PART TWO

ADVANCED SUMMARIES AND EXTENSIONS

5

Geometric Summaries

This chapter deals with data in a multidimensional space, that is, how to summarize a point set P in the d-dimensional Euclidean space $\mathbb{R}^d$. As many geometric summaries are just (carefully chosen) subsets of the points in P that approximately preserve certain properties of P, they are often called the *coresets* of P (for the respective properties).

We assume that the coordinates of the points in the data set are real numbers. For simplicity, we will assume that all computations involving real numbers are exact; in actual implementation, using floating point numbers (single or double precision) usually works fine; when rounding errors may cause robustness issues, or a higher precision is desired, some exact computation package (such as the one provided in `CGAL`) can be used. In this chapter, $\|\cdot\|$ always denotes the Euclidean norm.

5.1 ε-Nets and ε-Approximations

Brief Summary. ε-nets and ε-approximations are small summaries about a set of points P with respect to a certain class of ranges $\mathcal{R}$. In the plane, for example, $\mathcal{R}$ can be all the orthogonal rectangles, or all the halfplanes[1]. Let $|P| = n$. Given $0 < \varepsilon < 1$, a set $N \subseteq P$ is called an *ε-net* for P with respect to $\mathcal{R}$ if, for any range R of $\mathcal{R}$ with $|R \cap P| \geq \varepsilon n$, we have $N \cap R \neq \emptyset$, i.e., N "hits" every range in $\mathcal{R}$ that contains at least a fraction of ε of the points of P. This allows us to perform one-sided threshold tests for the ranges in $\mathcal{R}$: For any $R \in \mathcal{R}$, we simply check if R contains any point of N. If no, we are certain that R contains less than εn points

[1] A *halfplane* is the region on either side of an infinite straight line.

of P; but if yes, R may or may not contain at least εn points of P. In other words, an ε-net can be used to detect all the "heavy" ranges on P, although it might return some false positives.

A set $A \subseteq P$ is called an *ε-approximation* for $(P, \mathcal{R})$ if, for any $R \in \mathcal{R}$,

$$\left| \frac{|R \cap P|}{|P|} - \frac{|R \cap A|}{|A|} \right| \leq \varepsilon,$$

i.e., the fraction of points of P contained in R is approximately the same (at most ε apart) as that of A. Therefore, we can approximately count the number of points of P inside any range R by simply counting the number of points in $R \cap A$ and scaling back. The additive error will be at most εn.

Note that an ε-approximation is also an ε-net, but not vice versa.

The easiest way to construct ε-nets and ε-approximations is by simply drawing a random sample from P, and remarkably, the sample size only depends on ε and d, but not n, for almost all natural geometric range spaces (the notion will be made more precise below).

Operations on the summary. Suppose the range space $\mathcal{R}$ has VC-dimension d (see precise definition below), a random sample of size $\frac{d}{\varepsilon}(\log \frac{1}{\varepsilon} + 2 \log\log \frac{1}{\varepsilon} + 3)$ from P is an ε-net for $(P, \mathcal{R})$ with probability at least $1 - e^{-d}$, and a random sample of size $O(\frac{1}{\varepsilon^2}(d + \log \frac{1}{\delta}))$ is an ε-approximation with probability at least $1 - \delta$. Then one can simply use the random sampling algorithms to CONSTRUCT, UPDATE, or MERGE ε-nets or ε-approximations. Please refer to Section 2.2.

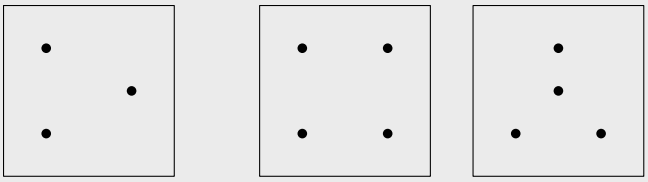
Note that although a random sample of a certain size is an ε-net or an ε-approximation with high probability, it is very expensive to actually verify that the obtained sample is indeed one [108]. In practice, we can only increase the sample size to reduce the probability of failure. Alternatively, there are deterministic algorithms for constructing ε-nets and ε-approximations, but they are also expensive and complicated, as mentioned under "History and Background" below.

Further Discussion. The proofs for the claimed bounds above are quite technical; below we give proofs for slightly weaker results, which nevertheless still capture all the basic ideas.

Consider ε-nets first. For any particular range R with $|R \cap P| \geq \varepsilon n$, a randomly sampled point from P hits R with probability at least ε, thus if we sample s points to form N, then N misses R with

probability at most $(1-\varepsilon)^s \leq e^{-\varepsilon s}$. This means that with a sample size of $O(1/\varepsilon)$, we can hit R with constant probability. However, the challenge is that there are infinitely many ranges in $\mathcal{R}$. A first observation is that if two ranges R and R' contain the same set of points in P, the two can be considered as the same. Let $\pi_{\mathcal{R}}(n)$ be the maximum number of distinct ranges for any P with $|P| = n$. Clearly, $\pi_{\mathcal{R}}(n)$ depends on $\mathcal{R}$: For halfplanes, $\pi_{\mathcal{R}}(n) = O(n^2)$, as each distinct halfplane must be uniquely defined by two points on its boundary; for ellipses, the problem is more complicated. In order to derive a general result, we need the concept of VC-dimensions for range spaces.

Consider a range space $\mathcal{R}$. A set of points $X \subset \mathbb{R}^d$ is *shattered* by $\mathcal{R}$ if all subsets of X can be obtained by intersecting X with members of $\mathcal{R}$, i.e., for any $Y \subset X$, there is some $R \in \mathcal{R}$ such that $Y = X \cap R$. For example, if $\mathcal{R}$ consists of all the halfplanes in two dimensions, then the following figure shows 3 point sets that can be either shattered or not shattered, respectively. On the other hand, if $\mathcal{R}$ is all the ellipses, then all 3 point sets can be shattered. Thus intuitively, ellipses are more powerful, hence have higher complexity, than halfplanes.



(i) (ii) (iii)

Shattered by halfplanes Not shattered by halfplanes

The *Vapnik-Chervonenkis dimension*, or *VC-dimension*, of a range space $\mathcal{R}$ captures this complexity, and is defined to be the size of the largest point set X that can be shattered by $\mathcal{R}$. It can be verified that no 4-point set can be shattered by halfplanes, so the range space of halfplanes has VC-dimension 3. On the other hand, the range space of all ellipses has VC-dimension at least 4. (To be precise, the actual VC-dimension of ellipses is 5: The 5 points on a regular pentagon is shattered by ellipses, and no 6 points can be shattered). Figuring out the exact VC-dimension for any given range space may not be

easy, but luckily for most natural geometric range spaces, the problem has already been studied. As a rule of thumb, if the ranges are defined by some shape of constant description size (e.g., simplices, balls, but not, say, arbitrary convex sets), then the VC-dimension is also a constant.

If there is no assumption on $\mathcal{R}$, $\pi_{\mathcal{R}}(n)$ can be as high as 2^n. It turns out if $\mathcal{R}$ has bounded VC-dimension, $\pi_R(n)$ can be effectively bounded.

Fact 5.1 *For any range space $\mathcal{R}$ of VC-dimension d, $\pi_{\mathcal{R}}(n) = O(n^d)$.*

A proof of this fact can be found in [220]. Intuitively, each distinct range can be characterized by up to d points in P, so $\pi_{\mathcal{R}}(n) \leq \binom{n}{0} + \binom{n}{1} + \cdots + \binom{n}{d} = O(n^d)$. Note that, however, Fact 5.1 may not give the tightest bound on $\pi_{\mathcal{R}}(n)$. For example, halfplanes have VC-dimension 3, but $\pi_{\mathcal{R}}(n)$ is actually only $O(n^2)$. Nevertheless, this does not affect the bounds on ε-nets and ε-approximations asymptotically.

Now with Fact 5.1, we can continue the analysis on the size of ε-net from above. Since a random sample of size s misses any particular heavy range with probability at most $e^{-\varepsilon s}$, it misses any one of the $\leq \pi_{\mathcal{R}}(n)$ heavy ranges with probability at most $n^d e^{-\varepsilon s}$, by a simple union bound. Thus, setting $s = O(\frac{d}{\varepsilon} \log \frac{n}{\delta})$ suffices for the random sample to hit all heavy ranges, i.e., become an ε-net, with probability at least $1 - \delta$.

The analysis for ε-approximations is similar. First, consider any particular range R, and suppose $\frac{|R \cap P|}{n} = \beta$. Recall that for a random sample A of size s to be an ε-approximation of P, $\frac{|A \cap R|}{s}$ should be within $\beta \pm \varepsilon$, which translates to $(\beta - \varepsilon)s \leq |A \cap R| \leq (\beta + \varepsilon)s$. However, each sampled point falls inside R with probability β, so we have $\mathsf{E}[|A \cap R|] = \beta s$. Since each sampled point being inside R or not is an independent Bernoulli event, we can use the Chernoff bound (Fact 1.5) to bound its probability that $|A \cap R|$ deviates from its expectation by more than εs as

$$\exp\left(-O\left(\left(\frac{\varepsilon}{\beta}\right)^2 \beta s\right)\right) = \exp\left(-O\left(\frac{\varepsilon^2 s}{\beta}\right)\right) \leq \exp(-O(\varepsilon^2 s)).$$

By the union bound, the probability that A deviates from the correct fraction for any of the $\pi_{\mathcal{R}}(n)$ ranges is at most $n^d \exp(-O(\varepsilon^2 s))$.

Thus, choosing $s = O(\frac{d}{\varepsilon^2} \log \frac{n}{\delta})$ gives us an ε-approximation with probability at least $1 - \delta$.

To get the tighter bound as mentioned previously, one has to use more careful analysis than simply applying the union bound. In particular, to remove the dependency on n, one has to more carefully bound the number of distinct ranges where two ranges should be considered the same if they differ by no more than εn points. Some pointers are given in the "History and Background" section below.

Finally, it is worth pointing out that all the analysis in this section can be carried over to any set system $(P, \mathcal{R})$, where P is any set of objects and $\mathcal{R}$ is a collection of subsets of P. For instance, we can define P to be all lines in the plane, and define $\mathcal{R}$ to be all the subsets of the following form: Each subset P_s included in $\mathcal{R}$ is the set of lines intersected by some segment s. This set system has VC-dimension 4, and all the results in this section continue to hold with $d = 4$.

History and Background. The notion of VC-dimension originated in statistics, introduced by Vapnik and Chervonenkis [220], who also established initial random sampling bound for ε-approximations. It has been subsequently applied and further developed in many areas such as computational learning theory, computational geometry, and combinatorics. The $O(\frac{1}{\varepsilon^2}(d + \log \frac{1}{\delta}))$ bound was proved by Talagrand [210] and Li *et al.* [164], who also showed that it is optimal within a constant factor.

The notion of ε-net is due to Haussler and Welzl [126]. The dependence on d was subsequently improved by Blumer *et al.* [30] and by Komlós *et al.* [155], who gave the result mentioned above, which is almost tight.

The easiest way to construct an ε-net or an ε-approximation is by random sampling. By using more careful constructions and/or exploiting the special properties of the range space, asymptotically smaller ε-nets and ε-approximations can be obtained. In one dimension with all the intervals as the range space, we can simply sort all the points of P and take every (εn)-th point. It is easy to see that this gives us an ε-net as well as an ε-approximation. So for this range space, the size of ε-net or an ε-approximation is $\Theta(1/\varepsilon)$. Note that such an ε-approximation is also a summary that supports ε-approximate rank and quantile queries as in Chapter 4, but this summary cannot be updated or merged.

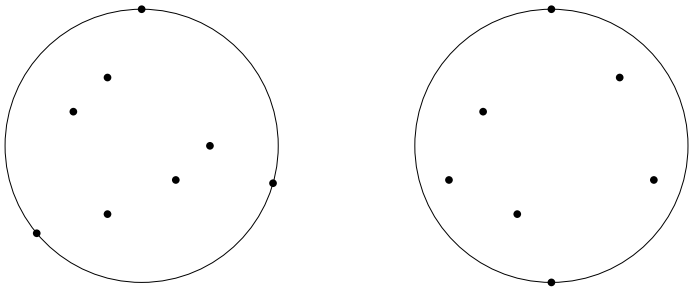


Figure 5.1 The minimum enclosing ball in two dimensions is defined by either 3 points (the example on the left) or 2 points (the example on the right) of P, assuming no 4 points are on the same circle.

In two and higher dimensions, asymptotically smaller ε-nets and ε-approximations are also known, but these results are mostly of a theoretical nature and the corresponding algorithms are quite complicated. For orthogonal rectangles in the plane, an ε-net of size $O(\frac{1}{\varepsilon}\log\log\frac{1}{\varepsilon})$ can be constructed [15, 191], while an ε-approximation of size $O(\frac{1}{\varepsilon}\log^{2.5}\frac{1}{\varepsilon})$ exists [206]. For a general range space with VC-dimension d, the size of an ε-net cannot be improved, but ε-approximations can be reduced to size $O(1/\varepsilon^{2-2/(d+1)})$ [172], where the hidden constant depends on d. With some extra logarithmic factors, such ε-approximations can be updated and merged [2, 207].

5.2 Coresets for Minimum Enclosing Balls

Brief Summary. Let $B_{o,r}$ be the ball of radius r centered at point $o \in \mathbb{R}^d$, i.e., $B_{o,r} = \{p \in \mathbb{R}^d : \|p - o\| \leq r\}$. Let P be a set of n points in $\mathbb{R}^d$. The *minimum enclosing ball* of P, denoted as MEB(P), is the minimum-radius ball containing P. Note that the center of MEB(P) may not be a point of P. Figure 5.1 gives two examples of MEBs in two dimensions. In general, in d dimensions, the MEB is defined by up to $d+1$ points of P. As it turns out, if some approximation is allowed, this number can be made independent of d, which is captured by the notion of an ε-coreset.

Given $\varepsilon > 0$, a subset $K \subseteq P$ is an *ε-coreset* (for MEB) of P if $P \subset B_{o,(1+\varepsilon)r}$, where $B_{o,r} = \text{MEB}(K)$, i.e., the $(1+\varepsilon)$-factor expansion of the MEB of K contains P. It turns out that such a coreset of size $O(1/\varepsilon)$ exists, which is independent of both n and d. This makes it especially appealing for large high-dimensional data. In particular, many kernel

Algorithm 5.1: ε-coreset for MEB: CONSTRUCT(P)

```
K ← {p}, where p is an arbitrary point in P;
while true do
    B_{o,r} ← MEB(K);
    if P ⊂ B_{o,(1+ε)r} then return K;
    q ← arg max_{x∈P} ||o − x||;
    K ← K ∪ {q};
```

Algorithm 5.2: ε-coreset for MEB: UPDATE(p)

```
A ← A ∪ {p};
if A is not full then return;
if A ⊈ ⋃_{i=1}^{u} (1 + ε)B_i then
    K' ← CONSTRUCT (⋃_{i=1}^{u} K_i ∪ A);
    B' ← MEB(K');
    foreach K_i ∈ 𝒦 do
        if r(K_i) ≤ ε/4 · r(B') then Remove K_i from 𝒦;
    Renumber the indexes of 𝒦 if necessary;
    Append K' to 𝒦;
A ← ∅;
```

methods in machine learning can be equivalently formulated as MEB problems in high dimensions, and much faster machine learning algorithms have been designed by using this coreset [219].

Operations on the summary. For a ball B, we use $(1+\varepsilon)B$ to denote its $(1+\varepsilon)$-expansion, and use $r(B)$ to denote the radius of B. To CONSTRUCT the ε-coreset of P, we start with $K = \{p\}$, where p is an arbitrary point in P. Then we add points from P into K iteratively. In each iteration, we first check if $(1+\varepsilon)\,\mathrm{MEB}(K)$ encloses all points of P. If yes, we are done and the current K must be an ε-coreset of P. Otherwise, we pick the furthest point in P from the center of $\mathrm{MEB}(K)$ and add it to K. The pseudocode of this algorithm is given in Algorithm 5.1. It has been shown that the algorithm always finishes in at most $2/\varepsilon$ iterations, so the resulting coreset has size as most $2/\varepsilon$ [39].

To be able to UPDATE the ε-coreset, we maintain a sequence of coresets $\mathcal{K} = (K_1, \ldots, K_u)$. We maintain their MEBs explicitly $B_i = \mathrm{MEB}(K_i)$

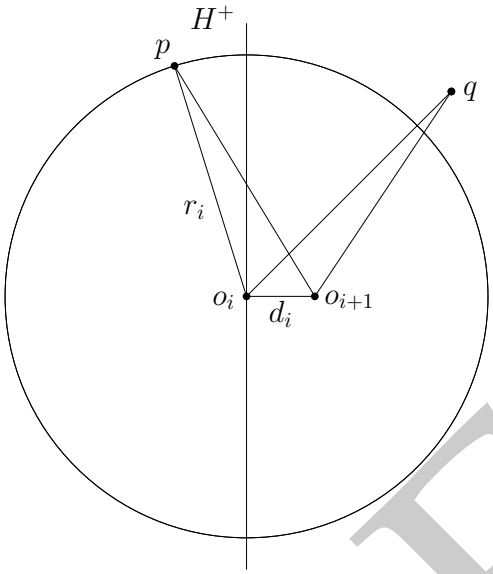


Figure 5.2 Algorithm 5.1 in the $(i+1)$-th iteration.

for $i = 1, \ldots, u$. The sequence is ordered such that $r(B_i) < r(B_j)$ for all $i < j$. There is also a buffer A for new points. Initially, the buffer is empty and $u = 0$. To UPDATE the summary with a new point p, we first add it to the buffer A. When the size of the buffer reaches a certain limit, we perform the following procedure. If all the points in A are inside $\bigcup_{i=1}^{u}(1+\varepsilon)B_i$, we just clear A and we are done. Otherwise, we CONSTRUCT an $(\varepsilon/3)$-coreset K' on $\bigcup_{i=1}^{u} K_i \cup A$, and add it $\mathcal{K}$. Let $B' = \text{MEB}(K')$. Then we clear A. Finally, we delete all K_i's in $\mathcal{K}$ for which $r(B_i) \leq \frac{\varepsilon}{4} r(B')$. This removes a prefix of $\mathcal{K}$, so we need to renumber the indexes of the K_i's. The pseudocode of this algorithm is given in Algorithm 5.2. The limit on the buffer A controls the tradeoff of the size of the summary and update time, and can be usually set to $\frac{1}{\varepsilon}\log\frac{1}{\varepsilon}$. To query the summary for the MEB of all the points P that have ever been added, we first flush the buffer A by following the procedure above, and then return $\text{MEB}\left(\bigcup_{i=1}^{u} B_i\right)$. However, this only returns an $(1.22 + O(\varepsilon))$-approximation of MEB(P), namely, we no longer have an ε-coreset for P. In fact, there is a lower bound stating that any summary that maintains an ε-coreset under insertions must have size $\Omega(\exp(d^{1/3}))$ [4].

It is not known if this summary can be merged.

Further Discussion. Now we show that Algorithm 5.1 always terminates in at most $2/\varepsilon$ iterations, resulting in a coreset of size at most $2/\varepsilon$.

Fact 5.2 *For any set of n points P in $\mathbb{R}^d$, Algorithm 5.1 terminates in at most $2/\varepsilon$ iterations.*

Proof We follow the argument in [39]. Let K_i be the coreset after the i-th iteration, let $B_{o_i,r_i} = \text{MEB}(K_i)$, and let $B_{o^*,r^*} = \text{MEB}(P)$. In the $(i+1)$-th iteration, the algorithm adds q, the furthest point in P from o_i, to K_i. Please refer to Figure 5.2. We must have $\|q - o_i\| > r^*$, otherwise $B_{o_i,\|q-o_i\|}$ would have been the MEB of P. Consider the distance that the center o_i moves in this iteration $d_i = \|o_{i+1} - o_i\|$. If $d_i = 0$, then the algorithm terminates, since $\text{MEB}(K_{i+1}) = B_{o_i,\|q-o_i\|}$ encloses all points of P. If $d_i > 0$, consider the hyperplane H that passes through o_i and is orthogonal to $\overline{o_i o_{i+1}}$. Let H^+ be the closed halfspace bounded by H that does not contain o_{i+1}. Since B_{o_i,r_i} is the MEB of K_i, there must be a point $p \in K_i \cap H^+$ that lies on the surface of B_{o_i,r_i} (see e.g., [116] for a proof for this property of MEBs). So we have $\|o_i - p\| = r_i$ and $r_{i+1} \geq \|o_{i+1} - p\| \geq \sqrt{r_i^2 + d_i^2}$. In addition, $r^* \leq \|o_i - q\| \leq d_i + \|o_{i+1} - q\| \leq d_i + r_{i+1}$. Therefore,

$$r_{i+1} \geq \max\left\{r^* - d_i,\ \sqrt{r_i^2 + d_i^2}\right\}. \tag{5.1}$$

The RHS of (5.1) is minimized when

$$r^* - d_i = \sqrt{r_i^2 + d_i^2},$$

or

$$d_i = \frac{r^* - r_i^2/r^*}{2}.$$

So we have

$$r_{i+1} \geq r^* - \frac{r^* - r_i^2/r^*}{2} = \frac{r^* + r_i^2/r^*}{2}. \tag{5.2}$$

Setting $r_i = \lambda_i r^*$ for all i, (5.2) becomes

$$\lambda_{i+1} \geq \frac{1 + \lambda_i^2}{2}. \tag{5.3}$$

Substituting $\lambda_i = 1 - \frac{1}{\gamma_i}$ in recurrence (5.3), we get

$$\gamma_{i+1} \geq \frac{\gamma_i}{1 - \frac{1}{2\gamma_i}} = \gamma_i \left(1 + \frac{1}{2\gamma_i} + \frac{1}{4\gamma_i^2} + \cdots\right) \geq \gamma_i + \frac{1}{2}.$$

Since $\lambda_0 = 0, \gamma_0 = 1$, so $\gamma_i \geq 1 + i/2$ and $\lambda_i \geq 1 - \frac{1}{1+i/2}$. Therefore, to get $\lambda_i > 1 - \varepsilon$, it is enough that $1 + i/2 \geq 1/\varepsilon$, or $i \geq 2/\varepsilon$. □

The analysis of Algorithm 5.2 is more involved. For the proofs of the following fact, we refer the reader to [4, 47].

Fact 5.3 *Algorithm 5.2 maintains $O(\log(1/\varepsilon))$ coresets $K_1, K_2, \ldots,$ such that $P \subset \text{MEB}\left(\bigcup_i \text{MEB}(K_i)\right)$, whose radius is at most $1.22 + O(\varepsilon)$ times that of $\text{MEB}(P)$.*

Implementation Issues. There are various ways to implement the algorithm efficiently with little or no impact on the quality of the resulting coreset. To start with, instead of initializing the algorithm with an arbitrary point p (line 1 of Algorithm 5.1), we can find the furthest neighbor of p in P, i.e., the point that maximizes the distance from p, say q, and start the algorithm with $K = \{q\}$. This usually reduces the size of the resulting coreset by 1.

Computing the MEB (line 3 of Algorithm 5.1) can be formulated as a quadratic programming problem, for which many efficient solvers are available. Most of them use some iterative method to gradually approach the optimal solution, which is the MEB in our case. For the purpose of this algorithm, finding the exact MEB is unnecessary. Kumar *et al.* [156] proved that it suffices to find the MEB in each iteration of the algorithm up to error $O(\varepsilon^2)$ to still guarantee that the resulting K is an ε-coreset , while in practice it seems that an error of ε works just fine. Moreover, since only one point is added to K in each iteration, the corresponding quadratic program only changes slightly. So one can use the MEB solution obtained from the previous iteration as the starting point, so that the quadratic programming solver can converge quickly.

Finally, instead of finding the furthest neighbor from the center of $\text{MEB}(K)$ (line 5 of Algorithm 5.1), which requires $O(n)$ time, we can take a random sample of the points in P, and choose the furthest one in the sample. This may increase the size of the resulting coreset slightly, but can significantly improve the running time. When all the sampled points are within the $(1 + \varepsilon)$-expansion of $\text{MEB}(K)$, we still have to ex-

amine all points in P. But when this happens, the algorithm will usually soon terminate in a few more iterations.

History and Background. The notion of ε-coreset for MEBs, as well as Algorithm 5.1 for constructing ε-coresets, was proposed by Bădoiu *et al.* [41], who showed that this algorithm terminates in $O(1/\varepsilon^2)$ rounds. Later, Bădoiu and Clarkson [39] improved the bound to $2/\varepsilon$ as described above. Independently, Kumar *et al.* [156] also showed an $O(1/\varepsilon)$ bound, as well as an efficient implementation with some experimental results. Their experimental results suggest that for typical inputs, the algorithm returns coresets that are much smaller than the $2/\varepsilon$ bound. Bădoiu and Clarkson [40] later gave an example on which the coreset has size $1/\varepsilon$, and presented an algorithm that always finds a coreset at most this size, so this problem can be considered as completely solved even up to the constant. But the new algorithm usually returns the same coreset as the one produced by Algorithm 5.1.

Algorithm 5.2 was proposed by Agarwal and Sharathkumar [4]. They showed that this algorithm maintains a sequence of $O(\log \frac{1}{\varepsilon})$ coresets (thus using space $O(\frac{d}{\varepsilon} \log \frac{1}{\varepsilon})$) and returns a $\frac{1+\sqrt{3}}{2}+\varepsilon = (1.366+\varepsilon)$-approximation to the MEB. The approximation ratio (of the same algorithm) was subsequently improved to $1.22 + \varepsilon$ by Chan and Pathak [47]. On the other hand, Agarwal and Sharathkumar [4] proved that any summary that returns a $\frac{1+\sqrt{2}}{2}(1-2/d^{1/3})$-approximation of the MEB while being able to support the addition of new points has to have size $\Omega(\exp(d^{1/3}))$, which rules out any small ε-coresets for MEBs in high dimensions. For low dimensions, ε-coresets that support addition of new points are known; please see the next section.

5.3 ε-Kernels

Brief Summary. Let P be a set of n points in $\mathbb{R}^d$. The ε-kernel is an approximation of the convex hull of P. Unlike the convex hull itself which can have as many as n points, its ε-kernel has a small size which is independent of n. In this section, d is assumed to be a constant.

Let u be a unit directional vector in d dimensions, i.e., $u \in \mathbb{R}^d$ with $\|u\| = 1$. For any direction u, define the *directional width* of P in direction u, denoted by $\omega(u, P)$, to be

$$\omega(u, P) = \max_{p \in P} \langle u, p \rangle - \min_{p \in P} \langle u, p \rangle,$$

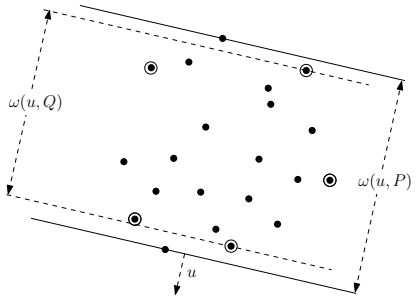


Figure 5.3 An example of a point set P and its ε-kernel (circled points). For any direction u, the width of the circled points (spanned by the two dashed lines) is only a $(1+\varepsilon)$-factor smaller than the width of the whole point set.

where $\langle \cdot, \cdot \rangle$ is the inner product. The directional width of P is thus the span of P when projected onto u. Let $\varepsilon > 0$ be an error parameter. A subset $Q \subseteq P$ is called an *ε-kernel* of P if for every direction u,

$$(1+\varepsilon)\omega(u, Q) \geq \omega(u, P).$$

Clearly, $\omega(u, Q) \leq \omega(u, P)$. Please refer to Figure 5.3 for an example.

The ε-kernel is a coreset for many measures on P that depend on the convex hull of P, such as diameter (the distance between the two furthest points), width (maximum directional width), radius of the minimum enclosing ball, and the volume of the minimum enclosing box, etc.

Operations on the summary. There are two steps to CONSTRUCT an ε-kernel. In step one, we compute an affine transformation π so that $\pi(P)$ is an *α-fat* point set, namely, P is contained in $[-1, 1]^d$ but its convex hull encloses $[-\alpha, \alpha]^d$, where $0 < \alpha < 1$ is a constant. This is done by finding a bounding box C (not necessarily orthogonal) of P, as follows. Find the two furthest points $p, q \in P$. The segment pq defines the direction and length of one side of C. Then project all points in P into the $(d-1)$-dimensional hyperplane perpendicular to $\overline{pq}$, and repeat the process above to find the remaining sides of C. The affine transformation π is thus the one that maps C to $[-1, 1]^d$. Please see Figure 5.4 for an example in two dimensions. This transformation gives a fatness α of at least $1/2$.

In step two, we compute the ε-kernel of $\pi(P)$. To compute the ε-kernel of an α-fat point set, we set $\delta = \sqrt{\varepsilon\alpha}$, and consider the sphere S of radius $\sqrt{d}+1$ centered at the origin. We place a set R of $O(1/\delta^{d-1}) = O(1/\varepsilon^{(d-1)/2})$

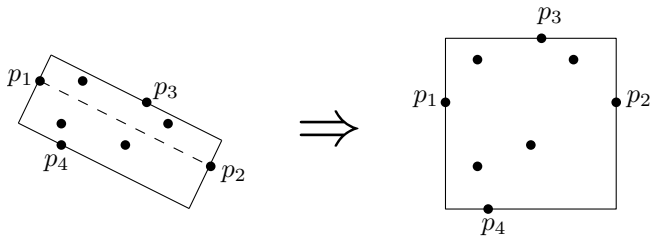


Figure 5.4 Transforming P into a fat point set in two dimensions. First, identify the two furthest points p_1 and $p_2 \in P$, which defines the first side of C. Then project all points onto the line $\overline{p_1 p_2}$ and find the two furthest points on this line, p_3 and p_4, which defines the second side of C. Finally, we take the affine transformation π that maps C to the unit square, and apply it to all points of P.

"equally spaced" points on the sphere such that for any point x on the sphere S, there is a point $y \in R$ such that $\|x - y\| \leq \delta$. Then, for each point in R, we find its nearest neighbor in $\pi(S)$ and add it to Q. The resulting set of chosen points Q becomes the ε-kernel of $\pi(P)$. Finally, we map the points in Q back to the original space by applying π^{-1}. Note that the size of the ε-kernel is $O(1/\varepsilon^{(d-1)/2})$. Figure 5.5 shows an example of how to construct an ε-kernel for a fat point set.

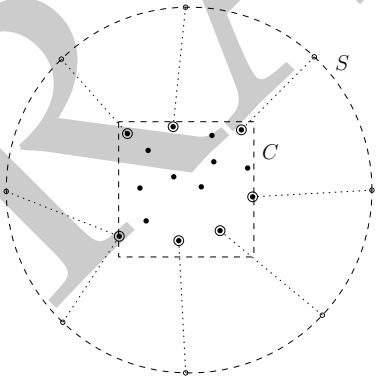


Figure 5.5 Constructing the ε-kernel for a fat point set in two dimensions: The points on the circle S are equally spaced and the distance between any two of them is at least δ. Each point on S picks its nearest neighbor in $\pi(P)$ (the circled points) and they form the ε-kernel of P. Note that multiple points in R on the sphere S can pick the same point in P as their nearest neighbor. In this case, it is stored only once in the ε-kernel.

To be able to UPDATE the ε-kernel, we apply a general method that only relies on the CONSTRUCT procedure, known as the *logarithmic method*. For simplicity we assume that $1/\varepsilon$ is an integer. Let $a/\varepsilon^{(d-1)/2}$ be the maximum size of the ε-kernel built by the CONSTRUCT algorithm above, for some constant a. For $i \geq 1$, define $\rho_i = \varepsilon/i^2$. Instead of maintaining one kernel, we maintain a series of kernels, each associated with a *rank*. To INITIALIZE the summary, we create only one kernel $Q_0 = \emptyset$ and set its *rank* to be 0. To UPDATE the summary with a new point p, we first add it to Q_0. If $|Q_0| \leq a/\varepsilon^{(d-1)/2}$, we are done. Otherwise we CONSTRUCT a ρ_1-kernel of Q_0, denoted by Q', and set its rank to 1. Then we clear $Q_0 = \emptyset$. Next, we check if there is another kernel Q'' in the summary with rank 1. If not, we are done; otherwise, we union Q' and Q'' together, and construct a ρ_2-kernel of their union, set its rank to 2, and delete Q' and Q''. We carry out this procedure iteratively: Whenever there are two kernels Q' and Q'' of the same rank i, we CONSTRUCT a ρ_{i+1}-kernel of their union, set its rank to $i + 1$, and delete Q' and Q''. Eventually, there is at most one kernel left for each rank. It should be clear that there are at most $O(\log n)$ kernels in the summary after n points have been inserted. To extract a kernel for all these points, we can simply return the union of these $O(\log n)$ kernels, of total size $O(\log n/\varepsilon^{(d-1)/2})$. If a smaller size is desired, we can CONSTRUCT an ε-kernel of the union of these $O(\log n)$ kernels.

Two ε-kernel summaries as described above can be similarly merged. We first merge the two kernels of rank 0 in the two summaries. If the resulting kernel has size more than $a/\varepsilon^{(d-1)/2}$, we CONSTRUCT a ρ_1-kernel for it and assign it rank 1. Next, we consider each rank i in the increasing order iteratively. For each rank i, there can be 0, 1, 2, or 3 (in case a new kernel of rank i has just been constructed by merging two kernels of rank $i - 1$) kernels. Whenever there are 2 or 3 kernels of rank i, we merge them together, and construct a ρ_{i+1}-kernel on their union. In the end, we still have at most one kernel at each rank.

Further Discussion. Below, we briefly sketch the proof that the CONSTRUCT algorithm computes a subset $Q \subseteq P$ that is an ε-kernel of P, following the argument in [231]. We assume that P is α-fat for some constant α. We also assume that in the last step of the algorithm, we find the exact nearest neighbors of the points in R. Fix a direction $u \in \mathbb{S}^{d-1}$. Let $\sigma \in P$ be the point that maximizes $\langle u, p \rangle$

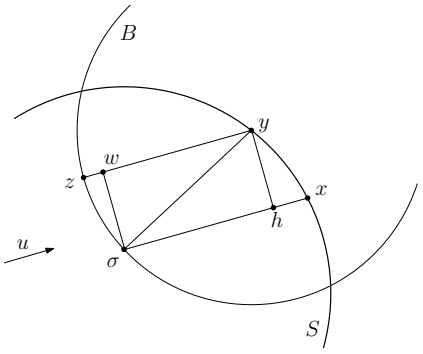


Figure 5.6 Correctness proof of the CONSTRUCT algorithm.

over all $p \in P$. Suppose the ray emanating from σ in direction u hits the sphere S at a point x. We know that there exists a point $y \in R$ such that $\|x - y\| \leq \delta$. Let $\varphi(y)$ denote the nearest neighbor of y in P. If $\varphi(y) = \sigma$, then $\sigma \in Q$ and

$$\max_{p \in P} \langle u, p \rangle - \max_{q \in Q} \langle u, q \rangle = 0.$$

Now suppose $\varphi(y) \neq \sigma$. Let B be the d-dimensional ball of radius $\|y - \sigma\|$ centered at y. Since $\|y - \varphi(y)\| \leq \|y - \sigma\|$, $\varphi(y) \in B$. Let us denote by z the point on the sphere ∂B that is hit by the ray emanating from y in direction $-u$. Let w be the point on zy such that $zy \perp \sigma w$ and h the point on σx such that $yh \perp \sigma x$; see Figure 5.6.

The hyperplane perpendicular to u and passing through z is tangent to B. Since $\varphi(y)$ lies inside B, $\langle u, \varphi(y) \rangle \geq \langle u, z \rangle$. Moreover, it can be shown that $\langle u, \sigma \rangle - \langle u, \varphi(y) \rangle \leq \alpha\varepsilon$. Thus, we can write

$$\max_{p \in P} \langle u, p \rangle - \max_{q \in Q} \langle u, q \rangle \leq \langle u, \sigma \rangle - \langle u, \varphi(y) \rangle \leq \alpha\varepsilon.$$

Similarly, we have $\min_{p \in P} \langle u, p \rangle - \min_{q \in Q} \langle u, q \rangle \geq -\alpha\varepsilon$.

The above two inequalities together imply that $\omega(u, Q) \geq \omega(u, P) - 2\alpha\varepsilon$. Since $[-\alpha, \alpha]^d$ is contained in the convex hull of P, $\omega(u, P) \geq 2\alpha$. Hence $\omega(u, Q) \geq (1 - \varepsilon)\omega(u, P)$. For sufficiently small ε (and also by scaling ε slightly), this is essentially $(1 + \varepsilon)\omega(u, Q) \geq \omega(u, P)$.

Analysis of the UPDATE *and* MERGE *algorithm.* We will only need the following two properties of kernels.

1. If P_2 is an ε_1-kernel of P_1, and P_3 is an ε_2-kernel of P_2, then P_3 is an $(1+\varepsilon_1)(1+\varepsilon_2)-1 = (\varepsilon_1+\varepsilon_2+\varepsilon_1\varepsilon_2)$-kernel of P_1;
2. If P_2 is an ε-kernel of P_1, and Q_2 is an ε-kernel of Q_2, then $P_2 \cup Q_2$ is an ε-kernel of $P_1 \cup Q_1$.

By simple induction, we can show that the kernel at rank i has error

$$\prod_{l=1}^{i}(1+\rho_i)-1 = \prod_{l=1}^{i}\left(1+\frac{\varepsilon}{l^2}\right)-1 \leq \exp\left(\sum_{l=1}^{i}\frac{\varepsilon}{l^2}\right)-1 \leq \exp\left(\frac{\pi^2\varepsilon}{6}\right)-1 = O(\varepsilon).$$

So, the union of all the $O(\log n)$ kernels, or after another CONSTRUCT algorithm, still has error $O(\varepsilon)$.

The total size of the data structure is

$$\sum_{i=0}^{\lfloor \log \varepsilon^{(d-1)/2} n \rfloor + 1} O\left(1/\rho_i^{(d-1)/2}\right) = \sum_{i=0}^{\lfloor \log \varepsilon^{(d-1)/2} n \rfloor + 1} O\left(\frac{i^{d-1}}{\varepsilon^{(d-1)/2}}\right) = O\left(\frac{\log^d n}{\varepsilon^{(d-1)/2}}\right).$$

Implementation Issues. The CONSTRUCT algorithm can be implemented with different choices in the various steps that can significantly reduce its running time without affecting the quality of the resulting kernel too much. In computing the bounding box C, we do not have to find the exact diameter of P (two furthest points), which takes $O(n^2)$ time, in deciding each side of C. Instead, one can use the following simple $O(n)$-time algorithm to find a constant-approximation of the diameter [91]. Pick an arbitrary point $p \in P$. Find its furthest neighbor $q \in P$. Then find the furthest neighbor of q in P, denoted by q'. Note that q, q' must be the two furthest points along the direction $\overline{qq'}$, which can then be used to decide one side of C.

Placing equally spaced points on a circle in 2D is easy, but the problem becomes nontrivial in higher dimensions. A good way is to start with an arbitrary set of k points on the sphere, regarding each point as an electron, and then using the gradient descent method to minimize the total potential energy of the point set by repeatedly fine-tuning the position of each point. The final configuration tends to be a regular distribution on the sphere. This method also gives us an explicit control on the kernel size, and there is no need to compute the actual value of α. The weaknesses of this approach are that it does not give a guarantee on the error of the resulting ε-kernel, and that it can be slow in high dimensions or when k is large. But since this procedure is independent of the point set, it just needs to be performed once, possibly in an offline stage, for each desired kernel size.

Finally, in the last step of the CONSTRUCT algorithm, we do not have to find the exact nearest neighbors of the points in R; some approximation can be tolerated. One can for example use the ANN package [185] for approximate nearest neighbor search.

History and Background. The notion of ε-kernels was introduced by Agarwal et al. [3]. The construction algorithm described above was proposed independently by Chan [46] and Yu et al. [231]. The $O(1/\varepsilon^{(d-1)/2})$ size of the resulting ε-kernels is optimal in the worst case. The practical methods presented here are from [231]; while theoretically, an ε-kernel can be constructed in time $O(n + 1/\varepsilon^{d-3/2})$ [46]. The experimental results in [231] show that the algorithm works extremely well in low dimensions (≤ 4) both in terms of the kernel size/error and running time. However, it does not scale to high dimensions due to the exponential dependency on d.

The UPDATE algorithm is also described in [3], which is a modification of the logarithmic method of Bentley and Saxe [24]. This algorithm is simple and practical, but it raises the theoretical question whether it is possible to maintain an ε-kernel using space that is independent of n. Chan [46] answered the question in the affirmative, by presenting an algorithm that uses $1/\varepsilon^{O(d)}$ space. This result has been subsequently improved by Agarwal and Yu [5] and Zarrabi-Zadeh [232], ultimately yielding an algorithm using $O(1/\varepsilon^{(d-1)/2} \log(1/\varepsilon))$ space.

5.4 k-Center Clustering

Brief Summary. Let $B_{o,r}$ be the ball of radius r centered at point $o \in \mathbb{R}^d$. Let P be a set of n points in $\mathbb{R}^d$. In the k-center clustering problem, we wish to find the minimum radius r and k balls with radius r centered at $o_1, \ldots, o_k \in \mathbb{R}^d$, such that $B_{o_1,r} \cup \cdots \cup B_{o_k,r}$ contain all points in P. Note that when $k = 1$, this degenerates to the minimum enclosing ball problem discussed in Section 5.2. However, unlike the minimum enclosing ball problem which can be solved in polynomial time, the k-center clustering problem is NP-hard. There is a simple 2-approximation algorithm for the k-center clustering problem: Let $O = \emptyset$ be a collection of centers. In each step, find the point $p \in P$ with the largest $d(p, O)$ and add it to O, where $d(p, O)$ denotes the minimum distance from p to any center in O. This step is repeated k times so that we have k centers in O in the end. This algorithm requires us to keep the entire set of points

P. Below, we show how to turn this idea into a small summary of size $O(k/\varepsilon \cdot \log(1/\varepsilon))$ that maintains a $(2+\varepsilon)$-approximation of the optimal k-center clustering.

Operations on the summary. If we knew the optimal radius r_{OPT}, we could turn the 2-approximation algorithm above into a summary of size k, as shown in Algorithm 5.3. It is easy to see why the summary size, $|O|$, never exceeds k when $r \geq r_{\text{OPT}}$. This is because all points can be covered by k balls with radius r, and the algorithm never adds more than one point from each ball into O. Meanwhile, we must not have seen any point that is $2r$ away from any center in O (otherwise the algorithm would have added it to O), so all points can be covered by balls with radius $2r$ centered at the centers in O. Thus, if we ran Algorithm 5.3 with $r = r_{\text{OPT}}$, we would find a 2-approximation of the optimal k-center clustering.

Algorithm 5.3: k-center-with-r: CONSTRUCT(r, P)

```
1 O ← ∅;
2 for p ∈ P do
3     if d(p, O) > 2r then O ← O ∪ {p};
4 return O;
```

However, the problem is that we do not know r_{OPT} in advance. Furthermore, when more points are added to the set of points on which we wish to summarize, r_{OPT} will gradually increase . Therefore, the idea is to run multiple instances of Algorithm 5.3 with different values of r. For $j = 0, 1, \ldots, j_{\max} - 1$ where $j_{\max} = \lceil \log_{1+\varepsilon}(1/\varepsilon) \rceil$, instance j maintains a collection of centers O_j with radius r_j. The r_j's will be a $(1+\varepsilon)$-factor apart from each other. When more points are added, r_{OPT} will increase, and some instances will fail, i.e., having more than k centers. When $|O_j| > k$ for some j, we re-run the algorithm with a larger $r_j \leftarrow r_j/\varepsilon$ on O_j. If with the new r_j, O_j still has more than k centers, we increase r_j again.

A technical problem is from what value of r we should start. If $n \leq k$, then the optimal clustering is just to put each point in a cluster on its own. When $n = k + 1$, we find the closest pair of points, say p_1, p_2, and set $r_0 = d(p_1, p_2)/2, r_j = (1+\varepsilon)^j r_0$ to start running the algorithm.

The complete algorithm is shown in Algorithm 5.4. The algorithm starts with $n = 0$ and $P = \emptyset$. Upon a QUERY, we return the O_j with the smallest r_j.

Algorithm 5.4: k-center: UPDATE(p)

```
1  n ← n + 1;
2  if n ≤ k + 1 then P ← P ∪ {p};
3  if n = k + 1 then
4      p_1, p_2 ← the two closest points in P;
5      r_0 ← d(p_1, p_2)/2;
6      O_0 ← CONSTRUCT (r_0, P);
7      for j = 1, ..., j_max − 1 do
8          r_j = (1 + ε)r_{j−1};
9          O_j ← CONSTRUCT (r_j, P);
10 else
11     for j = 0, ..., j_max − 1 do
12         if d(p, O_j) > 2r_j then O_j ← O_j ∪ {p};
13         while |O_j| > k do
14             r_j ← r_j/ε;
15             O_j ← CONSTRUCT (r_j, O_j);
```

It is not known if this summary can be merged.

Further Discussion. Now we show that Algorithm 5.4 returns a $(2+\varepsilon)$-approximation of the optimal k-center clustering. It is sufficient to prove the following:

Fact 5.4 *For any O_j maintained by Algorithm 5.4, $\bigcup_{o\in O_j} B_{o,(2+3\varepsilon)r_{\mathrm{OPT}}}$ covers all points.*

Proof Algorithm 5.4 is actually running $j_{\max}$ instances independently. Instance j starts with $r_j = (1+\varepsilon)^j r_0$, and increases $r_j \leftarrow r_j/\varepsilon$ and reconstructs O_j from O_j itself whenever $|O_j| > k$. Suppose

$$r_0(1+\varepsilon)^{j-1}/\varepsilon^i \le r_{\mathrm{OPT}} < r_0(1+\varepsilon)^j/\varepsilon^i,$$

for some integers $0 \le j \le j_{\max}-1$, and $i \ge 0$. Note that such i, j must exist as we set $j_{\max} = \lceil \log_{1+\varepsilon}(1/\varepsilon) \rceil$. Then instance j must succeed, i.e., $|O_j| \le k$ when it reaches $r_j = r_0(1+\varepsilon)^j/\varepsilon^i$. Below we argue that the balls centered O_j must be able to cover all points with a certain radius. Recall the earlier argument. If we had run the instance with

$r_j = r_0(1+\varepsilon)^j/\varepsilon^i$ from the beginning, then a radius of $2r_j$ is enough. However, every time we rebuild O_j with $r_j = r_0(1+\varepsilon)^j/\varepsilon^t$, we do not have all points available, so have only built it from O_j itself. Nevertheless, observe that any point must be within a distance of $2r_0(1+\varepsilon)^j/\varepsilon^{t-1}$ of the centers in O_j. Carrying out this argument inductively and using the triangle inequality, we will need in total a radius of

$$\begin{aligned}\sum_{t=0}^{i} 2r_0(1+\varepsilon)^j/\varepsilon^t &= 2r_0(1+\varepsilon)^j \frac{1/\varepsilon^i - 1}{1/\varepsilon - 1} \\ &\leq 2r_0(1+\varepsilon)^{j-1}/\varepsilon^i \cdot (1+\varepsilon)\frac{1}{1/\varepsilon - 1} \\ &\leq 2r_{\text{OPT}} \cdot \frac{1+\varepsilon}{1-\varepsilon} \\ &\leq 2(1+3\varepsilon)r_{\text{OPT}}, \qquad (\text{assuming } \varepsilon < 1/4)\end{aligned}$$

to cover all points. □

If we replace ε with $\varepsilon/3$ to run the algorithm, then a $(2+\varepsilon)$-approximation is guaranteed.

History and Background. The algorithm presented in this section and its analysis is from [122].

5.5 The (Sparse) Johnson-Lindenstrauss Transform

Brief Summary. The (Sparse) Johnson-Lindenstrauss transform (Sparse JLT) transforms a set of n points in $\mathbb{R}^d$ to a set P' of n points in $\mathbb{R}^k$ for $k = \Theta(1/\varepsilon^2 \log n)$, such that the pairwise distance of the points are preserved up to a multiplicative error of $1+\varepsilon$. More precisely, it states that for any $\varepsilon > 0$, there exists a linear transform $f : \mathbb{R}^d \to \mathbb{R}^k$ such that for all $x, y \in P$,

$$(1-\varepsilon)\|x-y\| \leq \|f(x)-f(y)\| \leq (1+\varepsilon)\|x-y\|.$$

It is thus a summary that reduces dimensionality, not the number of points in the data set P. It is a fundamental result concerning low-distortion embeddings of points from high-dimensional to low dimensional Euclidean space.

It turns out that the linear transform f can be obtained by simply

picking a random $k \times d$ matrix S from a properly designed probability distribution. It can be shown that for such a random S and $k = \Theta(1/\varepsilon^2 \log(1/\delta))$, we have

$$\Pr[(1-\varepsilon)\|x\| \leq \|Sx\| \leq (1+\varepsilon)\|x\|] > 1-\delta \tag{5.4}$$

for any $x \in \mathbb{R}^d$. Note that the transform S is independent of P and (5.4) holds for any $x \in \mathbb{R}^d$. Therefore, we can set $\delta = 1/n^2$, pick an S, and apply (5.4) to each of the $\binom{n}{2}$ pairs of the points in P so that all pairwise distances are preserved with probability at least $1/2$.

Operations on the summary. We now specify how the Sparse JLT linear transform S is decided. Let $x_1, \ldots, x_d$ denote the d coordinates of a point $x \in \mathbb{R}^d$. We use y to denote the transformed point $y = Sx$, which has k coordinates, for some $k = \Theta(1/\varepsilon^2 \log(1/\delta))$. Set $r = \Theta(1/\varepsilon \log(1/\delta))$ such that it divides k. We will view the k coordinates of y as an $r \times k/r$ array, i.e. r rows each of size k/r. The array is initialized to all 0. For each row j of the array, we use a $2\log(1/\delta)$-wise independent hash function $h_j : [d] \to [k/r]$, and a $\log(1/\delta)$-wise independent hash function $g_j : [d] \to \{-1, +1\}$. For each coordinate x_i of x and for each row j of the array, we add $x_i g_j(i)/\sqrt{r}$ to $y_{j,h_j(i)}$. From the right perspective, the Sparse JLT of a single point is essentially the same as the Count Sketch summary for that point, except that (1) the array has dimension $\Theta(1/\varepsilon \log(1/\delta)) \times \Theta(1/\varepsilon)$ as opposed to $\Theta(\log(1/\delta)) \times \Theta(1/\varepsilon^2)$ for Count Sketch; (2) we use hash functions of a higher degree of independence; and (3) we divide the values by $\sqrt{r}$ to give the correct scaling factor, as we do not take a median across rows but (effectively) average. Thus, this summary of a point can be updated and merged in basically the same way as with the Count Sketch, covered in Section 3.5. Finally, the QUERY function provides an estimate of $\|x\|$ by simply computing the norm of y, i.e., taking the squared sum of all entries in the array, as opposed to taking the median of the rows as in Count Sketch. Distances between points can similarly be computed by performing the corresponding operations between their Sparse JLT transformations, then applying the QUERY function.

Note that the Count Sketch also provides a summary that preserves pairwise Euclidean distances between points with the same sketch size. The time to perform an update is actually faster (it does not have the factor of $1/\varepsilon$), but since it uses the median operator to produce an estimate, it does not offer an embedding in to a lower-dimensional Euclidean space, which is needed in certain applications such as nearest-

neighbor search and some machine learning applications. For this reason, the Sparse JLT may often be preferred due to its stronger mathematical guarantees, even though the cost of UPDATE operations is higher.

Further Discussion. The analysis of this technique is not hugely difficult to understand, but involves some concepts from combinatorics and coding theory that go beyond the scope of this volume. Consequently, we choose to sketch out the outline.

A first step is to assume that we are dealing with an input vector x with $\|x\|_2 = 1$. This is because the transformation is linear, so any scalar multiple of x is passed through the estimation process. Assuming $\|x\|_2 = 1$ threatens to make the analysis redundant, since the objective is to form an estimate of $\|x\|_2$, but of course, the algorithm does not rely on this assumption, and so we can proceed.

As noted above, this version of a Sparse JLT transform is very similar in operation to the Count Sketch and the AMS Sketch. Consequently, we use the same notation to describe its analysis. As in Section 3.6, we can define a random variable X_j as the estimate obtained of $\|x\|_2^2$ from the jth row, as

$$X_j = \sum_{p=1}^{k/r} C[j,p]^2 = \sum_{p=1}^{k/r} \frac{1}{r} \left(\sum_{i:h_j(i)=p} x_i^2 + 2 \sum_{i \neq \ell, h_j(i)=h_j(\ell)=p} g_j(i) g_j(\ell) x_i x_\ell \right)$$

where, $C[j,p]$ denotes the contents of the pth cell in the jth row of the sketch; x_i are the entries of vector x; h_j is the hash function assigning indices to cells in row j; and g_j provides the $\{+1,-1\}$ value for each index in row j.

Observe that by the assumption that $\|x\|_2 = 1$, we can write a random variable Z for the total error in estimation as

$$Z = \frac{1}{r} \sum_{j=1}^{r} (X_j - 1) = \frac{2}{r} \sum_{j=1}^{r} \sum_{p=1}^{k/r} \sum_{i \neq \ell, h_j(i)=h_j(\ell)=p} g_j(i) g_j(\ell) x_i x_\ell$$

The analysis proceeds by bounding this random variable Z. For the analogous quantity in the analysis of AMS Sketch, it was possible to bound the second moment (i.e., to bound $\mathsf{E}[Z^2]$) in order to show a constant probability of deviating far from zero error. This was done by expanding out the square of Z, and arguing that

terms were bounded due to the independence properties of the hash functions. The approach in this case is essentially the same, but now we need to show a bound directly in terms of δ, rather than a constant probability. This requires taking a higher moment of Z — we apply Markov's inequality to show that

$$\Pr[|Z| > \epsilon] < \epsilon^{-q}\mathsf{E}[Z^q]$$

where q is chosen to be an even number greater than $\log(1/\delta)$.

However, Z^q does not lend itself to direct expansion and bounding, and so instead, we need a much more involved argument. The high level idea is to use a graph representation of the polynomial to encode the terms that arise: nodes represent indices, and edges link indices that appear in the expansion of the polynomial Z^q. This allows a more convenient notation and bounding of the terms that arise, so the desired inequality can be proven.

History and Background. The JL lemma was established by Johnson and Lindenstrauss [142] in 1984. There have been a number of different proofs of this lemma, based on different constructions of the transform matrix S. Early constructions had S very dense – for example, where every entry is drawn independently from a Gaussian or Rademacher (Bernoulli with +1 or −1 values) distribution [134]. In this case UPDATE operations take time $O(k)$ for each coordinate that is updated. This construction remains of importance, as it leads to results in the relatively new area of compressed sensing [85]. For a concise proof for this Johnson-Lindenstrauss transform, see the proof of Gupta and Dasgupta [78].

Due to the importance of the transform, considerable effort has been invested in making it fast, by sparsifying the matrix S. We avoid a complete history, and highlight some key points along the way. An initial effort was by Achlioptas [1], whose construction had entries chosen independently from $\{-1, 0, +1\}$. This achieved a constant degree of sparsity, i.e., a constant fraction of the entries were non-zero. Ailon and Chazelle achieved a much faster result by introducing more structure into the transformation. The "fast Johnson-Lindenstrauss" transform [8] writes the matrix S as the product of three matrices, $S = PHD$. P is a sparse matrix where only a roughly $\log^2 n/d$ fraction of the entries are non-zero: the non-zero entries are chosen independently from a Gaussian distribution. If we applied P directly on the input, we would achieve the desired result (preservation of the Euclidean norm) — if the input

vector was dense (i.e., had few zero entries, and no particularly large entries). The other parts of the transform are designed to preserve the Euclidean norm of the vector, while (almost certainly) giving the dense property. D is a diagonal matrix whose entries are randomly chosen as $\{-1, +1\}$. Observe that this does not change the norm of the vector, and is very similar to steps in the Count Sketch and Sparse JLT described above that randomly flip signs of entries. H is the Hadamard transform — this is an instance of a Fourier transform and so has some useful properties: (1) it is an orthonormal basis transformation, so again does not change the Euclidean norm of the vector (2) it can be computed quickly using a Fast Fourier Transform algorithm, in time $O(d \log d)$. The Hadamard transform ensures that if x was sparse in the original space, it will be dense in the transformed space. The result of this construction is an instance of a Johnson-Lindenstrauss transform that can be computed for a vector of dimension d in time $O(d \log d)$. However, this relies on computing the transformation for a whole vector in one go. When the vector is being defined incrementally, the cost of a single UPDATE operation is not guaranteed to be fast.

This limitation prompted the study of sparse JL matrices. The version presented here is given by Kane and Nelson [146]. They also describe alternative approaches based on how the mapping to the sketch allows overlaps, with differing guarantees and proofs. It is natural to ask whether the sparsity can be reduced below the $1/\varepsilon$ dependency, or whether the number of dimensions k can be reduced below $1/\varepsilon^2$. Both questions have been answered in the negative. The number of dimensions of the reduced space $k = \Theta(1/\varepsilon^2 \log(1/\delta))$ has been shown to be optimal [140]. Nelson and Nguyen show no construction can achieve sparsity less than $1/\varepsilon$, as this would give too high a probability that significant entries of the input vector are not picked up by the transform [188].