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 *MaxCut*. 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[1], 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.

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 such a 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. Let P be a point set in 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 all

*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.

[1]We use the $\widetilde{O}$-notation to suppress logarithms in the largest occurence of a (function of a) variable, i.e. for any function $f(x)$ we have $f(x)\cdot\log(f(x))=\widetilde{O}(f(x))$.

partitions of P into C_1 and C_2:

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

We furthermore define

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

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 by at most $\epsilon \cdot Opt$ 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, $0 < \varrho < 1$. The coreset algorithm depends on a parameter δ. For each grid $G^{(i)}$, a random sample $S(i)$ is chosen, where each point from P is added 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 $\delta \leq \frac{\epsilon \cdot Opt}{4\sqrt{d}\,(1+\log n)\, b} \left(\frac{\epsilon}{56\sqrt{d}}\right)^d$ the set P_C is a $(c \cdot \epsilon)$-coreset of P for some constant c. If $\frac{\epsilon \cdot Opt}{8\sqrt{d}\,(1+\log n)\, b} \left(\frac{\epsilon}{56\sqrt{d}}\right)^d \leq \delta \leq \frac{\epsilon \cdot Opt}{4\sqrt{d}\,(1+\log n)\, b} \left(\frac{\epsilon}{56\sqrt{d}}\right)^d$ 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 Box 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 $Opt = b$. 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. Notice that for a fixed value of j the sample sets $S_{i,j}$ correspond to the sample sets $S(i)$ needed to construct a coreset with value $\delta = \delta^* \cdot 2^j$. Lemma 2 therefore 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:

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 $\delta(j) = \delta^* \cdot 2^j$. 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] 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.