Discrete Comput Geom 31:545–565 (2004)
DOI: 10.1007/s00454-004-2822-7

Discrete & Computational
Geometry

Clustering Motion*

Sariel Har-Peled

Department of Computer Science, DCL 2111, University of Illinois,
1304 West Springfield Avenue, Urbana, IL 61801, USA
sariel@uiuc.edu
http://www.uiuc.edu/∼sariel/

"A slow sort of country!" said the Queen. "Now, HERE, you see, it takes all the running YOU can do, to keep in the same place. If you want to get somewhere else, you must run at least twice as fast as that!"

Through the Looking-Glass, *Lewis Carroll*

Abstract. Given a set of moving points in $\mathbb{R}^d$, we show how to cluster them in advance, using a small number of clusters, so that at any time this static clustering is competitive with the optimal k-center clustering at that time. The advantage of this approach is that it avoids updating the clustering as time passes. We also show how to maintain this static clustering efficiently under insertions and deletions.

To implement this static clustering efficiently, we describe a simple technique for speeding up clustering algorithms and apply it to achieve faster clustering algorithms for several problems. In particular, we present a linear time algorithm for computing a 2-approximation to the k-center clustering of a set of n points in $\mathbb{R}^d$. This slightly improves the algorithm of Feder and Greene, that runs in $\Theta(n \log k)$ time (which is optimal in the algebraic decision tree model).

1. Introduction

Clustering is a central problem in computer science. It is related to unsupervised learning, classification, databases, spatial range-searching, data-mining, etc. As such, it has received much attention in the last 20 years. There is a large literature on this topic with numerous variants, see [BE] and [DHS].

* A preliminary version of the paper appeared in *Proceedings of the* 42*nd Annual IEEE Symposium on the Foundations of Computer Science*, pages 84–93, 2001.

k-Center Clustering. One of the most natural definitions of clustering is the *min-max radius clustering* or *k-center clustering*. Here, given a set of n points in some metric space, one wishes to find k special points, called *centers*, such that the maximum distance of a point to a center is minimized. In the *continuous* k-center clustering, the centers can be located everywhere in the underlying space, and in the second variant, known as the *discrete* k-center, the k centers must be input points. An alternative interpretation of k-center clustering is that we would like to cover the points by k balls, where the radius of the largest ball is minimized. Intuitively, if such a tight clustering exists, it provides a partition of the point set into k classes, where the points inside each class are "similar."

There is a very simple and natural algorithm that achieves a 2-approximation for the discrete k-center clustering [G]: Repeatedly pick the point furthest away from the current set of centers as the next center to be added. This algorithm can be easily implemented in $O(nk)$ time [G], and, in fact, it is also a 2-approximation for the continuous case.

Feder and Greene [FG] showed that if the points are taken from $\mathbb{R}^d$, one can compute 2-approximation to the optimal k-center clustering in $\Theta(n \log k)$, by a different implementation of the greedy algorithm mentioned above, and this algorithm is optimal in the algebraic decision tree model. They also showed that computing a c-approximation is NP-hard for all $c \leq 1.822$.

Clustering Motion. Let $P[t]$ be a set of n moving points in $\mathbb{R}^d$, with a degree of motion μ; namely, for a point $p[t] \in P[t]$, we have $p(t) = (p_1(t), \ldots, p_d(t))$, where $p_j(t)$ is a polynomial of degree μ, and t is the time parameter, for $j = 1, \ldots, d$. If one wishes to answer spatial queries of the moving point set $P[t]$, one needs to construct a data structure for that purpose. Most such data structures for stationary points rely on space partition schemes, and while such partitions and clusterings are well understood for the case of stationary points, for the case of moving points considerably less is known (see [DG], [AAE], [HV1], [GGH$^+$], and [AH] for recent relevant results).

The difficulty in maintaining and computing such clustering is the one underlying most kinetic data structures [BGH]: Once the clusters are computed at a certain time, and we let time progress, the clustering changes and deteriorates. To remain a competitive clustering (i.e., the cluster sizes are small compared with the size of the optimal clustering), one needs to maintain the clustering by either reclustering the points every once in a while, or, alternatively, move points from one cluster to another. The number of such "maintenance" events dominates the overall running time of the algorithm, and usually the number of such events is prohibitively large. For example, it is easy to verify that a kinetic clustering of n linearly moving points must handle $\Omega(n^2)$ events in the worst case to remain competitive (up to any factor) with the optimal k-center clustering at any given time. See Fig. 1. (However, there are indications that such worst-case inputs might not be that common in practice [BGSZ], and that they do not happen if you use random inputs [BDIZ].)

Our Results. We demonstrate, in Section 2, that if one is willing to compromise on the number of clusters used, then clustering becomes considerably easier (computationally) and it can be done quickly. Furthermore, we can trade off between the quality of the clustering and the number of clusters used. As such, one can compute quickly a clustering with a large number of clusters, and cluster those clusters in a second stage, to get a reasonable k-clustering.

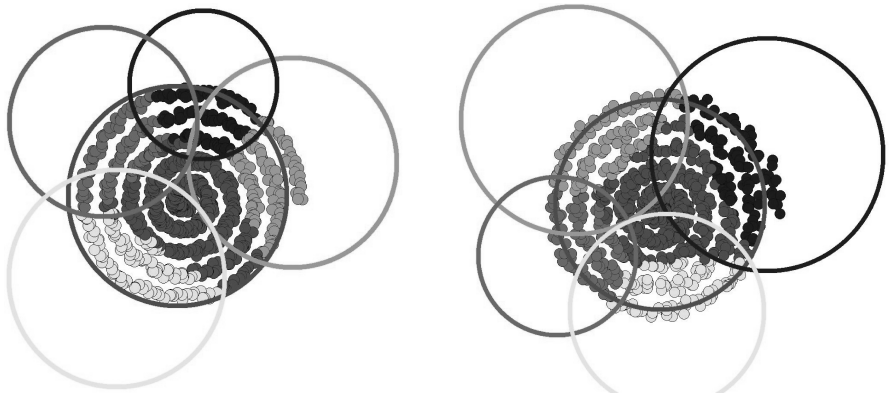

Fig. 1. 2-Approximate clustering changes completely, even when the change in the positions of the moving points is relatively "small." The clustering depicted was generated by running the algorithm of Gonzalez [G] on the two point sets. One can generate similar examples for the optimal k-center clustering, showing that it is unstable.

This relies on the observation that clustering a random sample results in a partial clustering of almost all input points (this is well known, see [VC], [ADPR], and [MOP]). Thus, by clustering the remaining uncovered points, we can get a clustering of the whole point set. Furthermore, by using a point-location data structure, one can quickly find out the points not covered by the clustering of the random sample. Putting this together results in a generic technique for speeding up clustering. The new approach can be viewed as a reduction of geometric clustering to the task of carrying out point-location queries quickly, together with approximate clustering [I], [ADPR], [MOP] and sampling techniques [VC], [HW]. We provide several applications of the new technique.

This naturally raises the question of whether one can get a similar tradeoff between the computational cost of maintaining a k-center clustering for moving points, and the number of centers used. In particular, in Section 3 we show that, for a point set moving with polynomial motion of degree μ, if one is allowed to use $k^{\mu+1}$ clusters (instead of k), then one can compute a clustering which is (constant factor) competitive with the optimal k-clustering of the point set at any time, where μ is the algebraic degree of motion of the points. This clustering is *static* and thus not required to handle any event. We also show that static clustering requires using $k^{\mu+1}$ clusters in the worst case. In Section 4 we present an algorithm to compute this clustering in $O(nk)$ time.

In Section 5, an algorithm for picking a "small" subset of the moving points is described. This is done by computing a very fine clustering, and picking a representative from each cluster. The size of this subset, known as *coreset*, is independent of n, and it represents the k-center clustering of the moving points at any time, up to a factor of $1+\varepsilon$. Namely, instead of clustering the points, we can cluster the representative points. This implies, that one can construct a data structure that can report the approximate clustering at any time, and performs insertions, deletions, and motion updates in $\text{poly}(k, 1/\varepsilon, \log n)$ time for all of those operations, where $\text{poly}(\cdot)$ denote a fixed-degree polynomial in the input parameters. Note that this performance is quite attractive as clustering can change completely even if only a few points change their motion, requiring a prohibitive price if one were to just kinetize the algorithm of Gonzalez [G] naively.

Finally, in Section 6, we show how one can implement the greedy clustering algorithm of Gonzalez [G] in expected linear time, for all $k = O(n^{1/3}/\log n)$. (In fact, all the expected time bounds in our paper also hold with high probability.) Our algorithm relies on a considerably stronger computation model than Feder and Greene [FG]. In particular, the algorithm uses constant-time hashing, the floor function, and randomization.

Concluding remarks are given in Section 7.

2. Fast Clustering Using Point-Location

Given a set P of n points in $\mathbb{R}^d$, we are interested in covering them with a set $\mathcal{C} \subset \mathcal{R}$ of k objects (i.e., clusters) that minimizes the target function $\mathrm{r}(\mathcal{C}) = \max_{c\in\mathcal{C}} \mathrm{r}(c)$, where $\mathrm{r}(c)$ is a function that quantifies the size of the cluster, and $\mathcal{R}$ is the set of all permissible clusters. For example, in the case of the k-center problem, the clusters are balls, and we wish to minimize the maximum radius of the k balls used to cover the points. Let $C_{\mathrm{opt}}(P, k)$ denote the optimal *continuous* k-clustering of P (i.e., the centers of the clusters can be any point of $\mathbb{R}^d$), and let $r_{\mathrm{opt}}(P, k) = \mathrm{r}(C_{\mathrm{opt}}(P, k))$ denote the radius of this clustering. Let $\mathcal{A} = (P, \mathcal{R})$ denote the underlying clustering space. For example, for the continuous k-center clustering, $\mathcal{R}$ is the set of all balls in $\mathbb{R}^d$.

Definition 2.1. A set of clusters $\mathcal{C}$ is a *partial k-clustering*, or just *partial clustering*, of a point set P if it covers *some* of the points of P and $|\mathcal{C}| = k$. It is a *k-clustering* if it covers *all* the points of P.

Let c be a cluster in a partial clustering $\mathcal{C}$ of P. We replace c with the *canonical cluster* c'; that is, the cluster that realizes $\min_{c'\in\mathcal{R}, c\cap P\subseteq c'} \mathrm{r}(c')$. For k-center clustering, the canonical cluster of a ball c is the smallest ball that contains $P \cap c$. In the following we assume that we work with canonical clusters.

In geometric settings a cluster c' cannot shrink any further (i.e., it is canonical) because it is "locked" in place by a set of points of P, and in fact c' is defined by this set of points. (Imagine a ball that cannot shrink any further—it contains at most $d+1$ points of P on its boundary, and this is the smallest ball that contains this set of points.) Let $D(c')$ be this set of points, called the *defining set* of c'. In the following we assume that every defining set defines only a constant number of different clusters. Let $\mu = \dim(\mathcal{A}) = \max_{c\in\mathcal{R}} |D(c)|$ be the *dimension* of the space $\mathcal{A} = (P, \mathcal{R})$. (This is a restricted variant of the concept of VC-dimension, and in fact all our arguments can be carried out using VC-dimension. However, avoiding VC-dimension arguments slightly simplifies the presentation.) A *canonical k-clustering* is a clustering of P by k clusters which are all canonical.

Example 2.2. Consider the clustering space for the discrete k-center clustering. Here, a canonical cluster c is defined by its center $p \in P$, and the point furthest away from the center $q \in P$ (i.e., the length of pq is the radius of c). Thus, a canonical cluster in this setting is defined by a set of two points $\{p, q\}$. However, this set corresponds to another canonical cluster, the one that has q as its center, and its radius is the length of pq. Observe that the size of the defining set in the discrete case is independent of the dimension of the underlying space $\mathbb{R}^d$.

Lemma 2.3. *If the dimension of $\mathcal{A} = (P, \mathcal{R})$ is $\mu = \dim(\mathcal{A})$, then the number of partial canonical k-clusterings of P is at most $c\rho n^{\mu k}$, where $n = |P|$, c is a constant, and $\rho = \rho(\mathcal{A})$ is a constant that depends on the underlying space $\mathcal{A}$.*

Proof. The number of different defining sets of $\mathcal{A}$ is at most $\sum_{i=0}^{\mu} \binom{n}{i} = O(n^{\mu})$. Each such set defines at most a constant number $\rho = \rho(\mathcal{A})$ of canonical clusters (usually one, but this constant depends on the space $\mathcal{A}$, see Example 2.2). Thus, the number of canonical clusters is $U = O(\rho n^{\mu})$.

Thus, a canonical k-clustering is defined by k such clusters. It follows that the number of canonical k-clustering of P is $\leq U^k = O(n^{\mu k})$. □

Definition 2.4. For a partial clustering $\mathcal{C}$, let $\bigcup \mathcal{C}$ denote the region covered by the clusters of $\mathcal{C}$. For a parameter ε, a partial canonical clustering $\mathcal{C}$ is *ε-good for P* if the number of points of P that are not covered by $\mathcal{C}$ is no larger than εn. Formally, $\mathcal{C}$ is ε-good if $|P \setminus \bigcup \mathcal{C}| < \varepsilon n$.

The following lemma shows that if one clusters a large enough sample of points from P, then this clustering covers almost all the points of P. A similar observation (in slightly different form) was originally made in [ADPR] and [MOP] (in essence, this observation can be traced back to the work of Vapnik and Chervonenkis [VC]), and we include it here for the sake of completeness.

Lemma 2.5. *Let S be a set of points computed by picking randomly and uniformly m points (with repetition) from P. If*

$$m \geq \frac{\log \rho + \mu k \log n + \beta \log n}{\varepsilon},$$

then any canonical k-clustering of S is ε-good with probability at least $1 - n^{-\beta}$, where $\mu = \dim(\mathcal{A})$, $\rho = \rho(\mathcal{A})$, $\mathcal{A} = (P, \mathcal{R})$, and ε, β are parameters.

Proof. Let $\mathcal{C}$ be a canonical partial k-clustering of P which is not ε-good. If S contains a point that $\mathcal{C}$ does not cover, then $\mathcal{C}$ cannot be a clustering of S. However, we have

$$\Pr\left[S \subseteq \bigcup \mathcal{C}\right] \leq \left(\frac{|P \cap \bigcup C|}{|P|}\right)^m \leq (1-\varepsilon)^m \leq e^{-\varepsilon m}.$$

However, by Lemma 2.3, the number of different canonical k-clusterings of P is at most $\rho n^{\mu k}$. Thus, the probability that any canonical partial k-clustering of P which is not ε-good to be a clustering of S, is bounded by $\rho n^{\mu k} e^{-\varepsilon m}$. In particular, if we want this probability to be smaller than $n^{-\beta}$, then we have $\log \rho + \mu k \log n - \varepsilon m \leq -\beta \log n$. Solving for m, we have $m \geq (\log \rho + \mu k \log n + \beta \log n)/\varepsilon$. □

In the following we are interested in problems where the clustering price is *monotone*; that is, $\mathrm{r}(P', k) \leq \mathrm{r}(P, k)$ for $P' \subseteq P$. If we wish to perform k-clustering (or approximate k-clustering) using Lemma 2.5, and we are willing to compromise on the number of

Algorithm `FastCluster`(P, k, ε)

P is a set of n points, k is number of clusters, $\varepsilon > 0$ is a parameter

(i) Compute a sample $S \subseteq P$ of size $O((k \log n)/\varepsilon)$.
(ii) Compute an (approximately) optimal k-clustering $\mathcal{C}$ of S.
(iii) Compute $P' = P \setminus \bigcup \mathcal{C}$.
(iv) Compute an (approximately) optimal k-clustering $\mathcal{C}'$ of P'.
(v) Return $\mathcal{C} \cup \mathcal{C}'$.

Fig. 2. Doing fast clustering by using random sampling.

clusters used, then we can use the algorithm `FastCluster` described in Fig. 2. The advantage of `FastCluster` is that it calls twice for a clustering algorithm, but in both cases the sets used are small (one of size $O((k \log n)/\varepsilon)$, and the other of size εn). In fact, `FastCluster` uses two subroutines: (a) given a (small) subset of the points, compute k-clustering of them (stages (ii) and (iv)), and (b) given a clustering, and a set of points, compute all the points that lie outside the clustering (stage (iii)).

Definition 2.6. A *point-location data structure* is a data structure that receives a set $\mathcal{C}$ of u clusters as input and preprocesses them in $T_P(u)$ time, so that given a point p it can decide in $O(\log u)$ time whether a point lies inside $\bigcup \mathcal{C}$.

Using such a point-location data structure it is now possible to compute P' in stage (iii) of `FastCluster` in $O(n \log k + T_P(k))$ time. Indeed, we preprocess $\mathcal{C}$, the set of k clusters computed in stage (ii), for point-location queries. This takes $T_P(k)$ time. Then, for each point of P, we can decide in $O(\log k)$ time whether it is contained in $\bigcup \mathcal{C}$.

Definition 2.7. A clustering $\mathcal{C}$ is a (c, m, k)-*clustering* of P if $\mathcal{C}$ is made out of m clusters, and $\mathcal{C}$ is c-*competitive* with the optimal k-clustering; namely, $\mathrm{r}(\mathcal{C}) \leq c \cdot r_{\mathrm{opt}}(P, k)$.

Theorem 2.8. *Assume that we have a clustering algorithm that, given a set of m points, can compute a $(\alpha, \varphi(k), k)$-clustering in $T_C(m)$ time, where α and $\varphi(\cdot)$ (the number of clusters generated) depend on the algorithm at hand. Using* `FastCluster`*, one can compute an $(\alpha, 2\varphi(k), k)$-clustering of P in*

$$O\left(T_C\left(\frac{k \log n}{\varepsilon}\right) + T_C(\varepsilon n) + T_P(\varphi(k)) + n \log k\right)$$

expected time, where $\varepsilon > 0$ is a parameter that can be fine-tuned in advance to optimize the running time.

The algorithm `FastCluster` can be viewed as two-round clustering, and it can be extended to η-round clustering. Indeed, we cluster a sample from the points which

are not currently covered, and remove all the points which are covered, by using a point-location data structure, and repeat this η times. Finally, we cluster the remaining points. The advantage of such an η-round approach is that we can use considerably smaller samples. The disadvantage is that instead of getting $(\alpha, 2\varphi(k), k)$-clustering we get $(\alpha, \eta\varphi(k), k)$-clustering. We summarize the result in the following:

Theorem 2.9. *Using* `FastCluster`, *one can compute an* $(\alpha, \eta\varphi(k), k)$*-clustering of* P *in*

$$O\left(\eta T_C\left(\frac{k\log n}{\varepsilon}\right) + T_C(\varepsilon^{\eta} n) + \eta T_P(\varphi(k)) + \eta n \log k\right)$$

expected time, where $\varepsilon > 0$ *and* η *are parameters that can be fine-tuned in advance to optimize the running time.*

`FastCluster` needs a subroutine for performing $(\alpha, \varphi(k), k)$-clustering. However, this can be replaced by an algorithm that extracts a single cluster which has radius smaller than the optimal clustering, but contains at least (say) $n/4k$ of the points. Such a subroutine can be easily implemented using random sampling, and a brute-force search over all possible canonical clusters of the sample. Using point-location data structures this results in a greedy clustering with $O(k \log n)$ clusters, and in running time $O(n \log k)$ (ignoring terms which are polynomial in $\log n$ and k). Although this seems to be a stronger algorithm than `FastCluster`, as it requires a weaker subroutine (i.e., a subroutine to extract one heavy cluster, instead of a subroutine for performing k-clustering), we are currently unaware of a case where this results in a faster algorithm.

For an example of how to use Theorem 2.9, consider the following clustering problem: We are given a set P of n points in the plane, and we would like to cover them with k minimum max-width strips. It is known that this problem is NP-complete to approximate within any constant factor [MT]. The fastest approximation algorithm currently known is due to Agarwal and Procopiuc [AP]. It runs in $O(nk^2 \log^4 n)$ time and covers the points with $O(k \log k)$ strips of width at most six times the optimal clustering using k strips. Using Theorem 2.8 we get the following improved result.

Lemma 2.10. *Given a set of* n *points in the plane, one can compute a clustering of* P *by* $O(k \log k)$ *strips of width at most* $r_{\text{opt}}(P, k)$, *where* $r_{\text{opt}}(P, k)$ *denotes the minimal width cover of* P *by* k *strips, and* $k \leq n^{1/6}$. *The expected running time is* $O(n \log k)$.

Proof. Note that a canonical strip is defined by three points of P, and the dimension of the space $(P, \mathcal{R})$ is $\mu = 3$, where $\mathcal{R}$ is the set of all strips in the plane. We set $\varepsilon = \sqrt{k/n}$, and use the algorithm of Agarwal and Procopiuc [AP] with a sample of size $m = O(\sqrt{nk} \log n)$. Their algorithm works in $T_C(m) = O(mk^2 \log^4 m)$ time (for $k^2 \log k \leq m$). Note that the algorithm of Agarwal and Procopiuc [AP] returns a clustering by $O(k \log k)$ strips of width at most $6r_{\text{opt}}(P, k)$, and by splitting each strip into six equal strips we get the required clustering by $O(k \log k)$ strips of width at most $r_{\text{opt}}(P, k)$. Given those strips, preprocessing those strips for point-location takes $T_P(k \log k) = O(k^2 \log^2 k)$ time using standard techniques [dBvKOS]. Overall, the algorithm computes

a clustering by $O(k \log k)$ strips, with width at most $r_{\text{opt}}(P, k)$, and the running time is

$$O\left(T_C\left(\frac{k \log n}{\varepsilon}\right) + T_C(\varepsilon n) + T_P(k \log k) + n \log k\right)$$
$$= O\left(k^2 \log^2 k + \sqrt{n/k}\, k^2 \log^4(n/k) + n \log k\right) = O(n \log k),$$

since $k \leq n^{1/6}$. □

3. Static Clustering of Moving Points

For a moving point set P in $\mathbb{R}^d$, let $r_{\text{opt}}(P[t], k)$ denote the radius of the optimal (continuous) k-center clustering of $P[t]$, where $P[t]$ is the set of points of P at time t. Formally,

$$r_{\text{opt}}(P[t], k) = \min_{\substack{P[t] \subseteq \cup \mathcal{C} \\ \mathcal{C} \in \mathcal{B}^k}} \mathrm{r}(\mathcal{C}),$$

where $\mathcal{B}$ is the set of all balls in $\mathbb{R}^d$, and $\mathrm{r}(\mathcal{C}) = \max_{c \in \mathcal{C}} \mathrm{r}(c)$.

In the following we omit P and k when they can be understood from the context. Let $t_{\min} = t_{\min}(P, k)$ be the time when $r_{\text{opt}}(t, k)$ is minimized, and let $r_{\min} = r_{\min}(P, k) = r_{\text{opt}}(P[t_{\min}], k)$. In the following we assume that the points of $P[t]$ are moving according to a *polynomial motion* of degree μ. Namely, the position of a point $p[t] \in P[t]$ is defined at time t to be $(p_1(t), \ldots, p_d(t))$, where $p_1(\cdot), \ldots, p_d(\cdot)$ are polynomials of degree at most μ.

Definition 3.1. Let P be a moving point set as above. A partition of P into m sets $\mathcal{U} = \{U_1, \ldots, U_m\}$ is a (c, m, k)-*static clustering* if

$$\mathrm{r}(\mathcal{U}[t]) = \max_i \mathrm{r}(U_i[t]) \leq c \cdot r_{\text{opt}}(P[t], k),$$

for any $t \in \mathbb{R}$, where $\mathrm{r}(U_i[t])$ is the radius of the smallest ball that contains the points of $U_i[t]$.

In this section we prove the existence of a small (c, m, k)-static clustering for points with a bounded degree motion, where c and m depend on the dimension and the degree of motion of the points. To appreciate the above result, note that the motion of the points changes the distances between the points continuously. Points which are at a certain time far, become close, and far again. Although the motion is not chaotic, the underlying structure of the optimal clustering of the moving points is unstable, changing discretely (and considerably) even if points move only a short distance. See Fig. 1.

We map each moving point $p[t] = (p_1(t), \ldots, p_d(t))$ to a curve $\gamma = \bigcup_t (p_1(t), \ldots, p_d(t), t)$ in $d + 1$ dimensions, so that the first d coordinates of the intersection point of γ with the axis-parallel hyperplane $x_{d+1} = t$ is the point $p[t]$. Abusing notation, we refer interchangeably to P as a set of curves in $\mathbb{R}^{d+1}$ or as a set of moving points in $\mathbb{R}^d$.

In the following we apply a sequence of transformations to each point of P. As we transform a point, we remember its index in the original set.

Lemma 3.2. *Let P' be a set of curves resulting by translating each curve of P horizontally by a vector of length $\leq \delta = \rho\, r_{\min}(P,k)$, where $\rho \leq 1$. Then, for any (c,m,k)-static clustering of P', the corresponding clustering of P is $((1+c)\rho + c, m, k)$-static clustering.*

Proof. Let $\mathcal{U}' = \{U'_1, \ldots, U'_m\}$ be a (c,m,k)-static clustering of P', and let $\mathcal{U} = \{U_1, \ldots, U_m\}$ be the corresponding clustering of P. We have by definition that

$$\mathrm{r}(\mathcal{U}'[t]) = \max_{U' \in \mathcal{U}'} \mathrm{r}(U'[t]) \leq c r_{\mathrm{opt}}(P'[t],k).$$

Thus,

$$\begin{aligned}\mathrm{r}(\mathcal{U}[t]) &\leq \mathrm{r}(\mathcal{U}'[t]) + \delta \leq c \cdot r_{\mathrm{opt}}(P'[t],k) + \delta \leq c(r_{\mathrm{opt}}(P[t],k) + \delta) + \delta \\ &= c \cdot r_{\mathrm{opt}}(P[t],k) + (c+1)\delta \leq c \cdot r_{\mathrm{opt}}(P[t],k) + (c+1)\rho \cdot r_{\mathrm{opt}}(P[t],k) \\ &\leq ((1+c)\rho + c) r_{\mathrm{opt}}(P[t],k).\end{aligned}$$

□

Definition 3.3. For a ball B in $\mathbb{R}^d$ and $x \in \mathbb{R}$, we refer to the set $B \times \{x\} \subseteq \mathbb{R}^{d+1}$ as a *horizontal disk*. For a moving point set P in $\mathbb{R}^d$, a set $\mathcal{D}$ of horizontal disks $D_1, \ldots, \mathcal{D}_m$ in $\mathbb{R}^{d+1}$ is a (c,m,k)-*stabbing* of P if each of the curves of P passes through one of the disks of $\mathcal{D}$, and $\mathrm{r}(\mathcal{D}) = \max_{D \in \mathcal{D}} \mathrm{r}(D) \leq c\, r_{\min}(P,k)$.

Similarly, a set of points $X \subseteq \mathbb{R}^{d+1}$ *stabs* P if each curve of P contains at least one point of X.

Example 3.4. For a moving point set P, let $\mathcal{D} = \{D_1, \ldots, D_k\}$ denote the set of k horizontal d-dimensional disks in $\mathbb{R}^d$ realizing the optimal k-clustering of P at time $t_{\min}(P,k)$. Clearly, $\mathcal{D}$ is a $(1,k,k)$-stabbing of P.

Lemma 3.5. *Let P be a set of moving points in $\mathbb{R}^d$, where the algebraic degree of motion is μ, and let $\mathcal{D}$ be a (ρ,m,k)-stabbing of P. Then one can translate each curve of P horizontally by a vector of length at most $\delta = \rho\, r_{\min}(P)$, and $r_{\mathrm{opt}}(P[t],k) \leq r_{\mathrm{opt}}(P'[t],k) + \delta \leq r_{\mathrm{opt}}(P[t],k) + 2\delta$, P' is stabbed by a set of points $X \subseteq \mathbb{R}^d$, and $|X| = m$. Moreover, for any (c,m,k)-static clustering of P', the corresponding clustering of P is $((1+c)\rho + c, m, k)$-static clustering.*

Proof. Let $D_1, \ldots, D_m$ denote the m horizontal d-dimensional disks in $\mathcal{D}$. Let $\mathcal{A}_1, \ldots, \mathcal{A}_k$ denote the partition of P into the sets as induced by $D_1, \ldots, D_m$. Namely, p is in $\mathcal{A}_i$ if p stabs D_i. If a moving point (i.e., a curve) is stabbing several such disks, we assign it to one of the candidate sets arbitrarily.

Let ξ_i denote the center of D_i. Let $\mathcal{B}_i$ denote the set of curves resulting from translating each curve of $\mathcal{A}_i$ *horizontally*, so that it passes through ξ_i. Let $P' = \bigcup_i \mathcal{B}_i$.

Observe that

$$r_{\mathrm{opt}}(P'[t],k) \leq r_{\mathrm{opt}}(P[t],k) + \delta \quad \text{and} \quad r_{\mathrm{opt}}(P[t],k) \leq r_{\mathrm{opt}}(P'[t],k) + \delta,$$

as the distance between a point in P and its translated image in P' is at most $\delta = r_{\min}(P)$ at any time.

Finally, given a (c, m, k)-static clustering of P', its corresponding clustering of P is $((1 + c)\rho + c, m, k)$-static clustering, as implied by Lemma 3.2. □

A natural interpretation of Lemma 3.5 is that it partitions the set of curves P' into m families, where each family of curves passes through a common point.

Let P be a set of moving points in $\mathbb{R}^d$, all passing through a common point ξ at time t'. We can consider $P[t]$ to be a set of curves in $\mathbb{R}^{d+1}$, and we are interested in finding a set of horizontal disks that stabs all the curves. Note that this measure in insensitive to translation. In particular, we can translate the curves of P so that (ξ, t') is mapped to the origin. Let P' denote the resulting point set. Clearly, P and P' are equivalent, and one can perform the clustering for P' instead of P. Furthermore, for $t = 0$ all the points of P' are located at the origin.

Lemma 3.6. *Let P be a set of moving points in $\mathbb{R}^d$ with algebraic degree of motion μ, such that all of them are at the origin at time $t = 0$. Then there exists a mapping $f(\cdot)$ of a point of $P[t]$ into a moving point of polynomial motion of degree $\mu - 1$, so that for any $A \subseteq P$, we have $\mathrm{r}(f(A[t])) = \mathrm{r}(A[t])/t$. In particular, given a (c, m, k)-static clustering $U_1, \ldots, U_m$ of $f(P)$, then $f^{-1}(U_1), \ldots, f^{-1}(U_m)$ is a static (c, m, k)-static clustering of P.*

Proof. Let $p[t] = (p_1(t), p_2(t), \ldots, p_d(t))$ be any point of $P[t]$. We have $p[0] = (0, 0, \ldots, 0)$ by assumption. In particular, $p_1(0) = \cdots = p_d(0) = 0$. Thus, all the polynomials $p_1(t), \ldots, p_d(t)$ have a constant term which is zero. Namely, $p_i(t)/t$ is a polynomial of degree at most $\mu - 1$. Let $f(p) = (p_1(t)/t, \ldots, p_d(t)/t)$.

Let $Q = f(P)$. By the above, Q is a set of points moving with motion of degree at most $\mu - 1$.

For any $p, q \in P$, we have $\|p[t]q[t]\| = \|f(p[t])f(q[t])\| \cdot t$. In particular, for any $A \subseteq P$, we have $\mathrm{r}(A[t]) = \mathrm{r}(f(A[t])) \cdot t$. In particular, $r_{\mathrm{opt}}(P[t], k) = r_{\mathrm{opt}}(Q[t], k) \cdot t$. Thus, given a (c, m, k)-static clustering of Q by $U_1, \ldots, U_m$, we then have

$$\mathrm{r}(f^{-1}(U_i[t])) = \mathrm{r}(U_i[t]) \cdot t \leq c \cdot t \cdot r_{\mathrm{opt}}(Q[t], k) = c \cdot r_{\mathrm{opt}}(P[t], k),$$

for all $t \in \mathbb{R}$. □

Lemma 3.7. *Let P be a set of moving points in $\mathbb{R}^d$, where the degree of motion is μ. There exists a partition of $P[t]$ into $m = k^{\mu+1}$ sets $P_1, \ldots, P_m$, so that $\mathrm{r}(P_i[t]) \leq (2^{\mu+1} - 1)r_{\mathrm{opt}}(P[t], k)$. Namely, there exists a $(2^{\mu+1} - 1, k^{\mu+1}, k)$-static clustering of P.*

Proof. The proof is by induction on μ. For $\mu = 0$, the points are stationary, and as such the claim trivially holds as the clustering is independent of time. In particular, we have for $\mu = 0$ an $(\alpha(0), \varphi(0), k)$-static clustering, where $\alpha(0) = 1$, and $\varphi(0) = k$.

For $\mu > 0$, by Example 3.4, there exists a $(1, k, k)$-stabbing of P and by Lemma 3.5 one can map the points of P (by translating each one of them horizontally by a vector of length $\leq r_{\min}(P, k)$) into a set P' so that the curves of P' are being stabbed by a set of k points (i.e., the centers of the horizontal disks of the $(1, k, k)$-stabbing). Let $\mathcal{A}'_1, \ldots, \mathcal{A}'_k$

be the partition of P' into sets, so that each set passes through a single common point. By Lemma 3.6 there is a mapping of each $\mathcal{A}'_i$ into a set of moving points $\mathcal{B}'_i$, so that the degree of motion of $\mathcal{B}'_i$ is $\mu - 1$. By induction each $\mathcal{B}'_i$ has an $(\alpha(\mu-1), \varphi(\mu-1), k)$-static clustering. By Lemma 3.6 the corresponding partition is also an $(\alpha(\mu-1), \varphi(\mu-1), k)$-static clustering of $\mathcal{A}'_i$. In particular, since $r_{\text{opt}}(\mathcal{A}'_i[t], k) \leq r_{\text{opt}}(P'[t], k)$, we have that the clustering resulting from doing this clustering process to each $\mathcal{A}'_i$ separately, results in an $(\alpha(\mu-1), k\varphi(\mu-1), k)$ static clustering of P'. However, by Lemma 3.5, such a clustering corresponds to a $(2\alpha(\mu-1)+1, k \cdot \varphi(\mu-1), k)$-static clustering of P. It is now follows that $\alpha(\mu) = 2\alpha(\mu-1)+1 = 2 \cdot (2^\mu - 1) + 1 = 2^{\mu+1} - 1$, and $\varphi(\mu) = k^{\mu+1}$. □

In essence, Lemma 3.7 reduces a rather obnoxious clustering problem (clustering of moving points) into a somewhat strange but conventional clustering problem.

Somewhat surprisingly, the $k^{\mu+1}$ term in Lemma 3.7 is tight if one is interested in a constant factor approximation. Indeed, let $\gamma_{i,j}$ be the line passing through the points $(i, 0)$ and $(j, 1)$, for some $1 \leq i, j \leq k$, and let L denote the set of all such lines. This corresponds to a set of linearly moving points in $\mathbb{R}$ (i.e., the y-axis is the time axis). Note that $r_{\text{opt}}(L[0], k) = r_{\text{opt}}(L[1], k) = 0$. Let $L_1, \ldots, L_m$ be any static clustering of L with $m < k^2$. One of those static clusters, L_j, must have at least two lines of L. In particular, $\max(\mathrm{r}(L_j(0)), \mathrm{r}(L_j(1))) \geq 1$ which implies that this is unbounded error compared with $r_{\text{opt}}(L(0))$ or $r_{\text{opt}}(L(1))$ which are both zero. This example can be easily extended to a point set with polynomial motion μ, showing that $k^{\mu+1}$ static clusters are necessary.

No static clustering can be competitive if insertions and deletions are allowed, even for non-moving points. Indeed, let us assume that we are given points in a online fashion. Once the point has been received, we have to assign it immediately to one of the static clusters $c_1, \ldots, c_m$, so that at any time this clustering is competitive with the optimal k-clustering. To see why such an algorithm is not possible, let the sequence of points (on the real line) be $x_i = (3+i)^i$, for $i = 1, \ldots, m+1$. Note that the optimal k-clustering for $P_i = \{x_1, \ldots, x_i\}$ has x_i as a singleton in the clustering (since the distance between x_i and x_{i-1} is larger than the distance between x_{i-1} and x_1). In particular, $x_1, \ldots, x_{m+1}$ must all be in disjoint clusters, but we are restricted to m clusters—a contradiction.

Using the same construction in the other direction shows that no static clustering can support deletions. Interestingly enough, one can combine the static clustering together with the kinetic data structure approach so one can support insertions and deletions. See Section 5. The case where only insertions (and mergings) are allowed for the stationary case was investigated by Charikar et al. [CCFM].

One can extend Lemma 3.7 by using tighter stabbings.

Theorem 3.8. *Let $\mathcal{A}$ be an algorithm that computes a (c, m, k)-stabbing of a moving point set. Then, for a moving point set P, with a degree of motion μ, one can compute a $((c+1)^{\mu+1} - 1, m^{\mu+1}, k)$-static clustering of P.*

Proof. The proof follows the same inductive argument of Lemma 3.7. For $\mu = 0$, the points are stationary, and $\mathcal{A}$ provides a (c, m, k)-static clustering of P. Let $\alpha(0) = c$ and $\varphi(0) = m$.

For $\mu > 0$, we compute, using $\mathcal{A}$, a (c, m, k)-stabbing of P, and we move each curve of P to the center of its stabbing disk. This partitions the resulting point set P' into m sets: $A_1, \ldots, A_m$. By Lemma 3.6 we can treat each of the A_i as having degree of motion $\mu - 1$. By induction, for each A_i one can compute an $(\alpha(\mu-1), \varphi(\mu-1), k)$-static clustering. Combining those sets together results in an $(\alpha(\mu-1), \varphi(\mu-1)m, k)$-static clustering of P'. By Lemma 3.2, this is a $((1+\alpha(\mu-1))c+\alpha(\mu-1), \varphi(\mu-1)m, k)$-static clustering of P.

Thus, $\varphi(\mu) = m^{\mu+1}$, and

$$\begin{aligned}
\alpha(\mu) &= (1+\alpha(\mu-1))c + \alpha(\mu-1) = c + (c+1)\alpha(\mu-1) \\
&= \sum_{i=1}^{\mu} c(c+1)^{i-1} + (c+1)^{\mu}\alpha(0) = c\left(\sum_{i=1}^{\mu}(c+1)^{i-1} + (c+1)^{\mu}\right) \\
&= c\sum_{i=1}^{\mu+1}(c+1)^{i-1} = c\frac{1-(c+1)^{\mu+1}}{1-(c+1)} \\
&= (c+1)^{\mu+1} - 1.
\end{aligned}$$

Overall, this results in a $((c+1)^{\mu+1}-1, m^{\mu+1}, k)$-static clustering of P. □

Lemma 3.9. *Given a (c, m, k)-stabbing $\mathcal{D}$ of a moving point set P in $\mathbb{R}^d$, one can compute an $(\varepsilon, O(m(c/\varepsilon)^d), k)$-stabbing of P.*

Proof. Cover each disk of $\mathcal{D}$ by $O(1/\varepsilon^d)$ equal size disks (for example, by covering such a disk by a grid of size $\leq (\varepsilon/\sqrt{d})r_{\min}(P, k)$), such that each one of them is of radius $\leq \varepsilon r_{\min}(P, k)$. The resulting set of disks is clearly an $(\varepsilon, O(m/\varepsilon^d), k)$-stabbing of P. □

Thus, to compute static clustering of a moving point set, we need to be able to compute a (c, m, k)-stabbing quickly. It is unclear how one can compute quickly (even approximately) a (c, m, k)-stabbing of a moving set of points. Nevertheless, it is quite easy to come up with a "slow" algorithm for computing such stabbing (intuitively, you just enumerate all possible stabbings, and return the best one). Thus, using this together with the speedup technique of Section 2 results in a reasonably fast algorithm for computing static clustering, as described in the following section.

4. A Motion-Clustering Algorithm

In this section we describe how to compute the static clustering of Section 3 using the techniques of Section 2.

Lemma 4.1. *Given a set of n moving points $P[t]$ with algebraic degree of motion μ, and a center $\xi[t] \in P[t]$, one can compute the time t' when $\mathrm{r}(P[t], \xi[t])$ is minimized, where $\mathrm{r}(P[t], \xi[t]) = \max_{p\in P}\|p[t]\xi[t]\|$. This takes $O(\lambda_{2\mu+2}(n)\log n)$ time, where $\lambda_t(n)$ is the maximum length of a Davenport–Schinzel sequence of order t.*

Proof. Let $f_i(t) = \|p_i[t]\xi[t]\|$, for $i = 1, \ldots, n$. Clearly, finding the time when the disk centered at $\xi[t]$ has minimum radius and covers all the points of $P[t]$, is equivalent to finding the lowest point in the upper-envelope $\max_i f_i(t)$. This can be done in $O(\lambda_{2\mu+2}(n) \log n)$ time [SA]. □

Remark 4.2. Lemma 4.1 can be extended to find the optimal radius for a static clustering. Namely, we are given k sets $P_1[t], \ldots, P_k[t]$, and respective centers $\xi_1[t] \in P_1[t], \ldots, \xi_k[t] \in P_k[t]$, and we wish to compute the time where the k disk's centers at $\xi_1[t], \ldots, \xi_k[t]$ cover their respective sets, and the maximum radius is minimized. Clearly, this boils down to finding the lowest point in an upper-envelope of n functions (assuming $|P_1| + \cdots + |P_k| = n$), and thus can be computed in $O(\lambda_{2\mu+2}(n) \log n)$ time.

Definition 4.3. Given a set of points $P[t]$ in $\mathbb{R}^d$, we associate with any pair of points $(p, q) \in P[t] \times P[t]$ the distance between the two points. The *signature* of $P[t]$ at time t is the ordering of all those $\binom{n}{2}$ pairs by their length.

Lemma 4.4. *Given a point set $P[t]$ whose signature is constant in the interval $t \in [a, b]$, the partition induced by the greedy clustering $[G]$ generated for any $t \in [a, b]$ is the same, and provides a $(2, k, k)$-static clustering in this interval. In particular, one can compute a 2-approximation to the minimum of $r_{\text{opt}}(P[t], k)$ on the interval $[a, b]$ in $O(n\lambda_{2\mu+2}(n) \log n)$ time.*

Proof. Since the algorithm of Gonzalez [G] bases its decisions on comparing distances between pairs of points of $P[t]$, it is clear that if the signature does not change, then the resulting partition remains the same for all $t \in [a, b]$.

This is a $(2, k, k)$-static clustering, because at any time $t \in [a, b]$ the algorithm of Gonzalez [G] generates a 2-approximation to $r_{\text{opt}}(P[t], k)$.

Computing the minimum radius of this partition in $[a, b]$ can be done in the time specified, using the algorithm of Remark 4.2. □

Lemma 4.5. *One can compute $O(n^4)$ sorted intervals that cover the real-line, in $O(n^4 \log n)$ time, so that inside each such interval $P[t]$ has the same signature.*

Proof. Consider a quadruple of points $p_1, p_2, q_1, q_2 \in P$. The equation $\|p_1[t]p_2[t]\| = \|q_1[t]q_2[t]\|$ has 2μ solutions, and those are the times when the pair p_1p_2 exchanges places with q_1q_2 in the signature. Repeating this for all possible quadruples results in all the times when the signature changes. Overall, there are $O(n^4)$ such events, and one can sort them in $O(n^4 \log n)$ time. □

Lemma 4.6. *Given a set of n moving points P in $\mathbb{R}^d$ with algebraic degree of motion μ, one can compute, in $O(n^4\lambda_{2\mu+2}(n) \log n)$ time, a $(2, k, k)$-stabbing of P.*

Proof. Use the algorithm of Lemma 4.5 to compute all the intervals where the signature remains the same. Apply for each interval the algorithm of Lemma 4.4. Return the set of disks computed with minimum radius. □

Lemma 4.7. *Given a set P of n moving points in $\mathbb{R}^d$, with algebraic degree of motion μ, one can compute a $(2, 15k, k)$-stabbing of P, for all $k = O(n^{1/14})$, in $O(nk)$ time.*

Proof. We use Theorem 2.9, but for the sake of simplicity we use the naive point-location data structure (i.e., for each curve, we scan the disks and decide whether it stabs the disk or not). We set $\eta = 14$ and $\varepsilon = (k/n)^{1/(\eta+1)}$. The samples used by the algorithm are of size $m = O((k \log n)/\varepsilon) = O(n^{1/(1+\eta)} k^{\eta/(1+\eta)} \log n)$. Using the algorithm of Lemma 4.6 such a sample can be 2-approximate k-clustered in

$$\begin{aligned} T_C(m) &= O(m^4 \lambda_{2\mu+2}(m) \log m) = O(m^{5.5}) = O((n^6 k^{6\eta})^{1/(1+\eta)}) \\ &= O((n^{6+6\eta/14})^{1/(1+\eta)}) = O(n^{(84+6\eta)/14(1+\eta)}) = O(n^{168/210}) = O(n) \end{aligned}$$

time, as $k = O(n^{1/14})$ and $\eta = 14$. Thus, the overall running time of the algorithm of Theorem 2.9 is $O(nk + (\eta + 1)T_C(m)) = O(nk)$, since we spend $O(k)$ on each point-location query, and we perform $O(\eta n)$ point-location queries. □

Lemma 4.7 can be easily improved, by using more advanced data structures. However, since such data structures are rather complicated, we opted for a slower algorithm which is reasonably fast.

Putting Lemma 4.7, together with Theorem 3.8, we have:

Theorem 4.8. *Let P be a set of moving points in $\mathbb{R}^d$, where the algebraic degree of motion is μ. One can compute a $(3^{\mu+1} - 1, (15k)^{\mu+1}, k)$-static clustering of P in $O(nk)$ time, for $k = O(n^{1/14})$.*

5. Using Coresets to Perform Insertions and Deletions

In this section we show how to maintain static clustering under insertions and deletions.

Theorem 5.1. *Let P be a set of moving points in $\mathbb{R}^d$ with algebraic degree of motion μ, and $0 < \varepsilon < \frac{1}{2}$ a parameter. One can compute an $(\varepsilon, O((k/\varepsilon^d)^{\mu+1}), k)$-static clustering of P in $O(nk/\varepsilon^d)$ time, for all $k = O(n^{1/14}\varepsilon^d)$.*

Proof. We convert a $(2, 15k, k)$-stabbing of Lemma 4.7 into an $(\varepsilon/c, O(k/\varepsilon^d), k)$-stabbing of P, using Lemma 3.9, where c is a constant to be specified shortly.

Using such a stabbing in the algorithm of Theorem 3.8 results in an (a, b, k)-static clustering $\mathcal{U}$ of P, where $a = (1 + \varepsilon/c)^{\mu+1} - 1$ and $b = O((k/\varepsilon^d)^{\mu})$. For $c = 2(\mu + 1)$, we have

$$a = \left(1 + \frac{\varepsilon}{c}\right)^{\mu+1} - 1 \leq \exp\left(\frac{\varepsilon}{2(\mu+1)}(\mu+1)\right) - 1 \leq 1 + 2\left(\frac{\varepsilon}{2}\right) - 1 = \varepsilon,$$

since $1 + x \leq e^x \leq 1 + 2x$ for $0 \leq x \leq \frac{1}{2}$. Thus, $\mathcal{U}$ is an $(\varepsilon, O((k/\varepsilon^d)^{\mu+1}), k)$-static clustering of P □

Theorem 5.1 implies that one can extract a small subset from P that represents the point set, up to factor of ε, as far as k-clustering at any time is concerned.

Definition 5.2. Let P be a point set in $\mathbb{R}^d$, and let $\frac{1}{2} > \varepsilon > 0$ be a parameter. A set $Q \subseteq P$ is an *ε-coreset* of P if for any set of k balls $\mathcal{C}$ that covers Q, we have that P is covered by $\mathcal{C}(\varepsilon r_{\text{opt}}(P, k))$, where

$$\mathcal{C}(\delta) = \{\text{ball}(p, r + \delta) | \text{ball}(p, r) \in \mathcal{C}\},$$

where $\text{ball}(p, r)$ denotes the ball of radius r centered at p.

If P' is a set of moving points, then $Q' \subseteq P'$ is an *ε-coreset* of P' if $Q'[t]$ is an ε-coreset of $P'[t]$, for all t.

For recent work on coresets, see [AH], [HV2], and [BHI].

Lemma 5.3. *For P, k as above, and $0 < \varepsilon < \frac{1}{2}$, the following hold*:

(i) *If Q is an ε-coreset of P, then $(1 - \varepsilon) r_{\text{opt}}(P, k) \leq r_{\text{opt}}(Q, k) \leq r_{\text{opt}}(P, k) \leq (1 + 2\varepsilon) r_{\text{opt}}(Q, k)$.*

(ii) *If Q is an ε-coreset of P and Q' is a coreset of P', then $Q \cup Q'$ is a coreset of $P \cup P'$.*

(iii) *If $Q_1 \subseteq Q_2 \subseteq \cdots \subseteq Q_m$, where Q_i is an ε-coreset of Q_{i+1}, for $i = 1, \ldots, m - 1$, then Q_1 is an δ-coreset of Q_m, where $\delta = \varepsilon m e^{2\varepsilon m}$.*

Proof. (i) Clearly, $r_{\text{opt}}(Q, k) \leq r_{\text{opt}}(P, k)$, as $Q \subseteq P$. Furthermore,

$$r_{\text{opt}}(Q, k) + \varepsilon r_{\text{opt}}(P, k) \geq r_{\text{opt}}(P, k),$$

as we can convert the optimal k-clustering of Q into a clustering of P, by expanding each ball by $\varepsilon r_{\text{opt}}(P, k)$. Thus, $r_{\text{opt}}(Q, k) \geq (1 - \varepsilon) r_{\text{opt}}(P, k)$. Also,

$$(1 + 2\varepsilon) r_{\text{opt}}(Q, k) \geq (1 + 2\varepsilon)(1 - \varepsilon) r_{\text{opt}}(P, k) \geq r_{\text{opt}}(P, k),$$

as $\varepsilon < \frac{1}{2}$.

(ii) Let $\mathcal{C}$ be a k-clustering of $Q \cup Q'$. By definition, $P \subseteq \mathcal{C}(\varepsilon r_{\text{opt}}(P, k))$ and $P' \subseteq \mathcal{C}(\varepsilon r_{\text{opt}}(P', k))$, as $\mathcal{C}$ is a k-clustering of Q and Q' separately. Finally, $r_{\text{opt}}(P, k)$, $r_{\text{opt}}(P', k)$
$\leq r_{\text{opt}}(P \cup P', k)$. Thus, $P \cup P' \subseteq \mathcal{C}(\varepsilon r_{\text{opt}}(P \cup P', k))$.

(iii) Let $\mathcal{C}_1$ be a clustering of Q_1 by k balls, and let $\mathcal{C}_i = \mathcal{C}_{i-1}(\varepsilon r_{\text{opt}}(Q_i, k))$, for $i = 2, \ldots, m$. By induction $Q_m \subseteq \mathcal{C}_m$. Furthermore, $\mathcal{C}_m = \mathcal{C}_1(\rho)$, where

$$\begin{aligned} \rho &= \sum_{i=2}^{m} \varepsilon r_{\text{opt}}(Q_i, k) \leq \varepsilon \sum_{i=2}^{m} (1 + 2\varepsilon)^{m-i} r_{\text{opt}}(Q_m, k) \leq \varepsilon m e^{2\varepsilon m} r_{\text{opt}}(Q_m, k) \\ &= \delta \cdot r_{\text{opt}}(Q_m, k), \end{aligned}$$

as $1 + x \leq e^x$, for $0 \leq x \leq 1$, and $\delta = \varepsilon m e^{2\varepsilon m}$. We conclude that $Q_m \subseteq \mathcal{C}_1(\delta r_{\text{opt}}(Q_m, k))$, and thus Q_1 is a δ-coreset of Q_m. □

Note that Lemma 5.3 holds verbatim if the point sets are moving.

Theorem 5.4. *Let P be a set of moving points in $\mathbb{R}^d$ with algebraic degree of motion μ. One can compute an ε-coreset of P of size $O((k/\varepsilon^d)^{\mu+1})$, in $O(nk/\varepsilon^d)$ time.*

Proof. Compute an $(\varepsilon/2, O((k/\varepsilon^d)^{\mu+1}), k)$-static clustering of P using the algorithm of Theorem 5.1. Next pick arbitrarily a representative point from each static cluster, and let Q be the resulting point set. It is now easy to verify that Q is an ε-coreset of P. □

Note that we can now maintain the k-clustering of a moving point set by computing its ε-coreset, and clustering the coreset whenever we want the exact clustering.

Theorem 5.5. *One can maintain an ε-coreset for k-clustering of a point set moving in bounded polynomial motion in $\mathbb{R}^d$. The operations insertion, deletion, and motion update can all be handled in time* $\text{poly}(k, 1/\varepsilon, \log n)$, *where n is the maximum number of points in the data structure at any time. Furthermore, one can extract a $(2+\varepsilon)$-approximate k-clustering, at any time, in the same time bound.*

Proof. Clearly, once we have an ε-coreset of the current point set in the data structure, we can report the $2+\varepsilon$ approximate clustering by just computing it directly from the coreset using the algorithm of Gonzalez [G]. This would take $O(kM)$, where M is the size of the coreset.

So, we need to show how we can maintain the ε-coreset of a moving point set, under insertions and deletions (motion updates can be performed by a deletion followed by an insertion). The basic idea was introduced in [AH]: Let T be a balanced binary tree that stores the points in its leaves, and store inside each interior node v of T a coreset of the points stored in the subtree of v. Let this set be denoted by $\text{coreset}(v)$.

If u and w are the two children of v, we can compute $\text{coreset}(v)$ by computing a δ-coreset of $\text{coreset}(u) \cup \text{coreset}(w)$ using the algorithm of Theorem 5.4, where δ is to be specified shortly. Clearly, insertions and deletions can be performed by doing rotations in T, and we can always fix the coresets of the changed nodes by recomputing them from their children coresets. Clearly, all this can be done while keeping the tree depth smaller than $5 \log n$, and the running time is $O(\text{poly}(1/\delta, k, \log n))$.

We now apply Lemma 5.3(iii), with $m = 5 \log n$ and δ being the approximation factor. Clearly, $\text{coreset}(root(T))$ is a ρ-coreset of the points stored in T, where

$$\rho = \delta m e^{2\delta m} = 5 e^{10\delta \log n} \delta \log n.$$

In particular, setting $\delta = \varepsilon/(20 \log n)$, we have $\rho = \varepsilon e^{1/2}/4 \leq \varepsilon$. Namely, $\text{coreset}(root(T))$ is an ε-coreset of all the points stored in the tree.

As for the running time, one can verify that all the operations can be performed in $O(\text{poly}(1/\varepsilon, k, \log n))$. Finally, computing the k-clustering from the root coreset takes $O(kM) = O(\text{poly}(k, 1/\varepsilon, \log n))$ time as $M = O(\text{poly}(k, 1/\varepsilon, \log n))$. □

The solution of Theorem 5.5 might be of limited use in practice, as sometime we would like every point to be marked by the current cluster that it is contained in. One

option is to take the ε-coreset maintained by the data structure of Theorem 5.5 and explicitly maintain its clustering, using a kinetic data structure (KDS) as done in the following (naive) result. For more information about KDS and relevant definitions, see [BGH].

Lemma 5.6. *Let P be a set of moving points in $\mathbb{R}^d$. One can maintain a 2-approximate k-center clustering of P, such that at any time there are $O(nk)$ active certificates. Furthermore, when one of the certificates is violated, the clustering and the associated set of certificates can be recomputed in $O(nk)$ time.*

Moreover, if the points move with polynomial motion of degree μ, then there would be at most $O(n^4)$ events throughout the execution of the algorithm.

Proof. The algorithm follows the following naive approach: We run Gonzalez's algorithm [G] on P at the current time t, and we compute for each comparison of the type $\|x[t]y[t]\| \leq \|z[t]w[t]\|$ a certificate (one can implement Gonzalez's algorithm so that it uses only this kind of comparison), where $x, y, z, w \in P$. Whenever one of these certificates is being violated, we stop and recompute the clustering and the set of certificates.

Finally, the bound on the number of events follows immediately from Lemma 4.6. Note that at any point in time, we know for every point which cluster it belongs to, and the current set of clusters. □

Note that this result is far from satisfactory from a KDS point of view, as the number of events and the time spent on each event are prohibitive. However, using coresets we can trade off between accuracy and the number of events. Namely, we can maintain the k-clustering of the coreset using Lemma 5.6. This results in a KDS handling $O(n \operatorname{poly}(k, 1/\varepsilon, \log n))$ events (instead of $O(n^4)$) that maintains a $(2+\varepsilon)$-approximate k-clustering. The details are straightforward, and we omit them here.

6. Approximate k-Center Clustering

In this section we show how one can compute a 2-approximate clustering in expected linear time. We achieve this by using the speed-up technique of Section 2, together with some ideas from Feder and Greene [FG].

Observe that in some clustering problems, one can replace the point-location data structure by an approximate point-location data structure. In this case we are allowed to inflate the clusters slightly, and the data structure has to decide whether the points are inside the inflated clusters (in fact, we allow uncertainty in this region).

Definition 6.1. A *γ-approximate point-location data structure* is a data structure that receives a set $\mathcal{C}$ of u clusters as input and preprocesses them in $T_{PA}(u)$ time, so that given a point p it can decide in $O(T_Q(m))$ time whether a point lies inside $\bigcup \mathcal{C}$, or alternatively is in distance larger than $\gamma \mathrm{r}(\mathcal{C})$ from $\bigcup \mathcal{C}$. If the point is in the middle, returning either result inside/outside is allowed.

In low dimensions, constructing such an approximate point-location data structure is quite easy, as the following lemma demonstrates.

Definition 6.2. Given a set Q of balls in $\mathbb{R}^d$, all of radius at most l, let G be the uniform grid of side-length l, let $Z(Q, l)$ denote the set of cubes of G that intersects the balls of Q, and let $\mathcal{U}_G(Q, l) = \left(\bigcup_{z \in Z(Q,l)} z\right)$ denote the union of those cubes. Note that if a point is inside $\mathcal{U}_G(Q, l)$, then it is in distance at most $(1 + \sqrt{d})l$ from one of the centers of Q.

Lemma 6.3. *Given a set Q of k balls in $\mathbb{R}^d$ of radius at most l, one can preprocess them in $O(k)$ time, so that given a query point, one can decide, in $O(1)$ time, whether a point is inside $\mathcal{U}_G = \mathcal{U}_G(Q, l)$. In particular, given a set P of n points, one can compute all the points in $P \setminus \mathcal{U}_G$ in $O(n + k)$ time.*

Proof. Compute $Z(Q, l)$ in $O(k)$ time. Since all the cubes of $Z(Q, l)$ are cells in a grid, one can answer a point-location query that decides for a point q whether or not it is inside $\bigcup Z(Q, l)$, in $O(1)$ time, by using a hash-table to check if the grid cell that contains q is in $Z(Q, l)$. □

Lemma 6.4. *Given a set P of n points in $\mathbb{R}^d$, one can compute a $(4 + 2\sqrt{d}, k, k)$-clustering of P in $O(n)$ expected time, for $k \leq \sqrt[3]{n}/\log n$.*

Proof. We implement `FastCluster` by using the algorithm of Gonzalez [G] as the clustering subroutine. Gonzalez's algorithm computes a $(2, k, k)$-clustering of m points in $O(mk)$ time. We set $\varepsilon = \sqrt{k/n}$, and use a sample of size $m = O((k \log n)/\varepsilon) = O(\sqrt{nk} \log n)$. Thus, this stage takes $O(mk) = O(\sqrt{n}k^{3/2} \log n) = O(n)$ time, and we compute a k-clustering $\mathcal{C}''$. Next, for the point-location stage, we use the data structure of Lemma 6.3. This yields a $((\sqrt{d} + 1)l, 2k, k)$-clustering $\mathcal{C}'$ of P, where $l = \mathrm{r}(\mathcal{C}'')/r_{\mathrm{opt}}(P, k) \leq 2r_{\mathrm{opt}}(P, k)/r_{\mathrm{opt}}(P, k) = 2$. Thus, $\mathcal{C}'$ is a $(2\sqrt{d} + 2, 2k, k)$-clustering of P. By clustering representative points (each point of P is in distance $\leq (2\sqrt{d} + 2)r_{\mathrm{opt}}(P, k)$ from its closest representative point) from each cluster of $\mathcal{C}'$, and applying Gonzalez's algorithm [G] we get a $(2\sqrt{d} + 4, k, k)$-clustering $\mathcal{C}$ of P in $O(n + k^2) = O(n)$ expected time. □

We now show how to transform a (large) constant approximation to the optimal k-center clustering into a 2-approximation. The following is a simple extension of the algorithm of Feder and Greene [FG], and is presented here as the implementation of Feder and Greene [FG] relies on a structure constructed in an earlier stage of their algorithm, and our version is independent of this structure.

Lemma 6.5. *Let P be a set of n points in $\mathbb{R}^d$, and let k, r be parameters, so that $r_{\mathrm{opt}} \leq r \leq c \cdot r_{\mathrm{opt}}$, where $c \geq 1$ is a constant and $r_{\mathrm{opt}} = r_{\mathrm{opt}}(P, k)$. Then one can compute a 2-approximation to the optimal k clustering of P in $O(n + k \log k)$ time.*

Proof. Let G be a partition of $\mathbb{R}^d$ into a uniform grid, with side length $\Delta = r/(2c\sqrt{d}) \geq r_{\mathrm{opt}}/(2c\sqrt{d})$. Note that $\Delta \leq c \cdot r_{\mathrm{opt}}/(2c\sqrt{d}) = r_{\mathrm{opt}}/(2\sqrt{d})$. Compute for each of the

points of P the cell of G that contains it, and compute for each grid cell the points assigned to it (using hashing and bucketing), in overall linear time (note that we need the floor function to implement this efficiently).

Since each ball of the optimal clustering can intersect at most $(2r_{\text{opt}}(P,k)/\Delta+1)^d = O((4c\sqrt{d}+1)^d) = O(1)$ cells of the grid, it follows that two non-empty grid cells in distance at least $2r$ from each other cannot participate in the same cluster in the 2-approximation to the optimal clustering. We compute for each non-empty grid cell c of G (there are at most $O(k)$ such non-empty cells) the $O(1)$ grid cells in distance at most $2r$ from c which are not empty. For each cell c, let $N(c)$ denote this cell list of neighbors.

We now implement the algorithm of Gonzalez [G] on P. We remind the reader that the algorithm of Gonzalez [G] repeatedly picks the point furthest away from the current set of centers as the next center to be added, and updates for each point its distance to its closest center. In our setting, each time we pick a new center p, which lies in a grid cell c, we visit all the neighbors $N(c)$ of c and update the distance to the closest point for all the points of P stored in those cells. (Note that we do not have to update cells that are further away from c as they lie on different clusters in any 2-approximate clustering.) Furthermore, a cell can contain only a single center point, as its diameter is at most $\sqrt{d}\Delta < r_{\text{opt}}$, and thus can initiate such a distance update only once. In particular, the points stored in a cell c might be scanned at most $O(|N(c)|) = O(1)$ times. Furthermore, the algorithm maintains for each cell c the point of P inside it which is furthest away from the current set of centers. This representative point is the point that the algorithm might pick for its next center (i.e., since the algorithm always picks the next center to be the point which is furthest away from the current set of centers, the representative is the only point that is relevant for the algorithm inside the cell).

Since each point is being scanned a constant number of times, it follows that the overall running time associated with this operation is $O(n)$. Also, the algorithm needs to maintain a heap of the points of P that might serve as the next center. Here, the heap is sorted by the distance of the points to their nearest center, and the top of the heap is the point that maximizes this quantity. Instead of maintaining a heap for n points, we instead maintain a heap for the $O(k)$ representative points. Since each cell has a single such representative, and the algorithm performs $O(k)$ insertions, weight-update, and delete-max operations on this heap, it follows that we can implement each heap operation in $O(\log k)$ time. Thus, the overall running time of the algorithm is $O(n + k\log k)$ time. □

Combining Lemmas 6.4 and 6.5, we establish the following result:

Theorem 6.6. *Given a set P of n points, one can compute a 2-approximate k-clustering of P, in $O(n)$ expected time, for $k = O(\sqrt[3]{n}/\log n)$.*

Note that the algorithm of Theorem 6.6 is very simple. It boils down to repeated usage of random sampling, hashing, and using the (rather simple) clustering algorithm of Gonzalez [G] as a subroutine.

7. Conclusions

In this paper we proved that there is a small static clustering of any set of moving points and presented an efficient algorithm for computing this clustering. We also show how to maintain static clustering efficiently under insertions and deletions. No previous results of this type were known for moving points, and this presents to our knowledge the first efficient dynamic data structure for maintaining (approximate) k-center clustering of a moving point set.

We also described a simple technique for speeding-up clustering algorithms. In a certain sense, our speeding-up technique implies that clustering is easy if one is willing to compromise on the number of clusters used.

We use it to derive a linear time algorithm for 2-approximate k-center clustering in Euclidean space. We believe that the ability to do 2-approximate clustering in (expected) linear time, even under our strong computation model, is interesting and surprising. Our result can be interpreted as extending the techniques of Rabin [S], [GRSS], [R] for closest-pair computation to clustering.

We note that our results about the small coreset for k-center clustering, implies that one can maintain such approximate clustering on a stream of points, using $O(\text{poly}(k, 1/\varepsilon, \log n))$ overall space (each insertion can be handled in similar time). This follows because any measure that has a small coreset can be streamed, see [AHV] for details.

One open question for further research is to develop an algorithm with a tradeoff between clustering quality and I/O efficiency.

Acknowledgments

The author thanks Pankaj Agarwal, Boris Aronov, Mark de Berg, Otfried Cheong, Alon Efrat, Jeff Erickson, Piotr Indyk, Mark Overmars, Lenny Pitt, Magda Procopiuc, and Micha Sharir for helpful discussions and suggestions. Neither last nor least, the author thanks Miranda R. Callahan for her comments on the manuscript. Finally, the author thanks again the anonymous referees for their insightful and useful comments on the writeup.

References

[AAE] P. K. Agarwal, L. Arge, and J. Erickson. Indexing moving points. In *Proc. 19th ACM Sympos. Principles Database Systems*, pages 175–186, 2000.

[ADPR] N. Alon, S. Dar, M. Parnas, and D. Ron. Testing of clustering. In *Proc. 41st Annu. IEEE Sympos. Found. Comput. Sci.*, pages 240–250, 2000.

[AH] P. K. Agarwal and S. Har-Peled. Maintaining the approximate extent measures of moving points. In *Proc. 12th ACM–SIAM Sympos. Discrete Algorithms*, pages 148–157, 2001.

[AHV] P. K. Agarwal, S. Har-Peled, and K. R. Varadarajan. Approximating extent measures of points. *J. Assoc. Comput. Mach.*, 2003, to appear.

[AP] P. K. Agarwal and C. M. Procopiuc. Approximation algorithms for projective clustering. In *Proc. 11th ACM–SIAM Sympos. Discrete Algorithms*, pages 538–547, 2000.

[BDIZ] J. Basch, H. Devarajan, P. Indyk, and L. Zhang. Probabilistic analysis for combinatorial functions of moving points. In *Proc. 13th Annu. ACM Sympos. Comput. Geom.*, pages 442–444, 1997.

[BE] M. Bern and D. Eppstein. Approximation algorithms for geometric problems. In D. S. Hochbaum, editor, *Approximationg Algorithms for NP-Hard Problems*, pages 296–345. PWS, Boston, MA, 1997.

[BGH] J. Basch, L. J. Guibas, and J. Hershberger. Data structures for mobile data. *J. Algorithms*, 31(1):1–28, 1999.

[BGSZ] J. Basch, L. J. Guibas, C. Silverstein, and L. Zhang. A practical evaluation of kinetic data structures. In *Proc. 13th Annu. ACM Sympos. Comput. Geom.*, pages 388–390, 1997.

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

[CCFM] M. Charikar, C. Chekuri, T. Feder, and R. Motwani. Incremental clustering and dynamic information retrieval. In *Proc. 29th Annu. ACM Sympos. Theory Comput.*, pages 626–635, 1997.

[dBvKOS] M. de Berg, M. van Kreveld, M. H. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*, 2nd edition. Springer-Verlag, New York 2000.

[DG] O. Devillers and M. Golin. Dog bites postman: point location in the moving Voronoi diagram and related problems. *Internat. J. Comput. Geom. Appl.*, 8:321–342, 1998.

[DHS] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification*, 2nd edition. Wiley-Interscience, New York, 2001.

[FG] T. Feder and D. H. Greene. Optimal algorithms for approximate clustering. In *Proc. 20th Annu. ACM Sympos. Theory Comput.*, pages 434–444, 1988.

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

[GGH+] J. Gao, L. Guibas, J. Hershberger, L. Zhang, and A. Zhu. Discrete mobile centers. In *Proc. 17th Annu. ACM Sympos. Comput. Geom.*, pages 188–196, 2001.

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

[HV1] S. Har-Peled and K. R. Varadarajan. Approximate shape fitting via linearization. In *Proc. 42nd Annu. IEEE Sympos. Found. Comput. Sci.*, pages 66–73, 2001.

[HV2] S. Har-Peled and K. R. Varadarajan. Projective clustering in high dimensions using core-sets. In *Proc. 18th Annu. ACM Sympos. Comput. Geom.*, pages 312–318, 2002.

[HW] D. Haussler and E. Welzl. ε-Nets and simplex range queries. *Discrete Comput. Geom.*, 2:127–151, 1987.

[I] P. Indyk. Sublinear time algorithms for metric space problems. In *Proc. 31st Annu. ACM Sympos. Theory Comput.*, pages 154–159, 1999.

[MOP] N. Mishra, D. Oblinger, and L. Pitt. Sublinear time approximate clustering. In *Proc. 12th ACM–SIAM Sympos. Discrete Algorithms*, pages 439–447, 2001.

[MT] N. Megiddo and A. Tamir. On the complexity of locating linear facilities in the plane. *Oper. Res. Lett.*, 1:194–197, 1982.

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

[S] M. Smid. Closest-point problems in computational geometry. In J.-R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*, pages 877–935. Elsevier North-Holland, Amsterdam, 2000.

[SA] M. Sharir and P. K. Agarwal. *Davenport–Schinzel Sequences and Their Geometric Applications*. Cambridge University Press, New York, 1995.

[VC] V. N. Vapnik and A. Y. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory Probab. Appl.*, 16:264–280, 1971.

Received September 19, 2001, *and in revised form May* 3, 2002, *and August* 24, 2003.
Online publication March 24, 2004.