an ε-kernel. We extend their method to maintain an ε-kernel, and at the same time reduce some of logarithmic factors. As an application, we obtain the first randomized dynamic streaming algorithm for the width problem (and related geometric optimization problems) that supports k *outliers*, using $\text{poly}(k, \log U)$ space and time.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases coresets, streaming algorithms, dynamic algorithms, polynomial method, randomization, outliers

Digital Object Identifier 10.4230/LIPIcs.SoCG.2016.27

1 Introduction

Given a set S of n points in a constant dimension d, define the *width* (or *extent*) of S along a given vector ξ to be $w(S, \xi) = \max_{s \in S} s^T \xi - \min_{s \in S} s^T \xi$. The *width* of S is the minimum width over all unit vectors ξ.

The seminal paper by Agarwal, Har-Peled, and Varadarajan [1] introduced the concept of *ε-kernels*, which are *coresets* for the width along all directions simultaneously: an ε-kernel of S is a subset $Q \subset S$ such that $w(Q, \xi) \geq (1-\varepsilon) w(S, \xi)$ for all vectors ξ. Agarwal et al. showed that an ε-kernel of constant $O((1/\varepsilon)^{(d-1)/2})$ size always exists and can be constructed efficiently in $O(n)$ time for any constant $\varepsilon > 0$ (the hidden dependency of the running time on ε was subsequently improved by the author [9]). Using ε-kernels, we can immediately obtain efficient $(1+\varepsilon)$-factor approximation algorithms not only for computing the width of a point set, but for numerous other geometric optimization problems, such as convex hull volume, minimum enclosing cylinder, minimum-volume bounding box, minimum-volume enclosing ellipsoid, minimum-width enclosing annulus, and minimum-width cylindrical shell, as well as variants of all these problems for moving points; see [1, 2]. For this reason, ε-kernels

* This work is supported by an NSERC Discovery Grant. This work was started during the author's visit to the Hong Kong University of Science and Technology.

have been recognized as one of the most powerful techniques in geometric approximation algorithms.

Using ε-kernels, Agarwal et al. [1] also obtained one-pass *streaming algorithms* for all these problems with $O(\text{polylog}\, n)$ space and $O(\text{polylog}\, n)$ time per insertion of a new point. This was later improved to $O(1)$ space and time for any constant $\varepsilon > 0$ by the author [9] in the real-RAM model (the hidden ε-dependencies were further improved by Zarrabi-Zadeh [16] and by Arya and the author [6]).

Agarwal et al. [1] also used ε-kernels to design *dynamic data structures* for all these problems to support both insertions and deletions in $O(\text{polylog}\, n)$ time, using $O(n)$ space. The update time was later improved to $O(\log n)$ by the author [10] (the ε-dependencies were further improved by Agarwal, Phillips, and Yu [4]).

These results naturally led to the question of whether these fully dynamic data structures can be made to use small space, as in those insertion-only streaming algorithms. The typical model for *dynamic streaming* algorithms in geometry is the *turnstile* model [15]: points are assumed to have integer coordinates in $[U] = \{0, \ldots, U-1\}$ (with $n \leq U^{O(1)}$); the update sequence is given as a stream; during both insertion and deletion of a point s, we are given its d coordinate values. We allow insertions of multiple copies of a point, but a point cannot be deleted more times than it is inserted (otherwise, the algorithm is not guaranteed to produce meaningful output). Several geometric results in this model were known, for example, on k-medians and Euclidean minimum spanning tree weight [12, 11].

However, finding a dynamic streaming algorithm for approximating the width of a point set, even in the two-dimensional case, turned out to be a challenge, and was posed by Agarwal and Indyk in an open problem list.[1] The difficulty is that unlike other problems such as k-medians whose objective function involves *sums*, the width function along a fixed direction involves *maximums/minimums*; for example, it is unclear if sampling techniques help. Furthermore, standard techniques for insertion-only streaming such as "merge-and-reduce" fail in the dynamic setting.

Andoni and Nguyen's method. This open problem was eventually resolved by Andoni and Nguyen [5] in SODA'12, who presented a method with $O(\text{polylog}\, U)$ space and update time for width of a point set in any constant dimension d with approximation factor $1+\varepsilon$ for any constant $\varepsilon > 0$.

The basic idea is remarkably simple: approximate the maximum with a sum of p-th powers, for a small integer $p = O(\log n)$. In order to apply this idea to the width function for all directions simultaneously, we treat the sum as a polynomial function; the sketch then consists of the coefficients of a degree-$O(p)$ polynomial in d variables. However, the number of logarithmic factors is large: the space/time bound is $O(\log^{Cd} U)$ for some unspecified absolute constant C. This constant C comes from the use of a "hammer"—namely, a solver for a system of polynomial inequalities over multiple real variables [8]—needed in the last step to minimize the width function over all possible directions. (Andoni and Nguyen showed how to avoid the polynomial-inequality solver using randomization, but only in the two-dimensional case.)

Though not claimed by Andoni and Nguyen, their dynamic streaming method can likely solve many of the other applications of ε-kernels, such as minimum-volume bounding box. This is because the polynomial maintained by Andoni and Nguyen encodes an approximation of the width along all directions simultaneously, and thus "acts" like an ε-kernel. The

[1] `http://sublinear.info/23`

invocation of the polynomial-inequality solver gets more complex for other problems such as minimum-volume bounding box, but in theory should still work.

Our method. The question of maintaining explicitly an ε-kernel in the dynamic streaming setting still remains. In some applications, it is desirable to find not only the value approximating the optimum, but an actual subset of the point set that realizes the approximate optimum (i.e., a coreset). With deterministic algorithms, this is impossible to do with sublinear space, since an adversary could delete the elements of the coreset and repeat to retrieve the entire point set.

Our main result is a *randomized* (Monte Carlo) dynamic streaming algorithm to maintain an ε-kernel with $O(\text{polylog}\, U)$ space and update time for any constant $\varepsilon > 0$. Thus, we can now obtain coresets for the width, the minimum-volume bounding box, and all the other applications. The key is an oracle to find not just an approximation of the width along a given direction, but approximate extreme points that realize this width. We follow Andoni and Nguyen's idea of approximating the maximum by a sum of p-th powers, but extend it to find a *witness* for an approximate maximum: in Section 2.1, we describe an interesting, simple randomized method to isolate such a witness via sampling. The parameter p is increased by a second logarithmic factor, and $O(\log n)$ polynomials are maintained, but at the end we get fewer logarithmic factors: the space and update bound is about $O(\log^{2d+3} U)$ (see Theorems 4 and 6). The reason for the improvement is that with explicit maintenance of an ε-kernel, we no longer need the polynomial-inequality solver. The avoidance of the "hammer" is also an advantage from the practical perspective.

More broadly speaking, although the "polynomial method" has seen wide-ranging applications in theoretical computer science and has been used in streaming algorithms even before Andoni and Nguyen's paper,[2] its occurrence in computational geometry is rarer (not counting a different "polynomial method" or the use of algebraic geometry in combinatorial geometry, advocated by Sharir and others in recent years). It is therefore worthy of more exposure in the SoCG community, in the author's opinion.

An alternative method. If an ε-kernel need not be reported after every update time, it is desirable to distinguish between update time (time to do an insertion/deletion) and query time (time to report an ε-kernel). For example, although Andoni and Nguyen's method has $O(\log^{Cd} U)$ query time, it has a better update time of about $O(\log^{d+1} U)$. We give an alternative method with an even better update time of about $O(\log^3 U)$, and with a query time of about $O(\log^{3d+4} U)$ (see Theorems 5 and 6).

In this method, we do not even need to store the polynomials explicitly. Instead, the sketch consists of just a few counters, namely, the number of data points (in various samples) that lie at each possible coordinate values modulo q, for each q in a family of small primes. As it turns out, such counters contain as much information as an ε-kernel! (This alternative method is an instance of a *linear sketch* [13].) The Chinese remainder theorem is invoked during the query algorithm.

Although the use of the Chinese remainder theorem is hardly original in the context of streaming algorithms, its occurrence in computational geometry is rarer, and noteworthy.

Outliers. One major application where explicit maintenance of an ε-kernel appears critical is in solving variants of geometric optimization problems that tolerate a certain number

[2] For example, just see Puzzle 1 in Muthukrishnan's survey [15].

k of *outliers*. For example, instead of the width of S, we may want to find the smallest width of $S \setminus T$ over all subsets T of size k. Such variants are well-motivated, since objective functions involving maximums/minimums rather than sums are particularly sensitive to outliers. The new problems can be solved by an extension of ε-kernels [3]: let $w_{a,b}(S, \xi) =$ (a-th largest of $\{s^T\xi\}_{s\in S}$)$-$(b-th smallest of $\{s^T\xi\}_{s\in S}$); a (k, ε)-*kernel* of S is a subset $Q \subset S$ such that $w_{a,b}(Q, \xi) \geq (1-\varepsilon)w_{a,b}(S, \xi)$ for every vector ξ and every $a, b \leq k$.

Agarwal, Har-Peled, and Yu [3] showed that a (k, ε)-kernel of $O((1/\varepsilon)^{(d-1)/2}k)$ size exists and can be constructed in $O(n + k^2)$ time for any constant $\varepsilon > 0$; the running time was improved by the author [10] to $O(n + k \log n)$ (ignoring ε-dependencies). Their algorithm is simple and is by *repeated peeling*: compute a standard ε-kernel, delete its points, and repeat $O(k)$ times.

Andoni and Nguyen's dynamic streaming algorithm cannot be used to solve the problem with k outliers. The key oracle concerns not just the maximum but the *top* k values, but the power-sum approach does not seem immediately applicable, at least without witness finding. With witness finding and repeated deletions though, the task becomes straightforward: since our method explicitly maintains an ε-kernel and supports deletions, Agarwal, Har-Peled, and Yu's peeling algorithm can be directly implemented, and we obtain the first dynamic streaming method for (k, ε)-kernels, with $O(k \operatorname{polylog} U)$ space and update time and $O(k^2 \operatorname{polylog} U)$ query time for any constant $\varepsilon > 0$ (see Corollary 7).

2 An Oracle for Approximate Extreme Points

The key to constructing ε-kernels is an oracle for answering approximate extreme-point queries:

▶ **Problem 1.** *Let $f(s_1, \ldots, s_d, x_1, \ldots, x_{d+1}) = s_1x_1 + \cdots + s_dx_d + x_{d+1}$. Maintain a small-space sketch for a set $S \subset [U]^d$ that supports insertions and deletions in S and can answer queries of the following kind: for any query vector $x \in (\pm[U])^{d+1}$, find an element $\widetilde{m} \in S$ with $|f(\widetilde{m}, x)| \geq (1-\varepsilon) \max_{s\in S} |f(s, x)|$.*

Andoni and Nguyen [5] suggested a simple way to approximate $\max_{s\in S} |f(s, x)|$, namely, by $\left(\sum_{s\in S} f(s, x)^p\right)^{1/p}$ for an even integer p. The approximation factor is at least $(\frac{1}{n})^{1/p}$, which can be made at least $1-\varepsilon$ by setting $p = 2\lceil \log_{1/(1-\varepsilon)} n \rceil = O((1/\varepsilon) \log n)$. The main observation is that $\sum_{s\in S} f(s, x)^p$ is a low-degree polynomial in x; we can maintain the coefficients of this polynomial easily in the dynamic streaming model. However, this solution gives only the value of an approximate maximum, not a "witness" $\widetilde{m} \in S$ that attains such an approximate maximum value.

2.1 Witness for Approximate Maximum

We start with a more basic version of the problem in one dimension:

▶ **Problem 2.** *Maintain a small-space sketch for a set $S \subset [U]$ of at most n elements, where each element s has a nonnegative value y_s, that supports insertions and deletions in S and can answer queries of the following kind: find an element $\widetilde{m} \in S$ with $y_{\widetilde{m}} \geq (1-\varepsilon) \max_{s\in S} y_s$.*

In what follows, we let $m \in S$ denote the element with the true maximum value $y_m = \max_{s\in S} y_s$, which we may assume is nonzero.

There are different ways to solve Problem 2, but not all methods can be generalized to solve Problem 1 later when we replace the y_s's with functions in x (for example, one solution

involves bucketing with exponential spacing, but it is unclear how one would assign functions to buckets). We will propose a solution based purely on maintaining power sums, akin to Andoni and Nguyen's, that can be generalized.

We warm up with a special case of Problem 2 where the maximum is known to have a much larger value than all other elements. Formally, we say that a set is α-*separated* if the ratio of the second largest to the largest in the set is at most α. Consider the following simple algorithm:

Algorithm 1: (preliminary version)
1. let $A = \sum_{s\in S} y_s$ and $C = \sum_{s\in S} s y_s$
2. if $\{y_s\}_{s\in S}$ is α-separated then return $\lfloor C/A \rceil$

Here, $\lfloor x \rceil$ denotes $\lfloor x + 1/2 \rfloor$ (the nearest integer to x). Both sums A and C are easy to maintain dynamically. Intuitively, as α approaches 0, either sum will be dominated by its maximum term and the algorithm would solve the problem *exactly*. An issue remains though: the α-separation condition is not easy to certify dynamically. We will replace it with a condition involving one more sum:

Algorithm 2: (second version)
1. let $A = \sum_{s\in S} y_s$, $B = \sum_{s\in S} y_s^2$, and $C = \sum_{s\in S} s y_s$
2. if $A^2 < (1 + 3n\alpha)B$ then return $\lfloor C/A \rceil$

Intuitively, as α approaches 0, the condition that the largest value is much larger than all other values roughly aligns with the condition that the square of the sum is close to the sum of the squares—this little trick is the key. We formally verify correctness for a concrete choice of α below:

▶ **Lemma 1.** *Let* $\alpha = \frac{1}{3n^2U}$.
(a) *If Algorithm 2 returns a number, then the returned number is exactly* m.
(b) *If* $\{y_s\}_{s\in S}$ *is* α*-separated, then Algorithm 2 returns a number.*

Proof. The proof involves just straightforward algebra. Let $A' = \sum_{s\in S\setminus\{m\}} y_s$. Then

$$\left(\sum_{s\in S} y_s\right)^2 \geq \sum_{s\in S} y_s^2 + 2\sum_{s\in S\setminus\{m\}} y_m y_s \qquad \Longrightarrow \qquad A^2 \geq B + 2y_m A'.$$

If $A^2 < (1 + 3n\alpha)B$, we then have

$$A' < \frac{3n\alpha B}{2y_m} \leq \frac{3n\alpha \cdot n y_m^2}{2y_m} = \frac{1}{2U} y_m$$

and thus

$$\frac{C}{A} \leq \frac{m y_m + U A'}{y_m} < m + \frac{1}{2}$$
$$\frac{C}{A} \geq \frac{m y_m}{y_m + A'} > \frac{m}{1 + \frac{1}{2U}} \geq m\left(1 - \frac{1}{2U}\right) \geq m - \frac{1}{2}.$$

This proves (a). For (b), just note that α-separation implies

$$A^2 \leq (y_m + n\alpha y_m)^2 < (1 + 3n\alpha) y_m^2 \leq (1 + 3n\alpha)B.$$

◀

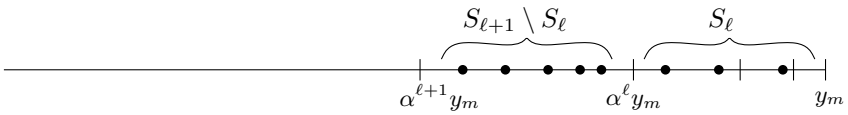

Figure 1 If exactly one element of S_ℓ is chosen and none of $S_{\ell+1} \setminus S_\ell$ is chosen, then the subset is α-separated.

In general, an arbitrary input would not satisfy the α-separation property. The next lemma shows that a *random sample* of a certain size can guarantee separation and at the same time contains an approximate maximum with good probability. The idea of sampling to isolate a unique solution is of course standard [14], but to adapt the idea in the approximate setting, we combine sampling with exponentially spaced bucketing in a nontrivial way—this is arguably the most interesting part of the entire section:

▶ **Lemma 2.** *Let R_j be a random sample of S where each element is selected independently with probability $1/2^j$. Fix any $\alpha > 0$. With probability $\Omega(1)$, for some $j \le \lceil \log n \rceil + 3$ the set $\{y_s\}_{s \in R_j}$ is α-separated and has maximum at least $\alpha^{\lceil \log n \rceil} y_m$.*

Proof. Let $S_\ell = \{s \in S : y_s \ge \alpha^\ell y_m\}$. Pick an index $\ell \le \lceil \log n \rceil$ with $|S_{\ell+1}| \le 2|S_\ell|$. Such an index exists: if not, $|S_{\lceil \log n \rceil}| > 2^{\lceil \log n \rceil} |S_0| \ge n$, which would be a contradiction. Set $j = \lceil \log |S_\ell| \rceil + 3$, so that $2^{j-3} \le |S_\ell| < 2^{j-2}$.

Let E be the event that exactly one element of S_ℓ is chosen in R_j and none of the elements of $S_{\ell+1} \setminus S_\ell$ is chosen in R_j (see Figure 1). If E is true, then $\{y_s\}_{s \in R_j}$ is α-separated, and furthermore has maximum at least $\alpha^\ell y_m$.

Let E_s denote the event that s is chosen in R_j. We can write $E = \bigcup_{s \in S_\ell} (E_s \cap \overline{\bigcup_{s' \in S_{\ell+1} - \{s\}} E_{s'}})$, which holds with probability at least

$$|S_\ell| \cdot \frac{1}{2^j} \cdot \left(1 - \frac{|S_{\ell+1}| - 1}{2^j}\right) > \frac{|S_\ell|}{2^j} \cdot \left(1 - \frac{2|S_\ell|}{2^j}\right) > \frac{1}{16}. \qquad \blacktriangleleft$$

Note that we use exponentially spaced bucketing only in the proof above, not in the algorithm itself.

There is still one complaint: the approximation factor $\alpha^{\lceil \log n \rceil}$ in Lemma 2 appears very weak, especially as the parameter α in Lemma 1 is very small. This can be fixed with one last trick: we can refine the approximation factor by simply replacing each y_s with its p-th power y_s^p for a sufficiently large even integer p. An approximation of $\max_{s \in S} y_s^p$ to within factor $\alpha^{\lceil \log n \rceil}$ yields an approximation of $\max_{s \in S} y_s$ to within factor $\alpha^{\lceil \log n \rceil / p}$, which can be made at least $1 - \varepsilon$ by setting $p = 2 \lceil \log n \rceil \lceil \log_{1/(1-\varepsilon)}(1/\alpha) \rceil$.

Putting everything together, we obtain our final algorithm to solve Problem 2:

Algorithm 3: (final version)

1. for $j = 0$ to $\lceil \log n \rceil + 3$ do
2. let R_j be a random sample of S where each element is selected independently with probability $1/2^j$
3. let $A_j = \sum_{s \in R_j} y_s^p$, $B_j = \sum_{s \in R_j} y_s^{2p}$, and $C_j = \sum_{s \in R_j} s y_s^p$
4. if $A_j^2 < (1 + 3n\alpha) B_j$ and ($\widetilde{m}$ is undefined or $y_{\lfloor C_j / A_j \rfloor} > y_{\widetilde{m}}$) then $\widetilde{m} = \lfloor C_j / A_j \rceil$

▶ **Theorem 3.** *Let $\alpha = \frac{1}{3n^2 U}$ and $p = 2 \lceil \log n \rceil \lceil \log_{1/(1-\varepsilon)}(1/\alpha) \rceil = O((1/\varepsilon) \log^2 U)$. With probability $\Omega(1)$, Algorithm 3 finds an element $\widetilde{m}$ with $y_{\widetilde{m}} \ge (1 - \varepsilon) y_m$. Furthermore, we always have $\widetilde{m} \in S$ whenever $\widetilde{m}$ is not undefined.*

Proof. The theorem follows immediately by combining Lemmas 1 and 2. $\blacktriangleleft$

2.2 First Solution via Polynomials

We are now ready to design a dynamic streaming algorithm to implement the oracle in Problem 1:

▶ **Theorem 4.** *For any constant d, we can solve Problem 1 with*
- $O((1/\varepsilon)^{d+1+o(1)}(\log U)^{2d+3+o(1)}\log(1/\delta))$ *words of space and update/query time,*

with error probability at most δ.

Proof. Our sketch consists of the coefficients of the following polynomials for each $j = 0, \ldots, \lceil \log n \rceil + 3$:

$$A_j(x) = \sum_{s \in R_j} f(s,x)^p, \quad B_j(x) = \sum_{s \in R_j} f(s,x)^{2p}, \quad \text{and} \quad C_j(x) = \sum_{s \in R_j} \lambda(s) f(s,x)^p,$$

where λ is a bijection from $[U]^d$ to $[U^d]$, e.g., $\lambda(s_1, \ldots, s_d) = s_1 U^{d-1} + s_2 U^{d-2} + \cdots + s_d$. Each such polynomial has degree $O(p)$ and consists of $O(p^d)$ monomials (since there are $O(p)$ choices for the degree of each of the $d+1$ variables, and the sum of these degrees is fixed). Each of the $O(p^d)$ coefficients of each of these $O(\log n)$ polynomials requires $O(p \log U)$ bits; the total space usage in words is $O(p^{d+1} \log n)$, where $p = O((1/\varepsilon)\log^2 U)$ (which remains true even after replacing U with U^d).

Insertion/deletion of an element s requires adding/subtracting a term to/from these polynomials. Each of the $O(p^d)$ coefficients of each of the $O(\log n)$ polynomials can be updated using $O(1)$ arithmetic operations on $O(p \log U)$-bit numbers; the total time is $O(p^{d+1+o(1)} \log n)$ in the $(\log U)$-bit word RAM model.

A query for a given vector x can be answered by evaluating these polynomials at x, via Algorithm 3 and Theorem 3. Each of the $O(\log n)$ evaluations requires $O(p^d)$ arithmetic operations on $O(p \log U)$-bit numbers; the total time is $O(p^{d+1+o(1)} \log n)$.

One technical issue concerns how the samples R_j are generated and stored. We can pick a random hash function $h : [N] \to [N]$ from a pairwise independent family[3] for some $N \in [U^d, 2U^d]$, and set $R_j = \{s \in S : h(\lambda(s)) \le \lfloor N/2^j \rfloor\}$. The proof of Lemma 2 only requires pairwise independence of the events E_s. Such a hash function h can be encoded in $O(\log U)$ bits and evaluated in constant time.

To lower the error probability from constant to δ, we can use $O(\log(1/\delta))$ independent versions of the data structure, increasing all bounds by an $O(\log(1/\delta))$ factor. ◀

2.3 Alternative Solution via Chinese Remainders

We also propose an alternative dynamic streaming algorithm for Problem 1 which does not require explicitly storing the polynomials $A_j(x), B_j(x), C_j(x)$ but instead uses the Chinese remainder theorem. The new version has better update time (which is desirable when there are more updates than queries).

▶ **Theorem 5.** *For any constant d, we can solve Problem 1 with*
- $O((1/\varepsilon)^{d+1}(\log U)^{3d+4}\log(1/\delta)/\log\log U)$ *words of space and query time, and*
- $O((1/\varepsilon)(\log U)^3 \log(1/\delta)/\log\log U)$ *expected update time,*

with error probability at most δ.

[3] For example, $h(i) = (r_1 + r_2 i) \bmod N$ for two random numbers $r_1, r_2 \in [N]$, assuming that N is prime.

Proof. Let $W = \max\{n(dU^2+U)^{2p}, nU^d(dU^2+U)^p\}$. Find a set Π of $O(\log W/\log\log W)$ primes, each at most $O(\log W)$, whose product exceeds $2W$. Our sketch simply consists of a counter

$$n_{j,q,(\sigma_1,\ldots,\sigma_d)} \;=\; |\{(s_1,\ldots,s_{d_0}) \in R_j : s_1 \equiv \sigma_1,\ \ldots,\ s_d \equiv \sigma_d \pmod q\}|$$

for each $j = 0,\ldots,\lceil\log n\rceil+3$, each $q \in \Pi$, and each $(\sigma_1,\ldots,\sigma_d) \in [q]^d$; the total space usage in words is $O(\log n \cdot (\log W/\log\log W)\cdot \log^d W)$, where $\log W = O(p\log U) = O((1/\varepsilon)\log^3 U)$.

Insertion/deletion of a point s affects an expected $O(1)$ number of samples R_j, each requiring increments/decrements of $O(\log W/\log\log W)$ counters of $O(\log n)$ bits and $O(\log W/\log\log W)$ divisions on $O(\log U)$-bit numbers; the total expected time is $O(\log W/\log\log W)$ in the $(\log U)$-bit word RAM model.

A query for a given point x can be answered by computing $A_j(x), B_j(x), C_j(x)$ as in the proof of Theorem 4, for each $j = 0,\ldots,\lceil\log n\rceil+3$. To this end, we compute, for each $q \in \Pi$,

$$A_j(x) \equiv \sum_{\sigma\in[q]^d} n_{j,q,\sigma} f(\sigma,x)^p \pmod q.$$

From these values, we can then reconstruct $A_j(x)$ by the Chinese remainder theorem, since $|A_j(x)| < W$. Similarly, we can compute $B_j(x)$ and $C_j(x)$. For each of the $O(\log n)$ indices j and each of the $O(\log W/\log\log W)$ primes q, the computation requires $O(\log^d W)$ evaluations of f and $O(\log^d W)$ arithmetic operations on $O(\log U)$-bit numbers; the total time is $O(\log n \cdot (\log W/\log\log W)\cdot \log^d W)$.

The technical issue of storing the samples R_j can be addressed as before. The error probability can be lowered to δ by increasing the bounds by an $O(\log(1/\delta))$ factor as before. ◀

3 Applications

3.1 ε-Kernels

We now apply the oracle from the previous section to compute an ε-kernel for a point set $S \subset [U]^d$ in a constant dimension d. This part is where geometry finally comes into play, although as it turns out, not much originality is required here. We adopt the following variant of one of Agarwal, Har-Peled, and Varadarajan's ε-kernel algorithms [1]:

0. $s_0 =$ any point in S
1. for $i = 1$ to d do
2. $\quad s_i =$ a point $s \in S$ that approximately maximizes the distance to the $(i-1)$-flat through $s_0,\ldots,s_{i-1}$, with constant approximation factor $1/c$
3. $\phi =$ the affine transformation that maps $s_0, s_1, \ldots, s_d$ to $e_0 = (0,\ldots,0), e_1 = (1,0,\ldots,0), \ldots, e_d = (0,\ldots,0,1)$
4. for each grid point ξ of a grid over $\partial[-1,1]^d$ of side length ε do
5. $\quad s_\xi =$ a point $s \in S$ that approximately maximizes $\phi(s)\cdot\xi$ with additive error $O(\varepsilon)$
6. return an ε-kernel of $O((1/\varepsilon)^{(d-1)/2})$ size for all these s_ξ's

Correctness. Because the algorithm is not original, we will only outline the correctness proof and point to references for the details. The algorithm consists of two phases. In the first phase (lines 0–3), we compute a transformation ϕ that makes $\phi(S)$ *fat*, by a simple iterative procedure with d steps (based on an idea of Barequet and Har-Peled [7]). Specifically, $\phi(S)$ satisfies the following properties:

- $\phi(S) \subset [-c, c]^d$. A proof can be found, for example, in [10, Lemma 2.2].
- $w(\phi(S), \xi) \geq 1/\sqrt{d}$ for all unit vectors ξ. This follows since $\phi(S)$ contains $\{e_0, \ldots, e_d\}$ by line 3.

In the second phase (lines 4–6), we compute an $O(\varepsilon)$-kernel of this fat point set $\phi(S)$, simply by finding extreme points along $O((1/\varepsilon)^{d-1})$ roughly equally spaced directions. A correctness proof can be found, for example, in [9, Theorem 2.5].

As noted in previous papers [1], an $O(\varepsilon)$-kernel of $\phi(S)$ yields an $O(\varepsilon)$-kernel of S for any invertible affine transformation ϕ, and an ε-kernel of an $O(\varepsilon)$-kernel is an $O(\varepsilon)$-kernel, and so the final output is an $O(\varepsilon)$-kernel of S.

Analysis. A naive implementation of lines 2 and 5 would require linear time. To achieve sublinear query time, we observe how both lines can be implemented using the extreme-point oracle for Problem 1.

For line 2, let A_{i-1} be the $d \times (i-1)$ matrix with column vectors $s_1 - s_0, \ldots, s_{i-1} - s_0$. Define the projection matrix $P_{i-1} = A_{i-1}(A_{i-1}^T A_{i-1})^{-1} A_{i-1}^T$ and let $P'_{i-1} = I - P_{i-1}$. The subproblem is equivalent to finding an $s \in S$ that approximately maximizes the expression $\|P'_{i-1}(s - s_0)\|$. It suffices to approximately maximize the expression $|P'_{i-1}(s - s_0) \cdot e_j|$ for each $j = 1, \ldots, d$ and take the maximum of the results; the approximation worsens by a $\sqrt{d}$ factor. Since

$$P'_{i-1}(s - s_0) \cdot e_j = s^T (P'_{i-1})^T e_j - s_0^T (P'_{i-1})^T e_j,$$

we can consult the oracle for the query vector $x = ((P'_{i-1})^T e_j, -s_0^T (P'_{i-1})^T e_j) \in \mathbb{R}^{d+1}$ (using a constant ε).

One issue is that this choice of x involves rational rather than integer coordinates, but we can convert the coordinates to integers by multiplying by the lowest common denominator. It can be checked that all integers involved are bounded by $U^{O(d)}$ (for example, by using the standard formula for matrix inverse in terms of determinants and co-factors); replacing U with $U^{O(d)}$ will not affect the space/time bounds.

For line 5, first note that $\phi(s) = A_d^{-1}(s - s_0)$. Since fatness implies that $|\phi(s) \cdot \xi| \leq dc$, it suffices to maximize $\phi(s) \cdot \xi + dc$ (which is nonnegative) with approximation factor $1 - \varepsilon$. Since

$$\phi(s) \cdot \xi + dc = s^T (A_d^{-1})^T \xi - s_0^T (A_d^{-1})^T \xi + dc,$$

we can consult the oracle for the query vector $x = ((A_d^{-1})^T \xi, -s_0^T (A_d^{-1})^T \xi + dc) \in \mathbb{R}^{d+1}$. Again, we can make the coordinates of x integers, bounded by $U^{O(d)}$.

Line 0 is easy (we can just query the oracle with an arbitrary x). Line 6 can be done by invoking a standard ε-kernel algorithm [1, 9] on $O((1/\varepsilon)^{d-1})$ points.

The total query time is thus dominated by the $O((1/\varepsilon)^{d-1})$ oracle calls in line 5. The error probability δ needs to be adjusted by an $O((1/\varepsilon)^{d-1})$ factor. Note that in Theorem 4 or 5, the query vectors should be independent of the random choices made by the data structure. This can be ensured by creating $d+1$ independent versions of the data structure, and using the i-th version for the i-th iteration in line 2, and the $(d+1)$-th version for line 5.

▶ **Theorem 6.** *For any constant dimension d, we can maintain an ε-kernel with*

- $O((1/\varepsilon)^{d+1+o(1)} (\log U)^{2d+3+o(1)} \log(1/\delta\varepsilon))$ *words of space and update time, and*
- $O((1/\varepsilon)^{2d+1+o(1)} (\log U)^{2d+3+o(1)} \log(1/\delta\varepsilon))$ *query time,*

or alternatively,

- $O((1/\varepsilon)^{d+1}(\log U)^{3d+4}\log(1/\delta\varepsilon)/\log\log U)$ *words of space,*
- $O((1/\varepsilon)(\log U)^3\log(1/\delta\varepsilon)/\log\log U)$ *expected update time, and*
- $O((1/\varepsilon)^{2d}(\log U)^{3d+4}\log(1/\delta\varepsilon)/\log\log U)$ *query time,*

with error probability at most δ.

3.2 ... with Outliers

For the variant of ε-kernels with k outliers, we invoke the peeling algorithm by Agarwal, Har-Peled, and Yu [3]: we compute an ε-kernel of $O((1/\varepsilon)^{(d-1)/2})$ size, delete its points, and repeat for $2k+2$ iterations; we output all the $O((1/\varepsilon)^{(d-1)/2}k)$ points deleted and reinsert them back.

Note that in Theorem 4 or 5, the update sequence for the data structure should be independent of the random choices made by the data structure. This can be ensured by creating $2k+2$ independent versions of the data structure, and using the i-th version for the i-th iteration. The space and update time bounds are increased by an $O(k)$ factor. The query time is $O(k)$ times the query time bound of our ε-kernel data structure, plus $O((1/\varepsilon)^{(d-1)/2}k)$ times the update time bound. The error probability needs to be adjusted by another $O(k)$ factor.

▶ **Corollary 7.** *For any constant dimension d, we can maintain a (k,ε)-kernel with*

- $O((1/\varepsilon)^{d+1+o(1)}k(\log U)^{2d+3+o(1)}\log(k/\delta\varepsilon))$ *words of space and update time, and*
- $O((1/\varepsilon)^{2d+o(1)}k(\log U)^{2d+3+o(1)}\log(k/\delta\varepsilon)+(1/\varepsilon)^{(3d+1)/2}k^2(\log U)^{2d+3}\log(k/\delta\varepsilon))$ *query time,*

or alternatively,

- $O((1/\varepsilon)^{d+1}k(\log U)^{3d+4}\log(k/\delta\varepsilon)/\log\log U)$ *words of space,*
- $O((1/\varepsilon)k(\log U)^3\log(k/\delta\varepsilon)/\log\log U)$ *expected update time, and*
- $O((1/\varepsilon)^{2d}k(\log U)^{3d+4}\log(k/\delta\varepsilon)/\log\log U+(1/\varepsilon)^{(d+1)/2}k^2(\log U)^3\log(k/\delta\varepsilon)/\log\log U)$ *query time,*

with error probability at most δ.

Acknowledgement. I thank Ke Yi for discussions that led to this work.

References

1 Pankaj K. Agarwal, Sariel Har-Peled, and Kasturi R. Varadarajan. Approximating extent measures of points. *J. ACM*, 51(4):606–635, 2004.

2 Pankaj K. Agarwal, Sariel Har-Peled, and Kasturi R. Varadarajan. Geometric approximation via coresets. In Emo Welzl, editor, *Current Trends in Combinatorial and Computational Geometry*, pages 1–30. Cambridge University Press, 2005.

3 Pankaj K. Agarwal, Sariel Har-Peled, and Hai Yu. Robust shape fitting via peeling and grating coresets. *Discrete & Computational Geometry*, 39(1-3):38–58, 2008. doi:10.1007/s00454-007-9013-2.

4 Pankaj K. Agarwal, Jeff M. Phillips, and Hai Yu. Stability of ε-kernels. In *Proceedings of the 18th Annual European Symposium on Algorithms, Part I*, pages 487–499, 2010. doi:10.1007/978-3-642-15775-2_42.

5 Alexandr Andoni and Huy L. Nguyen. Width of points in the streaming model. In *Proceedings of the 23rd Annual ACM–SIAM Symposium on Discrete Algorithms*, pages 447–452, 2012. *ACM Trans. Algorithms*, to appear.

6 Sunil Arya and Timothy M. Chan. Better ε-dependencies for offline approximate nearest neighbor search, Euclidean minimum spanning trees, and ε-kernels. In *Proceedings of the 30th Annual Symposium on Computational Geometry*, pages 416–425, 2014. doi:10.1145/2582112.2582161.

7 Gill Barequet and Sariel Har-Peled. Efficiently approximating the minimum-volume bounding box of a point set in three dimensions. *J. Algorithms*, 38(1):91–109, 2001.

8 Saugata Basu, Richard Pollack, and Marie-Françoise Roy. On the combinatorial and algebraic complexity of quantifier elimination. *J. ACM*, 43(6):1002–1045, 1996. doi:10.1145/235809.235813.

9 Timothy M. Chan. Faster core-set constructions and data-stream algorithms in fixed dimensions. *Comput. Geom.*, 35(1-2):20–35, 2006. doi:10.1016/j.comgeo.2005.10.002.

10 Timothy M. Chan. Dynamic coresets. *Discrete & Computational Geometry*, 42(3):469–488, 2009. doi:10.1007/s00454-009-9165-3.

11 Gereon Frahling, Piotr Indyk, and Christian Sohler. Sampling in dynamic data streams and applications. *Int. J. Comput. Geometry Appl.*, 18(1/2):3–28, 2008. doi:10.1142/S0218195908002520.

12 Piotr Indyk. Algorithms for dynamic geometric problems over data streams. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing*, pages 373–380, 2004. doi:10.1145/1007352.1007413.

13 Yi Li, Huy L. Nguyen, and David P. Woodruff. Turnstile streaming algorithms might as well be linear sketches. In *Proceedings of the 46th Annual ACM Symposium on Theory of Computing*, pages 174–183, 2014. doi:10.1145/2591796.2591812.

14 R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.

15 S. Muthukrishnan. Data streams: Algorithms and applications. *Foundations and Trends in Theoretical Computer Science*, 1(2), 2005.

16 Hamid Zarrabi-Zadeh. An almost space-optimal streaming algorithm for coresets in fixed dimensions. *Algorithmica*, 60(1):46–59, 2011. doi:10.1007/s00453-010-9392-2.