MAX-PLANCK-INSTITUT
FÜR
INFORMATIK

Static and dynamic algorithms for k-point clustering problems

Amitava Datta Hans-Peter Lenhof
Christian Schwarz Michiel Smid

MPI–I–93–108 February 1993

INFORMATIK

Im Stadtwald
66123 Saarbrücken
Germany

Static and dynamic algorithms for k-point clustering problems

Amitava Datta Hans-Peter Lenhof
Christian Schwarz Michiel Smid

MPI–I–93–108 February 1993

Static and dynamic algorithms for k-point clustering problems*

Amitava Datta Hans-Peter Lenhof Christian Schwarz
Michiel Smid

Max-Planck-Institut für Informatik
W-6600 Saarbrücken, Germany

February 18, 1993

Abstract

Let S be a set of n points in d-space and let $1 \leq k \leq n$ be an integer. A unified approach is given for solving the problem of finding a subset of S of size k that minimizes some closeness measure, such as the diameter, perimeter or the circumradius. Moreover, data structures are given that maintain such a subset under insertions and deletions of points.

1 Introduction

We consider clustering problems of the following type. Given a set S of n points in d-dimensional space and an integer k between one and n, find a subset of S of size k that minimizes some closeness measure. As an example, we may want to minimize the perimeter of the convex hull of the k points. This measure was considered by Dobkin et al. [4]. Other measures were considered by Aggarwal et al. [1]. To be more precise, they gave algorithms for finding k points such that their diameter, or their enclosing square, or the perimeter of their enclosing rectangle is as small as possible. Smid [11] also considered the case of minimizing the enclosing square.

Eppstein and Erickson [5] give a general framework for solving such k-point clustering problems. They start by computing for each point its $\Theta(k)$ nearest neighbors, where the constant depends on the problem. Then they use this information to reduce the original problem to $O(n/k)$ subproblems for only $O(k)$ points each. Every single subproblem is solved by some other algorithm $\mathcal{A}$ for the k-point clustering problem in question. (In this reduction, the parameter k remains the same, but the size of the point set is reduced.) If $T(n,k)$ resp. $S(n,k)$ denote the time resp. space complexity of algorithm $\mathcal{A}$ running on a set of size n, then the entire running time of their algorithm is bounded by

$$O(n \log n + nk + \frac{n}{k} T(O(k), k)) \quad \text{if } d = 2$$

*This work was supported by the ESPRIT Basic Research Actions Program, under contract No. 7141 (project ALCOM II).

and

$$O(nk \log n + \frac{n}{k} T(O(k), k)) \quad \text{if } d > 2.$$

Moreover, their algorithm uses space

$$O(n \log n + nk + S(O(k), k)) \quad \text{if } d = 2$$

and

$$O(nk + S(O(k), k)) \quad \text{if } d > 2.$$

In this paper, we improve the results of [5] by generalizing techniques that were designed for closest pair problems. Using the search technique of [7], we also reduce the problem to $O(n/k)$ subproblems for $O(k)$ points each. Our reduction, however, is more direct and it circumvents the necessity to compute $\Theta(k)$ neighbors for each point. For any dimension $d \geq 2$, the resulting algorithm has a running time of

$$O(n \log n + \frac{n}{k} T(O(k), k))$$

and it uses space

$$O(n + S(O(k), k)).$$

Hence, our algorithm uses strictly less space than the one in [5], and our time bound does not exceed that of [5].

Eppstein and Erickson also consider the problem of maintaining the optimal k-point subset if points are inserted. In the planar case, their result is a data structure of size $O(n \log n + S(O(k), k))$ with an insertion time of

$$O(\log^2 n + k \log n + T(O(k), k)).$$

They mention that for higher dimensions their method gives results that are only slightly better than brute force.

We give a data structure that, for any dimension $d \geq 2$, maintains the optimal k-point subset in

$$O(\log n + T(O(k), k))$$

time per insertion, using only $O(n + S(O(k), k))$ space.

Eppstein and Erickson mention that no fully dynamic solutions, i.e., solutions that maintain the optimal solution under insertions and deletions of points, are known. We show that the technique of [10] can be generalized to give such a fully dynamic data structure. It uses

$$O(n \log^d(n/k) + S(O(k), k))$$

space and it has an amortized update time of

$$O(\log n \log^{d-1}(n/k) + \log^d(n/k) \log \log n + T(O(k), k) \log^d(n/k)).$$

Note that it is not a surprise that techniques for closest pair problems can be applied here: If $k = 2$, finding k points with minimal diameter is exactly the closest pair problem.

This paper is organized as follows. In Section 2, we define the class of problems that we can solve and we give the general algorithm for solving them. In order to apply this general algorithm, we need a variant of a grid. If we use a standard grid, then we

need the non-algebraic floor function to identify the grid-cell that contains a given point. In Section 3, we introduce a degraded grid that has basically the same properties as a standard grid, but for which we do not need the floor function. In this way, we get algorithms that fall inside the algebraic decision tree model. The notion of degraded grid we use is simpler than the one in [7]. In Section 4, we use the search method of [7] to construct a degraded grid, such that each grid box contains $O(k)$ points and at least one box contains at least k points. This grid is needed to reduce the k-clustering problem for the n points of our input set S to $O(n/k)$ subproblems for $O(k)$ points each.

In Section 5, we give several applications of our general algorithm. The results improve the previously best known algorithms. For an overview of the results, see Table 1 in Section 5.

In Section 6, we give the data structure that maintains the optimal k-point subset under insertions. Section 7 gives a data structure that supports both insertions and deletions. The results of these two sections are summarized in Tables 2 and 3. We finish the paper in Section 8 with some concluding remarks.

2 A general approach

Let S be a set of n points in d-dimensional space and let k be an integer such that $1 \leq k \leq n$. A d-dimensional axes-parallel rectangle of the form

$$[a_1 : b_1) \times [a_2 : b_2) \times \ldots \times [a_d : b_d),$$

where a_i and b_i, $1 \leq i \leq d$, are real numbers is called a *box*. If $b_i = a_i + \delta$ for all i, then the box is called a *δ-box*. The closure of a box, i.e., the product of d closed intervals $[a_i : b_i]$ is called a *closed box*. Throughout this paper, we will use the following notations:

- μ denotes a function that maps a set V of points in d-space to a real number $\mu(V)$, the *measure* of V.
- $P(S, k)$ denotes the problem of finding a subset of S of size k whose measure is minimal among all k-point subsets.
- $\mu_{opt}(S)$ denotes this minimal measure.
- S_{opt} denotes a k-point subset of S such that $\mu(S_{opt}) = \mu_{opt}(S)$.
- $\mathcal{A}$ denotes an algorithm that solves problem $P(S, k)$.
- $T(n, k)$ resp. $S(n, k)$ denote the time resp. space complexity of algorithm $\mathcal{A}$.

As an example, we can take for $\mu(A)$ the diameter of the set A. Then $P(n, k)$ is the problem of finding a subset of size k whose diameter is minimal among all k-point subsets. We make the following assumptions about the measure μ.

Assumption 1 *There exists a closed $\mu_{opt}(S)$-box that contains the optimal solution S_{opt}.*

Assumption 2 *There exists an integer constant c such that for any $\delta < \mu_{opt}(S)/c$, any closed δ-box contains less than k points of S.*

This constant c will be used throughout this paper.

For example, in Section 5 we will show that the diameter measure satisfies these assumptions with $c = \lceil \sqrt{d} \rceil$.

Remark: If the optimal solution S_{opt} is contained in a $(c'\mu_{opt}(S))$-box for some constant c', then we define $\mu' := c'\mu$ and search for a k-point subset S'_{opt} of S such that $\mu'(S'_{opt})$ is minimal among all k-point subsets. It is clear that $S'_{opt} = S_{opt}$ and that Assumption 1 holds for μ'.

Lemma 1 *Let δ be a real number. Assume there exists a closed δ-box that contains at least k points of S. Then, $\mu_{opt}(S) \leq c\delta$ and there exists a closed $(c\delta)$-box that contains the optimal solution S_{opt}.*

Proof: By Assumption 1, we only have to prove that $\mu_{opt}(S) \leq c\delta$. This follows from Assumption 2. ■

We show how to reduce problem $P(n,k)$ to $O(n/k)$ subproblems $P(S',k)$ for subsets S' of size $O(k)$. Each of these subproblems is then solved using algorithm $\mathcal{A}$. First, we need the following

Definition 1 *Let δ be a positive real number, let $\alpha \leq \beta$ be positive integers, and let R be a collection of δ-boxes such that*

1. *each box in R contains at least one point of S,*
2. *each point of S is contained in exactly one box of R,*
3. *there is a box in R that contains at least α points of S,*
4. *each box in R contains at most β points of S.*

Then R is called an $(\alpha, \beta; \delta)$-covering of S.

Now we can give the algorithm.

Step 1. Compute a positive real number δ together with a $(k, 2^d k; \delta)$-covering R of S.

In Section 4, we show that such a δ and such a covering R exist and that they can be found in $O(n \log n)$ time using $O(n)$ space. Moreover, it will be shown there how this collection can be stored in a data structure of size $O(n)$ such that point location queries can be solved in $O(\log n)$ time. This data structure can be built in $O(n \log n)$ time.

Step 2. Initialize $\mu_{opt} := \infty$ and $S_{opt} := \emptyset$.

Step 3. For each box $B \in R$, do the following:

3.1 Find all boxes in R that overlap the $(2c+1)\delta$-box that is centered at B. These boxes are found as follows:

Let $(b_1, b_2, \ldots, b_d)$ be the "lower-left" corner of B. Then, in the data structure for R, locate the $(2c+1)^d$ points

$$(b_1 + \epsilon_1\delta, b_2 + \epsilon_2\delta, \ldots, b_d + \epsilon_d\delta),$$

for $\epsilon_i \in \{-c, -c+1, \ldots, c-1, c\}$, $1 \leq i \leq d$.

3.2 Let S' be the set of points of S that are contained in the boxes that are found in Step 3.1. If $|S'| \geq k$, solve problem $P(S', k)$ using algorithm $\mathcal{A}$. Let S'_{opt} be the optimal k-point subset of S'. If $\mu(S'_{opt}) < \mu_{opt}$, then set $\mu_{opt} := \mu(S'_{opt})$ and $S_{opt} := S'_{opt}$.

Step 4. Output μ_{opt} and S_{opt}.

Theorem 1 *The algorithm correctly solves problem $P(S, k)$. Moreover, there is a constant c' such that the algorithm takes $O(n \log n + (n/k)T(c'k, k))$ time and uses $O(n + S(c'k, k))$ space.*

Proof: By Lemma 1, there is a closed $(c\delta)$-box that contains the optimal solution. It is clear that this box must be contained in the $(2c+1)\delta$-box that is centered at some box of R. The algorithm checks all these $(2c+1)\delta$-boxes. If there are less than k points in such a box, then it does not contain the optimal solution. This proves the correctness of the algorithm.

Each box of R contains at most $2^d k$ points. Moreover, the point location queries in Step 3.1 find at most $(2c+1)^d$ boxes of R. Therefore, set S' in Step 3.2 has size at most $(2c+1)^d 2^d k$.

There are at most $(2c+1)^d n/k$ boxes $B \in R$ that yield a subset S' of size at least k. Hence, algorithm $\mathcal{A}$ is called at most $(2c+1)^d n/k$ times.

As mentioned already, we will show later that the real number δ and the covering R can be computed in $O(n \log n)$ time using $O(n)$ space. Moreover, in time $O(n \log n)$, this collection can be build into a data structure of size $O(n)$, such that point location queries can be solved in $O(\log n)$ time.

This proves that the running time of the algorithm is bounded by

$$O((2c+1)^d n \log n + (2c+1)^d (n/k) T((2c+1)^d 2^d k, k)),$$

and amount of space used is bounded by

$$O(n + S((2c+1)^d 2^d k, k)).$$

This completes the proof. ∎

3 Degraded grids

In the previous section we saw that we need a real number δ and a $(k, 2^d k; \delta)$-covering R for S. Moreover, we need a data structure for these boxes that support point location queries. Assume that the value of δ and a δ-box containing at least k points are known already. Then, of course, we can take a grid with mesh size δ containing this box, and take for R the set of non-empty grid cells. Then, however, we need the floor-function to find the cell that contains a given point. Hence, the algorithm falls outside the algebraic decision tree model.

In this section, we introduce so-called degraded grids, that have basically the same properties as standard grids. We can build and search in a degraded grid, however, without using the floor-function.

To give an intuitive idea, in a standard δ-grid, we divide d-space into slabs of width δ. The grid is then defined by fixing an arbitrary point of $\mathbb{R}^d$ to be a lattice point of the grid. So, if e.g. $(0, \ldots, 0)$ is a lattice point, then for $1 \leq i \leq d$, a slab along the i-th axis consists of the set of all points in d-space that have their i-th coordinates between $j\delta$ and $(j+1)\delta$ for some integer j. In a degraded δ-grid, we also have slabs. The difference is that slabs do not necessarily start and end at multiples of δ. Moreover, slabs have width at least δ, and slabs that contain points of S have width exactly δ. That is, while a δ-grid may be defined independently of the point set by fixing an arbitrary point of $\mathbb{R}^d$ to be a lattice point, the degraded δ-grid is defined in terms of the point set stored in it.

We give a formal definition, treating the case $d = 1$ first.

Definition 2 *Let S be a set of n real numbers and let δ be a positive real number. Let $a_1, a_2, \ldots, a_l$ be a sequence of real numbers such that*

1. *for all $1 \leq j < l$, $a_{j+1} \geq a_j + \delta$,*
2. *for all $p \in S$, $a_1 \leq p < a_l$,*
3. *for all $1 \leq j < l$, if there is a point $p \in S$ such that $a_j \leq p < a_{j+1}$, then $a_{j+1} = a_j + \delta$.*

The collection of intervals $[a_j : a_{j+1})$, $1 \leq j < l$, is called a one-dimensional degraded δ-grid for S.

Constructing a one-dimensional degraded δ-grid: Sort the elements of S. Let $p_1 \leq p_2 \leq \ldots \leq p_n$ be the sorted sequence. Let $a_1 := p_1$. Let $j \geq 1$, and assume that $a_1, \ldots, a_j$ are defined already.

If there is an element in S that lies in the half-open interval $[a_j : a_j + \delta)$, then we set $a_{j+1} := a_j + \delta$. Otherwise, we set a_{j+1} to the value of the smallest element in S that is larger than a_j. This construction stops if we have visited all elements of S.

Lemma 2 *Let S be a set of n real numbers and let δ be a positive real number. If the elements of S are sorted, then we can construct a degraded δ-grid for S in $O(n)$ time using $O(n)$ space. Given this degraded grid, for each element $x \in S$, we can find all ℓ elements of S that are contained in the interval of x in time $O(\log n + \ell)$.*

Proof: The proof follows immediately from the definition and the given algorithm. Note that we store the intervals in a balanced tree. With each interval, we store a list consisting of the elements of S that are contained in this interval. ■

We extend the definition of a degraded grid to higher dimensions.

Definition 3 *Let S be a set of n points in d-space and let δ be a positive real number. For $1 \leq i \leq d$, let S_i be the set of i-th coordinates of the points in S. Let*

$$[a_{ij} : a_{i,j+1}), \quad 1 \leq j < l_i,$$

be a one-dimensional degraded δ-grid for the set S_i. The collection of d-dimensional boxes

$$\prod_{i=1}^{d}[a_{ij_i} : a_{i,j_i+1}), \text{ where } 1 \leq j_i < l_i,$$

is called a d-dimensional degraded δ-grid for S.

See Figure 1 for an example. The following lemma follows immediately.

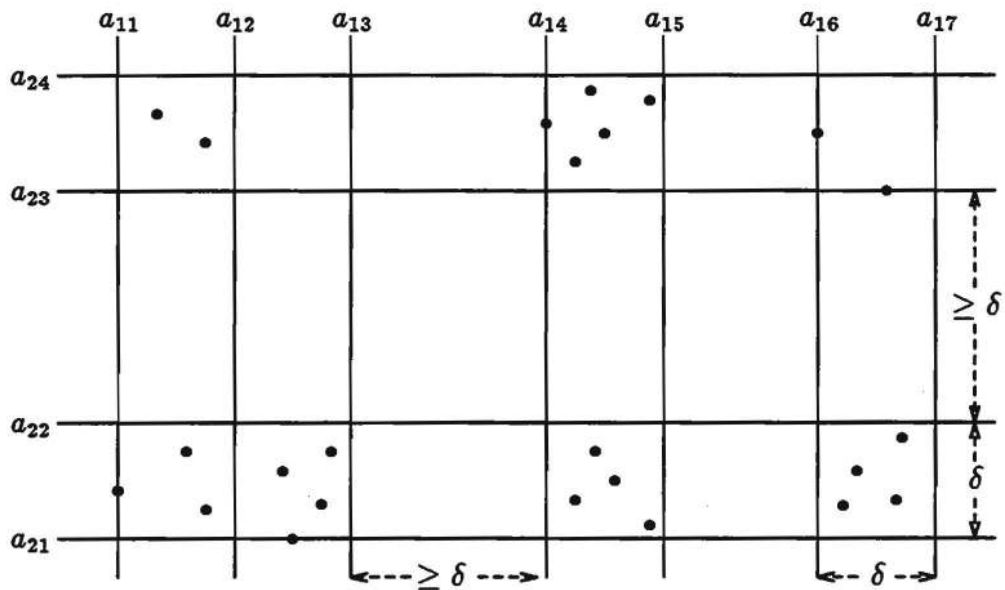

Figure 1: Example of a degraded δ-grid.

Lemma 3 *Let p be a point of S and let B be the box in the degraded δ-grid for S that contains p. Let c be an integer. All points of S that are within distance $c\delta$ from p are contained in B and in the $(2c+1)^d - 1$ boxes that surround B.*

Constructing a d-dimensional degraded δ-grid: Assume the points of S are stored in an array $\mathcal{S}$. For each $1 \leq i \leq d$, sort the elements of S_i. Give each element in S_i a pointer to its occurrence in $\mathcal{S}$.

For each $1 \leq i \leq d$, construct a one-dimensional degraded δ-grid $[a_{ij} : a_{i,j+1})$, $1 \leq j < l_i$, for the set S_i using the algorithm given above. During this construction, for each j and each element p_i—which denotes the i-th coordinate of point p—such that $a_{ij} \leq p_i < a_{i,j+1}$, follow the pointer to $\mathcal{S}$. Store with the point p in $\mathcal{S}$ the numbers a_{ij} and j.

At the end, each point in $\mathcal{S}$ stores with it two vectors of length d. If point p has vectors $(b_1, b_2, \ldots, b_d)$ and $(j_1, j_2, \ldots, j_d)$, then p is contained in the δ-box with lower-left corner $(b_1, b_2, \ldots, b_d)$. This δ-box is part of the j_i-th δ-slab along the i-th axis.

These vectors implicitly define the degraded δ-grid R. Note that each j_i is an integer in the range from 1 to n. Hence, we can sort the vectors $(j_1, j_2, \ldots, j_d)$ in $O(n)$ time by using radix-sort. This gives the non-empty boxes of the degraded grid, sorted in lexicographical order.

We summarize in the following lemma.

Lemma 4 *Let S be a set of n points in d-space and let δ be a positive real number. Assume the points of S are stored in an array $\mathcal{S}$. Moreover, assume that for each $1 \leq i \leq d$, the elements of S_i are sorted, and each element of this set contains a pointer to the corresponding point in $\mathcal{S}$.*

Then we can construct a d-dimensional degraded δ-grid for S in $O(n)$ time using $O(n)$ space. Moreover, we can preprocess this grid in $O(n)$ time, such that for any point p in S, we can report all ℓ points of S that are contained in the δ-box of p, in $O(\log n + \ell)$ time.

4 Constructing a degraded grid with $O(k)$ points per cell

In this section, we give the algorithm that computes the real number δ together with a corresponding $(k, 2^d k; \delta)$-covering R for S.

Recall the notion of weighted median: Let $x_1, x_2, \ldots, x_n$ be a sequence of n real numbers such that every element x_i has a weight w_i, which is a positive real number. Let $W = \sum_{j=1}^{n} w_j$. Element x_i is called a *weighted median* if

$$\sum_{j:x_j<x_i} w_j < W/2 \quad \text{and} \quad \sum_{j:x_j \leq x_i} w_j \geq W/2.$$

The weighted median can be computed in $O(n)$ time. (See e.g. [7] for a proof.)

Let S be a set of n points in d-space. In this section, we will use the following notations:

- Assume δ is a real number and R is a degraded δ-grid for S. Number the boxes of R (arbitrarily) $1, 2, \ldots, r = |R|$ and define n_i to be the number of points of S that are contained in the i-th box of R. Then we denote $M(R) = \max_{1 \leq i \leq r} n_i$.
- Let S' be a subset of S of size $2^d k$ with minimal L_∞-diameter among all $(2^d k)$-point subsets. Then, δ^* denotes the L_∞-diameter of S'.

Lemma 5 *Using these notations, the following holds:*

1. *For any $\delta \geq \delta^*$ and any degraded δ-grid R for S, we have $M(R) \geq k$.*
2. *For any $\delta \leq \delta^*$ and any degraded δ-grid R for S, we have $M(R) \leq 2^d k$.*

Proof: Let $\delta \geq \delta^*$ and let R be a degraded δ-grid for S. The set S' is contained in an axes-parallel square with sides of length δ^*. This square overlaps at most 2^d boxes of R. Since S' has size $2^d k$, there must be one box in R that contains at least k points of S. This shows that $M(R) \geq k$.

Let $\delta \leq \delta^*$ and let R be a degraded δ-grid for S. Assume that $M(R) > 2^d k$. Then there is a box in R that contains more than $2^d k$ points of S. Since this box is the product of half-open intervals of length δ, there are $2^d k$ points in S with L_∞-diameter less than δ. Since $\delta \leq \delta^*$, this contradicts the definition of δ^*. ■

The algorithm that is presented below searches for a real number δ together with a degraded δ-grid R for S such that $k \leq M(R) \leq 2^d k$. This grid is the $(k, 2^d k; \delta)$-covering we want. Lemma 5 implies that there is a δ for which such a covering exists, namely $\delta = \delta^*$. In fact, such a δ is contained in the set of all L_∞-distances between pairs of points in S. As in [7], we do a binary search in the larger set consisting of all possible differences $|p_i - q_i|$, where p and q are points of S and $1 \leq i \leq d$. Of course, we maintain the candidate differences in an implicit way.

The algorithm maintains the following information:

- Arrays $A_1, \ldots, A_d$ of length n, where A_i contains the points of S sorted w.r.t. their i-th coordinates. For each $1 \leq i \leq d$, each point in A_i contains a pointer to its occurrence in A_1.
- For each $1 \leq i \leq d$ and $1 \leq j < n$, we store with $A_i[j]$ an interval $[l_{ij} : h_{ij}]$, where l_{ij} and h_{ij} are integers, such that $j < l_{ij} \leq h_{ij} + 1 \leq n + 1$.

We define the set of *candidate differences* as follows. Let $p = (p_1, \ldots, p_d)$ and $q = (q_1, \ldots, q_d)$ be two distinct points in S, and let $1 \leq i \leq d$. Moreover, let j and j' be such that $A_i[j] = p$ and $A_i[j'] = q$. Assume w.l.o.g. that $j < j'$. Then $|q_i - p_i|$ is a candidate difference iff $l_{ij} \leq j' \leq h_{ij}$. Hence, the total number of candidate differences is equal to

$$\sum_{i=1}^{d} \sum_{j=1}^{n-1} (h_{ij} - l_{ij} + 1).$$

The algorithm makes a sequence of iterations. In each iteration, this summation is decreased by a factor of at least one fourth. The algorithm maintains the following

Invariant: At each moment, the value of δ^* is contained in the set of candidate differences.

Initialization: Build the arrays $A_1, \ldots, A_d$. Then, for each $1 \leq i \leq d$ and $1 \leq j < n$, store with $A_i[j]$ the interval $[l_{ij} : h_{ij}] = [j + 1 : n]$.

Now, the algorithm starts with the

Iteration:

Step 1. For each $1 \leq i \leq d$ and $1 \leq j < n$, such that $l_{ij} \leq h_{ij}$, take the pair

$$A_j[\lfloor (l_{ij} + h_{ij})/2 \rfloor] \quad \text{and} \quad A_i[j]$$

and take the (positive) difference of their i-th coordinates. Give this difference *weight* $h_{ij} - l_{ij} + 1$. This gives a sequence of at most $d(n-1)$ weighted differences.

Step 2. Compute a weighted median δ of these weighted differences.

Step 3. Construct a degraded δ-grid R for S, and compute $M(R)$. There are three possible cases.

3.1 If $k \leq M(R) \leq 2^d k$, then output δ and R, and stop.

3.2 If $M(R) < k$, then for each pair

$$A_i[\lfloor (l_{ij} + h_{ij})/2 \rfloor] \quad \text{and} \quad A_i[j]$$

selected in the first step such that the difference of their i-th coordinates is at most δ, set $l_{ij} := \lfloor (l_{ij} + h_{ij})/2 \rfloor + 1$. Go to Step 1.

3.3 If $M(R) > 2^d k$, then for each pair

$$A_i[\lfloor (l_{ij} + h_{ij})/2 \rfloor] \quad \text{and} \quad A_i[j]$$

selected in the first step such that the difference of their i-th coordinates is at least δ, set $h_{ij} := \lfloor (l_{ij} + h_{ij})/2 \rfloor - 1$. Go to Step 1.

Lemma 6 *The algorithm correctly maintains the invariant.*

Proof: After the initialization, the total number of candidate differences is equal to

$$\sum_{i=1}^{d} \sum_{j=1}^{n-1} (n - j) = d\binom{n}{2},$$

i.e., the set of candidate differences equals the set of all $d\binom{n}{2}$ differences $|p_i - q_i|$. Therefore, the invariant holds initially. Consider one iteration. First assume that Case 3.2 applies, i.e., $M(R) < k$. Then, Lemma 5 implies that $\delta < \delta^*$. The algorithm only removes differences $|p_u - q_u|$ from the set of candidate differences that are at most equal to δ. Hence, at the end of the iteration, the invariant still holds.

If Case 3.3 applies, then Lemma 5 implies that $\delta > \delta^*$. Hence, we can remove differences $|p_u - q_u|$ from the set of candidate differences that are at least equal to δ, without invalidating the invariant. ■

Lemma 7 *The algorithm makes at most* $\log_{4/3}(dn^2) = O(\log n)$ *iterations.*

Proof: At the start of the iteration, the set of candidate differences has size $d\binom{n}{2}$. In each iteration, the size of this set is decreased by a factor of at least one fourth. (See [7] for a precise proof of this.) ■

Theorem 2 *In $O(n \log n)$ time and using $O(n)$ space, we can compute a real number δ and a degraded δ-grid R for S, such that $k \leq M(R) \leq 2^d k$.*

Proof: It follows from the above that the algorithm computes a real number δ and a degraded δ-grid R such that $k \leq M(R) \leq 2^d k$. The initialization of the algorithm takes $O(n \log n)$ time. Moreover, $O(\log n)$ iterations are made, each taking $O(n)$ time. This proves that the entire algorithm has a running time of $O(n \log n)$. It is clear that the algorithm uses only linear space. ■

5 Applications

In this section, we consider several measures μ that satisfy Assumptions 1 and 2. For each measure, we improve the previously best known bounds for solving problem $P(n, k)$. The results are summarized at the end of this section in Table 1.

5.1 Minimum diameter k-point subset

In this problem, $\mu(V)$ is the L_2-diameter of the set V. Hence, we want to find k points that have a minimal diameter. In order to show that the algorithm of Section 2 can be applied, we only have to show that Assumptions 1 and 2 are satisfied. This is easily proved:

Lemma 8 *For μ the diameter measure, Assumptions 1 and 2 hold with $c = \lceil\sqrt{d}\rceil$.*

Proof: Consider the optimal k-point subset S_{opt}. If there is no closed $\mu_{opt}(S)$-box that contains S_{opt}, then there must be two points in S_{opt} that have L_2-distance larger than $\mu_{opt}(S)$. This proves that Assumption 1 holds. To prove Assumption 2, let $\delta < \mu_{opt}(S)/\lceil\sqrt{d}\rceil$. Assume there is a closed δ-box that contains at least k points of S. Then there are k points that have L_2-diameter at most $\sqrt{d}\delta < \mu_{opt}(S)$. This is a contradiction. ■

It follows that we can apply our general algorithm. Recall that we need an algorithm $\mathcal{A}$ that is called for subsets of size $\Theta(k)$. We take the algorithm of [5]. This algorithm solves the problem in time $T(n,k) = O(n^3 \log^2 n)$ using $S(n,k) = O(n)$ space in the planar case. For the d-dimensional case, the algorithm runs in time $T(n,k) = O(kn \log n + 2^{O(k)}n)$ and uses space $S(n,k) = O(kn)$. Applying Theorem 1 proves:

Theorem 3 *Given a set S of n points in d-space and an integer $1 \leq k \leq n$, we can find a subset of size k with minimal diameter*

1. *in $O(n \log n + nk^2 \log^2 k)$ time and $O(n)$ space, if $d = 2$,*
2. *in $O(n \log n + 2^{O(k)}n)$ time and $O(n + k^2)$ space, if $d > 2$.*

In [5], the running times are the same, but the space bounds are $O(n \log n + kn)$ if $d = 2$, and $O(kn)$ if $d > 2$.

5.2 Minimum L_∞-diameter k-point subset

We want to find k points with minimal L_∞-diameter, i.e., $\mu(V)$ is the L_∞-diameter of the set V. Note that this is the same as finding a smallest d-dimensional cube that contains at least k points of S. The following lemma follows immediately.

Lemma 9 *For μ the L_∞-diameter, Assumptions 1 and 2 hold with $c = 1$.*

Again, we take the algorithm $\mathcal{A}$ from [5]. This algorithm solves the problem in $O(n^{d/2} \log^2 n)$ time using $O(n^{d/2})$ space. Applying Theorem 1 proves:

Theorem 4 *Given a set S of n points in d-space and an integer $1 \leq k \leq n$, we can find a subset of size k with minimal L_∞-diameter*

1. *in $O(n \log n + n \log^2 k)$ time using $O(n)$ space, if $d = 2$,*
2. *in $O(n \log n + nk^{d/2-1} \log^2 k)$ time using $O(n + k^{d/2})$ space, if $d > 2$.*

In [5], the time resp. space bounds are $O(nk \log n + nk^{d/2-1} \log^2 k)$ resp. $O(nk + k^{d/2})$ for $d > 2$. For $d = 2$, they give a variant of the algorithm using time and space $O(n \log n + nk)$. The previously best linear space solution for the planar case was given by Smid [11]. He obtains a running time of $O(n \log n + nk \log^2 k)$.

5.3 Minimum perimeter k-point subset

For this problem, the points are planar. We want to find k points whose convex hull has minimal perimeter. That is, $\mu(V)$ is the perimeter of the convex hull of V.

Lemma 10 *For μ the perimeter measure, Assumptions 1 and 2 hold with $c = 4$.*

We take the algorithm $\mathcal{A}$ from [4]. This algorithm has a running time of $O(n^3k)$ and uses $O(nk)$ space. Then Theorem 1 gives:

Theorem 5 *Given a set S of n points in the plane and an integer $1 \leq k \leq n$, we can find a subset of size k with minimal perimeter in $O(n \log n + nk^3)$ time using $O(n + k^2)$ space.*

In [5], the same running time is obtained, but the space bound is $O(n \log n + nk + k^3)$.

Lemma 10 also holds if we take for μ the L_∞-perimeter. Then, we want to find k points such that the perimeter of their axes-parallel enclosing rectangle is minimal. We take for $\mathcal{A}$ the brute-force algorithm of [1]. This algorithm runs in time $O(n^3)$ and uses $O(n)$ space. Theorem 1 yields:

Theorem 6 *Given a set S of n points in the plane and an integer $1 \leq k \leq n$, we can find a subset of size k with minimal L_∞-perimeter in $O(n \log n + nk^2)$ time using $O(n)$ space.*

This result improves the space bound in [5] from $O(n \log n + nk)$ to $O(n)$. The time bound is the same as in [5].

5.4 Minimum circumradius k-point subset

This is a problem in d-space again. We want to find a smallest ball that contains at least k points. Hence, we can take for $\mu(V)$ the diameter of the smallest ball that contains V.

Lemma 11 *For μ the circumradius measure, Assumptions 1 and 2 hold with $c = \lceil \sqrt{d} \rceil$.*

We take the algorithm $\mathcal{A}$ from [5]. This algorithm runs in time $O(n^d \log^2 n)$ and uses space $O(n^d \log n)$. In the planar case, the time and space bounds are both $O(n^2 \log n)$. Applying Theorem 1 gives:

Theorem 7 *Given a set S of n points in d-space and an integer $1 \leq k \leq n$, we can find a subset of size k with minimal circumradius*

1. *in $O(n \log n + nk \log k)$ time using $O(n + k^2 \log k)$ space, if $d = 2$,*
2. *in $O(n \log n + nk^{d-1} \log^2 k)$ time using $O(n + k^d \log k)$ space, if $d > 2$.*

The running time is the same as in [5]. There, however, the space complexity is $O(n \log n + nk + k^2 \log k)$ if $d = 2$, and $O(nk + k^d \log k)$ if $d > 2$.

measure	dimension	time	space
diameter	2	$n \log n + nk^2 \log^2 k$	n
diameter	$d > 2$	$n \log n + 2^{O(k)} n$	$n + k^2$
L_∞-diameter	2	$n \log n + n \log^2 k$	n
L_∞-diameter	$d > 2$	$n \log n + nk^{d/2-1} \log^2 k$	$n + k^{d/2}$
perimeter	2	$n \log n + nk^3$	$n + k^2$
L_∞-perimeter	2	$n \log n + nk^2$	n
circumradius	2	$n \log n + nk \log k$	$n + k^2 \log k$
circumradius	$d > 2$	$n \log n + nk^{d-1} \log^2 k$	$n + k^d \log k$

Table 1: Static solutions.

6 Maintaining an optimal k-point subset under insertions

In this section, we consider the problem of maintaining the optimal solution S_{opt} if points are inserted into S. This problem was also considered in [5]. They solve it by applying the logarithmic method of Bentley [2]. For the planar case, the resulting data structure has size $O(n \log n + S(O(k), k))$ and an insertion time of $O(\log^2 n + k \log n + T(O(k), k))$. We improve both complexity bounds.

Lemma 12 *The value $\mu_{opt}(S)$ does not increase if we insert a point into S.*

Proof: It suffices to show that $S \subseteq S'$ implies $\mu_{opt}(S') \leq \mu_{opt}(S)$. This is clear, since $\mu_{opt}(S')$ is determined by more k-point subsets than $\mu_{opt}(S)$. ■

Recall the meaning of the constant c in Assumption 2.

Lemma 13 *Let B be a box that contains at least $(2c)^d k$ points of S. For $1 \leq i \leq d$, let m_i resp. M_i denote the minimal resp. maximal i-th coordinate of any point in $S \cap B$. Then there is an index i such that $M_i - m_i \geq 2\mu_{opt}(S)$.*

Proof: Assume the claim is false. Then there is a $\delta < 2\mu_{opt}(S)$ and a closed δ-box that contains all points of $S \cap B$. Partition this box into $(2c)^d$ closed subboxes with sides of length $\delta/(2c)$. Then one of these closed subboxes contains at least k points of S. This contradicts Assumption 2, because $\delta/(2c) < \mu_{opt}(S)/c$. ■

In [9], it is shown how a partition of d-space into at most n boxes can be maintained in $O(n)$ space, such that point location queries can be solved in $O(\log n)$ time, and such that a box can be split into two boxes in $O(\log n)$ amortized time. (The two new boxes must still be axes-parallel.) Using a technique described in [3], the time for a split operation can even be made worst-case.

In the following insertion algorithm, we assume that we have a partition of d-space into boxes such that

- each point of S is contained in exactly one box of the partition,
- each box in the partition has sides of length at least $\mu_{opt}(S)$,

- each box in the partition contains at most $(2c)^d k$ points of S.

Moreover, we assume that this partition is stored using the method of [3]. With each box of this partition, we store a list of all points of S that are contained in it.

The insertion algorithm: We denote the current value of $\mu_{opt}(S)$ by μ_{opt}. Recall that S_{opt} denotes the optimal k-point subset of the current set S. Let $p = (p_1, p_2, \ldots, p_d)$ be the point to be inserted.

Step 1. Find all boxes of the partition that overlap the $(2\mu_{opt})$-box that is centered at p. These boxes are found by performing 3^d point location queries with the points

$$(p_1 + \epsilon_1\mu_{opt}, p_2 + \epsilon_2\mu_{opt}, \ldots, p_d + \epsilon_d\mu_{opt}), \quad \epsilon_1, \epsilon_2, \ldots, \epsilon_d \in \{-1, 0, 1\}.$$

Let S' be the set of points of S that are contained in these boxes.

Step 2. If $|S'| \geq k$, solve problem $P(S', k)$ using algorithm $\mathcal{A}$. Let S'_{opt} be the optimal k-point subset of S'. If $\mu(S'_{opt}) < \mu_{opt}$ then set $\mu_{opt} := \mu(S'_{opt})$ and $S_{opt} := S'_{opt}$.

Step 3. Output μ_{opt} and S_{opt}.

Step 4. Insert p into the box of the partition that contains it. If this box contains $(2c)^d k + 1$ points, then split this box into two boxes with sides of length at least μ_{opt} such that both new boxes contain at most $(2c)^d k$ points. (By Lemma 13, this is possible.)

Theorem 8 *The insertion algorithm correctly maintains the optimal solution of problem $P(S, k)$. Moreover, there is a constant c' such that the insertion time is bounded by $O(\log n + T(c'k, k))$ and the amount of space used is bounded by $O(n + S(c'k, k))$.*

Proof: Let $\mu_{opt} = \mu_{opt}(S)$ and $\mu'_{opt} = \mu_{opt}(S \cup \{p\})$. We know from Assumption 1 that the optimal solution for the set $S \cup \{p\}$ is contained in a closed μ'_{opt}-box. It is clear that if the optimal solution changes because of the insertion of p, then this optimal solution must be contained in the $(2\mu'_{opt})$-box which is centered at p. Since $\mu'_{opt} \leq \mu_{opt}$, it suffices to consider all points that are contained in the $(2\mu_{opt})$-box centered at p. The algorithm indeed considers all these points. This proves that the optimal solution is correctly maintained.

Next we show that the partition of d-space is correctly maintained. If a box in the partition is not split, then it has sides of length at least $\mu_{opt} \geq \mu'_{opt}$. The two new boxes that arise because of a split operation have sides of length at least μ'_{opt}. It is clear that the other two requirements also hold.

It remains to prove the time and space bounds. The bound on the size of the data structure is clear. To insert a point, we perform 3^d point location queries, each taking $O(\log n)$ time. Then we solve a problem $P(S', k)$ for a subset S' of size at most $3^d(2c)^d k$. Finally, we may split a box. This takes $O(\log n + k)$ time. Hence, for an appropriate constant c', the entire insertion algorithm takes time $O(\log n + k + T(c'k, k)) = O(\log n + T(c'k, k))$, because $T(c'k, k) = \Omega(k)$. ■

Table 2 shows the results that follow from this theorem and from the results in the previous section.

measure	dimension	insertion time	space
diameter	2	$\log n + k^3 \log^2 k$	n
diameter	$d > 2$	$\log n + 2^{O(k)}$	$n + k^2$
L_∞-diameter	2	$\log n + k \log^2 k$	n
L_∞-diameter	$d > 2$	$\log n + k^{d/2} \log^2 k$	$n + k^{d/2}$
perimeter	2	$\log n + k^4$	$n + k^2$
L_∞-perimeter	2	$\log n + k^3$	n
circumradius	2	$\log n + k^2 \log k$	$n + k^2 \log k$
circumradius	$d > 2$	$\log n + k^d \log^2 k$	$n + k^d \log k$

Table 2: Semi-dynamic solutions.

7 A fully dynamic data structure

The algorithm of the previous section only works for insertions: The use of Lemma 12 is crucial. In this section, we show that the method of [10] can be adapted such that the optimal k-point subset can be maintained under insertions and deletions.

Recall that for V a set of points, $\mu_{opt}(V)$ denotes the minimal measure of any k-point subset of V. If V has size less than k, then $\mu_{opt}(V) = \infty$.

Let $1 \leq i \leq d$. The space $\mathbb{R}^i$ consists of 2^i quadrants. (For $i = 1$, the quadrants of $\mathbb{R}^i = \mathbb{R}$ are $(-\infty : 0]$ and $[0 : \infty)$.) Quadrants are assumed to be closed. We number them arbitrarily. For $1 \leq j \leq 2^i$, we denote the j-th quadrant by Q^i_j.

As in [10], we recursively define data structures of type i for $i = 0, 1, \ldots, d$. The data structure of type d maintains the optimal solution S_{opt} and its measure $\mu_{opt}(S)$. The data structure of type i stores a collection of 2^{d-i} sets. For $1 \leq j \leq 2^{d-i}$, the j-th set of this collection lies in $Q^{d-i}_j \times \mathbb{R}^i$.

We start with the data structure of type 0. For $1 \leq j \leq 2^d$, let V_j be a set of points that lies in the j-th quadrant of d-space, i.e., $V_j \subseteq Q^d_j$. (The 2^d quadrants can intersect in an arbitrary point of $\mathbb{R}^d$. W.l.o.g. we take this point to be the origin.) Let $n_j = |V_j|$ and $n = \sum_j n_j$. Some of the n_j's may be zero.

The data structure of type 0:

1. For $1 \leq j \leq 2^d$, the points of V_j are stored in the leaves of a balanced binary search tree T_j, sorted by their L_∞-distances to the origin. Points with equal L_∞-distance to the origin are stored in lexicographical order.

2. For $1 \leq j \leq 2^d$, let V'_j be the set of $\min((2c)^d k, |V_j|)$ smallest—i.e., leftmost—points in T_j. We store a variable η_0 having value

$$\eta_0 = \mu_{opt}\Big(\bigcup_j V'_j\Big).$$

The meaning of the variable η_0 will become clear later. We consider updates of the following type: If we insert or delete a point p that lies in the j-th quadrant of d-space, then we insert or delete p in the set V_j. If p lies on the boundary of several quadrants, then we insert or delete p in only one (arbitrary) set V_j. The following lemma is clear.

Lemma 14 *The data structure of type 0 has size $O(n)$. Moreover, there exists a constant c', such that the data structure can be built in $O(n \log n + T(c'k, k))$ time and can be maintained in $O(\log n + T(c'k, k))$ time per insertion and deletion. During the building and update algorithms, $O(n + S(c'k, k))$ space is needed.*

Note that the data structure itself has only $O(n)$ size. In order to build and maintain it, however, we call algorithm $\mathcal{A}$ for a set of size $O(k)$. This causes the extra term $S(c'k, k)$.

Now let $0 < i \leq d$ and assume that the data structure of type $(i-1)$ has been defined already. We define the data structure of type i.

For $1 \leq j \leq 2^{d-i}$, let V_j be a set of points that lie in $Q_j^{d-i} \times \mathbb{R}^i$. Let $n_j = |V_j|$ and $n = \sum_j n_j$. Some of the n_j's may be zero.

The data structure of type i: All points of the set $\bigcup_j V_j$ are stored in the leaves of one balanced binary search tree, sorted by their $(d-i+1)$-th coordinates. Points with equal $(d-i+1)$-th coordinate are stored in lexicographical order. In each node u of this tree, we store a hyperplane $x_{d-i+1} = \sigma_u$, where σ_u is the maximal $(d-i+1)$-th coordinate stored in its left subtree.

Each node u of this tree contains the following additional information.

1. If the subtree of u contains less than k points, then it contains a variable $\eta_i(u)$ with value ∞.

2. Assume the subtree of u contains at least k points. Let v resp. w be the left resp. right son of u.

   For $1 \leq j \leq 2^{d-i}$, let V_j^v resp. V_j^w be the subsets of V_j that are stored in the subtrees of v resp. w. Note that $V_j^v \subseteq Q_j^{d-i} \times (-\infty : \sigma_u] \times \mathbb{R}^{i-1}$, and $V_j^w \subseteq Q_j^{d-i} \times [\sigma_u : \infty) \times \mathbb{R}^{i-1}$. That is, w.r.t. an appropriate origin, we have $V_j^v \subseteq Q_j^{d-i} \times Q_1^1 \times \mathbb{R}^{i-1}$, and $V_j^w \subseteq Q_j^{d-i} \times Q_2^1 \times \mathbb{R}^{i-1}$.

   (a) We store in u a pointer to a data structure of type $(i-1)$ which stores the 2^{d-i+1} sets V_j^v and V_j^w, $1 \leq j \leq 2^{d-i}$. Let η_{i-1} be the variable that is stored with this data structure.

   (b) Let $\eta_i(v)$ resp. $\eta_i(w)$ be the variables that are stored with the nodes v resp. w. We store in u a variable $\eta_i(u)$ with value

$$\eta_i(u) = \min(\eta_i(v), \eta_i(w), \eta_{i-1}).$$

Finally, the data structure of type i stores a variable η_i with value $\eta_i = \eta_i(r)$, where r is the root of the tree.

We consider updates of the following type: If we insert or delete a point p that lies in $Q_j^{d-i} \times \mathbb{R}^i$, then we insert or delete p in the set V_j. If p lies on the boundary of several regions, then we insert or delete p in only one (arbitrary) set V_j.

Note that the data structure of type d stores one set of points. We show that this data structure for the set S stores the optimal solution to problem $P(S, k)$:

Lemma 15 *Consider the data structure of type d for the set S. The value η_d that is stored with this structure is equal to $\mu_{opt}(S)$.*

Proof: First note that all $\eta_i(\cdot)$-variables have value either ∞ or $\mu_{opt}(S')$ for some subset S' of S. Therefore, $\mu_{opt}(S) \leq \eta_d$.

The data structure of type d contains type 0 structures as substructures. If we can show that the η_0-variable of one of these type 0 substructures has value $\mu_{opt}(S)$, then it follows that $\eta_d \leq \mu_{opt}(S)$ and, hence, $\eta_d = \mu_{opt}(S)$.

We show that such a type 0 substructure exists. Consider the optimal k-point subset S_{opt} of S. Note that $\mu(S_{opt}) = \mu_{opt}(S)$. We inductively define a sequence $u_1, u_2, \ldots, u_d$ of nodes having the following properties:

- u_i is a node of a data structure of type $(d - i + 1)$.
- u_i is a node of the data structure of type $(d - i + 1)$ that is pointed to by u_{i-1}.
- The subtree of u_i contains all points of S_{opt}.
- u_i is the highest node in its tree such that both its left and its right subtree contain points of S_{opt}.

To define u_1, consider the data structure of type d. Then, u_1 is the highest node in its binary search tree such that both the left and the right subtree of u_1 contain points of S_{opt}. Let $1 < i \leq d$ and assume that $u_1, u_2, \ldots, u_{i-1}$ have been defined already. Then, u_i is the highest node in the binary tree of the data structure of type $d - i + 1$ that is pointed to by u_{i-1}, such that both the left and the right subtree of u_i contain points of S_{opt}. It is clear that the nodes defined in this way have the four given properties.

Consider node u_d. This node belongs to the binary tree of a data structure of type 1, and it contains a pointer to a data structure D of type 0. We claim that the variable η_0 that is stored with D has value $\mu_{opt}(S)$. This will complete the proof of the lemma.

Let V_j, $1 \leq j \leq 2^d$, be the sets that are stored in D. Note that $S_{opt} \subseteq \bigcup_j V_j$. Moreover, the sets V_j, some of which may be empty, lie in different quadrants that are defined by the hyperplanes that are stored in the nodes $u_1, u_2, \ldots, u_d$. We assume w.l.o.g. that these hyperplanes intersect in the origin.

Let V'_j, $1 \leq j \leq 2^d$, be the subsets that define the value of η_0. That is, $\eta_0 = \mu_{opt}(\bigcup_j V'_j)$. (See the definition of the data structure of type 0.) If we can show that $S_{opt} \subseteq \bigcup_j V'_j$, then we must have $\eta_0 = \mu_{opt}(S)$.

Assume this is not the case. Let p be a point of S_{opt} that does not belong to $\bigcup_j V'_j$. Assume w.l.o.g. that p lies in the first quadrant of $\mathbb{R}^d$, i.e., all coordinates of p are non-negative.

Since $|V'_1| = \min((2c)^d k, |V_1|)$ and since $p \in V_1 \setminus V'_1$, we have $|V'_1| = (2c)^d k$. Let δ be the maximal L_∞-distance between any point of V'_1 and the origin. Then, V'_1 is contained in the closed box $[0 : \delta]^d$. This closed box contains $(2c)^d k$ points of S. Partition it into $(2c)^d$ closed subboxes, each with sides of length $\delta/(2c)$. One of these closed subboxes contains at least k points of S. Hence, by Lemma 1, $\mu_{opt}(S) \leq c \cdot \delta/(2c) < \delta$.

Consider again point p. This point has L_∞-distance at least δ to the origin. Let $1 \leq l \leq d$ be an index such that p_l, i.e., the l-th coordinate of p, is at least equal to δ. Our choice of the nodes $u_1, u_2, \ldots, u_d$ implies that there is a point $q \in S_{opt}$ whose l-th

coordinate is non-positive. Hence, there are two points p and q in S_{opt} that are at L_∞-distance at least δ from each other. On the other hand, we know from Assumption 1, that the set S_{opt} is contained in some $\mu_{opt}(S)$-box. In particular, the L_∞-distance between p and q is at most $\mu_{opt}(S)$. This proves that $\delta \leq \mu_{opt}(S)$, which is a contradiction. Hence, we have shown that $S_{opt} \subseteq \bigcup_j V'_j$. This completes the proof. ∎

The data structure of type i is similar to a range tree. (See e.g. [8].) To maintain it under insertions and deletions, we take the binary trees from the class of BB[α]-trees. First, we analyze its space complexity. Let $G(n,i)$ be the size of a data structure of type i storing n points. Then, $G(n,0) = O(n)$. Let $i > 1$. The data structure of type i consists of a binary tree, having size $O(n)$. Each node of this tree whose subtree contains at least k points, has a pointer to a data structure of type $(i-1)$. For all nodes on one level, these type $(i-1)$ structures together have size at most $G(n,i-1)$. Since there are $O(\log(n/k))$ levels whose nodes contain pointers to type $(i-1)$ structures, we get the following recurrence: $G(n,i) = O(n + G(n,i-1)\log(n/k))$, which solves to $G(n,d) = O(n\log^d(n/k))$.

The update algorithm is virtually the same as in [10]. Rebalancing is done by means of rotations. Moreover, as in [10], we can apply dynamic fractional cascading. We refer the reader to that paper for the details. If $U(n,i)$ denotes the amortized update time, then we can show that $U(n,0) = O(\log n + T(O(k),k))$,

$$U(n,1) = O(\log n + \log(n/k)\log\log n + T(O(k),k)\log(n/k)),$$

and

$$U(n,i) = O(\log n + U(n,i-1)\log(n/k) + \log^i(n/k)\log\log n + T(O(k),k)/k\log^i(n/k)).$$

The last term in the bound on $U(n,1)$ and the last two terms in the recurrence for $U(n,i)$ are the amortized rebalancing costs. It follows that

$$U(n,d) = O(\log n\log^{d-1}(n/k) + \log^d(n/k)\log\log n + T(O(k),k)\log^d(n/k)).$$

Note that during the update algorithm, we need an extra amount $S(O(k),k)$ of space, because we call the algorithm $\mathcal{A}$ for subsets of size $O(k)$.

The data structure of type d, as presented so far, only maintains the optimal measure $\mu_{opt}(S)$. It does not give the optimal k-point subset S_{opt} that realizes this measure. We can easily extend the data structure such that it also maintains S_{opt}: Maintain a pointer to the node in the substructure of type 0 whose η_0-variable has value $\mu_{opt}(S)$. (That is, node u_d in the proof of Lemma 15.) Then after an update, if the value of $\mu_{opt}(S)$ has changed, we follow the pointer to this node and recompute $\mu_{opt}(S)$. This gives us the new optimal k-point subset S_{opt}.

We have proved the following:

Theorem 9 *There exists a data structure that maintains the optimal solution of problem $P(S,k)$ under insertions and deletions. For some constant c' this data structure uses $O(n\log^d(n/k) + S(c'k,k))$ space and it has an amortized update time of*

$$O(\log n\log^{d-1}(n/k) + \log^d(n/k)\log\log n + T(c'k,k)\log^d(n/k)).$$

Table 3 shows the results that follow from this theorem.

measure	dimension	update time	space
diameter	2	$f(n,k,2)+k^3\log^2 k\log^2(n/k)$	$n\log^2(n/k)$
diameter	$d>2$	$f(n,k,d)+2^{O(k)}\log^d(n/k)$	$n\log^d(n/k)+k^2$
L_∞-diameter	2	$f(n,k,2)+k\log^2 k\log^2(n/k)$	$n\log^2(n/k)$
L_∞-diameter	$d>2$	$f(n,k,d)+k^{d/2}\log^2 k\log^d(n/k)$	$n\log^d(n/k)+k^{d/2}$
perimeter	2	$f(n,k,2)+k^4\log^2(n/k)$	$n\log^2(n/k)+k^2$
L_∞-perimeter	2	$f(n,k,2)+k^3\log^2(n/k)$	$n\log^2(n/k)$
circumradius	2	$f(n,k,2)+k^2\log k\log^2(n/k)$	$n\log^2(n/k)+k^2\log k$
circumradius	$d>2$	$f(n,k,d)+k^d\log^2 k\log^d(n/k)$	$n\log^d(n/k)+k^d\log k$

Table 3: Fully dynamic solutions. The update times are amortized. $f(n,k,d)$ denotes the function $\log n\log^{d-1}(n/k)+\log^d(n/k)\log\log n$.

8 Concluding remarks

We have given a unified approach for solving k-point clustering methods. The main technique was to reduce the problem to subproblems for $O(n/k)$ points, which were solved by some other algorithm $\mathcal{A}$. In this way, it suffices to solve the problem for instances where n and k are proportional. Any improvement for such a problem gives improved static and dynamic solutions.

There remain some open problems. First, can the randomized data structure of Golin et al.[6], that maintains the closest pair in a point set, be extended to maintain the optimal k-point subset?

Second, our method does not apply for the problem of finding the axes-parallel rectangle of minimal area that contains at least k points. It is clear that this measure does not satisfy Assumption 1. Can our techniques be generalized such that such measures can also be handled?

References

[1] A. Aggarwal, H. Imai, N. Katoh and S. Suri. *Finding k points with minimum diameter and related problems.* J. Algorithms **12** (1991), pp. 38-56.

[2] J.L. Bentley. *Decomposable searching problems.* Inform. Proc. Lett. **8** (1979), pp. 244-251.

[3] R.F. Cohen and R. Tamassia. *Combine and conquer.* Report CS-92-19, Brown University, Providence, 1992.

[4] D.P. Dobkin, R.L. Drysdale and L.J. Guibas. *Finding smallest polygons.* In: F.P. Preparata (ed.), Advances in Computing Research, Vol. 1, Computational Geometry, J.A.I. Press, London, 1983, pp. 181-214.

[5] D. Eppstein and J. Erickson. *Iterated nearest neighbors and finding minimal polytopes.* Proc. 4th Annual ACM-SIAM Symp. on Discrete Algorithms, 1993, pp. 64-73.

[6] M. Golin, R. Raman, C. Schwarz and M. Smid. *Randomized data structures for the dynamic closest-pair problem.* Proc. 4th Annual ACM-SIAM Symp. on Discrete Algorithms, 1993, pp. 301-310.

[7] H.P. Lenhof and M. Smid. *Enumerating the k closest pairs optimally.* Proc. 33rd Annual IEEE Symp. Foundations of Computer Science, 1992, pp. 380-386.

[8] F.P. Preparata and M.I. Shamos. *Computational Geometry, an Introduction.* Springer-Verlag, New York, 1985.

[9] C. Schwarz, M. Smid and J. Snoeyink. *An optimal algorithm for the on-line closest pair problem.* Proc. 8th ACM Symp. on Computational Geometry, 1992, pp. 330-336.

[10] M. Smid. *Maintaining the minimal distance of a point set in polylogarithmic time.* Discrete Comput. Geom. 7 (1992), pp. 415-431.

[11] M. Smid. *Finding k points with a smallest enclosing square.* Report MPI-I-92-152, Max-Planck-Institut für Informatik, Saarbrücken, 1992.