

Coresets in Dynamic Geometric Data Streams

Gereon Frahling^{*}
Heinz Nixdorf Institute and
Computer Science Department
University of Paderborn
Germany
frahling@upb.de

Christian Sohler[†]
Heinz Nixdorf Institute and
Computer Science Department
University of Paderborn
Germany
csohler@upb.de

ABSTRACT

A dynamic geometric data stream consists of a sequence of m insert/delete operations of points from the discrete space $\{1, \dots, \Delta\}^d$ [26]. We develop streaming $(1 + \epsilon)$ -approximation algorithms for k -median, k -means, MaxCut, maximum weighted matching (MaxWM), maximum travelling salesperson (MaxTSP), maximum spanning tree (MaxST), and average distance over dynamic geometric data streams. Our algorithms maintain a small weighted set of points (a coreset) that approximates with probability $2/3$ the current point set with respect to the considered problem during the m insert/delete operations of the data stream. They use $\text{poly}(\epsilon^{-1}, \log m, \log \Delta)$ space and update time per insert/delete operation for constant k and dimension d .

Having a coreset one only needs a fast approximation algorithm for the weighted problem to compute a solution quickly. In fact, even an exponential algorithm is sometimes feasible as its running time may still be polynomial in n . For example one can compute in $\text{poly}(\log n, \exp(O((1 + \log(1/\epsilon)/\epsilon)^{d-1})))$ time a solution to k -median and k -means [21] where n is the size of the current point set and k and d are constants. Finding an implicit solution to MaxCut can be done in $\text{poly}(\log n, \exp((1/\epsilon)^{O(1)}))$ time. For MaxST and average distance we require $\text{poly}(\log n, \epsilon^{-1})$ time and for MaxWM we require $O(n^3)$ time to do this.

Categories and Subject Descriptors

F.2 [Theory of Computation]: Analysis of Algorithms and Problem Complexity

General Terms

Algorithms, Theory

^{*}Supported by the DFG Research Training Group GK-693 of the Paderborn Institute for Scientific Computation (PaSCo).

[†]Supported by IST programme of EC under contract no. IST-2002-001-907 (DELIS).

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

STOC'05, May 22-24, 2005, Baltimore, Maryland, USA.
Copyright 2005 ACM 1-58113-960-8/05/0005 ...\$5.00.

Keywords

Streaming Algorithms, Computational Geometry, Data Structures

1. INTRODUCTION

The increasing inter-connectivity of modern computer systems has led to the phenomenon of massive data sets occurring in the form of *data streams*. A prominent example for such a data stream is the internet traffic at a backbone router. Assume we want to maintain some statistics about the routed packets. It would be way too costly to store the required information (e.g., source and destination) for every packet routed. It seems to be much more attractive to maintain a small *sketch* (or *synopsis*) of the data seen so far. Such a sketch should contain an approximation of the information we are interested in. Applications of data streaming algorithms include sensor networks, analysis of astrophysical data, and (spatial) data bases. In the aforementioned applications data streams must be seen in a geometric context and the data items can be geometric objects, e.g. points in the \mathbb{R}^d .

In this paper we study dynamic geometric data streams that have first been introduced in [26]. Such a data stream consists of a sequence of INSERT and DELETE operations of points from the discrete space $\{1, \dots, \Delta\}^d$. We assume that the input sequence is consistent, i.e. a DELETE(p) can only occur, if p has previously been inserted. Also a point p cannot be inserted, if it is currently present in the point set (the current point set is not a multiset). The assumption that the points are from the discrete space $\{1, \dots, \Delta\}^d$ is not very common in Computational Geometry. It can be justified by the fact that computations are typically done using finite precision arithmetic. Equivalently, we could assume that the smallest and largest distance between two points (or a lower and upper bound) are known in advance. Therefore, from now on let P denote a point set in $\{1, \dots, \Delta\}^d$.

We develop streaming algorithms for the following fundamental problems: k -median, k -means clustering, MaxCut, maximum weighted matching, maximum travelling salesperson, maximum spanning tree, and average distance. Our algorithms are based on a new construction method for coresets. Such a coreset is a small weighted set of points that approximates the cost of any solution to the problem under consideration with relative error $(1 \pm \epsilon)$. We show that one can maintain a coreset under insertions and deletions.

Dynamic data streams have applications in the context of mobile ad-hoc networks, sensor networks, databases, web graph analysis, etc. For example, in a mobile ad-hoc network the participants may regularly broadcast updates of their current position. All participants want to maintain information about the distribution of the participants to maintain an efficient communication network. Since mobile devices have usually limited memory it would be nice to do

this using only a small amount of space. Of course, one can model an update as deleting the old position and inserting the new one. Therefore, the model of dynamic geometric data streams applies. In particular, k -median, k -means and average distance are interesting in this scenario as they give valuable information about the structure of the point set that can be used for the maintainance of an efficient communication network.

1.1 Related work

One of the first geometric problems studied in the streaming model was to approximate the diameter of a point set in 2D [12] using $O(1/\epsilon)$ space. Indyk considered the same problem in higher dimensions [25] and obtained an algorithm with space complexity $O(dn^{1/(c^2-1)})$ to maintain a c -approximate diameter for $c > \sqrt{2}$. Cormode and Muthukrishnan introduced the *radial histogram* [9], which can be used to approximate different geometric problems including diameter, convex hull, furthest neighbors, etc. Hershberger and Suri showed how to maintain a set of $2r$ points such that the distance from the true convex hull of the points seen so far is $O(D/r^2)$ where D is the current diameter of the sample set [22]. Chan used coresets to approximate different geometric problems including diameter and min-volume bounding box [6]. Together with Sadjad he considered the diameter and width of a point set in the sliding widow model [7]. Bagchi et al. showed how to maintain ϵ -nets and ϵ -approximations of a data stream [3]. Suri et al. developed data streaming algorithms for range counting [32]. Har-Peled and Mazumdar gave $(1 + \epsilon)$ -approximation algorithms for the k -median and k -means problem [21]. They also mention the extension of their results to the case of dynamic streaming algorithms as an interesting open problem.

Streaming algorithms for clustering problems have also been considered in the more general metric space setting [4, 18, 28]. The currently best known algorithm for the k -median problem is a $O(1)$ -approximation using $O(k \text{polylog} n)$ space [8].

Indyk introduced the model for dynamic geometric data streams used in the present paper [26]. He gave $O(d \cdot \log \Delta)$ -approximation algorithms for (the weight of) minimum weighted matching, minimum bichromatic matching and minimum spanning tree. He also showed how to approximate the weight of an optimal solution of the facility location problem within a factor of $O(d \log^2 \Delta)$. For the k -median problem he gave a $(1 + \epsilon)$ approximation algorithm with query time $O(\Delta^{kd} \cdot k \cdot \epsilon^{-d-1} (\log \Delta + \frac{1}{\epsilon} \log \frac{1}{\epsilon}))$ (exhaustive search). He also developed a $O(1)$ -approximation that needs $O(\Delta^d \cdot k \cdot \epsilon^{-d-1} (\log \Delta + \frac{1}{\epsilon} \log \frac{1}{\epsilon}))$ time to compute an approximation from the maintained data structure. He further gave a $(1 + \epsilon, O(\log \Delta (\log \Delta + \log(1/\epsilon)/\epsilon)))$ -approximation algorithm, i.e. an algorithm that returns $O(\log \Delta (\log \Delta + \log(1/\epsilon)/\epsilon)) \cdot k$ medians whose cost is at most $1 + \epsilon$ times the cost of an optimal algorithm. This algorithm has polylogarithmic query time. For the minimum spanning tree weight a $(1 + \epsilon)$ -approximation has been given [15].

For other work on streaming algorithms we refer to the survey by Muthukrishnan [29].

1.2 Our Contribution

We develop streaming algorithms for a number of fundamental problems over dynamic geometric data streams. Table 1 gives a summary of our results. Except for the k -median problem, we give the first algorithms in the dynamic geometric streaming model for all considered problems. For the k -median problem we develop the first $(1 + \epsilon)$ -approximation algorithm with efficient query time. The previous algorithm by Indyk was designed to show that in principle a $(1 + \epsilon)$ -approximation dynamic streaming algorithm using polylog space exists [26]. Its running time is $\tilde{O}(\Delta^{kd}/\epsilon^{d-1})$ and it

uses exhaustive search, i.e. it enumerates all sets of cardinality k of the input space.

All our algorithms maintain a coreset for the considered problem. Once we have the coreset we can run an arbitrary $(1 + \epsilon)$ -approximation algorithm on this small set and quickly get a good approximation for the original point set.

Apart from the specific results the main contribution of the paper is our new method for constructing a coreset. The coreset is obtained by combining statistics about the distribution of points in $\log \Delta$ nested grids. To obtain these statistics in the streaming context we use the fact that for a given cost Opt of the optimal solution the distribution of points within the grids cannot be arbitrary. This way we can get statistics with higher precision than it would be possible for arbitrary input distributions.

Our results can be modified to obtain similar results (with some additional $\log n$ factors and replacing $\log \Delta$ by $\log n$) for the insertion-only case where the input points have arbitrary positions in the \mathbb{R}^d . Our algorithms also have applications in a distributed scenario. Assume the input data is distributed over a number of data streams. Then we can compute a coreset for each of these data streams. Finally, we can aggregate the coresets at one computer and compute a solution for the union of the coresets. This solution will be a $(1 + \epsilon)$ -approximation of the original point set.

2. PRELIMINARIES

Let P denote a set of n points in $\{1, \dots, \Delta\}^d$. We will assume that d is a constant. Since $|P| \leq \Delta^d$ we have $\log n = O(\log \Delta)$. Let $d(p, q)$ denote the Euclidean distance between p and q . In the *k -median clustering problem* we try to find a set C of k points in \mathbb{R}^d such that $Median(P, C) = \sum_{p \in P} d(p, C)$ is minimized, where $d(p, C) = \min_{c \in C} d(p, c)$. In the *k -means clustering problem* we want to minimize $Means(P, C) = \sum_{p \in P} d(p, C)^2$. The extension of the definitions to weighted point sets is straightforward. The following definition is from [21].

DEFINITION 2.1 (CORESETS I.). [21] *Let P be a weighted set of n points in the \mathbb{R}^d . A weighted point set P_{core} in \mathbb{R}^d is an ϵ -coreset for the k -median problem, if for every set C of k centers $(1 - \epsilon) \cdot Median(P, C) \leq Median(P_{core}, C) \leq (1 + \epsilon) \cdot Median(P, C)$. In a similar way a weighted point set P_{core} is an ϵ -coreset for the k -means clustering problem, if for every set C of k centers $(1 - \epsilon) \cdot Means(P, C) \leq Means(P_{core}, C) \leq (1 + \epsilon) \cdot Means(P, C)$.*

In the Max-Cut problem the goal is to partition the set P into two sets C_1, C_2 such that the sum

$$MaxCut(P, C_1, C_2) = \sum_{p \in C_1, q \in C_2} d(p, q)$$

of inter-cluster distances is maximized. For the Max-Cut problem there is no set of centers and so we need a different notion for coresets. The idea is that the coreset is a multiset with a bijection into P . This multiset can be represented as a weighted set where the point weights stand for the multiplicity of a point. Since MaxCut is a maximization problem we only guarantee that for no partition the value of the solution changes by more than $\pm \epsilon Opt$, where Opt is the cost of an optimal partition.

DEFINITION 2.2 (CORESETS II.). *Let P be a weighted set of n points in the \mathbb{R}^d . A multiset S of n points with a bijection π in P is called ϵ -coreset for the Max-Cut problem, if for every partition of P into clusters C_1 and C_2 we have $MaxCut(P, C_1, C_2) - \epsilon \cdot Opt \leq MaxCut(S, \pi(C_1), \pi(C_2)) \leq MaxCut(P, C_1, C_2) + \epsilon \cdot Opt$.*

Problem	Space	Coreset construction
k-median	$O((\log \Delta + \log m)^3 k^2 \log^4 \Delta / \epsilon^{2d+4})$	$O((\log \Delta + \log m)^2 k \log n / \epsilon^{d+2})$
k-means	$O((\log \Delta + \log m)^3 k^2 \log^4 \Delta / \epsilon^{2d+6})$	$O((\log \Delta + \log m)^2 k \log n / \epsilon^{d+3})$
MaxCut	$O((\log \Delta + \log m)^3 \log^4 \Delta / \epsilon^{2d+4})$	$O((\log \Delta + \log m)^2 \log n / \epsilon^{d+2})$
Other	$O((\log \Delta + \log m)^3 \log^4 \Delta / \epsilon^{2d+4})$	$O((\log \Delta + \log m)^2 \log n / \epsilon^{d+2})$

Figure 1: Summary of our results in the dynamic streaming model. In the table m denotes the number of insert/delete operations in the data stream and n is the size of the current point set. The row for other problems summarizes the results for MaxWM, MaxTSP, MaxST, and average distance. Update time is $O(\log^2 \Delta (\log \Delta + \log m))$ for all problems. The approximation guarantee is $1 + \epsilon$.

The Euclidean versions of the following graph problems are defined on the complete Euclidean graph, i.e. the complete weighted graph over vertex set P whose edge lengths are the Euclidean distances between the corresponding vertices. The *maximum matching* problem asks to find a perfect matching of the points in P that maximizes the sum of the length of the matching edges. For a matching \mathcal{M} we denote its cost by

$$\text{MaxMatching}(P, \mathcal{M}) = \sum_{(p, q) \in \mathcal{M}} d(p, q) .$$

The *maximum travelling salesperson problem* is to find a simple cycle \mathcal{C} (a tour) of the points in P with maximal cost. We denote its cost by

$$\text{MaxTSP}(P, \mathcal{C}) = \sum_{(p, q) \in \mathcal{C}} d(p, q) .$$

The maximum spanning tree problem is to find a spanning tree \mathcal{T} with maximum cost. We denote its cost by

$$\text{MaxST}(P, \mathcal{T}) = \sum_{(p, q) \in \mathcal{T}} d(p, q) .$$

We can define ϵ -coresets for these problems in a similar way as for the Max-Cut problem.

3. CORESETS FOR k -MEDIAN

The first problem we consider is the k -median problem. In the following we describe and analyze our coreset construction. For the moment we assume that we know the cost $Opt = \text{Median}(P, C_{opt})$ of an optimal set C_{opt} of centers. Later we explain how to get rid of this assumption and how to maintain such a coreset in the dynamic streaming model.

To construct our coresets we impose $\log \Delta$ nested square grids $\mathcal{G}^{(1)}, \dots, \mathcal{G}^{(\log \Delta)}$ over the point space. The side length of the grid cells in grid $\mathcal{G}^{(i)}$ is 2^i . Our goal will be to identify for each grid $\mathcal{G}^{(i)}$ its *heavy cells*, i.e. cells that contain more than a certain threshold of points. This threshold depends on the size of the grid and grows with the inverse of the grid size (the smaller the cells, the larger the threshold). We will parametrize the threshold by some small value δ , which is specified later.

DEFINITION 3.1. *We call a cell of grid $\mathcal{G}^{(i)}$ heavy, if it contains at least $\delta \cdot \frac{Opt}{2^i}$ points of P . A grid cell that is not heavy is light. Further we call a grid cell ϵ -close to heavy, if it contains at least $(1 - \epsilon) \cdot \delta \cdot \frac{Opt}{2^i}$ points. A cell that is not ϵ -close to heavy is called ϵ -far from heavy.*

We start our construction with the coarsest grid $\mathcal{G}^{(\log \Delta)}$. First, we identify every heavy cell in $\mathcal{G}^{(\log \Delta)}$. Then we proceed similar to the process of building a quadtree. We subdivide every heavy cell \mathcal{C} into 2^d equal sized quadratic subcells. These subcells are contained in grid $\mathcal{G}^{(\log \Delta - 1)}$. We call \mathcal{C} the *parent cell* of these subcells. If none of these subcells is heavy we mark \mathcal{C} . Otherwise, we recurse

this process with all heavy subcells. The recursion eventually stops because at some point a heavy cell is required to have more than n points inside. At the end of this recursive process we choose the center of every marked cell to be its *representative point*. Then we want to replace every point by some representative point. In the end we get a set of points consisting of multiples of representative points. These multiples are then represented by weighted points instead.

The replacement is done as follows. We replace every point contained in a marked cell by the corresponding representative point. All other points must be contained in a light cell whose parent cell is heavy and not marked. We replace these points by an arbitrary representative point contained in the parent cell. For suitably chosen δ the resulting weighted set of points P_{core} is an ϵ -coreset for the k -median problem of size $O(k \cdot \log n / \epsilon^d)$.

We denote by $\mathcal{L}(i)$ the set of light cells of grid $\mathcal{G}^{(i)}$ whose parent cell is heavy.

CLAIM 3.2. *Any point $p \in \mathcal{L}(i)$ is moved a distance of at most $\sqrt{d}2^{i+1}$ during our coreset construction.*

Proof : If the parent cell of p is a marked heavy cell, then p is moved to the representative point of that cell. Hence it stays in the parent cell and is moved by a distance of at most $\sqrt{d}2^{i+1}$. If the parent cell is not marked, then p is moved to an arbitrary representative point in the parent cell. Again it stays in the parent cell and is moved by a distance of at most $\sqrt{d}2^{i+1}$. \square

From now on let C be an arbitrary fixed set of k centers. We partition the sets $\mathcal{L}(i)$ into two subsets $\mathcal{L}_{near}(i)$ and $\mathcal{L}_{dist}(i)$. $\mathcal{L}_{near}(i)$ contains all cells \mathcal{C} whose distance $\min_{q \in C} d(q, \mathcal{C})$ to the nearest center from C is at most $\frac{4\sqrt{d}}{\epsilon} \cdot 2^i$, i.e.

$$\mathcal{L}_{near}(i) = \{ \mathcal{C} \in \mathcal{L}(i) \mid \min_{q \in C} d(q, \mathcal{C}) \leq \frac{4\sqrt{d}}{\epsilon} \cdot 2^i \} .$$

$\mathcal{L}_{dist}(i)$ contains all other cells from $\mathcal{L}(i)$, i.e.

$$\mathcal{L}_{dist}(i) = \{ \mathcal{C} \in \mathcal{L}(i) \mid \min_{q \in C} d(q, \mathcal{C}) > \frac{4\sqrt{d}}{\epsilon} \cdot 2^i \} .$$

CLAIM 3.3.

$$\sum_i \sum_{p \in \mathcal{L}_{dist}(i)} \sqrt{d}2^{i+1} \leq \frac{\epsilon}{2} \text{Median}(P, C) .$$

Proof : We use a charging argument from [21]. Any point in $\mathcal{L}_{dist}(i)$ has a distance of more than $\frac{4\sqrt{d}}{\epsilon} \cdot 2^i$ from the nearest center in C . Therefore, we get

$$\begin{aligned} \sum_i \sum_{p \in \mathcal{L}_{dist}(i)} \sqrt{d}2^{i+1} &= \frac{\epsilon}{2} \sum_i \sum_{p \in \mathcal{L}_{dist}(i)} \frac{4\sqrt{d}}{\epsilon} \cdot 2^i \\ &\leq \frac{\epsilon}{2} \cdot \text{Median}(P, C) . \end{aligned}$$

□

CLAIM 3.4. For $\delta \leq \frac{1}{4k\sqrt{d} \cdot (\log n + 1)} \cdot \left(\frac{\epsilon}{14\sqrt{d}}\right)^d$ we get

$$\sum_i \sum_{p \in \mathcal{L}_{near}(i)} \sqrt{d} 2^{i+1} \leq \frac{\epsilon}{2} \text{Median}(P, C) .$$

Proof : We observe that the furthest point in a cell in $\mathcal{L}_{near}(i)$ can have a distance of at most $(2\sqrt{d} + \frac{4\sqrt{d}}{\epsilon}) \cdot 2^i$ to the nearest center. Since there are k centers we get

$$|\mathcal{L}_{near}(i)| \leq k \cdot (2 \cdot (1 + 2\sqrt{d} + \frac{4\sqrt{d}}{\epsilon}))^d \leq k \cdot (14\sqrt{d}/\epsilon)^d .$$

Each of the considered cells is light and so it contains at most $\delta \cdot \text{Opt}/2^i$ points. Hence,

$$\begin{aligned} & \sum_i \sum_{p \in \mathcal{L}_{near}(i)} \sqrt{d} 2^{i+1} \\ & \leq \sum_i k \cdot (14\sqrt{d}/\epsilon)^d \cdot \delta \cdot \text{Opt}/2^i \cdot \sqrt{d} 2^{i+1} \\ & \leq \sum_i \delta \cdot k \cdot 2\sqrt{d} \cdot (14\sqrt{d}/\epsilon)^d \cdot \text{Median}(P, C) \end{aligned}$$

Now observe that for every $i \geq \log(\delta \cdot \text{Opt})$ every cell that contains a point is heavy and so there are no light cells. Further we observe that for those $i < \log(\delta \cdot \text{Opt}/n)$ there are no heavy cells. Therefore, there are at most $\log n + 1$ grids that have light grid cells whose parent cells are heavy. We conclude that

$$\begin{aligned} & \sum_i \delta \cdot k \cdot 2\sqrt{d} \cdot (14\sqrt{d}/\epsilon)^d \cdot \text{Median}(P, C) \\ & \leq \delta \cdot (\log n + 1) \cdot k \cdot 2\sqrt{d} \cdot (14\sqrt{d}/\epsilon)^d \cdot \text{Median}(P, C) \\ & \leq \frac{\epsilon}{2} \text{Median}(P, C) \end{aligned}$$

for our choice of δ . □

LEMMA 3.5. The set P_{core} is an ϵ -coreset for $\delta \leq \frac{1}{4k\sqrt{d} \cdot (\log n + 1)} \cdot \left(\frac{\epsilon}{14\sqrt{d}}\right)^d$.

Proof : We first observe that $\bigcup_i \mathcal{L}(i)$ covers the input space and so we count the movement cost for every point. Further we observe that the cost of any set of k centers changes by at most $\pm D$ when the points of the point set P are moved by an overall distance of D . By Claims 3.2, 3.3 and 3.4 we get that the overall movement is at most $\epsilon \text{Median}(P, C)$. Hence the set P_{core} constructed by our algorithm is a coreset. □

3.1 Size of the Coreset

To determine the size of the coreset we count the number of heavy cells that are marked by our algorithm. For every grid $\mathcal{G}^{(i)}$ we define $\mathcal{M}(i)$ to be the set of marked heavy cells. Further we need the following notion of center cells.

DEFINITION 3.6. Let C_{opt} denote the optimal set of k centers for the k -median problem. A cell C in grid $\mathcal{G}^{(i)}$ is called a center cell, if its distance to the nearest center in C_{opt} is at most 2^{i-1} , i.e. if $\min_{q \in C_{opt}} d(q, C) \leq 2^{i-1}$.

We define $\mathcal{M}_{center}(i) = \{C \in \mathcal{M}(i) | C \text{ is center cell}\}$ to be the subset of marked heavy cells that are center cells. We use $\mathcal{M}_{external}(i) = \mathcal{M}(i) \setminus \mathcal{M}_{center}(i)$ to denote the remaining marked heavy cells. We call these cells *external* cells.

CLAIM 3.7. Every external cell contributes with $\delta \cdot \text{Opt}/2$ to the cost Opt of C_{opt} .

Proof : Every external cell C is a heavy cell and so it contains at least $\delta \cdot \text{Opt}/2^i$ points. Each point contributes with at least 2^{i-1} to the cost of the optimal solution. Hence the overall contribution of the points in C is at least $\delta \cdot \text{Opt}/2$. □

LEMMA 3.8. The size of the coreset is at most $\frac{2}{\epsilon} + (\log n + 2) \cdot k \cdot 2^d = O(k \log n / \epsilon^d)$.

Proof : From Claim 3.7 it follows that

$$|\bigcup_i \mathcal{M}_{external}(i)| \leq 2/\delta .$$

We also know that

$$|\mathcal{M}_{center}(i)| \leq k \cdot 2^d .$$

We observe that there cannot be any marked heavy cells in grids $\mathcal{G}^{(i)}$ for $i \geq \log(\delta \cdot \text{Opt}) + 1$. In that case the threshold for heavy cells is smaller than $1/2$ and so any heavy cell contains another heavy cell as a subcell. Thus it cannot be marked. For $i < \log(\delta \cdot \text{Opt}/n)$ the threshold for heavy cells is larger than n and so there cannot be any (marked) heavy cells in the corresponding grids either. Therefore, we get

$$|\bigcup_i \mathcal{M}_{center}(i)| \leq (\log n + 1) \cdot k \cdot 2^d .$$

The lemma follows since $\bigcup_i \mathcal{M}(i) = \bigcup_i (\mathcal{M}_{external}(i) \cup \mathcal{M}_{center}(i))$ is the set of marked heavy cells. □

3.2 Coresets in data streams

We now give a high level description of our streaming algorithm. Then in Subsection 3.3 we show in more detail how to implement the algorithm for the insertion-only case. In Subsection 3.4 we explain how to deal with deletions.

When we want to use our coreset construction for data streams we have to deal with the fact that we cannot determine the number of points in a cell exactly (at least not for many cells). Therefore, we cannot exactly determine which cells are heavy. Our approach is to pick a random sample of points from the current set of points. We choose our sample size in such a way that every heavy cell contains $\Omega(\epsilon^{-2} \log n + \log \Delta)$ sample points. Then it follows from a variant of Chernoff bounds [31] that we can approximate the number of points in every heavy cells up to a multiplicative error of $(1 \pm \epsilon)$. If a cell contains more points than $(1 - \epsilon)$ times the threshold for heavy cells, we classify it as heavy. This way, we detect every heavy cell but we also classify some light cells as heavy. This will increase the size of our coreset but it will also refine it (recall that we are allowed to move points in light cells to every representative point in its parent cell). We can also show that the size of the coreset increases at most by the number of cells that are ϵ -close to heavy. Since the contribution of any such cell to the optimal solution will be almost as high as the minimal contribution of a heavy cell we can easily modify the proof of Lemma 3.8 to show:

COROLLARY 3.9. Let $\epsilon < 1/2$ and assume that no cell that is ϵ -far from heavy is considered as heavy. Then the size of the coreset is at most $\frac{4}{\epsilon} + (\log n + 2) \cdot k \cdot 2^{d+1} = O(k \log n / \epsilon^d)$.

REMARK 3.10. We remark that in principle it would be possible to determine heavy cells using algorithms for finding frequent items in data streams [5, 10]. This can be done by identifying each

point with the cell it is contained in and then running these algorithms on the domain 'grid cells'. However, we have to use the fact that the distribution of points within the grids cannot be arbitrary (because the side length of the grid cells is a function of Opt). Otherwise, the given approximation guarantee will be too weak. Since the analysis of [10] is for general input distributions, we would have to modify the analysis taking our special input distribution into account.

The space complexity of the algorithm from [5] depends on certain parameters of the input distribution. Using Lemma 3.11 one can derive these parameters and get a sufficient approximation guarantee.

3.3 Insertions

We show how our streaming algorithm works in the insertion-only case. The algorithm takes a random sample from the points to find the heavy cells in grid $\mathcal{G}^{(i)}$ and to get an estimation for the number of points in every heavy cell in that grid. Since we do not know the value of Opt in advance we have to run an instance of our algorithm for every possible value 2^j of Opt with $j \in \{1, \dots, \lceil (d+1) \log(\Delta+1) \rceil\}$. We then try to determine the cells that contain more than $\delta \cdot 2^{j-i}$ points. If $2^j < Opt$ then we will determine all heavy cells but we will also categorize some light cells as heavy. Since our set of approximations contains a $1/2$ -approximation of Opt , it follows from Corollary 3.9 that for some value of j the coreset has size $O(k \log n / \epsilon^{d+1})$.

To select our random sample we want to take every element in the data stream with probability $\frac{\alpha}{\delta} \cdot 2^{i-j}$ into our sample, where

$$\alpha = 6 \cdot \epsilon^{-2} \ln(\rho^{-1}) + 1$$

and ρ is some variable related to the confidence probability of our streaming algorithm. Thus our random sample has expected size

$$s = \frac{\alpha}{\delta} \cdot n \cdot 2^{i-j}.$$

Hence, if $2^j < Opt$ then any heavy cell contains at least α sample points in expectation (or the threshold for heavy cells is at most 1 and our sample contains the whole stream). To pick our random sample we choose a hash function $h : \Delta^d \rightarrow \{1, \dots, \delta \cdot 2^{j-i} / \alpha\}$ from a class of α -independent hash functions and consider the set $S = h^{-1}(1) \cap P$. Since S may be of polynomial size we do not store S and instead proceed as follows. We maintain a counter for every cell in $\mathcal{G}^{(i)}$ that contains at least one point in S . The counter counts the number of points from S in the corresponding cell. When a new point p arrives we check if $h(p) = 1$. If this is the case we determine the cell the point is contained in. If it is the first point in the corresponding cell we initiate a new counter for the cell, otherwise we increase the corresponding old one. If the number of counters exceeds $\frac{8\alpha}{\delta} + k \cdot 2^d$ we cancel the instance. As we will see, in this case we have $2^j \ll Opt$ with overwhelming probability. When we want to compute a coreset we look at the instance with smallest j -value that is still alive. If the coreset for this value of j turns out to be larger than the upper bound on the coreset size given in Corollary 3.9 then we increment j as long as the coreset is larger than this bound.

LEMMA 3.11. *Let $2^j > Opt/2$. Then we have points from at most $\frac{8\alpha}{\delta} + k \cdot 2^d$ cells in our sample set with probability at least $1 - \rho$.*

Proof : We determine an upper bound on the number of points in non-center grid cells. Let us recall that every point except for those contained in the $k \cdot 2^d$ center cells has a distance of at least 2^{i-1} to

the nearest center in an optimal solution. Thus the overall number of points in non-center cells is at most 2^{j-i+2} . Let X_p denote the indicator random variable for the event that $h(p) = 1$. Let \mathcal{D} denote the set of non-center grid cells. We have $\mathbf{E}[\sum_{p \in \mathcal{D}} X_p] \leq \frac{4\alpha}{\delta}$. We will assume $\mathbf{E}[\sum_{p \in \mathcal{D}} X_p] = \frac{4\alpha}{\delta}$ as the deviation is maximized in this case. Applying Theorem ?? (see Appendix) we get

$$\begin{aligned} \Pr \left[\left| \sum_{p \in \mathcal{D}} X_p - \mathbf{E} \left[\sum_{p \in \mathcal{D}} X_p \right] \right| \geq \mathbf{E} \left[\sum_{p \in \mathcal{D}} X_p \right] \right] \\ \leq e^{-\min\{\lfloor \alpha/2 \rfloor, \lfloor \epsilon^2 \frac{\alpha}{\delta} / 3 \rfloor\}} \\ \leq \rho. \end{aligned}$$

Therefore, with probability at least $1 - \rho$ we have at most $\frac{8\alpha}{\delta}$ points from non-center cells in our sample. If no two of these $\frac{8\alpha}{\delta}$ points are contained in the same grid cell we get an upper bound of $\frac{8\alpha}{\delta}$ on the number of non-center cells that contain a sample point. The lemma follows from the fact that there are at most $k \cdot 2^d$ center cells. \square

LEMMA 3.12. *Let $\epsilon < 1/3$, let \mathcal{C} be an arbitrary grid cell in $\mathcal{G}^{(i)}$ and let j be fixed. Further let $n_{\mathcal{C}}$ denote the number of points in \mathcal{C} . Then the following events hold with probability at least $1 - \rho$*

- If \mathcal{C} contains at least $\delta \cdot 2^{j-i-1}$ points, then $(1 - \epsilon) \cdot n_{\mathcal{C}} \leq |S \cap \mathcal{C}| \cdot n/s \leq (1 + \epsilon) \cdot n_{\mathcal{C}}$.
- If \mathcal{C} contains less than $\delta \cdot 2^{j-i-1}$ points, then $|S \cap \mathcal{C}| \cdot n/s < (1 - \epsilon) \cdot \delta \cdot 2^{j-i}$.

Proof : The proof again follows from Theorem ??.

Let X_p denote the indicator random variable for the event that $h(p) = 1$. We want to show that $\sum_{p \in \mathcal{C}} X_p$ does not deviate much from its expectation. If a cell contains at least $\delta \cdot 2^{j-i-1}$ points then $\mathbf{E}[\sum_{p \in \mathcal{C}} X_p] \geq \alpha/2$. From Theorem ?? it follows

$$\Pr \left[\left| \sum_{p \in \mathcal{C}} X_p - \mathbf{E} \left[\sum_{p \in \mathcal{C}} X_p \right] \right| \geq \epsilon \cdot \mathbf{E} \left[\sum_{p \in \mathcal{C}} X_p \right] \right] \leq e^{-\min\{\lfloor \alpha/2 \rfloor, \lfloor \epsilon^2 \alpha / 6 \rfloor\}}$$

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 Theorem ?? to the case when \mathcal{C} contains $\delta \cdot 2^{j-i-1}$ points. In this case the expected number of points in the cell is $\alpha/2$ and the second part of the lemma follows. \square

To obtain the coreset we apply the algorithm described at the beginning of Section 3 and use our random sample to identify heavy cells. If for a cell \mathcal{C} we have $|S \cap \mathcal{C}| \cdot n/s \geq (1 - \epsilon) \cdot \delta \cdot 2^{j-i}$ then we classify it as heavy. For every heavy cell \mathcal{C} we use the value $\tilde{n}_{\mathcal{C}} = \frac{|S \cap \mathcal{C}| \cdot n}{s}$ as an approximation for the number $n_{\mathcal{C}}$ of points in \mathcal{C} . By Lemma 3.12 we know that with probability at least $1 - \rho$ we have $\tilde{n}_{\mathcal{C}} / (1 + \epsilon) \leq n_{\mathcal{C}} \leq \tilde{n}_{\mathcal{C}} / (1 - \epsilon)$. For every heavy cell we define $L_{\mathcal{C}} = \tilde{n}_{\mathcal{C}} / (1 + \epsilon)$ and $U_{\mathcal{C}} = \tilde{n}_{\mathcal{C}} / (1 - \epsilon)$. For every light cell we define $L_{\mathcal{C}} = 0$ and $U_{\mathcal{C}} = \delta 2^{j-i}$. We call a cell *useful*, if it is either heavy or a subcell of a heavy cell. In our coreset construction we need an approximation $E_{\mathcal{C}}$ for the number of points in every useful cell \mathcal{C} . We require that our estimation satisfies $L_{\mathcal{C}} \leq E_{\mathcal{C}} \leq U_{\mathcal{C}}$ and that the estimated number of points in a cell \mathcal{C} is the sum of the estimated number of points in its subcells. Let us consider an arbitrary point set P' that is distributed according to our estimations.

LEMMA 3.13. *For every set \mathcal{C} of k centers*

$$\begin{aligned} (1 - O(\epsilon)) \text{Median}(P, \mathcal{C}) &\leq \text{Median}(P', \mathcal{C}) \\ &\leq (1 + O(\epsilon)) \text{Median}(P, \mathcal{C}). \end{aligned}$$

Hence, an ϵ -coreset for P' is a $O(\epsilon)$ -coreset for P .

Proof : We construct a coreset for P' . We slightly modify our coreset construction from Section 3. We say a cell is heavy, if the cell is classified as heavy by our sampling procedure above. Using this definition of heavy we run our coreset construction. For a given coreset P_{core} of P we can construct a coreset P'_{core} of P' such that the point locations in P_{core} and P'_{core} are the same and the weight of every point differs by a factor of at most $(1 \pm \epsilon)$. It follows that $(1 - \epsilon)Median(P_{core}, C) \leq Median(P'_{core}, C) \leq (1 + \epsilon)Median(P'_{core}, C)$. By the definition of coresets, the lemma follows. \square

Hence, an ϵ -coreset for P' is a $O(\epsilon)$ -coreset for P .

To obtain the E_C values we first consider every useful cell in the finest grid. We group the useful cells into groups of (at most) 4 such that every cell in a group has the same parent cell. Then we consider every group separately. By adding the lower bounds for the number of points in the cells of a group we obtain a (possibly) new lower bound on the number of points in the parent cell. If this lower bound is stronger than the existing one for the parent cell we replace it. Similarly, we can obtain a new upper bound. This way we obtain new bounds for all cells in the second finest grid. We use these bounds to obtain new bounds for the 3rd finest grid and so on. After obtaining these upper and lower bounds for every cell we can find a feasible solution top-down starting with the single cell in the coarsest grid that contains all n points.

Lemma 3.12 shows that every heavy cell is approximated with sufficient precision. Additionally, Lemma 3.12 tells us that no cell that is 0.3-far from heavy is classified as heavy. By Corollary 3.9 we have a bound on the coreset size.

THEOREM 1. *Our streaming algorithm maintains a data structure for ϵ -coresets for the k -median of a data stream of point insertions. Our data structure needs $O(k\epsilon^{-(d+2)} \log^4 \Delta)$ space. An insertion can be processed in $O(\epsilon^{-2} \cdot \log^3 \Delta)$ time. An ϵ -coreset can be extracted in $O(k\epsilon^{-d} \log^3 \Delta)$ time.*

Proof : We set $\rho = 1/\Theta(m \cdot \Delta^{2d} \log^2 \Delta)$. By the union bound Lemma 3.12 and Lemma 3.11 hold with probability $1 - \frac{1}{\Delta^d \cdot m}$ for all choices of $C, i,$ and j . Thus with probability $1 - \frac{1}{\Delta^d}$ the algorithm maintains a coreset over the whole stream of size m .

We will now count the number of memory cells we use. For each of the $O(\log \Delta)$ estimations of Opt and for each of the $O(\log \Delta)$ grids we maintain at most $\frac{8\alpha}{\delta} + k \cdot 2^d = O(k\epsilon^{-(d+2)} \cdot (\log \Delta + \log m) \log n)$ counters according to Lemma 3.11. They occupy $O(k\epsilon^{-(d+2)} \cdot \log^2 \Delta \cdot (\log \Delta + \log m) \cdot \log n)$ memory cells which dominates the memory use.

When a new point arrives in the stream we have to do the following update steps: For each estimation of Opt and for each grid we have to apply the hash function h to decide if the new point is a sample point. This needs $O(\alpha \log^2 \Delta) = O(\epsilon^{-2} \log^2 \Delta (\log \Delta + \log m))$ time. Then we have to find the corresponding cell and counter, which can be realized using dictionaries in constant time for each grid. We get an overall insertion time of $O(\epsilon^{-2} \log^2 \Delta \cdot (\log \Delta + \log m))$.

We will now bound the coreset extraction time. Let us first assume we now the right value of Opt in advance. Since (according to the proof of Lemma 3.8) there are at most $O(k \log n / \epsilon^d)$ heavy cells in each grid, we have a bound of $O(k \log^2 \Delta / \epsilon^d)$ on the number of heavy cells. The marking process yielding to the actual coreset can be seen as a quadtree traversal. Since each inner node of this tree corresponds to a heavy cell, the tree traversal can

be done in time $O(k \log^2 \Delta / \epsilon^d)$. Since we have to do this process for each value of Opt (and then choose the minimum value with a small coreset), we get a total running time of $O(k \log^3 \Delta / \epsilon^d)$. \square

3.4 Deletions

In the presence of deletions we still have to maintain a random sample. We use in the following a slight modification of an algorithm proposed in [19] to get a random sample in the presence of deletions. Other methods for random sampling in dynamic data streams have been proposed by Indyk [26] and Cormode and Muthukrishnan [11].

When we allow deletions one problem occurs. We cannot stop an instance of our algorithm, if the number of counters exceeds $\frac{8\alpha}{\delta} + k \cdot 2^d$. For example, it could happen that in the first half of the stream many points are inserted and the number of counters is way too large. But then most of these points can be deleted in the second half of the stream such that we eventually have less occupied cells than our threshold for the number of counters. If we want to obtain a coreset at that point of time we have to know these values *and* the corresponding cells.

To overcome this problem we implement our counters using another hash function $h_2 : C^{(i)} \rightarrow \{1, \dots, 3 \cdot (2 \cdot \frac{8\alpha}{\delta} + k \cdot 2^d)^2\}$, where $C^{(i)}$ denotes the set of grid cells in $\mathcal{G}^{(i)}$. The hash function h_2 is chosen from a class of pairwise independent hash functions. For every bucket b we maintain a counter c_b and a sum S_b of point coordinates. If a point p is inserted we detect the cell C it is contained in. Then we increment the counter $c_{h_2(C)}$ and add the coordinates of p to S_b . If a point p is deleted we decrement the counter $c_{h_2(C)}$ and subtract the coordinates of p from S_b .

If at some point of time we have at most $2 \cdot \frac{8\alpha}{\delta} + k \cdot 2^d$ occupied cells then h_2 is likely to have no collisions (see Lemma 3.14 below). If there are no collisions we just check for every b , if $c_b > 0$. If this is the case then we compute the center of gravity S_b/c_b of the points that increased counter c_b . Since there are no collisions all points that increased c_b come from the same cell and so their center of gravity is also contained in that cell. Hence we can determine for every counter c_b the corresponding cell.

To determine whether there are at most $\frac{8\alpha}{\delta} + k \cdot 2^d$ occupied cells we can use a streaming algorithm to approximately count the number of distinct elements in a data stream [14, 1, 2]. In this case the elements are the grid cells in $\mathcal{G}^{(i)}$. If the algorithm tells us that there are less than $2 \cdot \frac{8\alpha}{\delta} + k \cdot 2^d$ occupied cells, we assume that there are no collisions in the corresponding hash function. It remains to calculate the collision probability of h_2 .

LEMMA 3.14. *Let $C \subseteq C^{(i)}$ be a subset of cells of grid $\mathcal{G}^{(i)}$. If $|C| \leq 2 \cdot \frac{8\alpha}{\delta} + k \cdot 2^d$ then*

$$\Pr[\exists C_1, C_2 \in C \text{ with } h_2(C_1) = h_2(C_2) \text{ and } C_1 \neq C_2] \leq 1/3$$

Proof : Since h_2 is from a class of pairwise independent hash functions we obtain for every pair of cells $C_1, C_2 \in C^{(i)}, C_1 \neq C_2$ that $\Pr[h_2(C_1) = h_2(C_2)] = \frac{1}{3(2 \cdot \frac{8\alpha}{\delta} + k \cdot 2^d)^2}$. By the union bound we get that

$$\begin{aligned} & \Pr[\exists C_1, C_2 \in C \text{ with } h_2(C_1) = h_2(C_2) \text{ and } C_1 \neq C_2] \\ & \leq \sum_{\substack{C_1, C_2 \in C \\ C_1 \neq C_2}} \Pr[h_2(C_1) = h_2(C_2)] \leq 1/3 \end{aligned}$$

\square

To achieve an error probability of at most ρ we use standard amplification techniques. We run $O(\log(\rho^{-1}))$ independent copies of the hashing scheme and use majority vote.

THEOREM 2. *Let m denote the number of insert/delete operations in the data stream. Our streaming algorithm maintains with probability at least $2/3$ a data structure for ϵ -coresets for the k -median of a dynamic geometric data stream. Our data structure needs $O((\log \Delta + \log m)^3 \cdot k^2 \cdot \log^4 \Delta / \epsilon^{2d+4})$ space and an insertion/deletion can be processed in $O(\log^2 \Delta (\log \Delta + \log m))$ time. An ϵ -coreset can be extracted in $O((\log \Delta + \log m)^2 \cdot k \cdot \log n / \epsilon^{d+2})$ time. Using the algorithm from [21] one can compute a $(1 + \epsilon)$ -approximation from the coreset in $O(k^5 \log^9 n + k^2 \cdot \log^5 n \cdot \exp(O((1 + \log(1/\epsilon)/\epsilon)^{d-1})))$ time. \square*

Proof : For every $1 \leq i, j \leq (d+1) \log \Delta$ we need to maintain $O(\log(\rho^{-1}))$ copies of h_2 and the associated $O((\alpha/\delta)^2) = O((\log \Delta + \log m)^2 k^2 \log^2 \Delta / \epsilon^{2d+4})$ counters. Since we want to have confidence probability at least $2/3$ we can use $\rho \leq \frac{1}{c \cdot \Delta^d \cdot m}$ for some constant c . Since the space to store the hash functions is negligible we need $O(\frac{(\log \Delta + \log m)^3 \cdot k^2 \cdot \log^4 \Delta}{\epsilon^{2d+4}})$ space.

To process insertion or deletions we have to apply for the $O(\log^2 \Delta)$ pairs of (i, j) each of the $O(\log(\rho^{-1}))$ copies of h_2 . This needs $O(\log^2 \Delta (\log \Delta + \log m) \alpha) = O(\log^2 \Delta (\log \Delta + \log m)^2 / \epsilon^2)$ time and dominates the cost of the update time.

To extract the coreset we first have to do the majority vote. This can be done by $O(\log(\rho^{-1}))$ comparisons of sparse vectors with at most $2 \cdot \frac{8\alpha}{\delta} + k \cdot 2^d$ entries. Therefore, the time for the majority vote is $O(\log^2(\rho^{-1}) \cdot k \log n / \epsilon^{d+2}) = O((\log \Delta + \log m)^2 \cdot k \cdot \log n / \epsilon^{d+2})$ time. This dominates the running time for the coreset construction. \square

4. K-MEANS CLUSTERING

One can modify our construction to obtain similar results for the k -means clustering problem. In the k -means clustering the contribution of a point at distance D from the nearest center is D^2 . Therefore, we have to change the definition of a heavy cell. For the k -means clustering we call a cell heavy, if it contains more than $\delta \cdot \frac{Opt}{2^{2d}}$ points. Using this definition the proofs are essentially similar to the case of the k -median problem.

THEOREM 3. *Let m denote the number of insert/delete operations in the data stream. Our streaming algorithm maintains with probability at least $2/3$ a data structure for ϵ -coresets for the k -means clustering problem in a dynamic geometric data stream. Our data structure needs $O((\log \Delta + \log m)^3 \cdot k^2 \cdot \log^4 \Delta / \epsilon^{2d+6})$ space and an insertion/deletion can be processed in $O(\log^2 \Delta \cdot (\log \Delta + \log m))$ time. An ϵ -coreset can be extracted in $O((\log \Delta + \log m)^2 \cdot k \cdot \log n / \epsilon^{d+3})$ time. Using the algorithm from [21] one can compute a $(1 + \epsilon)$ -approximation in $O(k^5 \log^9 n + k^{k+2} \cdot \epsilon^{-(2d+1)k} \cdot \log^{k+1} n)$ time. \square*

5. MAXCUT

We will show how to extend our techniques to find a coreset for MaxCut. We will scale the objective function by a factor of $1/n$ to obtain many similarities to the proofs for the k -median problem. In the following let Opt denote the maximum value of $\frac{1}{n} \cdot \sum_{p \in C_1} \sum_{q \in C_2} d(p, q)$.

We start with a lower bound that relates the cost of an optimal solution to the distance of the points from the center of gravity. A similar idea has been used in [13] to analyze a $(1 + \epsilon)$ -approximation algorithm for MaxCut. We use the following lemma which is proven in the proof of Lemma 2 in [13].

LEMMA 5.1. [13] *Let c denote the center of gravity. The*

$$d(p, c) \leq \frac{1}{n} \sum_{q \in P} d(p, q) .$$

COROLLARY 5.2. *Let c denote the center of gravity of all points. Then*

$$Opt \geq \frac{1}{4n} \cdot \sum_{p, q \in P} d(p, q) \geq \frac{1}{4} \cdot \sum_{p \in P} d(p, c)$$

Proof : To show the first inequality we consider a random cut C_1, C_2 . For each point $p \in P$ we flip a coin 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 $\frac{1}{4n} \sum_{p, q \in P} d(p, q)$. Since Opt denotes the maximum value of such a cut, the first inequality holds. From Lemma 5.1 the corollary follows. \square

We want to show that the algorithm from Section 3 computes a coreset for the MaxCut problem. We will briefly discuss the changes that have to be done in the proof of Lemma 3.5. We want to show that the cost of every partition of P changes by at most $\pm \epsilon \cdot Opt$ when we consider the corresponding partition of P_C . In the following let c be the fixed center of gravity. We proceed similar to the proof of Claims 3.3-3.4 and Lemma 3.5 and consider $\{c\}$ as a possible solution to the 1-median problem. We observe that the cost of the normalized objective function for the MaxCut problem changes by at most D when a point is moved a distance of D . Following the proof of Claims 3.3-3.4 we see that the overall movement costs for MaxCut are at most $\epsilon/2 \cdot (Median(P, \{c\}) + Opt)$. By Lemma 5.2 we have $Opt \geq \frac{1}{4} \sum_{q \in P} d(q, c) = \frac{1}{4} Median(P, \{c\})$. Hence we get an additive error of $5/2 \cdot \epsilon \cdot Opt$.

It remains to show that the coreset is of small size. Here we use again the fact $Opt \geq \frac{1}{4} Median(P, \{c\})$. Every heavy non-center cell contributes to $Median(P, \{c\})$ with at least $\delta \cdot Opt/2 \geq \frac{1}{8} Median(P, \{c\})$. Therefore, there can be at most $(\frac{8}{\delta} + \log n + 2) \cdot k \cdot 2^d = O(k \log n / \epsilon^d)$ heavy cells, which gives an upper bound on the size of the coreset.

To make our streaming algorithm work the problem must also have the property that changing the weight of the representative points by an ϵ -fraction will not significantly change the value of the solution of the problem (it will still be within a factor of $(1 \pm O(\epsilon))$). This property trivially holds for the MaxCut problem.

Finally, we remark that one can find a $(1 + \epsilon)$ -approximation to the MaxCut of a weighted point set of cardinality ℓ in time $O(\text{poly}(\ell) \cdot \exp(1/\epsilon))$. This can be done by using an approach from [24] that combines the techniques from [13] and [17] and has been used to obtain an $O(n + \exp(1/\epsilon))$ time $(1 + \epsilon)$ -approximation algorithm for metric MaxCut. In [13] metric MaxCut is solved by a reduction to MaxCut in dense graphs. One can run the MaxCut algorithm from [17] on this graph without explicit construction.

Since every representative point corresponds to a certain number of cells (one heavy cell and possibly one or more light cells) we get a partition of the input space into $O(k \log n / \epsilon^d)$ regions. We can always find a solution such that every copy of a representative point belongs to the same side of the cut. Therefore, each region corresponds to one side of the cut and we have an implicit solution to the MaxCut problem that assigns every point of P to the side that corresponds to the region it is contained in.

THEOREM 4. *Let m denote the number of insert/delete operations in the data stream. Our streaming algorithm maintains with probability at least $2/3$ a data structure for ϵ -coresets for the MaxCut of a dynamic geometric data stream. Our data structure needs*

$O(((\log \Delta + \log m)^3 \cdot \log^4 \Delta) / \epsilon^{2d+4})$ space and an insertion / deletion can be processed in $O(\log^2 \Delta (\log \Delta + \log m))$ time. An ϵ -coreset can be extracted in $O((\log \Delta + \log m)^2 \cdot \log n / \epsilon^{d+2})$ time. One can extract an implicit solution from the coreset in $\text{poly}(\exp((1/\epsilon)^{O(1)}), \log n)$ time.

6. OTHER PROBLEMS

We show how to adapt our method to find coresets for the problems maximum weighted matching, maximum spanning tree, maximum travelling salesperson and average distance. In the following c denotes the center of gravity of all points. The coreset construction is similar to the one for MaxCut. First for each value we will define the notion of Opt . For maximum matching, maximum spanning tree, and MaxTSP this will be the value of an optimal solution. In the case of the average distance problem Opt will have a slightly different meaning. For each problem we will show that the following two statements are true: (i) When we move a point p by a distance of at most D , the solution changes by at most $O(D)$. (ii) $Opt \geq \frac{1}{2} \sum_{p \in P} d(p, c)$.

Under these two assumptions we can show that the algorithm from Section 3 computes a coreset for the considered problem. This can be proven by slightly changing the proof for MaxCut. To make our streaming algorithm work the problem must also have the property that changing the weight of the representative points by an ϵ -fraction will not significantly change the value of the solution of the problem (it will still be within a factor of $(1 \pm O(\epsilon))$). This property holds for the four problems below.

6.1 Maximum Weighted Matchings

W.l.o.g. assume we have an even number of points. Let Opt denote the value of a maximum weighted matching. Look at a random matching constructed the following way. We connect the first point p to one point $q \in P \setminus \{p\}$ uniformly chosen from all other points. Then we delete both points from P and go on with this construction until all points are matched. Notice that each pair (p, q) of points is matched with probability $\frac{1}{n-1}$. When M denotes the aggregated weight of this matching, $\mathbf{E}[M] = \frac{1}{2(n-1)} \sum_{p, q \in P} d(p, q)$. We show that $Opt \geq \frac{1}{2} \sum_{p \in P} d(p, c)$. A similar argument as used for MaxCut shows that for each point p we have $\frac{1}{n} \sum_{q \in P} d(p, q) \geq d(p, c)$. We obtain

$$Opt \geq \mathbf{E}[M] \geq \frac{1}{2n} \sum_{p, q \in P} d(p, q) \geq \frac{1}{2} \sum_{p \in P} d(p, c)$$

Since moving a point by a distance of D changes the value of the optimal solution by at most D we have all properties to find a coreset for maximum weighted matching.

Having the coreset, one can compute a $(1 + \epsilon)$ -approximate solution to the Maximum Weighted Matching problem. We replace each coreset point $p \in P_{core}$ having weight $w(p)$ by $w(p)$ unweighted points and run the algorithm of Gabow [16], which needs $O(n^3)$ time.

6.2 Maximum Spanning Tree

Let Opt denote the value of the maximum spanning tree. Since we can complete each matching to a spanning tree, we get from the proof of Max Matching:

$$Opt \geq \mathbf{E}[M] \geq \frac{1}{2} \sum_{p \in P} d(p, c)$$

Clearly moving a point by a distance of D changes the value of the

optimal solution by at most D .

Having a coreset of size $O(k \cdot \log n / \epsilon^d)$ we can compute the Maximum Spanning Tree in time $O(k^2 \cdot \log^2 n / \epsilon^{2d})$ using Prim's algorithm [30].

6.3 Maximum Travelling Saleman

Similar to Maximum Spanning Tree we can complete each matching to a TSP tour. So again $Opt \geq \mathbf{E}[M] \geq \frac{1}{2} \sum_{p \in P} d(p, c)$ holds. Moving a point by a distance of D changes the value of the optimal solution by at most $2D$. We obtain a solution to the MaxTSP problem by exhaustive search.

6.4 Average Distance

In case we want to find a coreset which approximates Average Distance, we will define $Opt = \frac{1}{n-1} \sum_{p, q \in P} d(p, q)$. We will find a coreset which approximates the value of Opt up to a multiplicative error of $(1 + \epsilon)$. Note that this also leads to an approximation of average distance by dividing the value Opt by $2n$. From the definition of Opt we directly get

$$Opt \geq \mathbf{E}[M] \geq \frac{1}{2} \sum_{p \in P} d(p, c)$$

Moving a point by a distance of D changes the value of Opt by at most D .

We obtain a solution to average distance in $O((k \log n / \epsilon^d)^2)$ time by summing up all distances.

THEOREM 5. *Let m denote the number of insert/delete operations in the data stream. Our streaming algorithm maintains with probability at least $2/3$ a data structure for ϵ -coresets for MaxWM, MaxTSP, MaxST and average distance in a dynamic geometric data stream.*

Our data structure needs $O(((\log \Delta + \log m)^3 \cdot \log^4 \Delta) / \epsilon^{2d+4})$ space and an insertion/deletion can be processed in $O(\log^2 \Delta \cdot (\log \Delta + \log m))$ time. An ϵ -coreset can be extracted in $O((\log \Delta + \log m)^2 \cdot \log n / \epsilon^{d+2})$ time. \square

7. CONCLUSIONS

We have introduced a new method to maintain coresets for k -median, k -means, MaxCut and a number of other problems in dynamic data streams. From such a coreset one can quickly compute a $(1 + \epsilon)$ -approximation for the original point set. The size of our coreset is $O(k \log n / \epsilon^d)$ where n is the number of points in the original point set. Recently, Har-Peled and Kushal showed that one can compute a coreset for the k -median and k -means problem whose size does not depend on n [20]. One interesting open question is, whether it is also possible to maintain such a constant sized (for constant d, k and ϵ) coreset in dynamic data streams.

8. REFERENCES

- [1] N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. *Proc. 28th Annual ACM Symposium on Theory of Computing (STOC)*, pp. 20–29, 1996.
- [2] Z. Bar-Yossef, T.S. Jayram, R. Kumar, D. Sivakumar, and L. Trevisan. Counting distinct elements in a data stream. *Proceedings of the 6th International Workshop on Randomization and Approximation Techniques*, pages 1–10, 2002.
- [3] A. Bagchi, A. Chaudhary, D. Eppstein and M. T. Goodrich. Deterministic sampling and range counting in geometric data

- streams. *Proc. 20th Annual Symposium on Computational Geometry (SoCG)*, pp. 144–151, 2004.
- [4] M. Charikar, C. Chekuri, T. Feder, and R. Motwani. Incremental clustering and dynamic information retrieval. *Proc. 29th Annual ACM Symposium on Theory of Computing (STOC)*, 626–635, 1997.
- [5] M. Charikar, K. Chen, M. Farach-Colton. Finding Frequent Items in Data Streams. *Proc. . 29th Annual International Colloquium on Automata, Languages and Programming (ICALP)*, pp. 693–703, 2002.
- [6] T. M. Chan. Faster core-set constructions and data stream algorithms in fixed dimensions. *Proc. 20th Annual Symposium on Computational Geometry (SoCG)*, pp. 152–159, 2004.
- [7] T. Chan, B. Sadjad. Geometric Optimization Problems Over Sliding Windows. *Proc. 15th Annual International Symposium on Algorithms and Computation (ISAAC)*, pp. 246–258, 2004.
- [8] M. Charikar, L. O’Callaghan, and R. Panigrahy. Better streaming algorithms for clustering problems. *Proc. 35th Annual ACM Symposium on Theory of Computing (STOC)*, pp. 30–39, 2003.
- [9] G. Cormode and S. Muthukrishnan. Radial histograms for spatial streams. DIMACS Technical Report 2003-11, 2003.
- [10] G. Cormode and S. Muthukrishnan. Improved Data Stream Summaries: The Count-Min Sketch and its Applications. *Proc. 6th Latin American Theoretical Informatics (LATIN)*, pp. 29–38, 2004.
- [11] G. Cormode and S. Muthukrishnan and I. Rozenbaum. Summarizing and Mining Inverse Distributions on Data Streams via Dynamic Sampling. *DIMACS Technical Report 2005-11*, 2005.
- [12] J. Feigenbaum, S. Kannan, and J. Zhang. Computing diameter in the streaming and sliding window models. Technical Report YALEU/DCS/TR-1245, Yale University, 2002.
- [13] W. Fernandez de la Vega and C. Kenyon. A Randomized Approximation Scheme for Metric MAX-CUT. *J. Comput. Syst. Sci.*, 63(4):531-541, 2001.
- [14] P. Flajolet and G. Martin. Probabilistic counting algorithms for data base applications. *Journal of Computer and System Sciences*, 31:182–209, 1985.
- [15] G. Frahling, P. Indyk, and C. Sohler. Sampling in Dynamic Data Streams and Applications. To appear in *Proc. 21st Symposium on Computational Geometry*, 2005.
- [16] H.N. Gabow. Data structures for weighted matching and nearest common ancestors with linking *Proc. 1st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 434–443, 1990.
- [17] O. Goldreich, S. Goldwasser, D. Ron. Property testing and its connection to learning and approximation. *Journal of the ACM*, 45(4):653–750, 1998.
- [18] S. Guha, N. Mishra, R. Motwani, and L. O’Callaghan. Clustering data streams. *Proc. 41st IEEE Symposium on Foundations of Computer Science (FOCS)*, 359–366, 2000.
- [19] S. Gunguly, M. Garofalakis, R. Rastogi. Processing Set Expressions over Continuous Update Streams. *Proc. ACM SIGMOD*, pp. 265-276, 2003.
- [20] S. Har-Peled and A. Kushal. Smaller Coresets for k-Median and k-Means Clustering.
- [21] 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 (STOC)*, 291–300, 2004.
- [22] J. Hershberger and S. Suri. Adaptive sampling for geometric problems over data streams. *Proc. ACM Symposium on Principles of Database Systems (PODS)*, 2004.
- [23] P. Indyk. Personal communication, 2004.
- [24] P. Indyk. High-dimensional Computational Geometry. *Phd. thesis*, Stanford University, 2000.
- [25] P. Indyk. Better algorithms for high-dimensional proximity problems via asymmetric embeddings. *Proc. 14th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pp. 539–545, 2003.
- [26] P. Indyk. Algorithms for Dynamic Geometric Problems over Data Streams. *Proc. 36th Annual ACM Symposium on Theory of Computing (STOC)*, pp. 373–380, 2004.
- [27] P. Indyk. Better Algorithms for high-dimensional proximity problems via asymmetric embeddings. *Proc. 14th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 539–545, 2003.
- [28] A. Meyerson. Online facility location. *Proc. IEEE Symposium on Foundations of Computer Science (FOCS)*, pp. 426–431, 2001.
- [29] S. Muthukrishnan. Data streams: Algorithms and applications (invited talk at SODA’03). Available at <http://athos.rutgers.edu/~muthu/stream-1-1.ps>, 2003.
- [30] R. Prim. Shortest Connection Networks and some Generalizations. *Bell Systems Technical Journal*, 36:1389-1401, 1957.
- [31] J. Schmidt, A. Siegel, and A. Srinivasan. Chernoff-Hoeffding bounds for applications with limited independence. *SIAM Journal on Discrete Mathematics*, 8(2):223–250, 1995.
- [32] S. Suri, C. D. Toth, and Y. Zhou. Range counting over multidimensional data streams. *Proc. 20th Annual Symposium on Computational Geometry*, pp. 160–169, 2004.

APPENDIX