Net and Prune: A Linear Time Algorithm for Euclidean Distance Problems*

Sariel Har-Peled† Benjamin Raichel‡

November 8, 2012

Abstract

We provide a general framework for getting linear time constant factor approximations (and in many cases FPTAS's) to a copious amount of well known and well studied problems in Computational Geometry, such as k-center clustering and furthest nearest neighbor. The new approach is robust to variations in the input problem, and yet it is simple, elegant and practical. In particular, many of these well studied problems which fit easily into our framework, either previously had no linear time approximation algorithm, or required rather involved algorithms and analysis. A short list of the problems we consider include furthest nearest neighbor, k-center clustering, smallest disk enclosing k points, kth largest distance, kth smallest m-nearest neighbor distance, kth heaviest edge in the MST and other spanning forest type problems, problems involving upward closed set systems, and more.

1 Introduction

In many optimization problems, one is given a (say, geometric) input, and one is interested in computing the minimum of a function over this input. Such a function can be, for example, the minimum cost clustering of the input, the price of a minimum spanning tree, the radius of the smallest enclosing disk, the closest pair distance, and many other functions. Often for such optimization problems it is possible to construct a decision procedure, which given a query value can decide whether the query is smaller or larger than the minimum of the function. Naturally, one would like to use this decider to preform a binary search to compute the minimum. However, often this is inherently not possible as the set of possible solutions is a real interval. Instead one must identify a set of critical values at which the function changes. However, searching over these values directly can be costly as often the number of

*Work on this paper was partially supported by a NSF AF awards CCF-0915984 and CCF-1217462. The full updated version of this paper is available online [HR12].

†Department of Computer Science; University of Illinois; 201 N. Goodwin Avenue; Urbana, IL, 61801, USA; sariel@uiuc.edu; http://illinois.edu/~sariel.

‡Department of Computer Science; University of Illinois; 201 N. Goodwin Avenue; Urbana, IL, 61801, USA; raichel2@uiuc.edu; http://illinois.edu/~raichel2.

such critical values is much larger than the desired running time. Instead one attempts to preform an implicit search over them.

One of the most powerful techniques to solve optimization problems efficiently in Computational Geometry, using such an implicit search, is the ***parametric search*** technique of Megiddo [Meg83]. It is relatively complicated, as it involves implicitly extracting values from a simulation of a parallel decision procedure (often a parallel sorting algorithm). For this reason it is inherently not possible for parametric search to lead to algorithms which run faster than $O(n \log n)$ time. Nevertheless, it is widely used in designing efficient geometric optimization algorithms, see [AST94, Sal97]. Luckily, in many cases one can replace parametric search by simpler techniques (see prune-and-search below for example) and in particular, it can be replaced by randomization, see the work by van Oostrum and Veltkamp [vOV04]. Another example of replacing parametric search by randomization is the new simplified algorithm for the Fréchet distance [HR11]. Surprisingly, sometimes these alternative techniques can actually lead to linear time algorithms.

Linear time algorithms. There seems to be three main ways to get linear time algorithms for geometric optimization problems (exact or approximate):

(A) **Coreset/sketch**. One can quickly extract a compact sketch of the input that contains the desired quantity (either exactly or approximately). As an easy example, consider the problem of computing the axis parallel bounding box of a set of points - an easy linear scan suffices. There is by now a robust theory of what quantities one can extract a coreset of small size for, such that one can do the (approximate) calculation on the coreset, where usually the coreset size depends only on the desired approximation quality. This leads to many linear time algorithms, from shape fitting [AHV04], to $(1+\varepsilon)$-approximate k-center/median/mean clustering [Har01, Har04a, HM04, HK05] in constant dimension, and many other problems [AHV05]. The running times of the resulting algorithms are usually $O(n + \text{func}(\text{sketch size}))$. The limitation of this technique is that there are problems for which there is no small sketch, from clustering when the number of clusters is large, to problems where there is no sketch at all [Har04b] – for example, for finding the closest pair of points one needs all the given input and no sketching is possible.

(B) **Prune and search**. Here one prunes away a constant fraction of the input, and continues the search recursively on the remaining input. The paramount example of such an algorithm is the linear time median finding algorithm, but there are many other examples of such algorithms in Computational Geometry. For example, linear programming in constant dimension in linear time [Meg84], and its extension to LP-type problems [SW92, MSW96]. Intuitively, LP-type problems include low dimensional convex programming (a standard example is the smallest enclosing ball of a point set in constant dimension). However, surprisingly, such problems also include problems that are not convex in nature – for example, deciding if a set of (axis parallel) rectangles can be pierced by three points is an LP-type problem. Other examples of prune-and-search algorithms that work in linear time include (i) computing an ear in a triangulation of a polygon [EET93], (ii) searching in sorted matrices [FJ84], and (iii) ham-sandwich cuts in two dimensions [LMS94]. Of course, there are many other

examples of using prune and search with running time that is super-linear.

(C) **Grids.** Rabin [Rab76] used randomization, the floor function, and hashing to compute the closest pair of a set of points in the plane, in expected linear time. Golin *et al.* [GRSS95] presented a simplified version of this algorithm, and Smid provides a survey of algorithms on closest pair problems [Smi00]. Of course, the usage of grids and hashing to perform approximate point-location is quite common in practice. By itself, this is already sufficient to break lower bounds in the comparison model, for example for k-center clustering [Har04a]. The only direct extension of Rabin's algorithm the authors are aware of is the work by Har-Peled and Mazumdar [HM05] showing a linear time 2-approximation to the smallest ball containing k points (out of n given points).

There is some skepticism of algorithms using the floor function, since Schönhage [Sch79] showed how to solve a PSPACE complete problem, in polynomial time, using the floor function in the real RAM model – the trick being packing many numbers into a single word (which can be arbitrarily long in the RAM model, and still each operation on it takes only constant time). Note, that as Rabin's algorithm does not do any such packing of numbers (i.e., its computation model is considerably more restricted), this criticism does not seem to be relevant in the case of Rabin's algorithm and its relatives.

In this paper, we present a new technique that combines together all of the above techniques to yield linear time approximation algorithms.

Nets. Given a point set P, an r-net $\mathcal{N}$ of P is a subset of P that represents well the structure of P in resolution r. Formally, we require that for any point in P there is a net point in distance at most r from it, and no two net points are closer than r to each other, see Section 2.1 for a formal definition. Thus, nets provide a sketch of the point-set as far as distances that are r or larger. Nets are a useful tool in presenting point-sets hierarchically. In particular, computing nets of different resolutions and linking between different levels, leads to a tree like data-structure that can be used to facilitate many tasks, see for example the net-tree [KL04, HM06] for such a data-structure for doubling metrics. Nets can be defined in any metric space, but in Euclidean space a grid can sometimes provide an equivalent representation. In particular, net-trees can be interpreted as an extension of (compressed) quadtrees to more abstract settings.

Computing nets is closely related to k-center clustering. Specifically, Gonzalez [Gon85] shows how to compute an approximate net that has k points in $O(nk)$ time, which is also a 2-approximation to the k-center clustering. This was later improved to $O(n)$ time, for low dimensional Euclidean space [Har04a], if k is sufficiently small (using grids and hashing). Har-Peled and Mendel showed how to preprocess a point set in a metric space with constant doubling dimension, in $O(n \log n)$ time, such that an (approximate) r-net can be extracted in (roughly) linear time in the size of the net.

Our contribution.

In this paper, 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 linear time decider to approximate the optimum solution in linear time. As a first step towards a linear time approximation algorithm for such problems, we point out that one can compute nets in linear time in $\mathbb{R}^d$, see Section 2.1.

However, even if we could implicitly search over the super linear number of 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 $O(n \log n)$. 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 new technique. The new 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 two cases.

(A) **Net.** If r is too small then we zoom out to a resolution of r by computing an r-net and continuing the computation on the net instead of on the original point-set. That is, we net 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.

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.

Significance of results. Our basic framework is presented in a general enough manner to cover, and in many cases greatly simplify, many problems for which linear time algorithms had already been discovered. At the same time the framework provides new linear time algorithms for a large collection of problems, for which previously no linear time algorithm was known. The framework should also lead to algorithms for many other problems which are not mentioned.

At a conceptual level the basic algorithm is simple enough (with its basic building blocks already having efficient implementations) to be highly practical from an implementation

standpoint. Perhaps more importantly, with increasing shifts toward large data sets algorithms with super linear running time can be impractical. Additionally, our framework seems amenable to distributed implementation in frameworks like MapReduce. Indeed, every iteration of our algorithm breaks the data into grid cells, a step that is similar to the map phase. In addition, the aggressive thinning of the data by the algorithm guarantees that after the first few iterations the algorithm is resigned to working on only a tiny fraction of the data.

Framework and results. We provide a framework that classifies which optimization problems can be solved using the new algorithm. We get the following new algorithms (all of them have an expected linear running time, for any fixed ε):

(A) **k-center clustering** (Section 4.1). We provide an algorithm that 2-approximates the optimal k-center clustering of a point set in $\mathbb{R}^d$. Unlike the previous algorithm [Har04a] that was restricted to $k = O(n^{1/6})$, the new algorithm works for any value of k. This new algorithm is also simpler.

(B) **kth smallest distance** (Section 4.2). In the distance selection problem, given a set of points in $\mathbb{R}^d$, one would like to compute the kth smallest distance defined by a pair of points of P. It is believed that such exact distance selection requires $\Omega\left(n^{4/3}\right)$ time in the worst case [Eri95], even in the plane (in higher dimensions the bound deteriorates). We present an $O(n/\varepsilon^d)$ time algorithm that $(1+\varepsilon)$-approximates the kth smallest distance. Previously, Bespamyatnikh and Segal [BS02] presented $O(n \log n + n/\varepsilon^d)$ time algorithm using a well-separated pairs decomposition (see also [DHW12]).
Given two sets of points P, W with a total of n points, using the same approach, we can $(1+\varepsilon)$-approximate the kth smallest distance in a bichromatic set of distances $X = \left\{ \mathsf{d}(\mathsf{p},\mathsf{q}) \;\middle|\; \mathsf{p} \in \mathsf{P}, \mathsf{q} \in \mathsf{W} \right\}$, and in particular, we can compute exactly the closest bichromatic pair between P and Q.
Intuitively, the mechanism behind distance selection underlines many optimization problems, as it (essentially) performs a binary search over the distances induced by a set of points. As such, being able to do approximate distance selection in linear time should lead to faster approximation algorithms that perform a similar search over such distances.

(C) **The kth smallest m-nearest neighbor distance** (Section 4.3). For a set $\mathsf{P} = \{\mathsf{p}_1, \ldots, \mathsf{p}_n\} \subseteq \mathbb{R}^d$ of n points, and a point $\mathsf{p} \in \mathsf{P}$, its ***$m$th nearest neighbor*** in P is the mth closest point to p in P. In particular, let $\mathsf{d}_m(\mathsf{p},\mathsf{P})$ denote this distance. Here, consider the set of these distances defined for each point of P; that is, $X = \{\mathsf{d}_m(\mathsf{p}_1,\mathsf{P}), \ldots, \mathsf{d}_m(\mathsf{p}_n,\mathsf{P})\}$. We can approximate the kth smallest number in this set in linear time.

(D) **Furthest nearest neighbor** (Section 4.3.1). As an application of the above, one can compute in linear time, exactly, the furthest nearest neighbor distance; that is, the nearest neighbor distance of the point whose nearest neighbor is furthest away. This measure can be useful, for example, in meshing applications where such a point is a candidate for a region where the local feature size is large and further refinement is needed.
We are unaware of any previous work directly on this problem, although one can compute this quantity by solving the all-nearest-neighbor problem, which can be

done in $O(n \log n)$ time [Cla83]. This is to some extent the "antithesis" to Rabin's algorithm for the closest pair problem, and it is somewhat surprising that it can also be solved in linear time.

(E) **The kth longest MST edge** (Section 4.4). Given a set P of n points in $\mathbb{R}^d$, we can $(1+\varepsilon)$-approximate, in $O(n/\varepsilon^d)$ time, the kth longest edge in the MST of P.

(F) **Smallest ball with a monotone property** (Section 4.5.2). Consider a property defined over a set of points, P, that is monotone; that is, if $\mathsf{W} \subseteq \mathsf{Q} \subseteq \mathsf{P}$ has this property then Q must also have this property. Consider such a property that can be easily be checked, for example, whether the set contains k points, or if the points are colored, that all colors are present in the given point set. Given a point set P, one can $(1+\varepsilon)$-approximate, in $O(n/\varepsilon^d)$ time, the smallest radius ball b, such that $\mathsf{b} \cap \mathsf{P}$ has the desired property. For example, we get a linear time algorithm to approximate the smallest ball enclosing k points of P. The previous algorithm for this problem [HM05] was significantly more complicated. Furthermore, we can approximate the smallest ball such that all colors appear in it (if the points are colored), or the smallest ball such that at least t different colors appear in it, etc.

More generally, the kind of monotone properties supported are *sketchable*; that is, properties for which there is a small summary of a point-set that enables one to decide if the property holds, and furthermore, given summaries of two disjoint point sets, the summary of the union point-set can be computed in constant time. We believe that formalizing this notion of sketchability is a useful abstraction. See Section 4.5.1 for details.

(G) **Smallest connected component with a monotone property** (Section 4.5.3). Consider the connected components of the graph where two points are connected if they are distance at most r from each other. Using our techniques, one can approximate, in linear time, the smallest r such that there is a connected component of this graph for which a required sketchable property holds for the points in this connected component.

As an application, consider ad hoc wireless networks. Here, we have a set P of n nodes and their locations (say in the plane), and each node can broadcast in a certain radius r (the larger the r the higher the energy required, so naturally we would like to minimize it). Assume there are two special nodes. It is natural to ask for the minimum r, such that there is a connected component of the above graph that contains both nodes. That is, these two special nodes node can send message to each other, by message hopping (with distance at most r at each hop). We can approximate this connectivity radius in linear time.

(H) **Clustering for a monotone property** (Section 4.6). Imagine that we want to break the given point-set into clusters, such that the maximum diameter of a cluster is minimized (as in k-center clustering), and furthermore, the points assigned to each cluster comply with some sketchable monotone property. We present a $(4+\varepsilon)$-approximation algorithm for these types of problems, that runs in $O(n/\varepsilon^d)$ time. This includes lower bounded clustering (i.e., every cluster must contain at least α points), for which the authors recently presented an $O(n \log n)$ approximation algorithm [ERH12]. One can get a 2-approximation using network flow, but the running is significantly worse [APF+10]. See Section 4.6.1 for examples of clustering problems

that can be approximated using this algorithm.

(I) **Connectivity clustering for a monotone property** (Section 4.6.3). Consider the problem of computing the minimum r, such that each connected component (of the graph where points in distance at most r from each other are adjacent) has some sketchable monotone property. We approximate the minimum r for which this holds in linear time.
An application of this for ad hoc networks is the following – we have a set P of n wireless clients, and some of them are base stations; that is, they are connected to the outside world. We would like to find the minimum r, such that each connected component of this graph contains a base station.

(J) **Closest pair and smallest non-zero distance.** (Section 4.7). Given a set of points in $\mathbb{R}^d$, consider the problem of finding the smallest non-zero distance defined by these points. This problem is an extension of the closest pair distance, as there might be many identical points in the given point set. We provide a linear time algorithm for computing this distance *exactly*, which follows easily from our framework.

Paper organization. We describe how to compute nets in Section 2, and how to remove faraway points efficiently in Section 2.2. We define the abstract framework, and describe and analyze the new approximation algorithm, in Section 3. We describe the applications in Section 4. We conclude in Section 5.

2 Preliminaries

2.1 Computing nets quickly for a point set in $\mathbb{R}^d$

Definition 2.1. *For a point set P in a metric space with a metric d, and a parameter $r > 0$, an r-net of P is a subset $\mathcal{C} \subseteq \mathsf{P}$, such that (i) for every $\mathsf{p}, \mathsf{q} \in \mathcal{C}$, $\mathsf{p} \neq \mathsf{q}$, we have that $\mathsf{d}(\mathsf{p}, \mathsf{q}) \geq r$, and (ii) for all $\mathsf{p} \in \mathsf{P}$, we have that $\min_{\mathsf{q} \in \mathcal{C}} \mathsf{d}(\mathsf{p}, \mathsf{q}) < r$.*

Intuitively, an r-net represents P in resolution r. There is a simple algorithm for computing r-nets. Namely, let all the points in P be initially unmarked. While there remains an unmarked point, p, add p to $\mathcal{C}$, 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.

The following is implicit in previous work [Har04a], and we include it here for the sake of completeness.[1]

Lemma 2.2. *Given a point set $\mathsf{P} \subseteq \mathbb{R}^d$ of size n and a parameter $r > 0$, one can compute an r-net for P in $O(n)$ time.*

Proof: Let G denote the grid in $\mathbb{R}^d$ with side length $\Delta = r/\left(2\sqrt{d}\right)$. First compute for every point $\mathsf{p} \in \mathsf{P}$ the grid cell in G that contains p; that is, the cell containing p is uniquely

[1] Specifically, the algorithm of Har-Peled [Har04a] is considerably more complicated than Lemma 2.2, and does not work in this settings, as the number of clusters it can handle is limited to $O\left(n^{1/6}\right)$. Lemma 2.2 has no such restriction.

identified by the tuple of integers $\mathrm{id}(\mathsf{p}) = \left(\lfloor \mathsf{p}_1/\Delta \rfloor, \ldots, \lfloor \mathsf{p}_d/\Delta \rfloor\right)$, where $\mathsf{p} = (\mathsf{p}_1, \ldots, \mathsf{p}_d) \in \mathbb{R}^d$. Let $\mathcal{G}$ denote the set of non-empty grid cells of G. Similarly, for every non-empty cell $g \in \mathcal{G}$ 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 performed in constant time, as using hashing we can store a grid cell $\mathrm{id}(\cdot)$ in a hash table and in constant time hash each point into its appropriate bin. For a point $\mathsf{p} \in \mathsf{P}$ let $\mathsf{N}_{\leq r}(\mathsf{p})$ denote the set of grid cells in distance $\leq r$ from p, which is the ***neighborhood*** of p. Observe that $|\mathsf{N}_{\leq r}(\mathsf{p})| = O\Big((2r/(r/2\sqrt{d}) + 1)^d\Big) = O\Big((4\sqrt{d} + 1)^d\Big) = O(1)$.

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 net points, $\mathcal{C}$, and mark it and each point $\mathsf{q} \in \mathsf{P}$ such that $\mathsf{d}(\mathsf{p}, \mathsf{q}) < r$. Since the cells of $\mathsf{N}_{\leq r}(\mathsf{p})$ 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 net point, and a net point is only created if it is unmarked when visited, this implies that $\mathcal{C}$ is an r-net.

For the running time, observe that a grid cell, c, has its list scanned only if c is in the neighborhood of some created net point. From the discussion above we know that there are $O(1)$ cells which could contain a net point p such that $c \in \mathsf{N}_{\leq r}(\mathsf{p})$. Also, we create at most one net point per cell since the diameter of a grid cell is strictly smaller than r. Therefore c 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 net point, for a point $\mathsf{p} \in \mathsf{P}$, must be in one of its neighborhood grid cells. Since every grid cell can contain only a single net point, it follows that in constant time per point of P, one can compute its nearest net point. We thus have the following.

Corollary 2.3. *In $O(n)$ time one can not only compute an r-net, but also compute for each net point the set of points of P for which it is the nearest net point.*

This implies that for a weighted point set one can compute the following quantity in linear time.

Algorithm 2.4. *Given a weighted point set P, let $\mathbf{Net}(r, \mathsf{P})$ denote an r-net of P, where the weight of each net point p is the total sum of the weights of the points assigned to it. The running time of this algorithm is $O(|\mathsf{P}|)$.*

2.2 Notation and point removal

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 $\omega(S) = \sum_{\mathsf{p} \in S} \omega(\mathsf{p})$ denote the total weight of S.

For a given point $\mathsf{p} \in \mathsf{P}$ let $\mathsf{d}(\mathsf{p}, \mathsf{P})$ denote the distance of p to its nearest neighbor in $\mathsf{P} \setminus \{\mathsf{p}\}$, which can be computed naively in linear time by scanning the points. For a set of points P, and a parameter r, let $\mathsf{P}^{\geq r}$ denote the set of ***r-far*** points; that is, it is the set of all points in $\mathsf{p} \in \mathsf{P}$, such that their nearest-neighbor in P is more than distance r away (i.e., $\mathsf{d}(\mathsf{p}, \mathsf{P}) \geq r$). Similarly, $\mathsf{P}^{<r}$ is the set of ***r-close*** points; that is, all points $\mathsf{p} \in \mathsf{P}$, such that $\mathsf{d}(\mathsf{p}, \mathsf{P}) < r$.

Lemma 2.5 (algDelFar(ℓ, P)). *Given a weighted set $\mathsf{P} \subseteq \mathbb{R}^d$ of n points, and a distance $\ell > 0$, in $O(n)$ time, one can compute the sets $\mathsf{P}^{<r}$ and $\mathsf{P}^{\geq r}$.*

Proof: Build a grid where every cell has diameter ℓ/c, for some constant $c > 1$. Clearly any point $\mathsf{p} \in \mathsf{P}$ such that $\mathsf{d}(\mathsf{p}, \mathsf{P}) \geq \ell$ (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 $\mathsf{d}(\mathsf{q}, \mathsf{P}) \geq \ell$, one checks the points stored in all grid cells in distance $\leq \ell$ from it. If q has no such close neighbor, we mark q for inclusion in $\mathsf{P}^{\geq r}$. By the same arguments as in Lemma 2.2 the number of such cells is $O(c^2) = O(1)$. Again by the arguments of Lemma 2.2 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 $\mathsf{P}^{\geq r}$ (resp. $\mathsf{P}^{<r}$). ■

3 Approximating nice distance problems

3.1 Problem definition and an example

Definition 3.1. *Let P and Q be two sets of weighted points in $\mathbb{R}^d$ (of the same weight). The set Q is a* **Δ-translation** *of P, if Q can be built by moving each point of P by distance at most Δ. Formally, the earth mover distance between P and Q is at most Δ.*

Note that for a (potentially weighted) point set W, **Net**(Δ, W) is a Δ-translation of W.

Definition 3.2. *Given a function $f : X \to \mathbb{R}$, we call a procedure,* **decider**, *a* **t-decider** *for f, if for any $x \in X$ and $0 < r \in \mathbb{R}$,* **decider**(r, x) *returns one of the following: (i) $f(x) \in [\alpha, t\alpha]$, where α is some real number, (ii) $f(x) < r$, or (iii) $f(x) > r$.*

Definition 3.3 (NDP). *A pair (W, Γ) is an instance of a* **t-NiceDistanceProblem**, *where $\mathsf{W} \subseteq \mathbb{R}^d$ is a set of n distinct weighted points[2], and Γ is the* **context** *of the given instance (of size $O(n)$) and consists of the relevant parameters for the problem. The task is evaluating a function $f(\mathsf{W}, \Gamma) \to \mathbb{R}^+$, associated with this pair, that has the following properties:*

- *(P1) There exists an $O(n)$ time t-decider for f, for some* constant *t (see Definition 3.2).*
- *(P2) (Lipschitz.) Let Q be a Δ-translation of W. Then $|f(\mathsf{W}, \Gamma) - f(\mathsf{Q}, \Gamma)| \leq 2\Delta$.*
- *(P3) If $f(\mathsf{W}, \Gamma) < r$ then $f(\mathsf{W}, \Gamma) = f(\mathsf{W}^{<r}, \Gamma')$, where $\mathsf{W}^{<r}$ is the set of r-close points, and Γ' is an updated context which can be computed in $O(n)$ time.*

3.1.1 An example – k center clustering

As a concrete example of an NDP, lets consider the problem of k-center clustering. Formally, we have the following.

Problem 3.4 (kCenter). *Let W be a set of points in $\mathbb{R}^d$, and let $k > 0$ be an integer parameter. Find a set of* **centers** *$C \subseteq \mathsf{W}$ such that the maximum distance of a point of W to its nearest center in C is minimized, and $|C| = k$.*

[2]The case when W is a multiset can also be handled. See Remark 3.8.

Namely, the function of interest, $f_{cen}(\mathsf{W}, k)$ is the radius of the optimal k-center clustering of W. We now show that kCenter satisfies the properties of an NDP.

Lemma 3.5. *The following relations between* $|\mathbf{Net}(r, \mathsf{W})|$ *and* $f_{cen}(\mathsf{W}, k)$ *hold: (A) If we have* $|\mathbf{Net}(r, \mathsf{W})| \leq k$ *then* $f_{cen}(\mathsf{W}, k) < r$. *(B) If* $|\mathbf{Net}(r, \mathsf{W})| > k$ *then* $r < 2f_{cen}(\mathsf{W}, k)$.

Proof: (A) Observe that if $|\mathbf{Net}(r, \mathsf{W})| \leq k$, then, by definition, the set of net points of $\mathbf{Net}(r, \mathsf{W})$ are a set of $\leq k$ points such that all the points of W are in distance at most r from these centers.

(B) If $|\mathbf{Net}(r, \mathsf{W})| > k$ then W must contain a set of $k+1$ points whose pairwise distances are all at least r. In particular any solution to kCenter with radius $< r/2$ would not be a able to cover all these $k + 1$ points with only k centers. ■

Lemma 3.6. *An instance* (W, k) *of kCenter satisfies the properties of Definition 3.3, that is kCenter is an NDP.*

Proof: $(\mathrm{P1})_{\mathrm{p9}}$: We need to describe a decision procedure for kCenter clustering. To this end, given a distance r, the decider first calls $\mathbf{Net}(r, \mathsf{W})$. If we have $|\mathbf{Net}(r, \mathsf{W})| \leq k$, then by Lemma 3.5 the answer "$f_{cen}(\mathsf{W}, k) < r$" can be returned. Otherwise, call $\mathbf{Net}(2r, \mathsf{W})$. If $|\mathbf{Net}(2r, \mathsf{W})| \leq k$, then, by Lemma 3.5, we have $r/2 < f_{cen}(\mathsf{W}, k) < 2r$ and the interval $[r/2, 2r]$ can be returned by the decider. Otherwise $|\mathbf{Net}(2r, \mathsf{W})| > k$, and Lemma 3.5 implies that the answer "$r < f_{cen}(\mathsf{W}, k)$" can be returned by the decider.

(P2): Observe that if Q is a Δ-translation of W then each point, and its respective center, each move by distance at most Δ in the transition from W to Q. As such, the distance between a point and its center changes by at most 2Δ by this process. This argument also works in the other direction, implying that the k-center clustering radius of W and Q are the same, up to an additive error of 2Δ.

(P3): It suffices to show that if $f_{cen}(\mathsf{W}, k) < r$ then $f_{cen}(\mathsf{W}, k) = f_{cen}\left(\mathsf{W}^{<r}, k - \left|\mathsf{W}^{\geq r}\right|\right)$. Now, if $f_{cen}(\mathsf{W}, k) < r$ then any point of W whose neighbors are all $\geq r$ away must be a center by itself in the optimal k-center solution, as otherwise it would be assigned to a center $\geq r > f_{cen}(\mathsf{W}, k)$ away. Similarly, any point assigned to it would be assigned to a point $\geq r > f_{cen}(\mathsf{W}, k)$ away. Therefore, for any point $\mathsf{p} \in \mathsf{W}$ with no neighbor in distance $< r$, $f_{cen}(\mathsf{W}, k) = f_{cen}(\mathsf{W} \setminus \{\mathsf{p}\}, k - 1)$, repeating this observation implies the desired result. The context update (and computing $\mathsf{W}^{<r}$ and $\mathsf{W}^{\geq r}$) can by done in linear time using **algDelFar**. ■

3.2 A linear time approximation algorithm

We now describe the general algorithm which given an NDP, (W, Γ), and an associated target function f, computes, in linear time, a bounded spread interval containing $f(\mathsf{W}, \Gamma)$. In the following, let **decider** denote the given t-decider, and **contextUpdate** denote the context updater associated with the given NDP problem, see Definition 3.3. Both **decider** and **contextUpdate** run in linear time. The algorithm for bounding the optimum value of an NDP is shown in Figure 3.1.

ndpAlg(W, Γ)

1: Let $\mathsf{W}_0 = \mathsf{W}$, $\Gamma_0 = \Gamma$ and $i = 1$.
   // α is a constant, see Corollary 3.14 for its value.
2: **while** TRUE **do**
3: Randomly pick a point p from W_{i-1}.
4: $\ell_i \leftarrow \mathsf{d}(\mathsf{p}, \mathsf{W}_{i-1})$.
   // Use **decider** to estimate the value of $f_{i-1} = f(\mathsf{W}_{i-1}, \Gamma_{i-1})$
5: $res_> = \textbf{decider}(\ell_i, \mathsf{W}_{i-1}, \Gamma_{i-1})$
6: $res_< = \textbf{decider}(\alpha\ell_i, \mathsf{W}_{i-1}, \Gamma_{i-1})$
7: **if** $res_>$ = "$f_{i-1} \in [x, y]$" **then return** "$f(\mathsf{W}, \Gamma) \in [x/2, 2y]$".
8: **if** $res_<$ = "$f_{i-1} \in [x, y]$" **then return** "$f(\mathsf{W}, \Gamma) \in [x/2, 2y]$".
9: **if** $res_>$ = "$\ell_i < f_{i-1}$" and $res_<$ = "$f_{i-1} < \alpha\ell_i$" **then**
10: **return** $[\ell_i/2, 2\alpha\ell_i]$
   // Is the guess (i.e., ℓ_i) larger than opt?
11: **if** $res_>$ = "$\ell_i > f_{i-1}$" **then**
12: $\mathsf{W}_i \leftarrow \mathsf{W}_{i-1}^{<\ell_i}$ // computed by calling **algDelFar**$(\ell_i, \mathsf{W}_{i-1})$
13: $\Gamma_i = \textbf{contextUpdate}(\mathsf{W}_{i-1}, \Gamma_{i-1}, \mathsf{W}_i)$
   // Is the guess (i.e., ℓ_i) smaller than opt?
14: **if** $res_<$ = "$\alpha\ell_i < f_{i-1}$" **then**
15: $\mathsf{W}_i = \textbf{Net}(3\ell_i, \mathsf{W}_{i-1})$
16: $\Gamma_i = \textbf{contextUpdate}(\mathsf{W}_{i-1}, \Gamma_{i-1}, \mathsf{W}_i)$
17: $i = i + 1$

Figure 3.1: The approximation algorithm. We have an implicit target value $f = f(\mathsf{W}, \Gamma)$ that we are trying to approximate, where f is an NDP. Our only access to the function f is via the **decider** procedure.

Remark 3.7. For simplicity of exposition we assume that $f(\mathsf{W}, \Gamma) \neq 0$. The case when $f(\mathsf{W}, \Gamma) = 0$ can be handled with an additional check of the context in the algorithm. However, since all our applications have $f(\mathsf{W}, \Gamma) \neq 0$ we choose to make this simplifying assumption. Also note that if $f(\mathsf{W}, \Gamma) \neq 0$ then for each iteration i of while loop in **ndpAlg**, $f(\mathsf{W}_{i-1}, \Gamma_{i-1}) \neq 0$.

Remark 3.8. Note that the algorithm of Figure 3.1 can be modified to handle inputs where W is a multiset (namely, two points can occupy the same location). Specifically, it must be ensured that the distance computed in Line 4 is not zero, as this is required for the call to **decider**. This can be remedied (in linear time) by first grouping all the duplicates of the point sampled in Line 3 into a single weighted point (with the sum of the weights) before calling Line 4.

3.2.1 Analysis

A ***net iteration*** of the algorithm is an iteration where **Net** gets called. A ***prune iteration*** is one where **algDelFar** gets called. Note that the only other type of iteration is the one

where the algorithm returns.

Lemma 3.9. *Let* P *be a point set. A* $(2+\varepsilon)\ell$*-net of* P*, for any* $\varepsilon > 0$*, can contain at most half the points of* $\mathsf{P}^{\leq \ell}$*.*

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

Lemma 3.10. *Given an instance* (W, Γ) *of an NDP, the algorithm* **ndpAlg**(W, Γ) *runs in expected* $O(n)$ *time.*

Proof: In each iteration of the while loop the only non-trivial work done is in computing ℓ_i, the two calls to **decider**, and the one call to either **Net** or **algDelFar**. It has already been shown that all of these can be computed in $O(|\mathsf{W}_{i-1}|)$ time. Hence the total running time for the algorithm is $O\left(\sum_{i=0}^{i=k-1} |\mathsf{W}_i|\right)$, where k denotes the last (incomplete) iteration of the while loop.

So consider the beginning of iteration $i < k$ of the while loop. Let the points in W_{i-1} be labeled $\mathsf{p}_1, \mathsf{p}_2, \ldots, \mathsf{p}_m$ in increasing order of their nearest neighbor distance in W_{i-1}. Let j be the index of the point chosen in Line 3 and let $(\mathsf{W}_{i-1})^{\geq j}$ and $(\mathsf{W}_{i-1})^{\leq j}$ be the subset of the points with index $\geq j$ and index $\leq j$, respectively. Now since a point is randomly picked in Line 3, with probability $\geq 1/2$, $j \in [m/4, 3m/4]$. Lets call this event a ***successful iteration***. We have $\min\left(\left|(\mathsf{W}_{i-1})^{\geq j}\right|, \left|(\mathsf{W}_{i-1})^{\leq j}\right|\right) \geq |\mathsf{W}_{i-1}|/4$ for a successful iteration.

Since $i < k$ is not the last iteration of the while loop, either **algDelFar** or **Net** must get called. If **algDelFar**$(\ell_i, \mathsf{W}_{i-1})$ gets called (i.e. Line 12) then by Lemma 2.5, all of $(\mathsf{W}_{i-1})^{\geq j}$ gets removed. So suppose **Net** gets called (i.e. Line 15). In this case Lemma 3.9 implies that the call to **Net** removes at least $\left|(\mathsf{W}_{i-1})^{\leq j}\right|/2$ points.

Therefore, for any iteration $i < k$, at least $\nu_i = \min(\left|(\mathsf{W}_{i-1})^{\geq j}\right|, \left|(\mathsf{W}_{i-1})^{\leq j}\right|/2)$ points get removed. If an iteration is successful then $\nu_i \geq |\mathsf{W}_{i-1}|/8$. In particular, $\mathbf{E}\left[\nu_i \;\middle|\; |\mathsf{W}_{i-1}|\right] \geq |\mathsf{W}_{i-1}|/16$. Now, $|\mathsf{W}_i| \leq |\mathsf{W}_{i-1}| - \nu_i$ and as such $\mathbf{E}\left[|\mathsf{W}_i| \;\middle|\; |\mathsf{W}_{i-1}|\right] \leq (15/16)\,|\mathsf{W}_{i-1}|$.

Therefore, for $0 < i < k$,

$$\mathbf{E}\Big[|\mathsf{W}_i|\Big] = \mathbf{E}\Big[\mathbf{E}\Big[|\mathsf{W}_i| \;\Big|\; \mathsf{W}_{i-1}\Big]\Big] \leq \mathbf{E}\left[\frac{15}{16}\,|\mathsf{W}_{i-1}|\right] = \frac{15}{16}\,\mathbf{E}\Big[|\mathsf{W}_{i-1}|\Big].$$

Hence by induction on i, $\mathbf{E}[|\mathsf{W}_i|] \leq (15/16)^i\,|\mathsf{W}_0|$ and so, in expectation, the running time is bounded by

$$O\left(\mathbf{E}\left[\sum_{i=0}^{i=k-1} |\mathsf{W}_i|\right]\right) = O\left(\sum_{i=0}^{i=k-1} \mathbf{E}\Big[|\mathsf{W}_i|\Big]\right) = O\left(\sum_{i=0}^{i=k-1} (15/16)^i\,|\mathsf{W}_0|\right) = O(|\mathsf{W}_0|).$$

∎

Correctness The formal proof of correctness is somewhat tedious, but here is the basic idea: At every iteration, either far points are being thrown away (and this does not effect the optimal value), or we net the points. However, the net radius being used is significantly smaller than the optimal value, and throughout the algorithm execution the radii of the nets being used grow exponentially. As such, the accumulated error in the end is only a fraction of the target value, and it is thus being approximated correctly.

Before proving that **ndpAlg** returns a bounded spread interval containing $f(\mathsf{W}_0, \Gamma_0)$, several helper lemmas will be needed. For notational ease, in the rest of this section, we use $f(\mathsf{W}_i)$ as shorthand for $f(\mathsf{W}_i, \Gamma_i)$.

Lemma 3.11. *Suppose that* **Net** *is called (i.e. Line 15) in iteration i of the while loop. Then for any iteration $j > i$ we have, $\ell_j \geq 3\ell_i$.*

Proof: Consider the beginning of iteration j of the while loop. The current set of points, W_{j-1}, are a subset of the net points of a $3\ell_i$-net (it is a subset since Line 12 and Line 15 might have been executed in between rounds i and j). Therefore, being a net, the distance between any two points of W_{j-1} is $\geq 3\ell_i$, see Definition 2.1. In particular, this means that for any point p of W_{j-1}, we have $\mathsf{d}(\mathsf{p}, \mathsf{W}_{j-1}) \geq 3\ell_i$. ∎

Lemma 3.12. *For $i = 1, \ldots, k$, we have $|f(\mathsf{W}_i) - f(\mathsf{W}_0)| \leq 9\ell_i$.*

Proof: Consider an iteration i of the while loop. If **algDelFar** (Line 12) gets called then by (P3) of Definition 3.3_{p9} we have that $f(\mathsf{W}_i) = f(\mathsf{W}_{i-1})$. One can apply (P3) here since **algDelFar** only gets called if $f(\mathsf{W}_{i-1}) < \ell_i$ and moreover **algDelFar**$(\ell_i, \mathsf{W}_{i-1})$ preserves all points such that $\mathsf{d}(\mathsf{p}, \mathsf{W}_{i-1}) < \ell_i$. As such, the claim holds by induction.

So now suppose that **Net** (i.e Line 15) is called in iteration i, and let I be the set of indices of the net iterations up to (and including) the ith iteration. Then W_i is a $3\ell_i$-translation of W_{i-1} and so by (P2), $|f(\mathsf{W}_i) - f(\mathsf{W}_{i-1})| \leq 6\ell_i$. Therefore, $|f(\mathsf{W}_i) - f(\mathsf{W}_0)| \leq \sum_{j \in I} 6\ell_j \leq 9\ell_i$, as this summation behaves like a geometric series by Lemma 3.11. ∎

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

Lemma 3.13. *For any iteration i of the while loop such that* **Net** *gets called we have $\ell_i \leq f(\mathsf{W}_0)/\eta$, where $0 < \eta = \alpha - 9$.*

Proof: The proof will be by induction. Let $m_1, \ldots m_t$ be the indices of the iterations of the while loop in which **Net** gets called. For the base case, observe that in order for **Net** to get called $\eta\ell_{m_1} < \alpha\ell_{m_1} < f(\mathsf{W}_{m_1-1})$. However, since this is the first iteration in which **Net** is called it must be that $f(\mathsf{W}_0) = f(\mathsf{W}_{m_1-1})$ (since for all previous iterations **algDelFar** must have been called).

So now suppose that $\ell_{m_j} \leq f(\mathsf{W}_0)/\eta$ for all $m_j < m_i$. If a call to **Net** is made in iteration m_i then again $\alpha\ell_{m_i} < f(\mathsf{W}_{(m_i)-1}) = f(\mathsf{W}_{m_{(i-1)}})$. Thus, by Lemma 3.12 and induction, we have

$$\ell_{m_i} < \frac{f\left(\mathsf{W}_{m_{(i-1)}}\right)}{\alpha} \leq \frac{f(\mathsf{W}_0) + 9\ell_{m_{(i-1)}}}{\alpha} \leq \frac{f(\mathsf{W}_0) + 9f(\mathsf{W}_0)/\eta}{\alpha} = \frac{1 + 9/\eta}{\alpha} f(\mathsf{W}_0) = \frac{f(\mathsf{W}_0)}{\eta},$$

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

Setting $\alpha = 37$, results in $\eta = 28$, and by Lemma 3.13, for all i that correspond to a net iteration, $\ell_i \leq f(\mathsf{W}_0)/28$. By Lemma 3.12, for any net iteration i, we have $|f(\mathsf{W}_i) - f(\mathsf{W}_0)| \leq 9\ell_i \leq f(\mathsf{W}_0)/3$. In particular, we conclude that $|f(\mathsf{W}_i) - f(\mathsf{W}_0)| \leq f(\mathsf{W}_0)/3$ for any iteration i. We thus get the following.

Corollary 3.14. *For $\alpha \geq 37$, and any i, we have $(2/3)f(\mathsf{W}_0) \leq f(\mathsf{W}_i) \leq (4/3)f(\mathsf{W}_0)$.*

In particular, if $f(\mathsf{W}_i) \in [x, y]$ then $f(\mathsf{W}_0) \in [(3/4)x, (3/2)y] \subseteq [x/2, 2y]$.

Lemma 3.15. *For $\alpha \geq 37$, given an instance (W, Γ) of an NDP, **ndpAlg**(W, Γ) returns a bounded spread interval containing $f(\mathsf{W}, \Gamma)$.*

Proof: Consider the iteration of the while loop at which **ndpAlg** terminates. If Line 7 or Line 8 get executed at this iteration, then the correct interval is returned, by Corollary 3.14, and it is a bounded interval. The same argumentation holds if Line 10 gets executed. ■

3.3 The result

Theorem 3.16. *For a constant $t > 1$, given an instance of a t-NDP defined by a set of n points in $\mathbb{R}^d$, one can get a $\max\Big((1+\varepsilon), t\Big)$-approximation to its optimal value, in expected $O(n \log(1/\varepsilon))$ time.*

Proof: Let (W, Γ) be the given t-NDP instance, and let **decider** be the corresponding t-decider. By Lemma 3.10 and Lemma 3.15, in expected $O(n)$ time, one can get a bounded spread interval $[\gamma, c\gamma]$, for some $c = O(t) = O(1)$, such that $f = f(\mathsf{W}, \Gamma) \in [\gamma, c\gamma]$. Now preform a binary search over this interval.

Specifically, for $i = 0, 1, \ldots, m = \lfloor (c-1)/\varepsilon \rfloor$, let $\gamma_i = (1 + i\varepsilon)\gamma$ and let $\gamma_{m+1} = c\gamma$. Now perform a binary search over the γ_i's using **decider**. When a specific γ_i is queried, if **decider** returns an interval $[x, tx]$ then $f \leq tx \leq tf$ and so tx can be returned as a t-approximation. Otherwise, if $f < \gamma_i$ or $f > \gamma_i$ the binary search moves left or right, respectively. In the end we get an interval $(\gamma_i, \gamma_{i+1}) = ((1 + i\varepsilon)\gamma, (1 + i\varepsilon)\gamma + \varepsilon\gamma)$ which contains f. Specifically, $f \leq (1 + i\varepsilon)\gamma + \varepsilon\gamma \leq f + \varepsilon\gamma \leq (1 + \varepsilon)f$ and so $(1 + i\varepsilon)\gamma + \varepsilon\gamma$ is a $(1 + \varepsilon)$-approximation. ■

Theorem 3.17. *Given an instance of a $(1 + \varepsilon)$-NDP defined by a set of n points in $\mathbb{R}^d$, such that for any $\varepsilon > 0$ we have a $(1 + \varepsilon)$-decider with running time $O(n/\varepsilon^c)$, then one can $(1 + \varepsilon)$-approximate the optimal value for this NDP, in expected $O(n/\varepsilon^c)$ time.*

Proof: Theorem 3.16 directly implies that one can get a $(1 + \varepsilon)$-approximation in expected $O(n \log(1/\varepsilon)/\varepsilon^c)$ time. By using the same procedure as in the proof of Theorem 3.16 but with exponentially decreasing values for ε in the binary search, a factor of $\log(1/\varepsilon)$ can be avoided. This is a standard idea and was also used by Aronov and Har-Peled [AH08].

Let **decider**$_\varepsilon$ be our $(1+\varepsilon)$-decider. If ε is set to any constant (say 1), then by Lemma 3.10 and Lemma 3.15 in expected $O(n)$ time one can get a bounded spread interval $[\gamma, c\gamma]$, for some $c = O(t)$, such that $f = f(\mathsf{W}, \Gamma) \in \mathcal{I}_0 = [\gamma, c\gamma]$.

Set $\varepsilon_0 = c$ and $i = 1$. The algorithm now proceeds in rounds:

(A) Assume that in the beginning of the ith iteration, we know that

$$f \in \mathcal{I}_{i-1} = \Big[x_{i-1}, y_{i-1}\Big], \qquad \text{where} \qquad y_i = (1+\varepsilon_{i-1})\, x_{i-1}.$$

If $\varepsilon_{i-1} \leq \varepsilon$, then we found the desired approximation, and the algorithm stops. Otherwise, round i performs the following steps.

(B) Set $\varepsilon_i = \sqrt{1+\varepsilon_{i-1}} - 1$ and $m_i = (1+\varepsilon_i)\, x_{i-1}$.

(C) $R_i \leftarrow \mathbf{decider}_{\varepsilon_i}(m_i, \mathsf{W}, \Gamma)$, see Definition 3.2$_{\text{p9}}$.

(D) There are three possibilities:

(i) If $R_i =$ "$f < m_i$", then set $\mathcal{I}_i \leftarrow [x_{i-1}, m_i]$.

(ii) If $R_i =$ "$f > m_i$", then set $\mathcal{I}_i \leftarrow [m_i, y_i]$.

(iii) If $R_i =$ "$f \in \mathcal{I} = [x, (1+\varepsilon_i)x]$", then set $\mathcal{I}_i \leftarrow \mathcal{I}$.

(E) $i \leftarrow i+1$.

In each round, the algorithm computes an interval of spread $y_i/x_i = 1+\varepsilon_i$ that contains f, and

$$\varepsilon_i = \sqrt{1+\varepsilon_{i-1}} - 1 = \frac{1+\varepsilon_{i-1}-1}{\sqrt{1+\varepsilon_{i-1}}+1} \leq \frac{\varepsilon_{i-1}}{2}.$$

Since the main bottleneck in each iteration is calling $\mathbf{decider}_{\varepsilon_i}$, which runs in $O(n/\varepsilon_i^c)$ time, the total running time is bounded by,

$$\sum_{i=1} O(n/\varepsilon_i^c) = O(n/\varepsilon^c),$$

since the sum behaves like a geometric series and the last term is $O(n/\varepsilon^c)$. ∎

4 Applications – let me count the ways

We now show that Theorem 3.16 and Theorem 3.17 can be applied to a wide array of problems. In each case in order to apply the theorems it must first be shown that the given problem meets the requirements of an NDP, as was done for kCenter in Section 3.1.1.

4.1 k-center clustering

Since computing a k-center clustering is an NDP problem (Lemma 3.6), plugging this into Theorem 3.16, immediately yields a constant factor approximation to k-center in linear time. It is easy to convert such an approximation to a 2-approximation using a grid, see Har-Peled [Har04a]. Thus we get the following.

Theorem 4.1. *Given a set P of n points in $\mathbb{R}^d$, and a parameter k, $1 \leq k \leq n$, one can compute a 2-approximation to the optimal k-center clustering of P in (expected) linear time.*

A result similar to Theorem 4.1 was already known for the case $k = O\big(n^{1/3}/\log n\big)$ [Har04a], and the above removes this restriction. In addition, the new algorithm is both simpler and its analysis is simpler than the algorithm of Har-Peled [Har04a].

4.2 The kth smallest distance

Lemma 4.2. *Let P be a weighted point set in $\mathbb{R}^d$, and let $k > 0$ be an integer parameter. Let $\binom{\mathsf{P}}{2}$ denote the multi-set of pairwise distances determined by P.*[3] *Given an instance (P, k) the k**thDistance** *problem asks you to output the kth smallest distance in $\binom{\mathsf{P}}{2}$. Given such an instance, one can $(1+\varepsilon)$-approximate the kth smallest distance in $O(n/\varepsilon^d)$ time.*

Proof: Let $f(\mathsf{P}, k)$ be the function that returns the kth smallest distance. We prove that this function is an NDP (see Definition 3.3). For the decision procedure, given r, build a grid where every cell has diameter $\Delta = \varepsilon r/8$, and store the points of P in this grid. Now, for any non-empty grid cell c, let $\omega(c)$ be the total weight of points in $c \cap \mathsf{P}$, and register this weight with all the grid cells in distance at most r from it (i.e., $\mathsf{N}_{\leq r}(c)$). Let $\omega_{\mathsf{N}}(\square)$ denote the total weight registered with a cell $\square$ (which includes its own weight). Any point in $\square$ determines $\omega_{\mathsf{N}}(\square) - 1$ distances to other points in $\mathsf{N}_{\leq r}(\square)$. Therefore the total number of distances between points in $\square$ and points in $\mathsf{N}_{\leq r}(\square)$ is $\omega(\square)\Big(\omega_{\mathsf{N}}(\square) - 1\Big)$. Summing this over all cells (and dividing by 2 to avoid double counting) gives the quantity

$$W = \sum_{\square, \omega(\square) \neq 0} \frac{\omega(\square)}{2}\Big(\omega_{\mathsf{N}}(\square) - 1\Big).$$

Note that W counts all distances which are $\leq r$ and only distances which are $\leq r + 2\Delta$. So let the desired kth distance be denoted by ℓ_k. Then if $k \leq W$ then $\ell_k \leq r + 2\Delta = (1+\varepsilon/4)r$. Similarly, if $k > W$ then $\ell_k > r$. Therefore, to build a $(1+\varepsilon)$-decider for distance r, we run the above counting procedure on $r_1 = r/(1+\varepsilon/3)$ and $r_2 = r$. If the r_1 distance call returns that $\ell_k \leq (1+\varepsilon/4)r/(1+\varepsilon/3) < r$, then we return this result. Otherwise, if the r_2 call returns that $\ell_k \leq (1+\varepsilon/4)r_2 = (1+\varepsilon/4)r$, then $\ell_k \in [r/(1+\varepsilon/3), (1+\varepsilon/4)r]$, and we are done as this interval has spread $(1+\varepsilon/4)(1+\varepsilon/3) < 1+\varepsilon$. The only remaining possibility is that $\ell_k > r_2$, in which case we return that $\ell_k > r$.

Since the distance between any pair of points changes by at most 2Δ in a Δ-translation, the Lipschitz condition (i.e. $(\mathrm{P2})_{\mathrm{p9}}$) holds by Claim 4.3.

As for $(\mathrm{P3})_{\mathrm{p9}}$, by assumption, the kth smallest distance is determined by two distinct weighted points p and q. Clearly these points are not in $\mathsf{W}^{\geq r}$ since $\mathsf{d}(\mathsf{p}, \mathsf{q}) = f(\mathsf{W}, k) < r$. So consider any point $\mathsf{s} \in \mathsf{W}^{\geq r}$. Since removing s by does not remove the distance $\mathsf{d}(\mathsf{p}, \mathsf{q})$ from set of remaining distances, all is needed is to show up to update k. Clearly, s contributes $\omega(\mathsf{s}) \times \omega(\mathsf{P} \setminus \{\mathsf{s}\})$ distances $\geq r$ to $\binom{\mathsf{W}}{2}$, and $\binom{\omega(\mathsf{s})}{2}$ pairwise distances of value zero. Thus, after removing s from W, the new context is $k - \binom{\omega(\mathsf{s})}{2}$. ∎

Claim 4.3. *Let S and S' be subsets of $\mathbb{R}$ of size n, such that S' is obtained by taking each value in S and incrementing or decrementing it by less than Δ. Let v and v' be the kth smallest values in S and S', respectively. Then $|v - v'| \leq \Delta$.*

Proof: Suppose for contradiction that $|v - v'| > \Delta$. If $v' - v > \Delta$ then S' has at least $n - k + 1$ values strictly larger than $v + \Delta$ which implies S has at least $n - k + 1$ values

[3] In particular a point of weight m is viewed as m unit weight points when determining the values of $\binom{\mathsf{P}}{2}$. For simplicity we assume that the kth distance in P is not zero (i.e. it is determined by two distinct points).

strictly larger than v. Similarly if $v - v' > \Delta$ then S' has at least k values strictly smaller than $v - \Delta$ which implies S has at least k values strictly smaller than v. ■

The algorithm of Lemma 4.2 also works (with minor modifications) if we are interested in the kth distance between two sets of points (i.e., the bipartite version).

Corollary 4.4. *Given two sets* P *and* Q *of points in* $\mathbb{R}^d$*, of total size* n*, and parameters* k *and* $\varepsilon > 0$*, one can* $(1+\varepsilon)$*-approximate, in* $O(n/\varepsilon^d)$ *time, the following:*

(A) The k*th smallest distance in the bichromatic multiset of distances* $X = \left\{ \mathsf{d}(\mathsf{p},\mathsf{q}) \;\middle|\; \mathsf{p} \in \mathsf{P}, \mathsf{q} \in \mathsf{W} \right\}$.

(B) The closest bichromatic pair between P *and* Q.

4.3 The kth smallest m-nearest neighbor distance

For a set $\mathsf{P} = \{\mathsf{p}_1, \ldots, \mathsf{p}_n\} \subseteq \mathbb{R}^d$ of n points, and a point $\mathsf{p} \in \mathsf{P}$, its ***$m$th nearest neighbor*** in P is the mth closest point to p in P. In particular, let $\mathsf{d}_m(\mathsf{p},\mathsf{P})$ denote this distance. Here, consider the set of these distances defined for each point of P; that is, $X = \{\mathsf{d}_m(\mathsf{p}_1,\mathsf{P}), \ldots, \mathsf{d}_m(\mathsf{p}_n,\mathsf{P})\}$. For $m = 2$, the minimum number in X is the closet pair distance in P, which we already showed can be computed (exactly) in linear time using our framework. Interestingly, we can approximate the kth smallest number in this set in linear time.

As usual in order to use **ndpAlg** we must generalize the problem to handle positive integer weighted point sets. Namely, a point p of weight $\omega(\mathsf{p})$ will be treated as $\omega(\mathsf{p})$ distinct points at the same location. In particular the set X from the above is actually a multiset containing $\omega(\mathsf{p})$ copies of the value $\mathsf{d}_m(\mathsf{p},\mathsf{P})$.

Theorem 4.5. *Let* W *be a set of* n *weighted points in* $\mathbb{R}^d$*, and* m, k, ε *parameters. Then one can* $(1+\varepsilon)$*-approximate, in* $O(n/\varepsilon^d)$ *time, the* k*th smallest* m*-nearest neighbor distance in* W*. Formally, the algorithm* $(1+\varepsilon)$*-approximates the* k*th smallest number in the multiset* $X = \{\mathsf{d}_m(\mathsf{p}_1,\mathsf{W}), \ldots, \mathsf{d}_m(\mathsf{p}_n,\mathsf{W})\}$

Proof: Let $f(\mathsf{W}, k, m)$ denote the desired quantity. We first need a $(1+\varepsilon)$-decider for this problem. To this end, given r, k, m, ε and W as input, create a grid with cells of diameter $\varepsilon r/4$, and mark for each point of W all the grid cells in distance at most r from it. Each non-empty grid cell has a count of the total weight of the points in distance at most r from it. Thus, each point of W can compute the total weight of all points which are approximately in distance at most r from it. Now, a point $\mathsf{p} \in \mathsf{W}$, can decide in constant time if (approximately) $\mathsf{d}_m(\mathsf{p},\mathsf{W}) \leq r$. If the number of points which declare $\mathsf{d}_m(\mathsf{p},\mathsf{W}) \leq r$ (where a point p is counted $\omega(\mathsf{p})$ times) is greater than k then the distance r is too large, and if it is smaller than k than r is too small. Being slightly more careful about the details (as was done in the proof of Lemma 4.2) one can verify this leads to a $(1+\varepsilon)$-decider for this problem, thus establishing (P1) of Definition 3.3_{p9}.

Clearly the Lipschitz property (P2) holds in this case, and so we only need to show (P3) holds. Namely, it must be shown that k can be updated properly for a weighted point p whose nearest neighbor is further away than $f(\mathsf{W}, k, m)$. If $\omega(\mathsf{p}) \geq m$ then we are throwing away $\omega(\mathsf{p})$ points, all with $\mathsf{d}_m(\mathsf{p},\mathsf{W}) = 0$, and so k should be update to $k - \omega(\mathsf{p})$ when p is

removed. Similarly, if $\omega(\mathsf{p}) < m$ then all the points p corresponds to have $\mathsf{d}_m(\mathsf{p}, \mathsf{W})$ larger than the threshold, and k does not have to be updated. This establish (P3).

Plugging this into Theorem 3.17 implies the desired result. ∎

Remark 4.6. Theorem 4.5 can be easily extended to work in the bichromatic case. That is, there are two point sets P and Q, and we are interested in the mth nearest neighbor of a point $\mathsf{p} \in \mathsf{P}$ in the set Q. It is easy to verify that the same time bounds of Theorem 4.5 hold in this case.

In particular, setting $k = |\mathsf{P}|$ and $m = 1$ in the bichromatic case, the computed distance will be the minimum radius of the balls needed to be placed around the points of P to cover all the points of Q (or vice versa).

4.3.1 Furthest nearest neighbor

Using Theorem 4.5 with $m = 2$ and $k = n$ and $\varepsilon = 1$, results in a 2-approximation, in $O(n)$ time, to the ***furthest nearest neighbor distance***; that is, the nearest neighbor distance of the point whose nearest neighbor is furthest away.

So we have a quantity r that is larger than the furthest nearest neighbor distance, but at most twice larger. We build a grid with every cell having diameter $r/4$. Clearly, the point realizing the furthest nearest neighbor distance must be the only point in its grid cell. For such a point, we compute its distance to all the points stored in its cell's neighborhood (say, all cells in distance at most r from this cell). This computes for these "lonely" points their exact nearest neighbor distance, and the maximum such distance computed is the one realizing the furthest nearest neighbor distance. Clearly, every cell's list of points get scanned a constant number of times, so overall the running time is linear. We summarize the result.

Theorem 4.7. *Let P be a set of n points in $\mathbb{R}^d$. In $O(n)$ expected time, one can compute exactly the furthest nearest neighbor distance in P.*

4.4 The spanning forest partitions, and the kth longest MST edge

The net computation provides a natural way to partition the data into clusters, and leads to a fast clustering algorithm. One alternative partition scheme is based on distance connectivity.

Definition 4.8. *For a set of points P and a number $r > 0$, let $\mathcal{C}_{\leq r}(\mathsf{P})$ be the* ***r-connected components*** *of P; that is, it is a partition of P into connected components of the MST of P after all edges strictly longer than r are removed from it (alternatively, these are the connected components of the intersection graph where we replace every point of P by a disk of radius $r/2$ centered at that point).*

Consider two partitions $\mathcal{P}, \mathcal{Q}$ of P. The partition $\mathcal{P}$ is a refinement of $\mathcal{Q}$, denoted by $\mathcal{P} \sqsubseteq \mathcal{Q}$, if for any set $X \in \mathcal{P}$, there exists a set $Y \in \mathcal{Q}$ such that $X \subseteq Y$.

Lemma 4.9. *Given a set $\mathsf{P} \subseteq \mathbb{R}^d$, and parameters r and ε, one can compute, in $O\big(n/\varepsilon^d\big)$ time, a partition $\mathcal{P}$, such that $\mathcal{C}_{\leq r}(\mathsf{P}) \sqsubseteq \mathcal{P} \sqsubseteq \mathcal{C}_{\leq (1+\varepsilon)r}(\mathsf{P})$.*

Proof: Build a grid with every cell having diameter $\varepsilon r/4$. For every point in P mark all the cells in distance $r/2$ from it. That takes $O(n/\varepsilon^d)$ time. Next, for every marked cell create a node v, and let V be the resulting set of nodes. Next, create a bipartite graph $G = (V \cup \mathsf{P}, E)$ connecting every node of V, to all the points of P marking its corresponding grid cell. Now, compute the connected components in G, and for each connected component extract the points of P that belong to it. Clearly, the resulting partition $\mathcal{P}$ of P, is such that any two points in distance $\leq r$ are in the same set (since for such points there must be a grid cell in distance $\leq r/2$ from both). Similarly, if the distance between two subsets $X, Y \subseteq \mathsf{P}$ is at least $(1+\varepsilon)r$, then they are in different connected components of $\mathcal{P}$. In particular, in order for X and Y to be in the same component, there must exists points $x \in X$ and $y \in Y$ which marked the same grid cell. However, such a grid cell would need to be in distance at most $r/2$ from both x and y, implying $\mathsf{d}(x,y) \leq r/2 + r/2 + \varepsilon r/4 < (1+\varepsilon)r$, a contradiction. Thus, this is the desired partition. Clearly, this takes $O(n/\varepsilon^d)$ time overall. ∎

Theorem 4.10. *Given a set of n points P in $\mathbb{R}^d$, and parameters k and ε, one can output, in $O\big(n/\varepsilon^d\big)$ time, a $(1+\varepsilon)$-approximation to the kth longest (or kth shortest) edge in the MST of P.*

Proof: Consider the cost function $\ell_{MST}(\mathsf{P}, k)$ which returns the kth longest edge of the MST of P. This function is an NDP with the decision procedure being Lemma 4.9. It is easy to verify that $\ell_{MST}(\mathsf{P}, k)$ is the minimum r such that $\mathcal{C}_{\leq r}(\mathsf{P})$ has k connected components. Now, Lemma 4.9 provides a $(1+\varepsilon)$-decision procedure for $\ell_{MST}(\mathsf{P}, k)$ that works in linear time. Furthermore, $\ell_{MST}(\mathsf{P}, k)$ complies with $(\mathrm{P2})_{\mathrm{p9}}$, as can be readily verified. As for (P3), consider a point $\mathsf{p} \in \mathsf{W}^{\geq r}$, where $\ell_{MST}(\mathsf{W}, k) < r$. Clearly, any edge in the MST of W that involves p must be of length $> r$; that is, the point p is a singleton in $\mathcal{C}_{\leq \ell_{MST}(\mathsf{W},k)}(\mathsf{P})$. As such, it can indeed be thrown away, and k should be decreased by one, as removing p might replace edges longer than r in the MST by edges that are even longer. This establish that $\ell_{MST}(\mathsf{P}, k)$ is a NDP, and by plugging this into Theorem 3.17, we get the result. ∎

4.5 Computing the minimum cluster

4.5.1 Sketchable families

Definition 4.11 (Upward Closed Set System). *Let P be a finite ground set of elements, and let $\mathcal{F}$ be a family of subsets of P. Then $(\mathsf{P}, \mathcal{F})$ is an* **upward closed set system** *if for any $X \in \mathcal{F}$ and any $Y \subseteq \mathsf{P}$, such that $X \subseteq Y$, we have that $Y \in \mathcal{F}$. Such a set system is a* **sketchable family**, *if for any set $S \subseteq \mathsf{P}$ there exists a constant size* **sketch** $\mathrm{sk}(S)$ *such that:*

- *(A) For any $S, T \subseteq \mathsf{P}$ that are disjoint, $\mathrm{sk}(S \cup T)$ can be computed from $\mathrm{sk}(S)$ and $\mathrm{sk}(T)$ in $O(1)$ time.*
- *(B) There is a membership oracle for the set system based on the sketch. That is, there is a procedure $\mathcal{O}(\cdot)$ such that given the sketch of a subset $\mathrm{sk}(S)$, $\mathcal{O}(\mathrm{sk}(S))$ returns whether $S \in \mathcal{F}$ or not, in $O(1)$ time.*

An example for such a sketchable family, is the set system $(\mathsf{P}, \mathcal{F})$, where $S \subseteq \mathsf{P}$ is in $\mathcal{F}$ if $|S| \geq 10$. Here the sketch of the set is simply the number of elements in the set,

and combining two sketches $\mathrm{sk}(S)$ and $\mathrm{sk}(T)$ is adding the numbers to get $\mathrm{sk}(S \cup T)$ (for $S \cap T = \emptyset$).

We will be interested in two natural problems induced by such a family: (i) smallest cluster – find the smallest set in the family with certain properties, and (ii) min-max clustering – find disjoint sets in the family such that they cover the original set, and the maximum price of these sets is minimized.

Remark 4.12. Note it is not necessary for the descriptions to have $O(1)$ size or the oracle to run in $O(1)$ time, however, assuming otherwise will affect the running time of **ndpAlg** if it is used to solve a problem involving a sketchable family.

Example 4.13. *Consider associating a positive k-dimensional vector ψ_{p} with each point $\mathsf{p} \in \mathsf{P}$ (a vector is* **positive** *if it is non-zero, and all its coordinates are non-negative). A* **positive linear inequality** *is an inequality of the form $\sum_i \alpha_i x_i \geq c$, where the coefficients $\alpha_1, \ldots, \alpha_k$ are all non-negative. For such a linear inequality, consider the set system $(\mathsf{P}, \mathcal{F})$, where a set Q is in $\mathcal{F}$ if the linear inequality holds for the vector $\sum_{\mathsf{p} \in \mathsf{Q}} \psi_{\mathsf{p}}$. Clearly, this family is a sketchable family, the sketch being the sum of the vectors associated with the points of the set (here k is assumed to be a constant).*

It is easy to verify that sketchable families are closed under finite intersection. Specifically, given a collection of m such positive inequalities, the family of sets such that their sketch vector complies with all these inequalities is a sketchable family (of course, checking if a set, given its sketch, is in the family naively would take $O(mk)$ time).

As a concrete application, consider the scenario where every element in P has $k = 4$ attributes. One might be interested in subsets, where each set has at least a total sum of 1 unit for the first two attributes, and a total sum of 2 units for the last two attributes.

Of course, in general an membership oracle for the attributes space that has the property that if ψ is valid then $\psi + \psi'$ is also valid, for any positive ψ', would define a sketchable family. As a concrete example, consider the non-linear condition that the sum of at least two attributes is larger than 1. Clearly, this defines a sketchable family.

Clearly the above definition of sketchable family is very general and widely applicable. Though many more example could be given, we now instead show how **ndpAlg** can be used to approximate certain objectives over sketchable families.

4.5.2 Min cluster

We now consider the problem of minimizing the cluster size of a subset of a given point set subject to inclusion in a sketchable family. Specifically, we consider the case when the cluster is defined by a ball of radius r or when the cluster is defined by a connected component of $\mathcal{C}_{\leq r}(\mathsf{P})$ for a radius r. Note that as a ball or component grows both the cluster size and inclusion of the set of points (in the cluster) in the sketchable $\mathcal{F}$ are monotone properties. This correspondence is what allows us to apply our general framework.

Theorem 4.14. *For a set of n points $\mathsf{P} \subseteq \mathbb{R}^d$, and a sketchable family $(\mathsf{P}, \mathcal{F})$, one can $(1+\varepsilon)$-approximate, in $O\big(n/\varepsilon^d\big)$ time, the radius of the smallest ball, b, such that $\mathsf{b} \cap \mathsf{P} \in \mathcal{F}$.*

Proof: Let $f(\mathsf{P})$ be the radius of the smallest ball b in $\mathbb{R}^d$ such that $\mathsf{P} \cap \mathsf{b} \in \mathcal{F}$. We claim that $f(\cdot)$ is an NDP (Definition 3.3_{p9}). Indeed (P2) and (P3) readily hold for $f(\cdot)$. As for the decision procedure, given r and $\varepsilon > 0$, construct a grid with cells of diameter $\varepsilon r/4$, and register each point of P in all the grid cells in distance at most r from it. If there is a ball, b, of radius r such that $\mathsf{b} \cap \mathsf{P} \in \mathcal{F}$ then the set of points registered with the grid cell containing the center of this ball will be a superset of $\mathsf{b} \cap \mathsf{P}$ and hence the set is in $\mathcal{F}$, by the upward closed property of sketchable families. Moreover, the set registered at this grid cell requires a ball of radius at most $r + 2\varepsilon r/4$ (centered at any point in this grid cell) to cover it. Thus, the decision procedure simply checks the set of points associated with each grid cell to see whether or not it is in $\mathcal{F}$. The definition of sketchable families implies this can be done in linear time in the sizes of the sets (namely computing the sketch takes linear time and the oracle constant time). Furthermore, if there is no ball of radius $(1+\varepsilon)r$ whose point set is in $\mathcal{F}$ then the decision procedure would fail to find such a cell. Thus, this is a $(1+\varepsilon)$-decision procedure, and its running time is $O(n/\varepsilon^d)$. Thus fulfilling (P1). Plugging this into the algorithm of Theorem 3.17 implies the result. ∎

The following is a sample of what the above theorem implies.

Corollary 4.15. *We can $(1+\varepsilon)$-approximate, in $O(n/\varepsilon^d)$ time, the following problems for a set of n points in $\mathbb{R}^d$:*

(A) The smallest ball containing k points of P.

(B) The points of P are weighted and we are given a threshold α. Compute the smallest ball containing points of weight at least α.

(C) If the points of P are colored by k colors, the smallest ball containing points of P, such that they are colored by at least t different colors. (Thus, one can find the smallest non-monochromatic ball [$t = 2$], and the smallest ball having all colors [$t = k$].) The running time is $O\big(nk/\varepsilon^d\big)$, as the sketch here is a k-dimensional vector.

4.5.3 Min cluster in the spanning forest

Note that $\mathcal{C}_{\le r}(\mathsf{P})$ is a monotone partition as r increases, and it is natural to ask what is the minimum r, for which there is a connected component in $\mathcal{C}_{\le r}(\mathsf{P})$ that is in a sketchable family.

Theorem 4.16. *For a set of n points $\mathsf{P} \subseteq \mathbb{R}^d$, and a sketchable family $(\mathsf{P}, \mathcal{F})$, one can $(1+\varepsilon)$-approximate, in $O\big(n/\varepsilon^d\big)$ time, the minimum r, such that there is a connected component in $\mathcal{C}_{\le r}(\mathsf{P})$ that is in $\mathcal{F}$.*

Proof: The target function $f(\mathsf{P})$ returns the smallest r such that $\mathcal{C}_{\le r}(\mathsf{P})$ contains a set that is in $\mathcal{F}$. Since $\ell_{MST}(\mathsf{P}, 1)$ complies with $(\text{P2})_{\text{p9}}$ (see Section 4.4) it is immediate to see that the same holds for $f(\cdot)$. Also, throwing away far away isolated points is allowable, as can be easily verified, thus implying that (P3) holds. As for the decision procedure, $(1+\varepsilon)$-approximate the connected components of $\mathcal{C}_{\le r}(\mathsf{P})$, using Lemma 4.9, and for each approximate connected component, use their sketch to decide if they are in $\mathcal{F}$. If so, return that r is larger than the optimal value, otherwise return that it is too small. Clearly, this is $(1+\varepsilon)$-decider that works in $O(n/\varepsilon^d)$ time. Plugging this into Theorem 3.17 implies the result. ∎

One natural application for Theorem 4.16, is for ad hoc wireless networks. Here, we have a set P of n nodes and their locations (say in the plane), and each node can broadcast in a certain radius r (the larger the r the higher the energy required, so naturally we would like to minimize it). It is natural now to ask for the minimum r such that one of the connected components in the resulting ad hoc network has some desired property. For example, in $O(n/\varepsilon^d)$ time, we can $(1+\varepsilon)$-approximate the smallest r such that:

(A) One of the connected components of $\mathcal{C}_{\leq r}(\mathsf{P})$ contains half the points of P, or more generally if the points are weighted, that one of connected component contains points of total weight at least α, for a prespecified α.

(B) If the points are colored, the desired connected component contains all the colors (for example, each color represent some fraction of the data, and the cluster can recover the data if all the pieces are available), or at least two colors, or more generally a different requirement on each color.

4.6 Clustering for monotone properties

Definition 4.17 (Min-Max Clustering). *We are given a sketchable family $(\mathsf{P}, \mathcal{F})$, and a cost function $g : 2^{\mathsf{P}} \to \mathbb{R}^+$. We are interested in finding disjoint sets $S_1, \ldots, S_m \in \mathcal{F}$, such that (i) $\bigcup_i S_i = \mathsf{P}$, and (ii) $\max_i g(S_i)$ is minimized. We will refer to the partition realizing the minimum as the* **optimal clustering** *of P.*

Theorem 4.18. *Let P be a set of points in $\mathbb{R}^d$, and let $(\mathsf{P}, \mathcal{F})$ be a sketchable family. For a set $\mathsf{W} \in \mathcal{F}$, let $\mathsf{r}_{\min}(\mathsf{W})$ be the radius of the smallest ball centered at a point of and enclosing W. One can $(4+\varepsilon)$-approximate, in $O(n/\varepsilon^d)$ time, the min-max clustering under $\mathsf{r}_{\min}$ of P.*

That is, one can cover P by a set of balls, and assign each point of P to one of these balls, such that the set of points assigned to each ball is in $\mathcal{F}$, and the maximum radius of any of these balls is a $(4+\varepsilon)$-approximation to the minimum radius used by any such cover.

Proof: Let $\mathcal{P}_{\text{opt}}$ be the optimal partition with radius r_{opt}, and consider an r-net $\mathcal{N}$ for $r \geq 4r_{opt}$, computed using Corollary 2.3. Consider a point $\mathsf{p} \in \mathcal{N}$, and let $\mathsf{P}_{\mathcal{N}}[\mathsf{p}]$ be the set of points of P assigned to p by the nearest net-point assignment.

Next, consider the cluster $\mathsf{W} \in \mathcal{P}_{\text{opt}}$ that contains it. Clearly, $\operatorname{diam}(\mathsf{W}) \leq 2r_{opt}$, and the distance of p from all other net points in $\mathcal{N}$ is at least $4r_{opt}$. It follows that $\mathsf{W} \subseteq \mathsf{P}_{\mathcal{N}}[\mathsf{p}]$, and since $\mathsf{W} \in \mathcal{F}$, it follows that $\mathsf{P}_{\mathcal{N}}[\mathsf{p}] \in \mathcal{F}$.

A 4-decider for this problem works by computing the $4r$-net $\mathcal{N}$, and for each $\mathsf{p} \in \mathsf{P}$, checking the sketchable property for the set $\mathsf{P}_{\mathcal{N}}[\mathsf{p}]$. It is easy to verify that the properties of Definition 3.3_{p9} hold in this case. In particular, throwing a far away isolated point corresponds to a cluster that already fulfill the monotone property, and it is too far away to be relevant. Namely, computing r_{opt} is an NDP and so plugging this into Theorem 3.16 implies the result. ∎

4.6.1 Lower bounded center clustering

If the required sketchable property is that every cluster contains at least k points, then Theorem 4.18 approximates the ***lower bounded center*** problem. That is, one has to cover

the points by balls, such that every cluster (i.e., points assigned to a ball) contains at least k points. The price of this clustering is the radius of the largest ball used. A 2-approximation to this problem is known via the usage of flow [APF+10] but the running time is super quadratic. Recently, the authors showed a similar result to Theorem 4.18 with running time (roughly) $O(n \log n)$ [ERH12]. It is also proven that this problem cannot be approximated to better than (roughly) 1.8 even for points in the plane. Thus, the following immediate implication of Theorem 4.18 improves over this result.

Corollary 4.19. *Let P be a set of points in $\mathbb{R}^d$, and let k and $\varepsilon > 0$ be parameters. One can $(4+\varepsilon)$-approximate the lower bounded center clustering in $O(n/\varepsilon^d)$ time.*

4.6.2 Other clustering problems

One can plug-in any sketchable family into Theorem 4.18. For example, if the points have k colors, we can ask for the min-max radius clustering, such that every cluster contains (i) all colors, (ii) at least two different colors, or (iii) a different requirement on each color, etc.

As another concrete example, consider that we have n customers in the plane, and each customer is interested in k different services (i.e., there is a k-dimensional vector associated with each customer specifying his/her demand). There are t types of service centers that can be established, but each such center type requires a minimum level of demand in each of these k categories (i.e., each type is specified by a minimum demand k-dimensional vector, and a set of customers can be the user base for such a service center if the sum of their demands vector is larger than this specification). The problem is to partition the points into clusters (of minimum maximum radius), such that for every cluster there is a valid service center assigned to it. Clearly, this falls into the framework of Theorem 4.18, and can be $(4+\varepsilon)$-approximated in $O(nkt/\varepsilon^d)$ time.

4.6.3 Clustering into spanning forests

One can get a similar result to Theorem 4.18 for the connectivity version of clustering of P. Formally, a set of points $\mathsf{W} \subseteq \mathsf{P}$ is ***r-valid*** if W is contained in some set of $\mathcal{C}_{\leq r}(\mathsf{P})$. Given a sketchable family $(\mathsf{P}, \mathcal{F})$, a partition $\mathcal{P}$ of P is an ***r-connected clustering*** if all the sets in $\mathcal{P}$ are in $\mathcal{F}$, and are r-valid.

Theorem 4.20. *Let P be a set of points in $\mathbb{R}^d$, and let $(\mathsf{P}, \mathcal{F})$ be a sketchable family. One can $(1+\varepsilon)$-approximate r_{opt}, where r_{opt} is the minimum value such that there is a r_{opt}-connected clustering of P.*

Proof: It is easy to verify that this target function is NDP (see Definition 3.3$_{\text{p9}}$). Indeed, for the decider, given r, we use Lemma 4.9 to compute a partition $\mathcal{P}$ of P such that $\mathcal{C}_{\leq r}(\mathsf{P}) \sqsubseteq \mathcal{P} \subseteq \mathcal{C}_{\leq (1+\varepsilon)r}(\mathsf{P})$. If the sketchable property holds for each cluster we return that r is too large, otherwise we return that it is too small. As for the quality of this decider, observe that if the optimal partition has a cluster W that uses points from two different clusters of $\mathcal{P}$, than W is not r-valid, as otherwise these two points would be in the same cluster of $\mathcal{P}$ (namely, $r_{opt} > r$).

The Lipschitz (P2) condition readily follows. Similarly, if an isolated point exists, then it can be thrown away because the cluster of original points it corresponds to, is a valid cluster

that can be used in the final clustering, and it does not interact with any other clusters. Thus (P3) also holds.

Plugging this into Theorem 3.17 now implies the result. ∎

A nice application of Theorem 4.20 is for ad hoc networks. Again, we have a set P of n wireless clients, and some of them are base stations; that is, they are connected to the outside world. We would like to find the minimum r, such that each connected component of $\mathcal{C}_{\leq r}(\mathsf{P})$ contains a base station.

4.7 Smallest non-zero distance

Closest pair. By setting $k = 1$ and $\varepsilon = 1$ one can use Lemma 4.2_{p16} to get a 2-approximation to the closest pair distance in $O(n)$ time. Let the returned distance be r. It is not hard to see that one can then compute the closest pair exactly by throwing the points into a grid where the diameter of a cell is $< r/2$. Indeed, compute for every point that is the only point in a cell, its nearest neighbor in the neighboring cells in the grid. Clearly, the closest pair distance would be encountered, and the overall work is linear.

Smallest non-zero distance. If P contains duplicates then a couple modifications to **ndpAlg** must be made. First modify the algorithm so that for the selected point it finds the closest distinct nearest neighbor. Secondly, we modify **algDelFar** (see Lemma 2.5_{p9}) so that if a net point corresponds to several copies of the same point then it is being treated as a single point. With these two modifications, the above algorithm works verbatim, and we get the following.

Lemma 4.21. *Let P be a multiset of weighted points in $\mathbb{R}^d$. Then one can solve the Smallest-NonZeroDist problem* **exactly** *for P in linear time. In particular, if P contains no duplicates then this corresponds to computing the closest pair distance.*

Interestingly, the algorithm of Lemma 4.21 is a prune-and-search algorithm, as the net stage never gets executed. Observe, that it is not hard to extend the algorithm of Golin *et al.* [GRSS95] to solve this variant, and the result of Lemma 4.21 is included in the paper only for the sake of completeness.

5 Conclusions

There is still a lot of further research to be done in investigating this technique. For example, looking into the implementation of this new algorithm in both standard settings and MapReduce. Additionally, since one can now do approximate distance selection in linear time, maybe now one can get a speed up for other algorithms that do (not necessarily point based) distance selection.

Our framework provides a new way of looking at distance based optimization problems, in particular through the lens of nets. We know how to compute nets efficiently for doubling metrics and it seems one can compute approximate nets in near linear time for planar graphs. For example, it seems the new technique implies that approximate k-center clustering in

planar graphs can be done in near linear time. This provides fertile ground for future research.

References

[AH08] B. Aronov and S. Har-Peled. On approximating the depth and related problems. *SIAM J. Comput.*, 38(3):899–921, 2008.

[AHV04] P. K. Agarwal, S. Har-Peled, and K. R. Varadarajan. Approximating extent measures of points. *J. Assoc. Comput. Mach.*, 51(4):606–635, 2004.

[AHV05] P. K. Agarwal, S. Har-Peled, and K. Varadarajan. Geometric approximation via coresets. In J. E. Goodman, J. Pach, and E. Welzl, editors, *Combinatorial and Computational Geometry*, Math. Sci. Research Inst. Pub. Cambridge, 2005.

[APF+10] G. Aggarwal, R. Panigrahy, T. Feder, D. Thomas, K. Kenthapadi, S. Khuller, and A. Zhu. Achieving anonymity via clustering. *ACM Trans. Algo.*, 6(3), 2010.

[AST94] P. K. Agarwal, M. Sharir, and S. Toledo. Applications of parametric searching in geometric optimization. *J. Algorithms*, 17:292–318, 1994.

[BS02] S. Bespamyatnikh and M. Segal. Fast algorithms for approximating distances. *Algorithmica*, 33(2):263–269, 2002.

[Cla83] K. L. Clarkson. Fast algorithms for the all nearest neighbors problem. In *Proc. 24th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 226–232, 1983.

[DHW12] A. Driemel, S. Har-Peled, and C. Wenk. Approximating the Fréchet distance for realistic curves in near linear time. *Discrete Comput. Geom.*, 48:94–127, 2012.

[EET93] H. ElGindy, H. Everett, and G. Toussaint. Slicing an ear using prune-and-search. *Pattern Recogn. Lett.*, 14(9):719–722, 1993.

[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://valis.cs.uiuc.edu/~sariel/papers/12/lbc/`, 2012.

[Eri95] J. Erickson. On the relative complexities of some geometric problems. In *Proc. 7th Canad. Conf. Comput. Geom.*, pages 85–90, 1995.

[FJ84] G. N. Frederickson and D. B. Johnson. Generalized selection and ranking: sorted matrices. *SIAM J. Comput.*, 13:14–30, 1984.

[Gon85] T. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoret. Comput. Sci.*, 38:293–306, 1985.

[GRSS95] M. Golin, R. Raman, C. Schwarz, and M. Smid. Simple randomized algorithms for closest pair problems. *Nordic J. Comput.*, 2:3–27, 1995.

[Har01] S. Har-Peled. Clustering motion. In *Proc. 42nd Annu. IEEE Sympos. Found. Comput. Sci.*, pages 84–93, 2001.

[Har04a] S. Har-Peled. Clustering motion. *Discrete Comput. Geom.*, 31(4):545–565, 2004.

[Har04b] S. Har-Peled. No coreset, no cry. In *Proc. 24th Conf. Found. Soft. Tech. Theoret. Comput. Sci.*, 2004.

[HK05] S. Har-Peled and A. Kushal. Smaller coresets for k-median and k-means clustering. In *Proc. 21st Annu. ACM Sympos. Comput. Geom.*, pages 126–134, 2005.

[HM04] S. Har-Peled and S. Mazumdar. Coresets for k-means and k-median clustering and their applications. In *Proc. 36th Annu. ACM Sympos. Theory Comput.*, pages 291–300, 2004.

[HM05] S. Har-Peled and S. Mazumdar. Fast algorithms for computing the smallest k-enclosing disc. *Algorithmica*, 41(3):147–157, 2005.

[HM06] S. Har-Peled and M. Mendel. Fast construction of nets in low dimensional metrics, and their applications. *SIAM J. Comput.*, 35(5):1148–1184, 2006.

[HR11] S. Har-Peled and B. Raichel. The Fréchet distance revisited and extended. In *Proc. 27th Annu. ACM Sympos. Comput. Geom.*, pages 448–457, 2011. `http://www.cs.uiuc.edu/~sariel/papers/10/frechet3d/`.

[HR12] S. Har-Peled and B. Raichel. Net and prune: A linear time algorithm for euclidean distance problems. Manuscript, 2012.

[KL04] R. Krauthgamer and J. R. Lee. Navigating nets: simple algorithms for proximity search. In *Proc. 15th ACM-SIAM Sympos. Discrete Algorithms*, pages 798–807. Society for Industrial and Applied Mathematics, 2004.

[LMS94] C.-Y. Lo, J. Matoušek, and W. L. Steiger. Algorithms for ham-sandwich cuts. *Discrete Comput. Geom.*, 11:433–452, 1994.

[Meg83] N. Megiddo. Applying parallel computation algorithms in the design of serial algorithms. *J. Assoc. Comput. Mach.*, 30(4):852–865, 1983.

[Meg84] N. Megiddo. Linear programming in linear time when the dimension is fixed. *J. Assoc. Comput. Mach.*, 31:114–127, 1984.

[MSW96] J. Matoušek, M. Sharir, and E. Welzl. A subexponential bound for linear programming. *Algorithmica*, 16:498–516, 1996.

[Rab76] M. O. Rabin. Probabilistic algorithms. In J. F. Traub, editor, *Algorithms and Complexity: New Directions and Recent Results*, pages 21–39. Academic Press, 1976.

[Sal97] J. Salowe. Parametric search. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, chapter 37, pages 683–698. CRC Press LLC, Boca Raton, FL, 1997.

[Sch79] A. Schönhage. On the power of random access machines. volume 71 of *Lecture Notes Comput. Sci.*, pages 520–529, 1979.

[Smi00] M. Smid. Closest-point problems in computational geometry. In Jörg-Rüdiger Sack and Jorge Urrutia, editors, *Handbook of Computational Geometry*, pages 877–935. Elsevier, 2000.

[SW92] M. Sharir and E. Welzl. A combinatorial bound for linear programming and related problems. In *Proc. 9th Sympos. Theoret. Aspects Comput. Sci.*, volume 577 of *Lect. Notes in Comp. Sci.*, pages 569–579. Springer-Verlag, 1992.

[vOV04] R. van Oostrum and R. C. Veltkamp. Parametric search made practical. *Comput. Geom. Theory Appl.*, 28(2-3):75–88, 2004.