Sampling in Dynamic Data Streams and Applications

Gereon Frahling, Piotr Indyk, Christian Sohler

2005
Coresets in Dynamic Geometric Data Streams

Gereon Frahling *
Computer Science Department
University of Paderborn
Germany
frahling@upb.de

Christian Sohler †
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 \([1, \ldots, \Delta]^d\). 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 \(poly(\epsilon^{-1}, \log m, \log \Delta)\) 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 \(poly(\log n, \exp(O((1+\log(1/\epsilon)/\epsilon)^{d-1}))\) 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 explicit solution to MaxCut can be done in \(poly(\log n, \exp((1/\epsilon)^{O(1)})\) time. For MaxST and average distance we require \(poly(\log n, \epsilon^{-1})\) time and for MaxWM we require \(O(n^2)\) 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).

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 to 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 \([1, \ldots, \Delta]^d\). 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 \([1, \ldots, \Delta]^d\) 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 \([1, \ldots, \Delta]^d\).

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 maintenance 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 $O(dn^{1/(d-1)})$ to maintain a $c$-approximate diameter for $c > \sqrt{d}$.

Cormode and Muthukrishnan introduced the radial histogram [9], which can be used to approximate different geometric problems including diameter, convex hull, furthest neighbors, etc. 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.

Indyk introduced the model for dynamic geometric data stream used in the present paper [26]. He gave $O(d \cdot \log \Delta / \epsilon)$-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 $O((\Delta^d \cdot k \cdot \epsilon^{-d-1}) \cdot (\log \Delta + \frac{1}{\epsilon} \log \frac{1}{\epsilon}))$ (exhaustive search). He also developed a $O(1)$-approximation that needs $O((\Delta^d \cdot k \cdot \epsilon^{-d-1}) \cdot (\log \Delta + \frac{1}{\epsilon} \log \frac{1}{\epsilon}))$ time to compute an approximation from the maintained data structure. He further gave a $(1 + \epsilon)$, $O((\log \Delta \log \Delta + \log^3(1/\epsilon)/\epsilon))$-approximation algorithm, i.e. an algorithm that returns $O((\log \Delta \log \Delta + \log^3(1/\epsilon)/\epsilon)) \cdot k$ 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 I 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 $O(\Delta^{kd}/\epsilon^{d-1})$ 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 $\mathbb{R}^d$. 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 $[1, \ldots, \Delta]^d$. We will assume that $d$ is a constant. Since $|P| \leq \Delta^d$ we have $\log \Delta = O(\log \Delta)$.

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 $\text{Median}(P, C) = \sum_{p \in C} d(p, C)$ is minimized, where $d(p, C) = \min_{c \in C} d(p, c)$. In the $k$-means clustering problem we want to minimize $\text{Means}(P, C) = \sum_{p \in C} d(p, c)^2$. 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 $\mathbb{R}^d$. A weighted point set $P_{core}$ in $\mathbb{R}^d$ is an $\epsilon$-coreset for the $k$-median problem, if for every set $C$ of $k$ centers $(1 - \epsilon) \cdot \text{Median}(P_{core}, C) \leq \text{Median}(P_{core}, C) \leq (1 + \epsilon) \cdot \text{Median}(P_{core}, C)$. In a similar way a weighted point set $P_{core}$ is an $\epsilon$-coreset for the $k$-means clustering problem, if for every set $C$ of $k$ centers $(1 - \epsilon) \cdot \text{Means}(P_{core}, C) \leq \text{Means}(P_{core}, C) \leq (1 + \epsilon) \cdot \text{Means}(P_{core}, C)$. In the MaxCut problem the goal is to partition the set $P$ into two sets $C_1, C_2$ such that the sum 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 multisets 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 of the value space the change of solutions by more than $\pm \epsilon \cdot \text{Opt}$, where $\text{Opt}$ is the cost of an optimal partition.*

**Definition 2.2 (Coresets II.)**[21] *Let $P$ be a weighted set of $n$ points in $\mathbb{R}^d$. A multisets $S$ of $n$ points with a bijection $\pi$ in $P$ is called $\epsilon$-coreset for the Max-Cut problem, if for every partition of $P$ into clusters $C_1$ and $C_2$ we have $\text{MaxCut}(P, C_1, C_2) - \epsilon \cdot \text{Opt} \leq \text{MaxCut}(S, \pi(C_1), \pi(C_2)) \leq \text{MaxCut}(P, C_1, C_2) + \epsilon \cdot \text{Opt}$.***
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 \textit{maximum matching} problem asks to find a perfect matching of the points in \( P \) with maximum total distances between the corresponding vertices. The \textit{maximum travelling salesperson problem} is to find a simple tour (a tour) of the points in \( P \) with maximum cost. We denote its cost by \( \text{MaxMatching}(P) \).

We can define an \( \epsilon \)-coreset for the \textit{Max-Cut} problem. For the following we describe and analyze our coreset construction. For the \textit{Max-Cut} problem.

More specifically, we first introduce a grid \( G \) over \( P \) with grid size \( \Delta \). Let \( \mathcal{C}_\text{opt} \) be an optimal set of centers. Later we explain how to get rid of this assumption and how to maintain such a coreset in the dynamic streaming model.

We can approximate the \textit{Max-Cut} problem with \( \epsilon \)-coresets. For the \textit{Max-Cut} problem.

First, we introduce a grid \( G \) over \( P \) with grid size \( \Delta \). Let \( \mathcal{C}_\text{opt} \) be an optimal set of centers. Later we explain how to get rid of this assumption and how to maintain such a coreset in the dynamic streaming model.

We can approximate the \textit{Max-Cut} problem with \( \epsilon \)-coresets. For the \textit{Max-Cut} problem.

First, we introduce a grid \( G \) over \( P \) with grid size \( \Delta \). Let \( \mathcal{C}_\text{opt} \) be an optimal set of centers. Later we explain how to get rid of this assumption and how to maintain such a coreset in the dynamic streaming model.

We can approximate the \textit{Max-Cut} problem with \( \epsilon \)-coresets. For the \textit{Max-Cut} problem.

First, we introduce a grid \( G \) over \( P \) with grid size \( \Delta \). Let \( \mathcal{C}_\text{opt} \) be an optimal set of centers. Later we explain how to get rid of this assumption and how to maintain such a coreset in the dynamic streaming model.

We can approximate the \textit{Max-Cut} problem with \( \epsilon \)-coresets. For the \textit{Max-Cut} problem.
whose parent cells are heavy. We conclude that
\[ n \leq \log \frac{d}{\epsilon} \] we get
\[
\sum_i \sum_{p \in L_{\text{near}}(i)} \sqrt{d} \cdot 2^{i+1} \leq \frac{\epsilon}{2} \text{Median}(P, C).
\]

Proof: We observe that the furthest point in a cell in \( L_{\text{near}}(i) \) 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
\[
|L_{\text{near}}(i)| \leq k \cdot (2 \cdot (1 + 2 \sqrt{d} + \frac{4\sqrt{d}}{\epsilon})) \cdot 2^i \leq k \cdot (14\sqrt{d}/\epsilon)^d.
\]
Each of the considered cells is light and so it contains at most \( \delta \cdot Opt/2^i \) points. Hence,
\[
\sum_i \sum_{p \in L_{\text{near}}(i)} \sqrt{d} \cdot 2^{i+1} \leq \sum_i k \cdot (14\sqrt{d}/\epsilon)^d \cdot \delta \cdot Opt/2^i \cdot \sqrt{d} \cdot 2^{i+1} \leq \sum_i \delta \cdot k \cdot 2 \sqrt{d} \cdot (14\sqrt{d}/\epsilon)^d \cdot \text{Median}(P, C)
\]
Now observe that for every \( i \geq \log(\delta \cdot Opt) \) every cell that contains a point is heavy and so there are no light cells. Further we observe that for those \( i < \log(\delta \cdot Opt/\epsilon) \) 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
\[
\sum_i \delta \cdot k \cdot 2 \sqrt{d} \cdot (14\sqrt{d}/\epsilon)^d \cdot \text{Median}(P, C) \leq \delta \cdot (\log n + 1) \cdot k \cdot 2 \sqrt{d} \cdot (14\sqrt{d}/\epsilon)^d \cdot \text{Median}(P, C) \leq \frac{\epsilon}{2} \text{Median}(P, C)
\]
for our choice of \( \delta \).

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

Proof: We first observe that \( \bigcup_i L(i) \) 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 \( P_{\text{core}} \) 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 \( G_i \) we define \( M(i) \) to be the set of marked heavy cells. We further need the notion of center cells.

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

We define \( M_{\text{center}}(i) = \{ C \in M(i) | C \text{ is center cell} \} \) to be the subset of marked heavy cells that are center cells. We use \( M_{\text{external}}(i) = M(i) \setminus M_{\text{center}}(i) \) 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 of \( \text{Opt} \).

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
\[
\left| \bigcup_i M_{\text{external}}(i) \right| \leq \frac{2}{\epsilon} \cdot \frac{\text{Opt}}{2}.
\]
We also know that
\[
\left| M_{\text{center}}(i) \right| \leq k \cdot 2^d.
\]
We observe that there cannot be any marked heavy cells in grids \( G_i \) 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 \( i < \log(\delta \cdot \text{Opt}/\epsilon) \) the threshold for heavy cells is larger than \( \epsilon \) and so there cannot be any (marked) heavy cells in the corresponding grids either. Therefore, we get
\[
\left| \bigcup_i M_{\text{center}}(i) \right| \leq (\log n + 1) \cdot k \cdot 2^d.
\]
The lemma follows since \( \bigcup_i M(i) = \bigcup_i (M_{\text{external}}(i) \cup M_{\text{center}}(i)) \) 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 \( \Omega(\epsilon^{-2} \log n \cdot \log \Delta) \) 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 \( \epsilon \)-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 \( \epsilon \)-far from heavy is considered as heavy. Then the size of the coreset is at most \( \frac{2}{\epsilon} + (\log n + 2) \cdot k \cdot 2^d + 1 = O(k \log n/\epsilon^d) \).

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 $G^{11}$ 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 $2^i$ of Opt with $i \in \{1, \ldots, \lfloor (d + 1) \log (\Delta + 1) \rfloor \}$. We then try to determine the cells that contain more than $\delta \cdot 2^i - 1$ points. If $2^i < \text{Opt}$ then we will determine all heavy cells but we will also categorize some light cells as heavy. Since our set of approximations contains a 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 $O(\log n / \epsilon d^j)$. To select our random sample we want to take every element in the data stream with probability $\frac{\delta}{\epsilon^2}$ into our sample, where

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

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

$$s = \frac{\alpha}{\delta} \cdot n \cdot 2^{-i - 1}.$$ 

Hence, if $2^i < \text{Opt}$ then any heavy cell contains at least $\alpha$ 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 $h : \Delta^d \rightarrow \{1, \ldots, \delta \cdot 2^i - 1 / \alpha \}$ from a class of $\alpha$-independent hash functions and consider the set $S = h^{-1}(1) \cap \mathbb{P}$. 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 $G^{11}$ 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 $\frac{\alpha}{\delta} + k \cdot 2^{i - 1}$ we cancel the instance. As we will see, in this case we have $2^i - \text{Opt}$ 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 $2^i > \text{Opt} / 2$. Then we have points from at most $\frac{\alpha}{\delta} + k \cdot 2^{i - 1}$ 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 $2^{i - 1} - 2$. Let $X_p$ denote the indicator random variable for the event that $h(p) = 1$. Let $D$ denote the set of non-center grid cells. We have $\text{E}[\sum_{p \in D} X_p] \leq \frac{\alpha}{\delta}$. We will assume $\text{E}[\sum_{p \in D} X_p] \leq \frac{\alpha}{\delta}$ as the deviation is maximized in this case. Applying Theorem 6 (see Appendix) we get

$$\Pr \left[ \left| \sum_{p \in D} X_p - \text{E} \left( \sum_{p \in D} X_p \right) \right| \geq \epsilon \cdot \text{E} \left( \sum_{p \in D} X_p \right) \right] \leq e^{-\min \{ \frac{\alpha}{2}, \lceil \epsilon^2 \frac{\alpha}{2} \rceil \}} \leq \rho.$$ 

Therefore, with probability at least $1 - \rho$ we have at most $\frac{\alpha}{\delta}$ points from non-center cells in our sample. If no two of these points are contained in the same grid cell we get an upper bound of $\frac{\alpha}{\delta}$ 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. □

**Lemma 3.12.** Let $\epsilon < 1 / 3$, let $C$ be an arbitrary grid cell in $G^{11}$ and let $j$ be fixed. Further let $n_C$ denote the number of points in $C$. Then the following events hold with probability at least $1 - \rho$:

- If $C$ contains at least $\delta \cdot 2^i - 1$ points, then $| \{1 - \epsilon \} \cdot n_C \leq |S \cap C| \cdot n/s \leq \lceil 1 - \epsilon \rceil \cdot n_C$.

- If $C$ contains less than $\delta \cdot 2^i - 1$ points, then $|S \cap C| \cdot n/s \leq \lceil 1 - \epsilon \rceil \cdot \delta \cdot 2^i - 1$.

**Proof:** The proof again follows from Theorem 6. Let $X_p$ denote the indicator random variable for the event that $h(p) = 1$. We want to show that $\sum_{p \in C} X_p$ does not deviate much from its expectation. If a cell contains at least $\delta \cdot 2^i - 1$ points then $\text{E}[\sum_{p \in C} X_p] \geq \alpha / 2$. From Theorem 6 it follows

$$\Pr \left[ \left| \sum_{p \in C} X_p - \text{E} \left( \sum_{p \in C} X_p \right) \right| \geq \epsilon \cdot \text{E} \left( \sum_{p \in C} X_p \right) \right] \leq e^{-\min \{ \frac{\alpha}{2}, \lceil \epsilon^2 \frac{\alpha}{2} \rceil \}}$$

and the first part of the lemma follows for the chosen value of $\alpha$. 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 6 to the case when $C$ contains $\delta \cdot 2^i - 1$ points. In this case the expected number of points in the cell is $\alpha / 2$ and the second part of the lemma follows. □

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 $C$ we have $|S \cap C| \cdot n/s \geq (1 - \epsilon) \cdot \delta \cdot 2^i - 1$ then we classify it as heavy. For every heavy cell $C$ we use the value $n_C = \frac{\delta \cdot 2^i \cdot n}{s}$ as an approximation for the number $n_C$ of points in $C$. By Lemma 3.12 we know that with probability at least $1 - \rho$ we have $n_C / (1 + \epsilon) \leq n_C / (1 - \epsilon)$. For every heavy cell we define $L_C = n_C / (1 + \epsilon)$ and $U_C = n_C / (1 - \epsilon)$. For every light cell we define $L_C = 0$ and $U_C = \delta^2 \cdot 2^i - 1$. 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 $L_C$ for the number of points in every useful cell $C$. 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. Let us consider an arbitrary point set $P'$ that is distributed according to our estimations.

**Lemma 3.13.** For every set $C$ of $k$ centers

$$(1 - O(\epsilon)) \text{Median}(P, C) \leq \text{Median}(P', C) \leq (1 + O(\epsilon)) \text{Median}(P, C).$$
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 $(1 \pm \varepsilon)$. It follows that $(1 - \varepsilon)\text{Median}(P_{core}, C) \leq \text{Median}(P_{core}', C) \leq (1 + \varepsilon)\text{Median}(P_{core}', C)$. By the definition of coresets, the lemma follows.

Hence, an $\varepsilon$-coreset for $P'$ is a $O(\varepsilon)$-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 $n$ in 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. When we allow deletions one problem occurs. We cannot stop an instance of our algorithm, if the number of counters exceeds $\frac{8\alpha}{3} + k \cdot 2^d$. 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 $h_2 : \mathcal{C}^{(1)} \rightarrow \{1, \ldots, 3 \cdot 2^{\frac{8\alpha}{3}} + k \cdot 2^d\}$, where $\mathcal{C}^{(1)}$ denotes the set of grid cells in $G^{(1)}$. 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 $2 \cdot \frac{8\alpha}{3} + k \cdot 2^d$ 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 $2 \cdot \frac{8\alpha}{3} + k \cdot 2^d$ 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 $G^{(1)}$. If the algorithm tells us that there are less than $2 \cdot \frac{8\alpha}{3} + k \cdot 2^d$ 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 $C \subseteq \mathcal{C}^{(1)}$ be a subset of cells of grid $G^{(1)}$. If $|C| \leq 2 \cdot \frac{8\alpha}{3} + k \cdot 2^d$ 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 $C_1, C_2 \in \mathcal{C}^{(1)}$, $C_1 \neq C_2$ that $\Pr[h_2(C_1) = h_2(C_2)] = \frac{1}{3 \cdot 2^{\frac{8\alpha}{3}} + k \cdot 2^d}$. By the union bound we get that $\Pr[\exists C_1, C_2 \in C \text{ with } h_2(C_1) = h_2(C_2) \text{ and } C_1 \neq C_2] \leq \sum_{C_1 \neq C_2} \Pr[h_2(C_1) = h_2(C_2)] \leq 1/3$.

To achieve an error probability of at most $\rho$ 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 \( \epsilon \)-coresets for the \( k \)-median of a dynamic geometric data stream. Our data structure needs \( O((\log \Delta + \log m)^3 \cdot k^2 \cdot \log^2 \Delta / \epsilon^{2d+4}) \) space and an insertion/deletion can be processed in \( O((\log \Delta + \log m)^2 \cdot k \cdot \log n / \epsilon^{d+2}) \) time. An \( \epsilon \)-coreset can be extracted in \( O((\log \Delta + \log m)^2 \cdot k^2 \cdot \log^2 n \cdot \exp(O((1 + \log(1/\epsilon)) \cdot \log^{-1}))) \) time. 

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

To process insertions or deletions we have to apply for the \( O(\log^2 \Delta) \) pairs of \( (i, j) \) each of the \( O(\log(p^{-1})) \) copies of \( h_2 \). This needs \( O(\log \Delta \log \Delta + \log m) \) entries. Since we want to have confidence probability at least 2/3 we can use \( \rho = \frac{1}{\epsilon^{2d+4}} \) for some constant \( c \). Since the space to store the hash functions is negligible we need \( O((\log \Delta + \log m)^3 \cdot k^2 \cdot \log^2 \Delta) \) space.

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

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 \( \frac{\delta \log \Delta}{\epsilon^d} \) points. Using this definition the proofs are essentially similar to the case of the k-means 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 \( \epsilon \)-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 \( O((\log \Delta + \log m)^2 \cdot k \cdot \log n / \epsilon^{d+4}) \) time. An \( \epsilon \)-coreset can be extracted in \( O((\log \Delta + \log m)^2 \cdot k \cdot \log n / \epsilon^{d+4}) \) time. Using the algorithm from [21] one can compute a \((1 + \epsilon)\)-approximation in \( O(k^2 \log^9 n + k^{k+2} \cdot \epsilon^{-(2d+1)} k \cdot \log^{k+1} n) \) time.

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 \( 1/\pi \) to obtain many similarities to the proofs for the \( k \)-median problem. In the following let \( \text{Opt} \) denote the maximum value of \( \frac{\pi}{2} \sum_{p \in C_1} \sum_{q \in C_2} d(p, q) \).

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

\[
d(p, c) \leq \frac{1}{\pi} \sum_{q \in P} d(p, q) .
\]

COROLLARY 5.2. Let \( c \) denote the center of gravity of all points. Then

\[
\text{Opt} \geq \frac{1}{4\pi} \sum_{p, q \in P} d(p, q) \geq \frac{1}{4} \sum_{p \in P} d(p, c)
\]

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 \( \frac{1}{4\pi} \sum_{p, q \in P} d(p, q) \). Since \( \text{Opt} \) denotes the maximum value of such a cut, the first inequality holds. From Lemma 5.1 the corollary follows.

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 \( \pm \epsilon \cdot \text{Opt} \) when we consider the corresponding partition of \( P \). In the following let \( c \) 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 \( \text{Opt} \) 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 \( \epsilon/2 \cdot (\text{Median}(P, [c]) + \text{Opt}) \). By Lemma 5.2 we have \( \text{Opt} \geq \frac{1}{2} \sum_{q \in P} d(q, c) = \frac{1}{2} \text{Median}(P, [c]) \).

Hence we get an additive error of \( 5/2 \cdot \epsilon \cdot \text{Opt} \).

It remains to show that the coreset is of small size. Here we use again the fact \( \text{Opt} \geq \frac{1}{2} \text{Median}(P, [c]) \). Every heavy non-center cell contributes to \( \text{Median}(P, [c]) \) with at least \( \delta \cdot \text{Opt}/2 \geq \frac{1}{2} \text{Median}(P, [c]) \). Therefore, there can be at most \( (\frac{\delta}{\epsilon} + \log n + 2) \cdot k \cdot 2^d = O(k \log n / \epsilon^d) \) 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 \( \epsilon \)-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 \( \Delta \) in time \( O((\log \Delta) \cdot \exp(O(1/\epsilon))) \). 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 \( \epsilon \)-coresets for the MaxCut of a dynamic geometric data stream. Our data structure needs
\[ O\left(\left(\log \Delta + \log m\right)^3 \cdot \log^4 \Delta / \varepsilon^{2d+4}\right) \text{ space and an insertion / deletion can be processed in } O\left(\left(\log^2 \Delta (\log \Delta + \log m)\right)\right) \text{ time. An } \varepsilon\text{-coreset can be extracted in } O\left(\left(\log \Delta + \log m\right)^2 \cdot \log n / \varepsilon^{d+2}\right) \text{ time.} \]

### 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 \( \Delta \), the solution changes by at most \( O\left(\left(\Delta + \log m\right)\right) \). (ii) \( Opt \geq \frac{1}{2} \sum_{p \in P} d(p, c) \).

#### 6.1 Maximum Weighted Matchings

We can complete each matching to a spanning tree, we get from the proof of Max Matching:

\[ \text{Moving a point by a distance of } \Delta \text{ changes the value of the optimal solution by at most } D. \]

Having a coreset of size \( O\left(\left(\log n / \varepsilon^d\right)\right) \) we can compute the Maximum Spanning Tree in time \( O\left(k^2 \cdot \log n / \varepsilon^d\right) \) using Prim’s algorithm [30].

#### 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:

\[ \text{Moving a point by a distance of } \Delta \text{ changes the value of the optimal solution by at most } D. \]

#### 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 \((1 + \varepsilon)\)-approximation for the original point set. The size of our coreset is \( O\left(k \cdot \log n / \varepsilon^d\right) \) where \( n \) is the number of points in the original point set. Recently, Har-Even 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 \( \varepsilon \)) coreset in dynamic data streams.

#### 8. REFERENCES


Section A: Chernoff Bounds with Limited Independence

We frequently use the following variant of Chernoff bounds for k-wise independent random variables.

**Theorem 6 (Theorem 5, [31]).** If \( X \) is the sum of \( k \)-wise independent random variables, each of which is confined to the interval \([0, 1]\) with \( \mu = E[X] \), then:

- For \( \delta \leq 1 \):
  
  - if \( k \leq \lceil \delta \mu e^{-1/3} \rceil \), then \( \Pr[|X - \mu| \geq \delta \mu] \leq e^{-\frac{1}{2}k} \).
  
  - if \( k = \lceil \delta^2 \mu e^{-1/3} \rceil \), then \( \Pr[|X - \mu| \geq \delta \mu] \leq e^{-\frac{1}{2}k}\mu^3/3} \).

- For \( \delta \geq 1 \):
  
  - if \( k \leq \lceil \delta \mu e^{-1/3} \rceil \), then \( \Pr[|X - \mu| \geq \delta \mu] \leq e^{-\frac{1}{2}k}\mu^2/3} \).
  
  - if \( k = \lceil \delta \mu e^{-1/3} \rceil \), then \( \Pr[|X - \mu| \geq \delta \mu] \leq e^{-\frac{1}{2}k}\mu^3/3} \).

- For \( \delta \geq 1 \) and \( k = \lceil \delta \mu \rceil \):

\[
\Pr[|X - \mu| \geq \delta \mu] \leq e^{-\frac{k}{2} + \frac{k \mu}{2}} < e^{-\frac{k \mu}{2}}.
\]