Chapter 2

The grid strike back: Dimension, grids, packing, and pruning

By Sariel Har-Peled, July 7, 2017[①]

> Peter Piper picked a peck of pickled peppers.
> A peck of pickled peppers Peter Piper picked.
> If Peter Piper picked a peck of pickled peppers,
> Where's the peck of pickled peppers that Peter Piper picked?
> – Nursery rhyme.

> "I tried to read this book, Huckleberry Finn, to my grandchildren, but I couldn't get past page six because the book is fraught with the 'n-word.' And although they are the deepest-thinking, combat-ready eight- and ten-year-olds I know, I knew my babies weren't ready to comprehend Huckleberry Finn on its own merits. That's why I took the liberty to rewrite Mark Twain's masterpiece. Where the repugnant 'n-word' occurs, I replaced it with 'warrior' and the word 'slave' with 'dark-skinned volunteer.'"
> – The sellout, Paul Beatty.

In this chapter, we further investigate the power of low dimension, by illustrating one its most powerful properties, namely, that one can not pack too many points into a domain if the required distance between points is close to the domain diameter. This packings are a powerful tool in summarizing data if we are interested only in a certain resolution. In particular, we demonstrate how to compute such packings quickly using grids in low dimensions. We then show how this technique leads to linear time approximation algorithms for several problems in low dimensions. These algorithms use grids, together with alternating between packing and pruning the data to get linear time approximation algorithms for a large number of problems.

2.1. Metric spaces

Definition 2.1.1. A ***metric space*** is a pair $(\mathcal{X}, \mathrm{d})$ where $\mathcal{X}$ is a set and $\mathrm{d} : \mathcal{X} \times \mathcal{X} \to [0, \infty)$ is a ***metric*** satisfying the following axioms: (i) $\mathrm{d}_{\mathcal{M}}(x, y) = 0$ if and only if $x = y$, (ii) $\mathrm{d}_{\mathcal{M}}(x, y) = \mathrm{d}_{\mathcal{M}}(y, x)$, and (iii) $\mathrm{d}_{\mathcal{M}}(x, y) + \mathrm{d}_{\mathcal{M}}(y, z) \geq \mathrm{d}_{\mathcal{M}}(x, z)$ (triangle inequality).

For example, $\mathbb{R}^2$ with the regular Euclidean distance is a metric space. In the following, we assume that we are given ***black-box access*** to $\mathrm{d}_{\mathcal{M}}$. Namely, given two points $p, q \in \mathcal{X}$, we assume that $\mathrm{d}_{\mathcal{M}}(p, q)$ can be computed in constant time.

Another standard example for a finite metric space is a graph G with non-negative weights $\omega(\cdot)$ defined on its edges. Let $\mathrm{d}_G(x, y)$ denote the shortest path (under the given weights) between any $x, y \in V(G)$. It is easy to verify that $\mathrm{d}_G(\cdot, \cdot)$ is a metric. In fact, any ***finite metric*** (i.e., a metric defined over a finite set) can be represented by such a weighted graph.

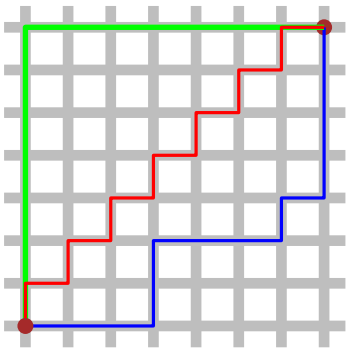
The ***L_p-norm*** defines the distance between two points $p, q \in \mathbb{R}^{\mathsf{d}}$ as

$$\left\| p - q \right\|_p = \left(\sum_{i=1}^{\mathsf{d}} |p_i - q_i|^p \right)^{1/p},$$

for $p \geq 1$. The ***L_2-norm*** is the regular Euclidean distance.

The ***L_1-norm***, also known as the ***Manhattan distance*** or ***taxicab distance***, is

$$\left\| p - q \right\|_1 = \sum_{i=1}^{\mathsf{d}} |p_i - q_i| .$$

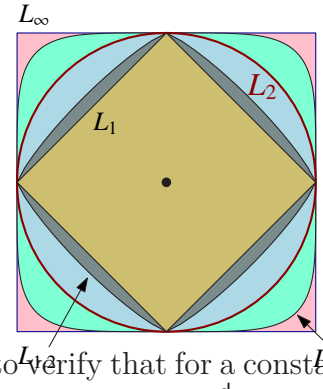


The L_1-norm distance between two points is the minimum path length that is axis parallel and connects the two points. For a uniform grid, it is the minimum number of grid edges (i.e., blocks in Manhattan) one has to travel between two grid points. In particular, the shortest path between two points is no longer unique; see the picture on the right. Of course, in the L_2-norm the shortest path between two points is the segment connecting the two points.

The ***L_∞-norm*** is

$$\left\| p - q \right\|_\infty = \lim_{p \to \infty} \left\| p - q \right\|_p = \max_{i=1}^{\mathsf{d}} |p_i - q_i| .$$

The triangle inequality holds for the L_p-norm, for any $p \geq 1$ (it is called the ***Minkowski inequality*** in this case). In particular, L_p is a metric for $p \geq 1$. Specifically, $\mathbb{R}^{\mathsf{d}}$ with any L_p-norm (i.e., $p \geq 1$) is another example of a metric space.



It is useful to consider the different unit balls of L_p for different value of p; see the figure on the right. The figure implies (and one can prove it formally) that for any point $p \in \mathbb{R}^{\mathsf{d}}$, we have that $\|p\|_p \leq \|p\|_q$ if $p > q$.

Lemma 2.1.2. *For any $p \in \mathbb{R}^{\mathsf{d}}$, we have that $\|p\|_1 / \sqrt{\mathsf{d}} \leq \|p\|_2 \leq \|p\|_1$.*

Proof: Indeed, let $p = (p_1, \ldots, p_{\mathsf{d}})$, and assume that $p_i \geq 0$, for all i. It is easy to verify that for a constant α, the function $f(x) = x^2 + (\alpha - x)^2$ is minimized when $x = \alpha/2$. As such, setting $\alpha = \|p\|_1 = \sum_{i=1}^{\mathsf{d}} |p_i|$, we have, by symmetry and by the above observation on $f(x)$, that $\sum_{i=1}^{\mathsf{d}} p_i^2$ is minimized under the condition $\|p\|_1 = \alpha$, when all the coordinates of p are equal. As such, we have that $\|p\|_2 \geq \sqrt{\mathsf{d}(\alpha/\mathsf{d})^2} = \|p\|_1 / \sqrt{\mathsf{d}}$, implying the claim. ■

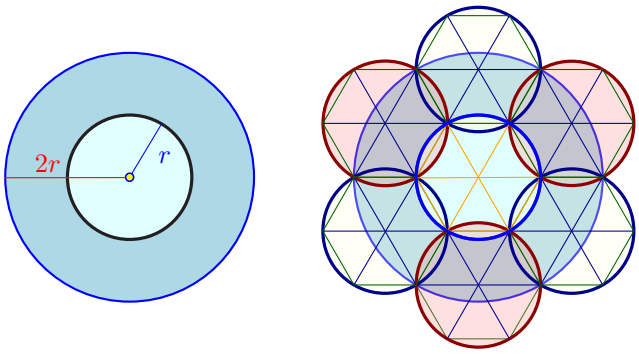


Figure 2.1: A disk of radius $2r$, can be covered by 7 disks of radius r.

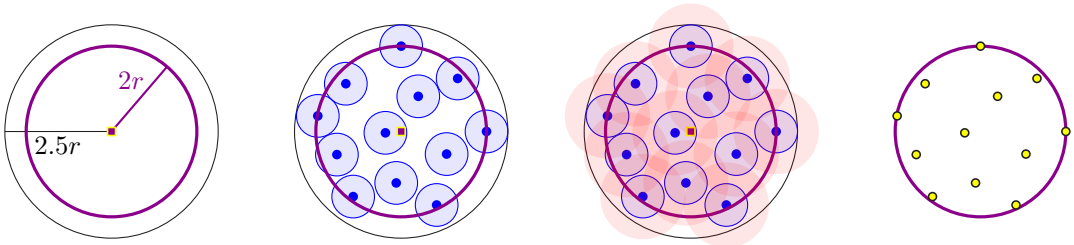


Figure 2.2: (A) The setup. (B) A maximal packing of balls of radius $r/2$ inside a ball of radius $2.5r$. (C) Expanding the balls covers the ball of radius $2r$. (D) The resulting r-packing of the ball of radius $2r$.

2.2. Covering, dimension and packings

2.2.1. Covering a ball by balls of half the radius

Fact 2.2.1. Let b be a ball of radius r in $\mathbb{R}^d$ (under the L_2 norm). It is known that $\mathrm{Vol}(\mathsf{b}) = \beta_d r^d$, where β_d is a constant that depends only on d (specifically, $\beta_d \leq (18/d)^{d/2}$) – this follows by a standard integration argument.

Lemma 2.2.2. *Let $\mathfrak{c}_d$ be the minimum number of balls of radius r in $\mathbb{R}^d$ that cover a ball of radius $2r$. Then, the constant $\mathfrak{c}_2 = 7$, and more generally $\mathfrak{c}_d \leq 5^d$. The constant $\mathfrak{c}_d$ is the* **doubling constant** *of $\mathbb{R}^d$.*

Proof: The planar case is demonstrated in Figure 2.1, and it is easy to prove that one can not cover the ball of radius $2r$ by six disks of radius r.

For $d > 2$, let $\mathsf{b} = \mathrm{ball}(\mathsf{o}, 2.5r) \subseteq \mathbb{R}^d$, where o denotes the origin. Let $\mathcal{B}$ be the maximum cardinality set of balls of radius $r/2$ that are (i) contained in b, and (ii) are interior disjoint. See Figure 2.2.

As such, it follows that $|\mathcal{B}| \leq \mathrm{Vol}\Big(\mathrm{ball}\big(2.5r\big)\Big)/\mathrm{Vol}\Big(\mathrm{ball}\big(r/2\big)\Big) = \beta_d(2.5r)^d/\big(\beta_d r^d\big) = 5^d$. Let $\mathcal{B}'$ be the set of balls resulting from replacing every ball in $\mathcal{B}$, by a ball with the same center, but of radius r. We claim that $\mathcal{B}'$ covers $\mathsf{b}_{2r} = \mathrm{ball}(\mathsf{o}, 2r)$. Indeed, if there is a point $p \in \mathsf{b}_{2r}$ that is not covered by any ball of $\mathcal{B}'$, then p is in distance at least $r/2$ from all the balls of $\mathcal{B}$. But then, one can add $\mathrm{ball}(p, r/2)$ to $\mathcal{B}$, forming a larger set of interior disjoint balls inside b. A contradiction. ∎

The doubling constant can be defined in any abstract metric space, and it leads to a natural definition of dimension.

Definition 2.2.3. Consider a ball b in some given metric space $(\mathcal{X}, \mathrm{d})$, its doubling constant is the minimum number of balls of radius radius(b)/2. The ***doubling constant*** $\mathfrak{c}$ of $\mathcal{X}$ is the maximum doubling constant over all the possible balls in $\mathcal{X}$. The ***doubling dimension*** of $\mathcal{X}$ is $\mathfrak{d} = \log_2 \mathfrak{c}$.

By Lemma 2.2.2, we have that the doubling dimension of $\mathbb{R}^d$ is $\mathfrak{d}(\mathbb{R}^d) = O(d)$. It is not hard to verify that the doubling dimension of $\mathbb{R}^d$ is $\Omega(d)$. As such, one can think of the doubling dimension as an abstraction of the standard notion of Euclidean dimension.

2.2.2. Packings

The above used a volume packing argument. In particular, it implicitly used the following notion.

Definition 2.2.4. Consider a metric space $(\mathcal{X}, \mathrm{d})$, and a set $P \subseteq \mathcal{X}$. A set $N \subseteq P$ is an ***r-packing*** for P if the following two properties hold.

(i) ***Covering property***: All the points of P are within a distance $< r$ from the points of N.

(ii) ***Separation property***: For any pair of points $p, q \in N$, we have that $\mathrm{d}_{\mathcal{M}}(p, q) \geq r$.

(One can relax the separation property by requiring that the points of N be at a distance $\Omega(r)$ apart.)

Intuitively, an r-packing is a summary of P in resolution r. It tell us how the point-set looks like roughly if we do not care about distances that are significantly smaller than r. In low dimensional Euclidean space, packings can not be too dense. This follows readily from the proof of Lemma 2.2.2.

Lemma 2.2.5. *Let S be an r-packing of a set P in a metric space with doubling dimension $\mathfrak{d}$, and consider a ball $\mathsf{b} = \mathrm{ball}(p, R)$, where $R \geq r$. Then b can contain at most $(2R/r)^{O(\mathfrak{d})}$ points of S. In particular, for $P \subseteq \mathbb{R}^d$, we have that $|S| \leq (R/r)^{O(d)}$.*

Proof: Let $\mathfrak{c} = 2^{\mathfrak{d}}$ be the doubling dimension of the given metric space. Cover b by balls of radius $r/4$. This is done hierarchically – we first cover b by $\mathfrak{c}$ balls of radius $R/2$, and we continue in this refinement process till all the balls are of radius at most $r/4$. This requires $t = \lceil \lg(4R/r) \rceil \leq 3 + \lg(R/r)$ levels of such refinement, and the resulting set of balls $\mathcal{B}$ contains at most $\mathfrak{c}^t = 2^{\mathfrak{d}t} \leq (2R/r)^{O(\mathfrak{d})}$ balls. Now, the diameter of a ball of $\mathcal{B}$ is $2(r/4) \leq r/2$, which implies that such a ball can contain only a single point of an r-packing, since the minimum distance between points in an r-packing is $\geq r$. ■

2.3. Computing packings quickly for a point set in $\mathbb{R}^d$

There is a simple algorithm for computing r-packings. Namely, let all the points in P be initially unmarked. While there remains an unmarked point, p, add p to $\mathbf{C}$, and mark it and all other points in distance $< r$ from p (i.e. we are scooping away balls of radius r). By using grids and hashing one can modify this algorithm to run in linear time.

Lemma 2.3.1. *Given a point set $P \subseteq \mathbb{R}^d$ of size n and a parameter $r > 0$, one can compute an r-packing for P in $O(n)$ time.*

Proof: Let G denote the grid in $\mathbb{R}^d$ with side length $\alpha = r/2\sqrt{d}$. First compute for every point $p \in P$ the grid cell in G that contains p; that is, $\mathrm{id}(p)$, see Definition 2.7.2. Let $\mathcal{G}$ denote the set of grid cells of G that contain points of P. Similarly, for every cell $\square \in \mathcal{G}$ we compute the set of points of P which

it contains. This task can be performed in linear time using hashing and bucketing assuming the floor function can be computed in constant time. Specifically, store the $\mathrm{id}(\cdot)$ values in a hash table, and in constant time hash each point into its appropriate bin.

Scan the points of P one at a time, and let p be the current point. If p is marked then move on to the next point. Otherwise, add p to the set of packing points, $\mathbf{C}$, and mark it and each point $q \in P$ such that $\|p - q\| < r$. Since the cells of $\mathsf{N}_{\le r}(p)$, see Definition 2.7.3, contain all such points, we only need to check the lists of points stored in these grid cells. At the end of this procedure every point is marked. Since a point can only be marked if it is in distance $< r$ from some packing point, and a packing point is only created if it is unmarked when visited, this implies that $\mathbf{C}$ is an r-packing.

As for the running time, observe that a grid cell, $\square$, has its list scanned only if $\square$ is in the neighborhood of some created packing point. As $\alpha = \Theta(r)$, there are only $O(1)$ cells which could contain a packing point p such that $\square \in \mathsf{N}_{\le r}(p)$. Furthermore, at most one packing point lies in a single cell since the diameter of a grid cell is strictly smaller than r. Therefore each grid cell had its list scanned $O(1)$ times. Since the only real work done is in scanning the cell lists and since the cell lists are disjoint, this implies an $O(n)$ running time overall. ■

Observe that the closest packing point, for a point $p \in P$, must be in one of its neighborhood's grid cells. Since every grid cell can contain only a single packing point, it follows that in constant time per point of P, one can compute each point's nearest packing point. We thus have the following.

Corollary 2.3.2. *For a set $P \subseteq \mathbb{R}^d$ of n points, and a parameter $r > 0$, one can compute, in linear time, an r-packing of P, and furthermore, for each packing point the set of points of P for which it is the nearest packing point.*

In the following, a ***weighted point*** is a point that is assigned a positive integer weight. For any subset S of a weighted point set P, let $|S|$ denote the number of points in S and let $\omega(S) = \sum_{p \in S} \omega(p)$ denote the total weight of S.

In particular, Corollary 2.3.2 implies that for a weighted point set one can compute the following quantity in linear time.

Algorithm 2.3.3 (**pack**). Given a weighted point set $P \subseteq \mathbb{R}^d$, let $\mathbf{pack}(r, P)$ denote an r-packing of P, where the weight of each packing point p is the total sum of the weights of the points assigned to it. We slightly abuse notation, and also use $\mathbf{pack}(r, P)$ to designate the algorithm (of Corollary 2.3.2) for computing this packing, which has linear running time.

2.3.1. Identifying close and far points

For a given point $p \in P$, let

$$\mathsf{nn}(p, P) = \arg \min_{q \in P \setminus \{p\}} \|p - q\| \qquad \text{and} \qquad \mathrm{d}(p, P) = \min_{q \in P \setminus \{p\}} \|p - q\|, \tag{2.1}$$

denote the nearest neighbor of p in $P \setminus \{p\}$, and the distance to it, respectively. The quantity $\mathrm{d}(p, P)$ can be computed (naively) in linear time by scanning the points. For a set of points P, and a parameter r, let $P^{\ge r}$ denote the set of ***r-far*** points; that is, it is the set of all points $p \in P$, such that the nearest-neighbor of p in $P \setminus \{p\}$ is at least distance r away (i.e., $\mathrm{d}(p, P) \ge r$). Similarly, $P^{< r}$ is the set of ***r-close*** points; that is, all points $p \in P$, such that $\mathrm{d}(p, P) < r$.

Lemma 2.3.4. *Given a weighted set $P \subseteq \mathbb{R}^d$ of n points, and a distance $\alpha > 0$, in $O(n)$ time, one can compute the sets $P^{<\alpha}$ and $P^{\ge \alpha}$. Let $\mathbf{delFar}(\alpha, P)$ denote the algorithm which computes these sets and returns $P^{<\alpha}$.*

Proof: Build a grid where every cell has diameter α/c, for some constant $c > 1$. Clearly any point $p \in P$ such that $\mathrm{d}(p, P) \geq \alpha$ (i.e., a far point) must be in a grid cell by itself. Therefore to determine all such "far" points only grid cells with singleton points in them need to be considered. For such a point q, to determine if $\mathrm{d}(q, P) \geq \alpha$, one checks the points stored in all grid cells in distance $\leq \alpha$ from it. If q has no such close neighbor, we mark q for inclusion in $P^{\geq r}$. By the same arguments as in Lemma 2.3.1, the number of such cells is $O(1)$. Again by the arguments of Lemma 2.3.1 every non-empty grid cell gets scanned $O(1)$ times overall, and so the running time is $O(n)$. Finally, we copy all the marked (resp. unmarked) points to $P^{\geq \alpha}$ (resp. $P^{<\alpha}$). The algorithm **delFar**(α, P) then simply returns $P^{<\alpha}$. ■

2.4. Approximating nice distance problems

The problem. Here, we consider problems of the following form: Given a set P of weighted points in $\mathbb{R}^d$, one wishes to solve an optimization problem whose solution is one of the pairwise distances of P (or "close" to one of these values). Problems of this kind include computing the optimal k-center clustering, or the length of the kth edge in the MST of P, and many others. Specifically, we are interested in problems for which there is a fast approximate decider. That is, given a value $r > 0$ we can, in linear time, decide if the desired value is (approximately) smaller than r or larger than r. The goal is then to use this decider to approximate the optimum solution in linear time.

Critical values. Naively, one can start with an interval $[0, \alpha]$ that should contain the optimal solution, and then perform a binary search using the decider, at each point picking a middle point of the interval, asking the decider which half of the current interval contains the desired optimal value, and continuing the search into this subinterval. Potentially, such a search might require a huge number of steps, since α might be very large (i.e., think as α as infinite). To overcome this, a standard approach is to identify the ***critical values*** where the optimal value might be attained. For example, for the MST, all its edge lengths are taken from the pairwise distances of pairs of points. As such, finding the longest edge in the MST, can be achieved by doing a binary search over the set of pairwise distances of points in P. The challenge is that the number of such distances might be quite large (e.g., $\Theta(n^2)$), and computing them explicitly might be quite expensive.

The need to summarize data. Even if we could implicitly search over the critical values (which we cannot) then we still would require a logarithmic number of calls to the decider which would yield a running time of $O(n \log n)$, as the number of critical values is at least linear (and usually polynomial). So instead we use the return values of the decision procedure as we search to thin out the data so that future calls to the decision procedure become successively cheaper. However, we still cannot search over the critical values (since there are too many of them) and so we also introduce random sampling in order to overcome this.

Outline of the algorithm. The algorithm works by randomly sampling a point and computing the distance to its nearest neighbor. Let this distance be r. Next, we use the decision procedure to decide if we are in one of the following three cases.

(A) **Pack.** If r is too small then we zoom out to a resolution of r by computing an r-packing and continuing the computation on the packing instead of on the original point set. That is, we pack the point set into a smaller point set, such that one can solve the original problem (approximately) on this smaller sketch of the input.

(B) **Prune.** If r is too large then we remove all points whose nearest neighbor is further than r away (of course, this implies we should only consider problems for which such pruning does not affect the solution). That is, we isolate the optimal solution by pruning away irrelevant data – this is similar in nature to what is being done by prune-and-search algorithms.

(C) **Done.** The value of r is the desired approximation.

We then continue recursively on the remaining data. In either case, the number of points being handled (in expectation) goes down by a constant factor and thus the overall expected running time is linear. In particular, getting this constant factor decrease is the reason we chose to sample from the set of nearest neighbor distances rather than from the set of all inter-point distances.

2.4.1. Problem definition and an example

Definition 2.4.1. Let P and R be two sets of weighted points in $\mathbb{R}^d$ (of the same weight). The set R is a ***Δ-drift*** of P, if R can be constructed by moving each point of P by distance at most Δ (and not altering its weight). Formally, there is an onto mapping $f : P \to R$, such that (i) For $p \in P$, we have that $\|p - f(p)\| \leq \Delta$, and (ii) for any $q \in R$, we have that $\omega(q)$ is the sum of the weights of all points $p \in P$ such that $f(p) = q$.

Note that for a (potentially weighted) point set Q, **pack**(Δ, Q) is a Δ-drift of Q.

Definition 2.4.2. Given a set X and a function $f : X \to \mathbb{R}$, a procedure **decider** is a ***φ-decider*** for f, if for any $x \in X$ and $r > 0$, **decider**(r, x) returns one of the following: (i) $f(x) \in [\alpha, \varphi\alpha]$, where α is some real number, (ii) $f(x) < r$, or (iii) $f(x) > r$.

Definition 2.4.3 (φ-NDP). An instance of a ***φ-NiceDistanceProblem*** consists of a pair (Q, Γ), where $Q \subseteq \mathbb{R}^d$ is a set of n distinct weighted points, and Γ is the ***context*** of the given instance (of size $O(n)$) and consists of the relevant parameters for the problem[2]. For a fixed ***φ-NDP***, the task is to evaluate a function $f(Q, \Gamma) \to \mathbb{R}^+$, defined over such input pairs, that has the following properties:

- Decider: There exists an $O(n)$ time φ-decider for f, for some *constant* $\varphi \geq 1$.
- Lipschitz: Let R be any Δ-drift of Q. Then $|f(Q, \Gamma) - f(R, \Gamma)| \leq 2\Delta$.
- Prune: If $f(Q, \Gamma) < r$ then $f(Q, \Gamma) = f(Q^{<r}, \Gamma')$, where $Q^{<r}$ is the set of r-close points, and Γ' is an updated context which can be computed, in $O(n)$ time, by a procedure denoted **contextUpdate**.

2.4.1.1. An example: k far points

Computing the closest pair of points (see Definition 2.7.4Definition 2.7.4) in a point set is an NDP, but it is somewhat uninteresting because of some technicalities. Instead, as a concrete example of an NDP, consider the problem of finding k-far points.

Problem 2.4.4 (FarPoints: max-min facility dispersion). Let P be a set of points in $\mathbb{R}^d$, and let $k > 0$ be an integer parameter. Find a set of ***points*** $\mathbf{C} \subseteq P$ such that the distance of any pair of points of $\mathbf{C}$ is as large as possible, and $|\mathbf{C}| = k$. For $k = 2$, the problem is to compute the pair of points realizing the diameter of P. Formally, the function of interest is

$$f_{far}(P, k) = \max_{\substack{U \subseteq P \\ |P| = k}} \mathrm{d}_{\min}(U),$$

where $\mathrm{d}_{\min}(P) = \min_{p \neq q,\, p,q \in P} \|p - q\|$. A ***k-far*** set of P, is a set $U \subseteq P$ of size k realizing $f_{far}(P, k)$ – namely, $\mathrm{d}_{\min}(U) = f_{far}(P, k)$.

[2] In almost all cases the context is a set of integers, usually of constant size.

Remark 2.4.5. Usually one defines the input for FarPoints to be a set of unweighted points. However, since the definition of NDP requires a weighted point set, we naturally convert the initial input into a set of unit weight points, Q. Also, for simplicity we assume $f_{far}(Q,k) \neq 0$, i.e. $|Q| > k$ (which can easily be checked).

Lemma 2.4.6. *Let $r > 0$ be a given number, and let $N = \mathbf{pack}(r, Q)$. We have the following*

(A) If $|N| < k$ then $f_{far}(Q,k)/2 < r$.

(B) If $|N| \geq k$ then $f_{far}(Q,k) \geq r$.

Proof: (A) Let $\mathcal{O} \subseteq P$ be an optimal k-far set, and let $N = \mathbf{pack}(r, Q)$ be the given r-packing. Since $k = |\mathcal{O}| > |N|$, it follows that there must be two points of $p, q \in P$, such that their closest point in N is the same. That is $\overline{c} = \mathsf{nn}(p, N) = \mathsf{nn}(q, N)$. Furthermore, by construction $\|\overline{c} - p\| < r$ and $\|\overline{c} - q\| < r$. As such, by the triangle inequality, we have

$$f_{far}(Q,k) \leq \|p - q\| \leq \|p - \overline{c}\| + \|\overline{c} - q\| < r + r = 2r.$$

(B) Since $|N| \geq k$, by definition, every pair of points of N are in distance $\geq r$ from each other, In particular, there are k such points in N, which implies the claim. ■

Lemma 2.4.7. *For any instance (Q,k) of FarPoints, $f = f_{far}(Q,k)$ satisfies the properties of Definition 2.4.3. Formally, FarPoints is a $(4+\varepsilon)$-NDP, for any $\varepsilon > 0$.*

Proof: We need to verify the required properties, see Definition 2.4.3.

Decider: We need to describe a decision procedure for FarPoints clustering. To this end, given a distance r, the decider computes $N_{r/2} = \mathbf{pack}(r/2, Q)$ and $N_r = \mathbf{pack}(r, Q)$, see Algorithm 2.3.3. Now, the decider make a decision according to Lemma 2.4.6:

(I) If $|N_{r/2}| < k$, then "$f < r$" is returned.

(II) If $|N_r| \geq k$, then "$f \geq r$" is returned.

(III) Otherwise, $|N_{r/2}| \geq k > |N_r|$, which implies that $r/2 \leq f_{far}(Q,k) < 2r$. The decider returns "$f \in [r/2, 2r)$".

Lipschitz: Observe that if Q' is a Δ-drift of Q, then for any two points $p, q \in Q$, their images in Q', denoted by p', q', respectively, have the property that $\|p' - q'\| - 2\Delta \leq \|p - q\| \leq \|p' - q'\| + 2\Delta$. As such, let $\mathcal{O}$ be the optimal k-far set of points in Q and let $\mathcal{O}'$ be the corresponding set of points in Q'. Observe that $\mathrm{d}_{\min}(\mathcal{O}) - 2\Delta \leq \mathrm{d}_{\min}(\mathcal{O}') \leq \mathrm{d}_{\min}(\mathcal{O}) + 2\Delta$. Namely, this (specific) solution has moved by at most distance 2Δ when moving from Q to Q'. This argument also works in the other direction, implying that $f_{far}(Q,k)$ and $f_{far}(Q',k)$ are the same, up to an additive error of 2Δ.

Prune: It suffices to show that if $f_{far}(Q,k) < r$ then $f_{far}(Q,k) = f_{far}\left(Q^{<r}, k - |Q^{\geq r}|\right)$.

Let $\mathcal{O} \subseteq Q$ be an optimal k-far set, and let p, p' be the two points of $\mathcal{O}$ realizing $\mathrm{d}_{\min}(\mathcal{O})$. Observe that $r > f_{far}(Q,k) = \mathrm{d}_{\min}(\mathcal{O}) = \|p - p'\|$.

Consider a point $s \in Q^{\geq r}$. If $s = p$ then by definition $\|p - p'\| \geq \mathsf{nn}(p, Q) \geq r$, which is impossible. As such $s \neq p$ and $s \neq p'$. In particular, $\mathcal{O} - s = \mathcal{O} \setminus \{s\}$ is a set of size $\geq k - 1$ with $\mathrm{d}_{\min}(\mathcal{O} - s) = \|p - p'\| = f_{far}(Q,k) < r$ in $Q - s$. As such, $f_{far}(Q - s, k-1) \geq \mathrm{d}_{\min}(\mathcal{O} - s) = f_{far}(Q,k)$.

Now, an optimal $(k-1)$-far set $U \subseteq Q - s$ with $\mathrm{d}_{\min}(U) < r$, corresponds to a set $U + s = U \cup \{s\}$ in Q of size k, with $f_{far}(Q - s, k-1) = \mathrm{d}_{\min}(U) = \mathrm{d}_{\min}(U + s) \leq f_{far}(Q,k)$. (The case $\mathrm{d}_{\min}(U) \geq r \implies f_{far}(Q,k) \geq r$, which is a contradiction.)

We conclude that $f_{far}(Q,k) = f_{far}(Q - s, k-1)$. Applying this argument repeatedly to all the points of $Q^{\geq r}$ implies the claim.[3] ■

[3] It is the benefit of the writer, that the reader does not know how long the writer spent on proving the most obvious things. Apropos nothing, of course.

2.4.2. A linear time approximation algorithm

We now describe the general algorithm which given a φ-NDP, (Q, Γ), and an associated target function, f, computes in linear time a $O(\varphi)$ spread interval containing $f(Q, \Gamma)$. In the following, let **decider** denote the given φ-decider, where $\varphi \geq 1$ is some parameter. Also, let **contextUpdate** denote the context updater associated with the given NDP problem, see Definition 2.4.3. Both **decider** and **contextUpdate** run in linear time. The algorithm for bounding the optimum value of an NDP is the following.

Algorithm 2.4.8. **ndpAlg**(Q, Γ):

1: p: Pick a random point from Q.
   $\alpha \leftarrow \mathrm{d}(p, Q)$.
   `// Next, estimate the value of` $f = f(Q, \Gamma)$
   $res_{>} =$ **decider**(α, Q, Γ)
   $res_{<} =$ **decider**$(c_{\mathrm{pack}}\alpha, Q, \Gamma)$
2: **if** $res_{>} =$ "$f \in [x, y]$" **then return** "$f(Q, \Gamma) \in [x/2, 2y]$".
3: **if** $res_{<} =$ "$f \in [x', y']$" **then return** "$f(Q, \Gamma) \in [x'/2, 2y']$".
4: **if** $res_{>} =$ "$\alpha < f$" and $res_{<} =$ "$f < c_{\mathrm{pack}}\alpha$" **then return** "$f(Q, \Gamma) \in [\alpha/2, 2c_{\mathrm{pack}}\alpha]$".
   if $res_{>} =$ "$f < \alpha$" **then**
5: $R \leftarrow$ **delFar**(α, Q) `// Prune`
   else `// Must be:` $res_{<} =$ "$c_{\mathrm{pack}}\alpha < f$"
6: $R =$ **pack**$(3\alpha, Q)$ `// Pack`
   return ndpAlg$\Big(R,$ **contextUpdate**$(Q, \Gamma, R)\Big)$

Remark 2.4.9. Two low level technicalities: (A) For the sake of simplicity of exposition we assume that $f(Q, \Gamma) \neq 0$. (B) Algorithm 2.4.8 can be modified to handle inputs where Q is a multiset (namely, two points can occupy the same location), and we still assume (as done above) that $f(Q, \Gamma) \neq 0$. Modifying the algorithm to have this property is easy if somewhat tedious[④].

2.4.2.1. Analysis

This algorithm has a tail recursion, and as such, the notion of the ith level of the recursion is well defined. A ***packing level*** of the algorithm is a recursive call where **pack** gets called. Similarly, a ***prune level*** is one where **delFar** gets called. In the following, $P^{\leq\alpha}$ is defined analogously to $P^{<\alpha}$ (see Section 2.3.1).

Lemma 2.4.10. *Let P be a point set. A $(2+\varepsilon)\alpha$-packing of P, for any $\varepsilon > 0$, can contain at most half the points of $P^{\leq\alpha}$.*

Proof: Consider any point p in $P^{\leq\alpha}$ which became one of the packing points. Since $p \in P^{\leq\alpha}$, a disk of radius α centered at p must contain another point q from $P^{\leq\alpha}$ (indeed, $p \in P^{\leq\alpha}$ only if its distance from its nearest neighbor in P is at most α). Moreover, q cannot become a packing point since it is too close to p. Now if we place a ball of radius α centered at each point of $P^{\leq\alpha}$ which became a packing point, then these balls will be disjoint because the pairwise distance between packing points is $\geq (2+\varepsilon)\alpha$. Therefore each point of $P^{\leq\alpha}$ which became a packing point can charge to at least one point of $P^{\leq\alpha}$ which did not make it into the packing, such that no point gets charged twice. ■

[④]The interested reader can figure out how to do it. The uninterested reader can move on. The interested reader might now choose to be uninterested.

Notations. In the following, a subscript of a variable of **ndpAlg** denotes its value in the beginning of the ith level of the recursion, where the initial call is treated as $i = 0$. As such, the original input is (Q_0, Γ_0).

Lemma 2.4.11. *Given an instance (Q, Γ) of an NDP, the algorithm* **ndpAlg**(Q, Γ) *runs in expected $O(n)$ time.*

Proof: Ignoring the recursive call, the only non-trivial work done is in computing α, the two calls to **decider**, and the one call to either **pack** or **delFar**. It has already been shown that all of these can be computed in $O(|Q|)$ time. Hence the total running time for the algorithm is $O\left(\sum_{i=0}^{k-1} |Q_i|\right)$, where k denotes the depth of the recursion.

So consider the beginning of the $(i-1)$th level of the recursion, of $i < k$. Let the points in Q_{i-1} be labeled $p_1, p_2, \ldots, p_m$ in increasing order of their nearest neighbor distance in Q_{i-1}. Let j be the index of the point chosen in Line 1 and let $(Q_{i-1})^{\geq j}$ and $(Q_{i-1})^{\leq j}$ be the subset of the points with index $\geq j$ and index $\leq j$, respectively. Now since a point is randomly picked in Line 1, with probability $\geq 1/2$, $j \in [m/4, 3m/4]$. Lets call this event a ***successful level***. We have $\min\left(|(Q_{i-1})^{\geq j}|, |(Q_{i-1})^{\leq j}|\right) \geq |Q_{i-1}|/4$ for a successful level.

Since $i < k$ is not the last level of the recursion, either **delFar** or **pack** must get called. If **delFar**(α_i, Q_{i-1}) gets called (i.e. Line 5) then by Lemma 2.3.4, all of $(Q_{i-1})^{\geq j}$ gets removed. So suppose **pack** gets called (i.e. Line 6). In this case Lemma 2.4.10 implies that the call to **pack** removes at least $\left|(Q_{i-1})^{\leq j}\right|/2$ points.

Therefore, for any level $i < k$, at least $\nu_i = \min\left(|(Q_{i-1})^{\geq j}|, |(Q_{i-1})^{\leq j}|/2\right)$ points get removed. If this level of the recursion is successful then $\nu_i \geq |Q_{i-1}|/8$. In particular, $\mathbf{E}[\nu_i \,|\, |Q_{i-1}|] \geq |Q_{i-1}|/16$. Now, $|Q_i| \leq |Q_{i-1}| - \nu_i$ and as such $\mathbf{E}[\,|Q_i| \,|\, |Q_{i-1}|] \leq (15/16)\,|Q_{i-1}|$.

Therefore, for $0 < i < k$,

$$\mathbf{E}\Big[|Q_i|\Big] = \mathbf{E}\Big[\mathbf{E}\big[|Q_i| \,|\, Q_{i-1}\big]\Big] \leq \mathbf{E}\left[\frac{15}{16}|Q_{i-1}|\right] = \frac{15}{16}\,\mathbf{E}\Big[|Q_{i-1}|\Big].$$

Hence, by induction on i, $\mathbf{E}\big[|Q_i|\big] \leq (15/16)^i |Q_0|$ and so, in expectation, the running time is bounded by

$$O\left(\mathbf{E}\left[\sum_{i=0}^{k-1} |Q_i|\right]\right) = O\left(\sum_{i=0}^{k-1} \mathbf{E}\Big[|Q_i|\Big]\right) = O\left(\sum_{i=0}^{k-1} (15/16)^i |Q_0|\right) = O\Big(|Q_0|\Big). \qquad \blacksquare$$

Correctness. The formal proof of correctness is somewhat tedious, but here is the basic idea: At every level, either far points are being thrown away (and this does not effect the optimal value), or we packing the points. However, the packing radius being used is always significantly smaller than the optimal value, and throughout the algorithm execution the radii of the packings being used grow exponentially. As such, the accumulated error in the end is proportional to the radius of the last packing computation before termination, which itself is also much smaller than the optimal value.

Before proving that **ndpAlg** returns a bounded spread interval containing $f(Q_0, \Gamma_0)$, several helper lemmas will be needed. For notational ease, in the rest of this section, we use $f(Q_i)$ as shorthand for $f(Q_i, \Gamma_i)$.

Lemma 2.4.12. *Suppose that* **pack** *is called in level i of the recursion (i.e., Line 6 in Algorithm 2.4.8). Then for any deeper level of the recursion $j > i$ we have, $\alpha_j \geq 3\alpha_i$.*

Proof: Consider the beginning of the jth level recursive call. The current set of points, Q_{j-1}, are a subset of the packing points of a $3\alpha_i$-packing (it is a subset since Line 5 and Line 6 might have been executed in between rounds i and j). Therefore, being a packing, the distance between any two points of Q_{j-1} is $\geq 3\alpha_i$, see Definition 2.2.4. In particular, this means that for any point p of Q_{j-1}, we have $\mathrm{d}(p, Q_{j-1}) \geq 3\alpha_i$. ∎

Lemma 2.4.13. *For $i = 1, \ldots, k$, we have $|f(Q_i) - f(Q_0)| \leq 9\alpha_i$.*

Proof: Let I be the set of indices of the packing levels up to (and including) the ith level. Similarly, let $\bar{I}$ be the set of recursive levels where **delFar** get called.

If **pack** was called in the jth level, then Q_j is a $3\alpha_j$-drift of Q_{j-1} and so by the Lipschitz property, $|f(Q_j) - f(Q_{j-1})| \leq 6\alpha_j$. On the other hand, if **delFar** gets called in the jth level, then $f(Q_j) = f(Q_{j-1})$ by the Prune property. Let $m = \max I$, we have that

$$\begin{aligned}\left|f(Q_i) - f(Q_0)\right| &\leq \sum_{j=1}^{i} \left|f(Q_j) - f(Q_{j-1})\right| = \sum_{j\in I} \left|f(Q_j) - f(Q_{j-1})\right| + \sum_{j\in \bar{I}} \left|f(Q_j) - f(Q_{j-1})\right| \\ &\leq \sum_{j\in I} 6\alpha_j + \sum_{j\in \bar{I}} 0 \leq 6\alpha_m \sum_{j=0}^{\infty} \frac{1}{3^j} \leq 9\alpha_m \leq 9\alpha_i,\end{aligned}$$

by Lemma 2.4.12. ∎

The following lemma testifies that the radii of the packings computed by the algorithm are always significantly smaller than the value we are trying to approximate.

Lemma 2.4.14. *For any level i of the recursion such that* **pack** *gets called, we have $\alpha_i \leq f(Q_0)/c_2$, where $0 < c_2 = c_{\mathrm{pack}} - 9$.*

Proof: The proof will be by induction. Let $m_1, \ldots m_t$ be the indices of the levels of the recursion in which **pack** gets called. For the base case, observe that in order for **pack** to get called, we must have $c_2\alpha_{m_1} < c_{\mathrm{pack}}\alpha_{m_1} < f(Q_{m_1-1})$. However, since this is the first level in which **pack** is called it must be that $f(Q_0) = f(Q_{m_1-1})$ (since for all previous levels **delFar** must have been called).

So now suppose that $\alpha_{m_j} \leq f(Q_0)/c_2$ for all $m_j < m_i$. If a call to **pack** is made in level m_i then again $c_{\mathrm{pack}}\alpha_{m_i} < f(Q_{(m_i)-1}) = f(Q_{m_{(i-1)}})$. Thus, by Lemma 2.4.13 and induction, we have

$$\alpha_{m_i} < \frac{f\left(Q_{m_{(i-1)}}\right)}{c_{\mathrm{pack}}} \leq \frac{f(Q_0) + 9\alpha_{m_{(i-1)}}}{c_{\mathrm{pack}}} \leq \frac{f(Q_0) + 9f(Q_0)/c_2}{c_{\mathrm{pack}}} = \frac{1 + 9/c_2}{c_{\mathrm{pack}}} f(Q_0) = \frac{f(Q_0)}{c_2},$$

if $c_2 = \dfrac{c_{\mathrm{pack}}}{1 + 9/c_2}$. This in turn is equivalent to $c_2 + 9 = c_{\mathrm{pack}}$, which is true by definition. ∎

Setting $c_{\mathrm{pack}} = 37$, results in $c_2 = 28$, and by Lemma 2.4.14, for all i that correspond to a packing level, $\alpha_i \leq f(Q_0)/28$. By Lemma 2.4.13, for any packing level i, we have

$$|f(Q_i) - f(Q_0)| \leq 9\alpha_i \leq f(Q_0)/3.$$

In particular, we conclude that $|f(Q_i) - f(Q_0)| \leq f(Q_0)/3$ for any level i. We thus get the following.

Corollary 2.4.15. *For $c_{\mathrm{pack}} \geq 37$, and any i, we have:*

(A) $(2/3)f(Q_0) \le f(Q_i) \le (4/3)f(Q_0)$.
(B) *If* $f(Q_i) \in [x, y]$ *then* $f(Q_0) \in [(3/4)x, (3/2)y] \subseteq [x/2, 2y]$.
(C) *If* $f(Q_0) > 0$ *then* $f(Q_i) > 0$.

Lemma 2.4.16. *For* $c_{\text{pack}} \ge 37$, *given an instance* (Q, Γ) *of a* φ*-NDP,* **ndpAlg**(Q, Γ) *returns an interval* $[x', y']$ *containing* $f(Q, \Gamma)$, *where* $\Phi([x', y']) \le 4 \max(\varphi, c_{\text{pack}})$.

Proof: Consider the iteration of the while loop at which **ndpAlg** terminates, see Algorithm 2.4.8. If the algorithm returns because of the checks in Line 2 or Line 3 in this level, then the interval $[x, y]$ was computed by the φ-decider, and has spread $\le \varphi$. As such, by Corollary 2.4.15, the returned interval $[x', y'] = [x/2, 2y]$ contains the optimal value, and its spread is $\le 4\varphi$.

A similar argumentation holds if Line 4 gets executed. Indeed, the returned interval contains the desired value, and its spread is $4c_{\text{pack}}$. ■

2.4.3. The result

In Definition 2.4.3 required that the decision procedure runs in $O(n)$ time. However, since our analysis also applies for decision procedures with slower running times, we first state the main result is these more general settings.

Lemma 2.4.17. *Given an instance of a* φ*-NDP defined by a set of* n *points in* $\mathbb{R}^d$, *one can get a* φ*-approximation to its optimal value, in* $O(T(n))$ *expected time, where* $T(n)$ *is the running time of the* **decider** *procedure, and* $\varphi \ge 3/2$.

If $\varphi = 1 + \varepsilon$, *for some* $\varepsilon \in (0, 1)$, *(i.e.,* **decider** *is* $(1+\varepsilon)$*-decider), then one can compute a* $(1+\varepsilon)$*-approximation to the optimal value, and the expected running time is* $O\left(T(n) \log \varepsilon^{-1}\right)$.

Proof: Let (Q, Γ) be the given φ-NDP instance, and let **decider** be the corresponding φ-decider. Now, by Lemma 2.4.11 and Lemma 2.4.16, in expected $O(n)$ time, one can get a bounded spread interval $[\gamma, c\gamma]$, for some constant $c \ge 1$ such that $c = O(\varphi)$, such that $f = f(Q, \Gamma) \in [\gamma, c\gamma]$. If $c \le \varphi$ then we are done. Otherwise, perform a binary search over this interval.

Specifically, for $i = 0, 1, \ldots, m = \lfloor \log_\varphi c \rfloor$, let $\gamma_i = \varphi^i \gamma$ and let $\gamma_{m+1} = c\gamma$. Now perform a binary search over the γ_i's using **decider**. If any of the calls returns an interval $[x, \varphi x]$ that contains the optimal value, then $f \le \varphi x \le \varphi f$ and so φx can be returned as the desired φ-approximation. Otherwise, if $f < \gamma_i$ or $f > \gamma_i$ the binary search moves left or right, respectively. In the end, the search ends up with an interval $(\gamma_i, \gamma_{i+1}) = (\gamma_i, \varphi\gamma_i)$ which must contain the optimal value, and again, its spread is φ, and it thus provide the required approximation.

The running time is dominated by the calls to the decider procedure (we are assuming here that $T(n) = \Omega(n)$). Clearly, the number of calls to the decider performed by this algorithm is $O(\log m) = O\left(\log \log_\varphi 4\varphi\right) = O(1)$, if $\varphi \ge 3/2$. Otherwise, if $\varphi = 1 + \varepsilon$, then

$$O(\log m) = O\left(\log \log_\varphi 4\varphi\right) = O\left(\ln\left(1 + \frac{\ln 4}{\ln(1+\varepsilon)}\right)\right) = O\left(\log\left(1 + \frac{1}{\varepsilon}\right)\right) = O\left(\log \frac{1}{\varepsilon}\right),$$

since $\ln(1+\varepsilon) \ge \ln e^{\varepsilon/2} = \varepsilon/2$, for $\varepsilon \in (0, 1/2)$, as can be easily verified. ■

2.5. Bibliographical notes

Section 2.3 is from [HR15], but similar algorithms were described earlier, see [Har04, ERH12].

2.6. Exercises

2.7. From other lectures

Theorem 2.7.1. *Let $X_1, \ldots, X_n$ be independent Bernoulli trials, where $\mathbf{Pr}[X_i = 1] = p_i$ and $\mathbf{Pr}[X_i = 0] = q_i = 1 - p_i$, for $i = 1, \ldots, n$. Furthermore, let $X = \sum_{i=1}^{b} X_i$ and $\mu = \mathbf{E}[X] = \sum_i p_i$. Then we have $\mathbf{Pr}\big[X < (1-\delta)\mu\big] < \exp\big(-\mu\delta^2/2\big)$.*

Definition 2.7.2. Let $p = (x, y)$ be any point in a grid cell $\square \in \mathsf{G}_\alpha$, and consider the pair of integer numbers $\mathrm{id}(\square) = \mathrm{id}(p) = (\lfloor x/\alpha \rfloor, \lfloor y/\alpha \rfloor)$. The pair $\mathrm{id}(p)$ is the ***grid ID*** of p.

Definition 2.7.3. For a number $r \geq 0$, and a point p let $\mathsf{N}_{\leq r}(p)$ denote the set of grid cells of G_α in distance $\leq r$ from p, which is the ***neighborhood*** of p. Note, that the neighborhood also includes the grid cell containing p itself, and if $\alpha = \Theta(r)$ then $|\mathsf{N}_{\leq r}(p)| = \Theta\big((2 + \lceil 2r/\alpha \rceil)^d\big) = \Theta(1)$.

Definition 2.7.4. For a set P of n points in the plane, let $\mathrm{d}_{\min}(P)$ denote the minimum distance between a pair of points in P. Formally, we have

$$\mathrm{d}_{\min}(P) = \min_{p \neq q,\, p,q \in P} \|p - q\| .$$

The pair of points realizing this minimum, is the ***closest pair*** in P, and it is denoted by $\mathcal{CP}(P)$.

Bibliography

[ERH12] A. Ene, B. Raichel, and S. Har-Peled. *Fast Clustering with Lower Bounds: No Customer too Far, No Shop too Small*. In submission. http://sarielhp.org/papers/12/lbc/. 2012.

[Har04] S. Har-Peled. *Clustering motion*. *Discrete Comput. Geom.*, 31(4): 545–565, 2004.

[HR15] S. Har-Peled and B. Raichel. *Net and prune: a linear time algorithm for euclidean distance problems*. *J. Assoc. Comput. Mach.*, 62(6): 44:1–44:35, 2015.