Efficient kinetic data structures for MaxCut

Artur Czumaj *† Gereon Frahling ‡ Christian Sohler §¶

Abstract

We develop a randomized kinetic data structure that maintains a partition of the moving points into two sets such that the corresponding cut is with probability at least $1-\varrho$ a $(1-\epsilon)$-approximation of the Euclidean *Max-Cut*. The data structure answers queries of the form *"to which side of the partition belongs query point p?"* in $O(2^{1/\epsilon^{O(1)}} \log^2 n/\epsilon^{2(d+1)})$ time. Under linear motion the data structure processes $\widetilde{O}(n \log(\varrho^{-1})/\epsilon^{d+3})$ events, each requiring $O(\log^2 n)$ expected time except for a constant number of events that require $\widetilde{O}(n \cdot \ln(\varrho^{-1})/\epsilon^{d+3})$ time. A flight plan update can be performed in $O(\log^3 n \cdot \ln(\varrho^{-1})/\epsilon^{d+3})$ average expected time, where the average is taken over the worst case update times of the points at an arbitrary point of time. No efficient kinetic data structure for the MaxCut has been known before.

1 Introduction

The problem of clustering data sets according to some similarity measures belongs to the most extensively studied optimization problems. In this paper we will focus on clustering moving points as described in the framework of *kinetic data structures* (KDS). The framework of kinetic data structures has been introduced by Basch et al. [3] and it has been used since as the central model of studying geometric objects in motion, see, e.g., [1, 3, 11, 12] and the references therein. In the kinetic setting, we consider a set of points in $\mathbb{R}^d$ that are continuously moving. Each point follows a (known) trajectory that is defined by a continuous function of time; for simplicity of presentation, we will assume that it is a linear function. Additionally, we allow the points to change their trajectory, i.e., to perform a *flight plan update*. The KDSs are data structures to maintain a certain attribute (for example, in the case of a clustering problem, assignment of the points to the clusters) under movement of the points. The main idea underlying the framework of KDSs is that even if the input objects are moving in a continuous fashion, the underlying combinatorial structure of the moving objects changes only at discrete times. Therefore, there is no need to maintain the data structure continuously but rather only when certain combinatorial *events* happen.

To measure the quality of a KDS, we will consider the following two most important performance measures (for more details, see, e.g., [11, 12]): the time needed to update the KDS when an event occurs and a bound on the number of events that may occur during the motion. Another important measure is the time to handle flight plan updates.

MaxCut problem. In this paper, we consider the Euclidean version of the MaxCut problem. For metric graphs (and hence also for geometric instances), Fernandez de la Vega and Kenyon [7] designed a PTAS. For the Euclidean version of the MaxCut that we study in this paper, it is still not known if the problem is NP-hard but a very fast PTAS can be obtained using a recent construction of small coresets for MaxCut [8].

In this paper, we develop the first efficient KDS for approximate Euclidean *MaxCut* for moving points. Our KDS is based on a coreset construction for MaxCut from [8]. In [8], it was shown in the context of data streaming algorithms that one can obtain a coreset from the distribution of certain sample sets of the point set in nested grids. Our KDS is based on the idea of maintaining these distribution under motion. The main difficulty of applying that approach lies in the interplay between a lower bound on the cost of the solution and the number of events, which requires some new ideas. Our KDS is not only the first efficient KDS for approximate Euclidean MaxCut, but it also puts the MaxCut problem into a very small set of complex geometric problems for which there exists a KDS requiring only $\widetilde{O}(n)$ events; many geometric problems, some even surprisingly simple ones, are known to have no KDS with $o(n^2)$ events.

2 Previous results used by our algorithm

We review a coreset construction from [8] and focus on the MaxCut problem. (Since some details of that construction that we need in the current paper differ from the presentation in [8], we will present some more formal arguments in Appendix A.) Let P be a point set in

*Department of Computer Science, University of Warwick, czumaj@dcs.warwick.ac.uk

†Research supported in part by the *Centre for Discrete Mathematics and its Applications (DIMAP)*, University of Warwick.

‡Google Research, New York, gereon@google.com

§Heinz Nixdorf Institute and Institute for Computer Science, University of Paderborn, csohler@upb.de

¶Supported by DFG grant Me 872/8-3.

the $\mathbb{R}^d$. For simplicity of presentation, we normalize the cost of the optimal solution of the MaxCut problem by dividing the cost by the number of points n, and define for each partition of P into C_1 and C_2:

$$\begin{aligned} M(C_1, C_2) &:= \tfrac{1}{n} \cdot \mathit{MaxCut}(P, C_1, C_2) \\ &= \tfrac{1}{n} \sum_{q_1 \in C_1, q_2 \in C_2} d(q_1, q_2) \ . \end{aligned}$$

We furthermore define

$$\mathit{Opt} := \tfrac{1}{n} \cdot \max_{C_1, C_2} \mathit{MaxCut}(P, C_1, C_2) = \max_{C_1, C_2} M(C_1, C_2) \ ,$$

and for weighted point sets C_1, C_2 with weight functions $w_1 : C_1 \to \mathbb{N}$ and $w_2 : C_2 \to \mathbb{N}$, we define

$$M(C_1, C_2) := \frac{\sum_{q_1 \in C_1, q_2 \in C_2} w_1(q_1) \cdot w_2(q_2) \cdot d(q_1, q_2)}{\sum_{q_1 \in C_1} w_1(q_1) + \sum_{q_2 \in C_2} w_2(q_2)} \ .$$

Definition 1 (ϵ-coresets) *A point set Q with integer weights $w(q)$ is an **ϵ-coreset** for P if there exists a mapping π from P to Q such that (i) $\pi^{-1}(q) = w(q)$ for every $q \in Q$ and (ii) the objective value $M(C_1, C_2)$ for any partition C_1, C_2 of P differs at most $\epsilon \cdot \mathit{Opt}$ from the objective value $M(\pi(C_1), \pi(C_2))$ of the corresponding partition of Q (think of $\pi(C_1) = \{\pi(p) | p \in C_1\}$ as a set with weights $w(q) = |\{p \in C_1 | \pi(p) = q\}|$).*

Let b be the largest side width of the bounding box of P. In [8] a family of nested grids $G^{(i)}$ is used, where $G^{(i)}$ denotes a grid of cell width $b/2^i$. Let ϱ be a confidence parameter, $0 < \varrho < 1$, and let δ be a parameter of the algorithm introduced in Lemma 17. For each grid $G^{(i)}$, a random sample $S(i)$ is chosen, where each point from P is taken to $S(i)$ with probability $\frac{\alpha}{\delta 2^i}$, where $\alpha = 12\epsilon^{-2} \ln(\varrho^{-1}) + 1$. Thus, the random sample $S(i)$ has expected size $s = \frac{\alpha}{\delta 2^i} \cdot n$.

Lemma 2 (Coresets for MaxCut [8]) *There is an algorithm that takes as input the number of points from $S(i)$ in each grid cell $C \in G^{(i)}$ and computes a weighted set of points P_C which satisfies the following constraints with probability at least $1 - \varrho$: If $\delta \leq \frac{\epsilon \cdot \mathit{Opt}}{4\sqrt{d}\,(1+\log n)\, b} \left(\frac{\epsilon}{56\sqrt{d}}\right)^d$ the set P_C is a $(c \cdot \epsilon)$-coreset of P for some constant c. If $\frac{\epsilon \cdot \mathit{Opt}}{8\sqrt{d}\,(1+\log n)\, b} \left(\frac{\epsilon}{56\sqrt{d}}\right)^d \leq \delta \leq \frac{\epsilon \cdot \mathit{Opt}}{4\sqrt{d}\,(1+\log n)\, b} \left(\frac{\epsilon}{56\sqrt{d}}\right)^d$ then the size of P_C is at most $\frac{34\sqrt{d}\,(1+\log n)}{\epsilon} \left(\frac{56\sqrt{d}}{\epsilon}\right)^d$.*

Note that a good choice for parameter δ depends on the cost of an optimal solution.

Theorem 3 (Kinetic Heaps [2]) *Let P be an initially empty set of points moving along linear trajectories in $\mathbb{R}^1$. Let $\sigma = \sigma_1, \ldots, \sigma_m$ be a sequence of m operations σ_i of the form* Insert(p, t_i) *and* Delete(p, t_i)*, such that for any two operations σ_i, σ_j with $i < j$ we have $t_i < t_j$ (the operations are performed sequentially in time). An* Insert(p, t_i) *inserts at time t_i point p into P. A* Delete(p, t_i) *removes p from P at time t_i. A kinetic heap maintains the biggest element of P. It requires $O(\log m)$ time to process an event and the expected number of events is $O(m \log m)$. Insertions and deletions are performed in expected time $O(\log^2 m)$.*

Theorem 4 (Bounding Cube Approximation [1]) *Let P be a set of n points moving in $\mathbb{R}^d$. If P is moving linearly, then after $O(n)$ preprocessing, we can construct a kinetic data structure of size $O(d)$ that maintains a 2-approximation of the smallest orthogonal box containing P. The data structure processes $O(d^2)$ events, and each event takes $O(1)$ time. The sides of the maintained box are moving linearly between the events.*

It can be decided in constant time if a flight plan update of a point p changes the data structure. At each point of time only flight plan updates of $O(d)$ points can potentially change the data structure.

3 Kinetic data structures for MaxCut

In this section we describe a KDS to maintain a $(1-\epsilon)$-approximation of a maximum cut. Our data structure supports queries of the type "*to which side of the partition belongs query point p?*". To support such a query the algorithm computes a coreset that has complexity $O(\log n/\epsilon^{d+1})$. Our data structure depends on a parameter $K = \alpha/\delta^*$, where $\delta^* = \frac{\epsilon \left(\frac{\epsilon}{56\sqrt{d}}\right)^d}{4\sqrt{d}\,(1+\log n)}$ is a lower bound for the value of δ, which can be obtained by setting $\mathit{Opt} = b$. We first create a sample set $S_{i,j}$ for every $0 \leq i, j \leq \log(Kn)$. $S_{i,j}$ is obtained from P by choosing each point $p \in P$ independently at random with probability $\min\{\frac{K}{2^{i+j}}, 1\}$.

We define $G^{(0)}$ as a 2-approximated bounding cube of P and $G^{(i)}$ as a partition of this bounding cube into 2^{id} equal sized (hyper-)cubes. For each $1 \leq i, j \leq \log(Kn)$, we maintain the set of all cells $C \in G^{(i)}$ containing sample points from $S_{i,j}$ and the number of sample points in each non-empty cell. Lemma 2 shows that at least for one value of j it is possible to compute a small coreset from the maintained information using the approach of [8].

The data structure. We assume that the cells in grid $G^{(i)}$ are numbered from 1 to 2^{id}. For each sample set $S_{i,j}$ we maintain a search tree $T_{i,j}$ that stores the cells in grid $G^{(i)}$ that contain at least one point from $S_{i,j}$. For each non-empty cell we maintain $2d$ kinetic heaps. For $1 \leq k \leq d$ we maintain one kinetic max-heap and one kinetic min-heap, where the priority of points is given by their k-th coordinate.

We maintain a 2-approximation of the bounding cube using the KDS from [1]. The $O(d^2)$ events of this KDS are called *major events*. Between any major events all movements of points and cell borders are linear.

The events. Additionally to the major events and the heap events (events caused by the kinetic heaps), our data structure stores the following (possible) events in a global event queue: For each grid $G^{(i)}$ and each non-empty cell we have an event for each dimension k, $1 \leq k \leq d$, when the maximum or minimum point with respect to that dimension crosses the corresponding cell boundary in that dimension. These events are called *minor* events. At each *major* event the movement of the grid boundaries changes and we must update every event that involves a boundary, i.e., every minor event.

Time to process events. We first consider minor events, when a point p in some set $S_{i,j}$ moves from one cell C_1 of the grid into another cell C_2. p is then deleted from $2d$ heaps corresponding to C_1 and is inserted into the $2d$ heaps corresponding to C_2. If the point moves into a cell that was previously empty, we must insert the index of C_2 into the search tree $T_{i,j}$ and initialize the $2d$ heaps. If p was the only point in C_1 we have to delete the $2d$ heaps. Since in $O(\log^2 n)$ time one can insert a point in a heap or search tree and since any insertion in a randomized kinetic heap creates $O(\log n)$ new events in expectation, we get:

Lemma 5 *Any minor event can be processed in $O(d \log^2 n)$ time. It creates $O(d \log n)$ new events in randomized kinetic heaps in expectation.* □

Lemma 6 *Any major event can be processed in expected time $O(d\,K\,n\,\log n)$.*

Proof. The time to setup our data structure at a major event is dominated by the time to setup the kinetic heaps for the boundary events. Since each kinetic heap consisting of m points can be constructed in time $O(m \cdot \log m)$ we have to count the number of sample points in all kinetic heaps. Each sample point is inserted into $2d$ kinetic heaps. The expected number of points in $S_{i,j}$ is $Kn/2^{i+j}$. By linearity of expectation we get that the total number of points in all kinetic heaps is $\sum_{i,j} 2d \cdot \frac{Kn}{2^{i+j}} = O(dKn)$. □

Lemma 7 *Between major events, every point crosses at most $d \cdot (2^i - 1)$ cells in grid $G^{(i)}$.*

Proof. Let us consider an arbitrary point p. We regard the cell boundaries in each dimension separately. In grid $G^{(i)}$ we have $2^i - 1$ internal boundaries. Since both p and the boundaries move linearly in time, p can cross each boundary at most once. Since this can happen in each of the dimensions, the lemma follows. □

Corollary 8 *The expected number of minor events is $O(d^3\,K\,n \cdot \log(Kn))$.*

Proof. The expected number of minor event involving points from $S_{i,j}$ is at most $\frac{K\,n}{2^{i+j}} \cdot d \cdot 2^i = d\,K\,n/2^j$. Summing up over all i, j we get that there are at most $O(d\,K\,n \log(Kn))$ events. □

Corollary 9 *The expected number of heap events is $O(d^4 \cdot K \cdot n \cdot \log^2(Kn))$.*

Proof. Every minor event creates an expected number of $O(d \log n)$ new events in randomized kinetic heaps. Linearity of expectation implies that the expected number of events in kinetic heaps is $O(d^4 \cdot K \cdot n \cdot \log^2(Kn))$. □

Flight plan updates. In KDS it is typically assumed that at certain points of time the *"flight plan"* of an object can change. The data structure is notified that a point now moves in another direction (possibly at a different speed) and we have to update all events in the event queue that involve this particular point. In our case we distinguish between two types of points. First, there are the two points that currently define the size of the bounding cube within the data structure from [1]. If the movement of one of these points is changed, the movement of all cells change and we have to update every event that involves a cell boundary (this is similar to the case of major events). Additionally, we have to update every 1-dimensional bounding cube we maintain.

If the flight plan of any other point is updated we simply have to update all events it is involved in and the bounding cube data structure. Since it requires $O(\log^2 n)$ time to update a kinetic heap we have to compute the expected number of such heaps a point is involved in. Every point is stored in $2d$ heaps for each set $S_{i,j}$ it is contained in. These are $O(dK)$ kinetic heaps in expectation (analogous to proof of Lemma 6)

Assume we fix some point of time and specify for each point an arbitrary flight plan update. If we choose one of these updates uniformly at random then the expected time to perform the update is small, i.e., the average cost of a flight plan update is low (proof in Appendix):

Lemma 10 *A flight plan update can be done in $O(\log^3 n \cdot \ln(\varrho^{-1})/\epsilon^{d+3})$ average expected time.*

Extracting the coreset and a solution. We can do a binary search on the different values of $\delta(j) = \delta^*/2^j$. The coreset technique described in [8] is capable to identify a value of δ, which leads to a small coreset having the desired approximation guarantees of Lemma 2. We then apply the MaxCut computation method described in [8] (also described in the Appendix in detail) to extract a solution on the coreset in $\widetilde{O}(n^2 \cdot 2^{1/\epsilon^{O(1)}})$ time.

We finally obtain our main theorem, where we assume that d is a constant:

Theorem 11 *There is a kinetic data structure that maintains a $(1+\epsilon)$-approximation for the Euclidean MaxCut problem, which is correct with probability $1-\varrho$. The data structure answers queries of the form 'to which side of the partition belongs query point p?' in $O(\log^2 n \cdot \epsilon^{-2(d+1)} \cdot 2^{1/\epsilon^{O(1)}})$ time. Under linear motion the data structure processes $\widetilde{O}(\frac{n \log(\varrho^{-1})}{\epsilon^{d+3}})$ events, which require $O(\log^2 n)$ expected time except for a constant number of events that require $\widetilde{O}(n \cdot \ln(\varrho^{-1})/\epsilon^{d+3})$ time. A flight plan update can be performed in $O(\log^3 n \cdot \ln(\varrho^{-1})/\epsilon^{d+3})$ average expected time, where the average is taken over the worst case update times of the points at an arbitrary point of time.*

4 Conclusions

In this paper we developed the first kinetic data structure for the Euclidean MaxCut problem. Our KDS is based on a coreset construction from [8]. For the streaming problems, the construction in [8] works also for other problems like k-median and k-means clustering, maximum matching, MaxTSP, and maximum spanning tree. Our KDS can be extended to the three maximization problems mentioned above (maximum matching, MaxTSP, and maximum spanning tree). However, the runtime to compute a solution from the coreset (which has to be done for each query to the data structure, or, alternatively with each event) can differ significantly. For the maximum spanning tree problem we can easily obtain similar results as for MaxCut; for the MaxTSP we do not know how to do the computation efficiently (and hence we do not obtain a very efficient KDS).

Extending our KDS to k-median and k-means clustering requires additional ideas. The technical problem is here that one cannot get a lower bound on the solution from the width of the bounding box. Hence, it is not clear how to get an upper bound on the number of events.

References

[1] P. K. Agarwal, S. Har-Peled, and K. R. Varadarajan. Approximating extent measures of points. *Journal of the ACM*, 51(4):606–635, July 2004.

[2] J. Basch. *Kinetic Data Structures*. Ph.D. thesis, Stanford University, 1999.

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

[4] J. Basch, L. J. Guibas, and G. Ramkumar. Sweeping lines and line segments with a heap. *Proc. 13th Annual ACM Symposium on Computational Geometry*, pp. 469–471, 1997.

[5] S. Bespamyatnikh, B. Bhattacharya, D. Kirkpatrick, and M. Segal. Mobile facility location. *Proc. 4th DIAL M*, pp. 46–53, 2000.

[6] G. S. Brodal and R. Jacob. Dynamic planar convex hull. *Proc. 43rd IEEE Symposium on Foundations of Computer Science*, pp. 617–626, 2002.

[7] W. Fernandez de la Vega and C. Kenyon. A randomized approximation scheme for metric MAX-CUT. *Proc. 39th IEEE Symposium on Foundations of Computer Science*, pp. 468–471, 1998.

[8] G. Frahling and C. Sohler. Coresets in dynamic geometric data streams. *Proc. 37th Annual ACM Symposium on Theory of Computing*, pp. 209–217, 2005.

[9] M. X. Goemans and D. P. Williamson. Improved approximation algorithms for Maximum Cut and satisfiability problems using semidefinite programming *Journal of the ACM*, 42:1115–1145, 1995.

[10] O. Goldreich, S. Goldwasser, and D. Ron. Property testing and its connection to learning and approximation. *Journal of the ACM*, 45(4):653–750, 1998.

[11] L. J. Guibas. Kinetic data structures — a state of the art report. *Proc. 3rd Workshop on the Algorithmic Foundations of Robotics*, pp. 191–209, 1998.

[12] L. J. Guibas. Modeling motion. In *Handbook of Discrete and Computational Geometry*, edited by J. E. Goodman and J. O'Rourke, 2nd edition, Chapter 50, pp. 1117–1134, 2004.

[13] S. Har-Peled and S. Mazumdar. Coresets for k-means and k-medians and their applications. *Proc. 36th Annual ACM Symposium on Theory of Computing*, pp. 291–300, 2004.

[14] S. Har-Peled. Clustering motion. *Discrete & Computational Geometry*, 31:545–565, 2004.

[15] J. Hershberger. Smooth kinetic maintenance of clusters. *Computational Geometry, Theory and Applications*, 31(1–2):3–30, 2005.

[16] P. Indyk. *High-dimensional Computational Geometry. PhD thesis*, Stanford, 2000.

[17] H. Kaplan, R. E. Tarjan, and K. Tsioutsiouliklis. Faster kinetic heaps and their use in broadcast scheduling. *Proc. 12th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pp. 834–844, 2001.

Appendix

A Formal arguments from [8] as used in the paper

For the sake of completeness and since some of the details of the coreset construction used in this paper differ in their presentation from [8] (which makes their use in the context of kinetic data structures simpler), in this section, we will present some proofs of the results used in our paper.

We have the following simple lemma.

Lemma 12 [7, 8] *Let C_1, C_2 be a partition of P, $p \in C_1$ and $\tilde{p}$ be an arbitrary point with $d(p,\tilde{p}) \leq D$. Then $|M(C_1,C_2) - M((C_1 \setminus \{p\}) \cup \{\tilde{p}\}, C_2)| \leq D$ (i.e., moving a point at distance at most D changes the cost of MaxCut by at most D).*

Let $\Psi = \frac{1}{n} \cdot \sum_{p \in P} p$ be the center of gravity of P. Then $Opt \geq \frac{1}{4} \sum_{p \in P} d(p,\Psi)$.

Proof. The first property follows directly from the definition of M and the triangle inequality.

To prove the second property, we use an inequality shown by Fernandez de la Vega and Kenyon [7] (in the proof of Lemma 2): For each point p: $d(p,\Psi) \leq \frac{1}{n} \sum_{q \in P} d(p,q)$.

Consider a random cut C_1, C_2 of P (for each point we flip a coin at random to decide whether it belongs to C_1 or to C_2). Since for every pair of points $p, q \in P$, the edge (p,q) is in the cut with probability $\frac{1}{2}$, the expected value of the resulting cut is $\frac{1}{4} \sum_{p,q \in P} d(p,q)$. Since $Opt \cdot n$ is the maximum value of such a cut, we conclude that $Opt \geq \frac{1}{4n} \sum_{p,q \in P} d(p,q) \geq \frac{1}{4} \sum_{p \in P} d(p,\Psi)$. □

Definition 13 (Heavy and light cells) *We say that a cell in grid $G^{(i)}$ is δ-**heavy**, if it contains more than $\delta\, 2^i$ points. A cell that is not δ-heavy is called δ-**light**.*

We describe the construction of an ϵ-coreset Q for P [8]. We say a cell $\mathcal{C}_1$ in grid $G^{(i)}$ is the *parent* of a cell $\mathcal{C}_2$ in grid $G^{(i+1)}$, if $\mathcal{C}_1$ contains $\mathcal{C}_2$. We define the coreset Q by taking to Q the center of every δ-heavy cell $\mathcal{C}$ that has no δ-heavy subcell. To determine the weights of the points in Q, we construct a mapping π from P to Q. Every point p is contained in a δ-light cell whose parent cell $\mathcal{C}$ is δ-heavy. Then p is assigned to an arbitrary coreset point in $\mathcal{C}$. We use an arbitrary mapping π that satisfies this condition. The weight of a point $q \in Q$ is $|\pi^{-1}(q)|$, i.e., the number of points assigned to q. The following theorem describes main properties of the construction.

Theorem 14 [8] *If $\delta \leq \frac{\epsilon \cdot Opt}{4\sqrt{d}\,(1+\log n)\,b} \left(\frac{\epsilon}{56\sqrt{d}}\right)^d$ then any set Q constructed as described above is an ϵ-coreset for P. If additionally $\delta \geq \frac{\epsilon \cdot Opt}{8\sqrt{d}\,(1+\log n)\,b} \left(\frac{\epsilon}{56\sqrt{d}}\right)^d$ then the number of heavy cells (and the size of the coreset) is at most $\frac{17\sqrt{d}\,(1+\log n)}{\epsilon} \left(\frac{56\sqrt{d}}{\epsilon}\right)^d$.*

To prove the Theorem, let us define $L(i)$ to be the set of occupied light cells in grid $G(i)$ whose parent cell is heavy. We partition $L(i)$ into two sets $N(i)$ and $D(i)$ according to their distance to the center of gravity Ψ:

$$N(i) \;=\; \left\{ \mathcal{C} \in L(i) : d(\mathcal{C},\Psi) \leq \frac{16\sqrt{d}\,b}{\epsilon\, 2^i} \right\}$$

and

$$D(i) \;=\; \left\{ \mathcal{C} \in L(i) : d(\mathcal{C},\Psi) > \frac{16\sqrt{d}\,b}{\epsilon\, 2^i} \right\} .$$

We observe that any point in a cell of $L(i)$ is moved at most $\sqrt{d}\,b/2^{i-1}$ during our coreset construction, because it remains in the parent cell. Furthermore, each point is contained in exactly one cell from $\bigcup L(i)$ Let us begin our analysis with points in $\bigcup D(i)$. We have the following claim.

Claim 15

$$\sum_i \sum_{p \in D(i)} \sqrt{d}\,b/2^{i-1} \;\leq\; \frac{\epsilon}{2}\, Opt \ .$$

Proof. We use a charging argument from [13]. Any point in $D(i)$ has a distance of more than $\frac{16\sqrt{d}\,b}{\epsilon\, 2^i}$ to the center of gravity Ψ. Hence we get

$$\begin{aligned} \sum_i \sum_{p \in D(i)} \sqrt{d}\,b/2^{i-1} &\leq \frac{\epsilon}{8} \sum_i \sum_{p \in D(i)} d(p,\Psi) \\ &\leq \frac{\epsilon}{8} \frac{4 \cdot Opt}{1} \;=\; \frac{\epsilon}{2} \cdot Opt \ , \end{aligned}$$

where the second inequality follows from our assumption that $Opt \geq 1/4 \sum_{p \in P} d(p,\Psi)$. □ □

Next, we consider points in $\bigcup N(i)$. We obtain the following claim.

Claim 16 *If $\delta \leq \frac{\epsilon\, Opt}{4\sqrt{d}\,k\,(1+\log n)\,b} \cdot \left(\frac{\epsilon}{56\sqrt{d}}\right)^d$, then*

$$\sum_i \sum_{p \in N(i)} \sqrt{d}\,b/2^{i-1} \;\leq\; \frac{\epsilon}{2} \cdot Opt \ .$$

Proof. By the definition of $N(i)$, every point in a cell in $N(i)$ has distance to the center of gravity Ψ of at most $\left(2\sqrt{d} + \frac{8\sqrt{d}}{\epsilon}\right) b/2^i$. Since each cell in $N(i)$ is disjoint and has side-length $b/2^i$, simple packing arguments imply the following inequality:

$$|N(i)| \;\leq\; \left(2\left(1 + 2\sqrt{d} + \frac{16\sqrt{d}}{\epsilon}\right)\right)^d \;\leq\; \left(\frac{56\sqrt{d}}{\epsilon}\right)^d \ .$$

Since each of the considered cells is light, it contains at most $\delta\, 2^i$ points. Hence,

$$\begin{aligned}
&\sum_i \sum_{p\in N(i)} \sqrt{d}\, b/2^{i-1} \\
&\leq \sum_{i:N(i)\neq\emptyset} \left(\frac{56\sqrt{d}}{\epsilon}\right)^d \delta\, 2^i\, \sqrt{d}\; Opt/2^{i-1} \\
&= \sum_{i:N(i)\neq\emptyset} 2\sqrt{d}\,\delta \left(\frac{56\sqrt{d}}{\epsilon}\right)^d \cdot Opt \ .
\end{aligned}$$

Next, let us observe that the threshold $\delta\, 2^i$ for heavy cells doubles with each grid and that we can have an occupied light cell only if the threshold is bigger than one. If the threshold is bigger than $2n$ the parent cell can not be heavy. Therefore, there are at most $1+\log n$ grids that have light grid cells whose parent cells are heavy. We conclude that,

$$\begin{aligned}
&\sum_i \sum_{p\in N(i)} \sqrt{d}\, b/2^{i-1} \\
&\leq \sum_{i:N(i)\neq\emptyset} \delta\, 2\sqrt{d} \left(\frac{56\sqrt{d}}{\epsilon}\right)^d Opt \\
&\leq (1+\log n)\,\delta\, 2\sqrt{d} \left(\frac{56\sqrt{d}}{\epsilon}\right)^d \cdot Opt \\
&\leq \frac{\epsilon}{2} \cdot Opt \ ,
\end{aligned}$$

for our choice of δ. □ □

We observe that $\bigcup_i L(i)$ covers all points and so we count the movement cost for every point. By our initial assumption, we know that if we move the points from P by an overall distance of D, then the cost of any solution changes (increases or decreases) by at most D. Therefore Claims 15 and 16 imply that the overall movement is at most $\epsilon\, Opt$. Hence, the set Q constructed by our algorithm is a coreset.

The bound for the size of the coreset follows from the observation that in each grid $G(i)$ there can be at most 2^d cells whose distance to Ψ is smaller than half of the width of the cells in $G(i)$. Hence, except for these 2^d cells, every heavy cell contributes with at least $\delta\, b/2$. We conclude that the number of marked heavy cells (which is also an upper bound on the number of coreset points) is at most $\frac{2\, Opt}{\delta\, b} + (1+\log n)\, 2^d$. Therefore, if we set $\delta \geq \frac{\epsilon\, Opt}{8\sqrt{d}\,(1+\log n)\, b} \left(\frac{\epsilon}{56\sqrt{d}}\right)^d$ and $\delta \leq \frac{\epsilon\, Opt}{4\sqrt{d}\,(1+\log n)\, b} \left(\frac{\epsilon}{56\sqrt{d}}\right)^d$, then since we assume that d is constant, we obtain the size of the coreset to be upper bounded by

$$\frac{17\sqrt{d}\,(1+\log n)}{\epsilon} \left(\frac{56\sqrt{d}}{\epsilon}\right)^d \quad \leq \quad O(k\, \log n\, \epsilon^{-(d+1)}) \ .$$

This concludes the proof of Theorem 14.

A.1 Proof of Lemma 2

The proof requires some auxiliary lemmas.

Lemma 17 [8] *Let $\epsilon < 1/3$, and let C be an arbitrary grid cell in $G^{(i)}$. The following events hold with probability at least $1-\rho$:*

- *If C contains at least $\frac{1}{2}\delta\cdot 2^i$ points, then $(1-\epsilon)\cdot n_C \leq \widetilde{n}_C \leq (1+\epsilon)\cdot n_c$.*
- *If C contains less than $\frac{1}{2}\delta\cdot 2^i$ points, then $\widetilde{n}_C \leq (1-\epsilon)\cdot\delta\cdot 2^i$.*

Proof. Let X_p be the indicator random variable that p is a sample point. Our goal is to show that $\sum_{p\in C} X_p$ does not deviate much from its expectation. If a cell contains at least $\frac{1}{2}\delta 2^i$ points then $\mathbf{E}[\sum_{p\in C} X_p] \geq \alpha/2$. From the Chernoff bound, it follows

$$\mathbf{Pr}\big[\big|\sum_{p\in C} X_p - \mathbf{E}[\sum_{p\in C} X_p]\big| \geq \epsilon\cdot\mathbf{E}[\sum_{p\in C} X_p]\big] \leq 2e^{-\epsilon^2\alpha/6} \ ,$$

and the first part of the lemma follows for the chosen value of α. To prove the second part we observe that the absolute deviation decreases when the number of points in the cell decreases. Therefore, we apply the Chernoff bound to the case when C contains $\frac{1}{2}\delta\cdot 2^i$ points. In this case the expected number of points in the cell is $\alpha/2$ and the second part of the lemma follows. □

To obtain a coreset we use the estimations $\widetilde{n}_C$ of the number of points in heavy cells to identify heavy cells (all cells having $\widetilde{n}_C \geq (1-\epsilon)\delta\, 2^i$). Since the weight of a coreset point will also depend on the number of points in some light cells, we have to estimate the number of points in these cells. To get an estimate for all required cells we use the following procedure. We require that the estimate for the number of points in a heavy cell is a $(1\pm\epsilon)$-approximation and that in every light cell there are no more points than the threshold for light cells (our coreset construction uses only these assumptions).

By Lemma 17 we know for each heavy cell C with probability $1-\rho$ we have $\widetilde{n}_C/(1+\epsilon) \leq n_C \leq \widetilde{n}_C/(1-\epsilon)$. For every heavy cell we define $L_C = \widetilde{n}_C/(1+\epsilon)$ and $U_C = \widetilde{n_C}/(1-\epsilon)$. For every light cell we define $L_C = 0$ and $U_C = \delta 2^i$ (so for every cell we know that $L_C \leq n_C \leq U_C$). We call a cell *useful*, if it is either heavy or a direct subcell of a heavy cell. We have to deal with the fact that the sum of the total estimated number of points $\sum_{C_i \text{subcell of } C} n_{C_i}$ in the subcells of C can exceed the estimated number of points in C. Therefore we will use the bounds L_C and U_C to compute new estimates E_C of the number of points in all useful cells. We require that our estimation satisfies $L_C \leq E_C \leq U_C$ and that the estimated number of points in a cell C is the sum

of the estimated number of points in its subcells. The estimates E_C can be computed bottom-up by adjusting the bounds L_C and U_C in cases of conflicts.

Corollary 18 [8] *Let $\epsilon < 1/2$. For each cell C identified as heavy we have $(1-4\epsilon)n_C \le E_C \le (1+4\epsilon)n_C$.*

Proof. The claim follows directly from the following two sequences of inequalities.

$$E_C \;\ge\; L_C \;\ge\; \widetilde{n_C}/(1+\epsilon) \;\ge\; \frac{1-\epsilon}{1+\epsilon} n_C \;\ge\; (1-2\epsilon)n_C$$

and

$$E_C \;\le\; U_C \;\le\; \widetilde{n_C}/(1-\epsilon) \;\le\; \frac{1+\epsilon}{1-\epsilon} n_C \;\le\; (1+4\epsilon)n_C \;.$$

□

We now apply the algorithm described in Section 2 to our estimations E_C and compute a coreset.

Lemma 19 [8] *If $\delta \le \frac{\epsilon \cdot Opt}{4\sqrt{d}\,(1+\log n)\,b} \left(\frac{\epsilon}{56\sqrt{d}}\right)^d$ the coreset computed with respect to the values E_C is a $(1 + O(\epsilon))$-coreset of P.*

Proof. Let P' be a point set that is distributed according to our estimations E_C (so for every useful cell C we have $|P \cap C| = E_C$). The proof of Theorem 14 shows that the coreset computed by our algorithm is a $(1+\epsilon)$-coreset for P'. Let $Q = \{q_1, \ldots, q_m\}$ be the computed coreset points. We will show that knowing the point sets P and P', our method can compute mappings $\pi : P \to Q$ and $\pi' : P' \to Q$ and weight functions $w : Q \to \mathbb{N}$ and $w' : Q \to \mathbb{N}$, such that (π, w) is a coreset for P and (π', w') is a coreset for P' and for all $q_i \in Q$ we have $(1-4\epsilon)w(q_i) \le w'(q_i) \le (1+4\epsilon)w(q_i)$. From that it easily follows that each solution on the point set P' differs by at most a factor of $(1 + O(\epsilon))$ from the solution on the point set P. Since the computed coreset is a $(1+\epsilon)$-coreset for P' it follows that it is a $(1 + O(\epsilon))$-coreset for P.

Let us construct the mappings π and π'. Theorem 14 shows that we construct a coreset when we map each point p to a coreset point in the smallest heavy cell it is contained in. We now start the assignment of points to coreset points within the smallest useful cells. Since the smallest useful cells are not heavy we do not assign them any points. We now proceed to assign points in the useful cells at the next higher level. Going through the levels bottom-up we will maintain the invariant that the number $w(q)$ of points in P mapped to a coreset-point q by π is approximately equal to the number of points $w'(q)$ mapped by π' to the coreset point:

$$(1-4\epsilon)w(q) \le w'(q) \le (1+4\epsilon)w(q) \;.$$

Let C be a heavy cell. If there is no heavy subcell, the algorithm introduces a new coreset point q. We map all E_C points from P' to q and all n_C points from P to q. Then $w(q) = E_C$ and $w'(q) = n_C$ and the invariant follows from Corollary 18. Let us now consider the case that C has already k coreset points $q_1, \ldots, q_k \in Q$ with weights $w(q_i)$ and $w'(q_i)$, respectively. Let $l := n_C - \sum_{i=1}^k w(q_i)$ resp. $l' := E_C - \sum_{i=1}^k w'(q_i)$ be the number of points which have to be assigned to the coreset points q_i by π resp. π'. We consider four cases:

- $l = 0$ *and* $l' = 0$: In this case nothing has to be assigned and the invariant holds.
- $l > 0$ *and* $l' = 0$: Then
$$\begin{aligned}&(1-4\epsilon)\sum_{i=1}^k w(q_i)\\ &= (1-4\epsilon)(n_C - l) \;<\; (1-4\epsilon)n_C\\ &\le\; E_C = \sum_{i=1}^k w'(q_i) \;.\end{aligned}$$
Therefore for one q_i we have $(1-4\epsilon)w(q_i) < w'(q_i)$ and can assign a small fraction of the points from P to q_i by π without violating the invariance. After that assignment either $l = 0$ or we find another q_i where we can assign points to. We go on with this assignment until $l = 0$.
- $l = 0$ *and* $l' > 0$: Then
$$\begin{aligned}&\sum_{i=1}^k w'(q_i) = E_C - l' \;<\; E_C \;\le\; (1+4\epsilon)n_C\\ &= (1+4\epsilon)(n_C - l) = (1+4\epsilon)\sum_{i=1}^k w(q_i) \;.\end{aligned}$$
Therefore for one q_i we have $w'(q_i) < (1+4\epsilon)w(q_i)$ and can assign a small fraction of the points from P' to q_i by π' without violating the invariance. After that assignment either $l' = 0$ or we find another q_i where we can assign points to. We go on with this assignment until $l' = 0$.
- $l > 0$ *and* $l' > 0$: We will assign $\min\{l, l'\}$ points from P to q_1 by π and $\min\{l, l'\}$ points from P' to q_1 by π'. This does not violate the invariance. After the assignment we are in the second or third case.

□

Lemma 20 [8] *If $\frac{\epsilon \cdot Opt}{8\sqrt{d}\,(1+\log n)\,b} \left(\frac{\epsilon}{56\sqrt{d}}\right)^d \le \delta \le \frac{\epsilon \cdot Opt}{4\sqrt{d}\,(1+\log n)\,b} \left(\frac{\epsilon}{56\sqrt{d}}\right)^d$ then the number of cells considered as heavy (and the size of the computed coreset) is at most $\frac{34\sqrt{d}\,(1+\log n)}{\epsilon} \left(\frac{56\sqrt{d}}{\epsilon}\right)^d$.*

Proof. The proof follows exactly the proof of Theorem 14. Since we only have a lower bound on the number of points in cells considered as heavy of $\frac{1}{2}\delta 2^i$ (instead of $\delta 2^i$), the number of cells considered as heavy can change by a factor of 2. □

Now, we can summarize the discussion in this section and observe that Lemma 2 follows directly from Lemmata 19 and 20.

B Proof of Lemma 10

Proof of Lemma 10 : It requires $O(\log^2 n)$ time to do a flight plan update of a kinetic heap. In expectation every point is stored in $O(d\,K)$ kinetic heaps. Hence the expected time required to update these heaps is $O(d\,K\,\log^2 n) = O(d \cdot \log^3 n \cdot \log(\varrho^{-1})/\epsilon^{d+3})$. Additionally, we have to update the $2d$ KDS that are used to maintain the 1-dimensional bounding cubes. This can be done in $O(d)$ time. Finally, we have to deal with updates of the 2 points that are currently defining the size of the d-dimensional bounding cube. By Lemma 6 we can process such an event in $O(dKn \log n) = O(n \cdot \log^2 n \ln(\varrho^{-1})/\epsilon^{d+3})$ expected time. Averaging over all point we get that the average expected update time is $O(d \cdot \log^3 n \cdot \log(\varrho^{-1})/\epsilon^{d+3})$. □

C Extracting a coreset from the data structure

The only problem in extracting a coreset from our grid statistics is that we do not know the cost of an optimal solution. However, Theorem 14 guarantees that for certain δ there exists a small coreset and as we have seen, one can compute this coreset from random samples. We now define $\delta(j) = \delta^*/2^j$. We extend our KDS such that it counts the number of heavy cells for each $\delta(j)$. This can be done without changing the asymptotic running time since we only have to check whether the number of points in a cell becomes more or less than $(1-\epsilon)\delta 2^i$ points each time when a point crosses a cell boundary (at each minor event). Thus we know how many heavy cells (and hence how many coreset points) exist for a certain $\delta(j)$. We choose the smallest $\delta(j)$ such that the number of heavy cells is at most $34\sqrt{d}\,(1+\log n)(56\sqrt{d})^d/\epsilon^{d+1}$.

D Computing a solution on the coreset

We now describe how to compute a MaxCut from a weighted set of points. We use an observation from [8] that an algorithm from [16] for metric MaxCut can be generalized to weighted MaxCut. The algorithm from [16] builds on a reduction from metric MaxCut to MaxCut in dense weighted graphs [7]. We follow the approach from [7, 16] and extend it to weighted MaxCut as proposed in [8]. For every point p let $w(p)$ denote the weight of point p and let $N = \sum_{p\in P} w(p)$. We will consider the point set P' of cardinality N where each point of P is replaced by $w(p)$ copies. We scale the distances such that $\sum_{p,q} w(p) \cdot w(q) \cdot d(p,q) = N^2$. For every point $p' \in P'$ we create $w'(p') := \sum_{q'\in P} d(p',q')$ *clone vertices* p_i^*, $1 \le i \le w'(p')$. Between any pair of clones p_i^*, q_j^* we create an edge with weight $\frac{d(p',q')}{w'(p')w'(q')}$. It was shown in [7] that for this choice of weights the obtained graph G is a dense weighted graph (the maximum weight exceeds the average weight by at most a constant factor) and that the weight of a MaxCut in G is equal to the weight of a MaxCut in the original metric space. For us the following observation will be crucial. Given two vertices from G we can compute the weight between them in constant time (after some preprocessing) without actually constructing G. Following the approach described in [16] one can round the weights to integers and apply the MaxCut algorithm from [10] to find a MaxCut in G. This algorithm samples a set S of $1/\epsilon^{O(1)}$ vertices and considers all partitions of these vertices into two sets. For each such partition it creates an oracle that for any remaining vertex v decides to which side of the partition it belongs by inspecting the edges from v to S. This decision is only based on these edges and a partition of the vertex set into $O(1/\epsilon)$ different sets. Since each clone of a point p' is connected in the same way to S we have to check at most $O(1/\epsilon)$ different clones to determine the partition of clones. It has been shown in [10] that for at least one of the partitions the oracle gives a $(1-\epsilon)$-approximation of the MaxCut. We can simply compute the cost of the partitions induced by the oracles and take the best one. This approach can be further improved following [10]. In the computed solution it may be that clones of the same point are assigned to both sides of the cut. Following [16], let f_a denote the fraction of clones that is assigned to one fixed side of the cut. Then we can assign all clones with probability f_a to this side and with probability $(1-f_a)$ to the other side. The expected cost of the cut will be similar to the cost of the computed cut and repeating this assignment $O(1/\epsilon)$ times will ensure with constant probability that the best of these assignments is only a factor $(1-O(\epsilon))$ away from the MaxCut. Since every coreset point corresponds to an area of the plane, the cut also induces a partition of the plane.

Theorem 21 [8] *Given a point set P with integer weights and of cardinality n one can compute a MaxCut of P in $\widetilde{O}(n^2 \cdot 2^{1/\epsilon^{O(1)}})$ time.* □