Sampling in Dynamic Data Streams and Applications

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

Piotr Indyk
Laboratory for Computer Science
MIT, USA
indyk@theory.lcs.mit.edu

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

ABSTRACT

A dynamic geometric data stream is a sequence of m ADD/REMOVE operations of points from a discrete geometric space $\{1,\ldots,\Delta\}^d$ [21]. ADD(p) inserts a point p from $\{1,\ldots,\Delta\}^d$ into the current point set, REMOVE(p) deletes p from P. We develop low-storage data structures to (i) maintain ϵ-approximations of range spaces of P with constant VC-dimension and (ii) maintain an ϵ-approximation of the weight of the Euclidean minimum spanning tree of P. Our data structures use $O(\log^3\Delta \cdot \log^3(1/\delta)\cdot \log(1/\epsilon)/\epsilon^2)$ and $O(\log(1/\delta)\cdot(\log\Delta/\epsilon)^{O(d)})$ bits of memory, respectively (we assume that the dimension d is a constant), and they are correct with probability $1-\delta$. These results are based on a new data structure that maintains a set of elements chosen (almost) uniformly at random from P.

Categories and Subject Descriptors

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

General Terms

Algorithms, Theory

Keywords

Streaming Algorithms, Computational Geometry, Data Structures

1. INTRODUCTION

The prevalence of high-rate information sources in todays society results in massive data sets occuring 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 is too costly to store the required information (e.g., source and destination) for every packet routed. A much more attractive solution is to design an algorithm which computes the desired statistic using only very limited space. Such streaming algorithm maintains only a *sketch* (or *synopsis*) of the data seen so far.

Applications of data streaming algorithms include sensor networks, astrophysical data, and (spatial) data bases. In the latter applications data streams are of geometric nature, and thus the stream items represent a set P of geometric objects, e.g. points in a d-dimensional plane. In many cases, the stream is *dynamic*, that is, it contains insertions as well as deletions of points. Algorithms over such data streams can be viewed as low-storage data structures maintaining certain statistics under insertions and deletions of points.

In this paper we provide randomized streaming algorithms for two well-studied geometric problems over dynamic geometric streams:

- Maintaining an ϵ-approximation of P; that is, a subset $S \subset P$, such that for any *range* R from a fixed family of ranges of bounded VC dimension (e.g., set of all rectangles), we have $\frac{|R\cap S|}{|S|} = \frac{|R\cap P|}{|P|} \pm \epsilon$. We show how to maintain an approximation of size $O(\log(1/\delta)\cdot\log(1/\epsilon)/\epsilon^2)$. Our algorithm uses $O(\log^3\Delta\cdot\log^3(1/\delta)\cdot\log(1/\epsilon)/\epsilon^2)$ space, where Δ is akin to the *spread* of the points in P (see definition below), and is correct with probability $1-1/\delta$. The approximations have applications to many other problems, including Tukey depth, simplicial depth, regression depth, the Thiel-Sen estimator, and the least median of squares [2].
- Maintaining an $(1+\epsilon)$-approximation of the cost of minimum weight tree spanning the points in P. This quantity in turn enables to achieve approximation for other problems, such as TSP or Steiner tree cost. Our algorithms use space $O(\log(1/\delta)\cdot(\log\Delta/\epsilon)^{O(d)})$, and is correct with probability $1-\delta$.

The above results are obtained by using a subroutine "Dynamic Sampling", which is formally defined in Section 2 and does the following. Consider a sequence of insertions and deletions of elements of a finite Universe $[U]=\{0,\ldots,U-1\}$. In the geometric setting $[U]$ corresponds to the set of all possible point

*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).

coordinates. Each element can be inserted as well as deleted multiple times, i.e., the stream represents a multiset P. The subroutine maintains a random element, chosen (almost) uniformly at random from P. It is immediate that such a procedure yields an ϵ-approximation of desired size.

To compute the weight of the Euclidean minimum spanning tree the above sampling procedure is used in a more subtle way. It is known that the EMST weight can be expressed as a formula depending on the number of connected components in certain subgraphs of the complete Euclidean graph of the current point set [6, 11]. We use an algorithm from [6] to count the number of connected components in these subgraphs. This algorithm is based on a BFS-like procedure starting at a randomly chosen point p. The BFS runs for a constant number of rounds only and one can show that it can never leave a ball around p of certain radius. Therefore, it suffices to maintain a random sample point and all points in a certain radius around this sample point. This task can be also approximately performed by a variant of our sampling routine.

It might initially appear that any algorithm performing the above task of maintaining a random element might require $\Omega(U)$ space. This is because, given a state of the algorithm, one could retrieve all the points in the multiset P by repeatedly choosing random element of P and deleting it. To avoid this problem, we assume that the stream of update operations is *oblivious* to the random bits used by our algorithm. That is, we assume that the data stream is constructed *before* the algorithm chooses its random bits. We show that, in this case, one can maintain with probability $1-\delta$ a random element using only $O(\log^2(UM/\delta))$ space (where M is the maximum multiplicity of an element from $[U]$).

Using a similar argument, one can observe that any *deterministic* algorithm for maintaining ϵ-approximation in the dynamic setting must use $\Omega(\Delta^d)$ space. This is because we can retrieve all elements in P, by repeatedly removing points belonging to the approximation. This implies that the deterministic approaches used for the insertions-only case (see Previous work) cannot be applied here.

1.1 Previous work

The idea of random sampling under insertions and deletions appeared earlier in [14]. However, their algorithm was only able to perform a (much more restrictive) task of *implicit sampling*. In particular, it was not able to return an actual element of the set P, but only some of its properties. The data structure we use to obtain random samples is similar to a structure from [16], which has been developed for the purpose of estimating the cardinality of set expressions in data streams. With some amount of modifications it can be used to obtain random samples. However, this was not the goal of [16]. Additionally, in our application of the EMST weight we need a random sample *and* some information about the distribution of points in some radius around the sample points. We also remark that independent of our work Cormode et al. developed a similar method for random sampling and used it to answer certain queries about the inverse distribution of the data in a stream [9]. Their construction only requires pairwise independent hash functions.

One of the first geometric problems studied in the streaming model was to approximate the diameter of a point set in 2D [13] using $O(1/\epsilon)$ space. Later this problem has been also considered in higher dimensions [22], where an algorithm with space complexity $O(dn^{1/(c^2-1)})$ to maintain a c-approximate diameter for $c > \sqrt{2}$ has been obtained. Chan and Sadjad proposed an algorithm to maintain an approximation of the diameter in the sliding window model [5]. Cormode and Muthukrishnan introduced the *radial histogram* [8], 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 [19]. Chan used coresets to approximate different geometric problems including diameter and min-volume bounding box [4]. Har-Peled and Mazumdar gave $(1+\epsilon)$-approximation algorithms for the k-median and k-means problem [18].

Bagchi et al. gave a deterministic streaming algorithm that maintains ϵ-nets and ϵ-approximations [2]. They apply their algorithm to approximate several robust statistics in data streams including Tukey depth, simplicial depth, regression depth, the Thiel-Sen estimator and the least median of squares. Since their algorithm is deterministic it cannot be extended to the dynamic streaming model. Other previous work includes the framework of *iceberg queries* [12]. In [23] a $(1+\epsilon)$-approximation for the frequency counts of items that occur more than ϵm times in a data stream of length m is given. Suri et al. gave both deterministic and randomized algorithms to compute a (weighted) ϵ-approximation for ranges that are axis-aligned boxes [26].

The model of dynamic geometric data streams has been introduced in [21]. In [21] $O(d \cdot \log \Delta)$-approximation algorithms for (the weight of) minimum weighted matching, minimum bichromatic matching and minimum spanning tree. Further results were given for the facility location problem and the k-median problem. In [14] $(1+\epsilon)$-approximation algorithms were given for a number of problems including k-median, k-means, MaxCut, maximum spanning tree, and MaxTSP.

For other work on streaming algorithms we refer to the survey by Muthukrishnan [24].

The problem of estimating the weight of a minimum spanning tree has been considered in the context of sublinear time approximation algorithms. The first such algorithm for the minimum spanning tree weight is designed for sparse graphs and computes a $(1+\epsilon)$-approximation [6]. It has a running time of $\widetilde{O}(D \cdot W/\epsilon^2)$ when the edge weights are in a range from 1 to W and the average degree of the input graph is D. In the geometric context a $\widetilde{O}(\sqrt{n}/\epsilon^{O(1)})$ time $(1+\epsilon)$-approximation algorithm was given, if the point set can be accessed using certain complex data structures [10]. In the metric case one can compute a $(1+\epsilon)$-approximation of the minimum spanning tree weight in $O(n/\epsilon^{O(1)})$ time [11].

1.2 Discrete Geometric Space

To achieve our results, we assume that our input points come from the discrete d-dimensional space $\{1,\ldots,\Delta\}^d$. Alternatively, we could have assumed that all interpoint distances are between 1 and Δ. Although such an assumptions are not very common in computational geometry, they are typically satisfied in practice when bounded precision arithmetic is used. In streaming algorithms the assumption of bounded precision is common because otherwise the notion of storage is not well defined (e.g., one cannot use discrete communication complexity tools to prove lower bounds).

2. DYNAMIC SAMPLING

We consider the following data structure problem: build a low-storage data structure that enables certain operations on a vector $x : [U] \to [M]$. Initially $x = 0$. We assume that, at any moment, $0 \le x_i \le M - 1$ for $i \in [U]$. Let Supp(x) be the set of indices i such that $x_i > 0$. We use notation $\|x\|_0 = |\text{Supp}(x)|$. The data structure is parametrized by two numbers $\delta', \delta'' > 0$. The operations are as follows:

- UPDATE(i, a): performs $x_i = x_i + a$, where $i \in [U]$, $a \in \{-M \dots M\}$. We assume that all update operations lead to an x_i within the range $\{0, \dots M-1\}$. Our structure does not verify this assumption.
- SAMPLE: returns either a pair (r, v), where $r \in [U]$, $v = x_r$ or a flag FAIL. The procedure satisfies the following constraints:
  - If a pair (r, v) is returned, then r is chosen at random from Supp(x) such that for any $i \in$ Supp(x), **Pr**$[r = i] = \frac{1}{\|x\|_0} \pm \delta'$
  - The probability of returning FAIL is at most δ''.

Our goal is to design such a data structure with δ'' strictly separated from 1. The probability of reporting FAIL can be then reduced to be arbitrarily small, by running several data structures in parallel.

We implement our data structure as follows. We will use the following data structures.

Unique Element (UE). The data structure supports two operations on a vector $x : [U] \to [M]$:

- UPDATE(i, a): as above
- REPORT: if $\|x\|_0 = 1$, then it returns the pair (i, x_i) such that $x_i > 0$; otherwise, it can return any pair (j, v).

The data structure keeps counters c and C which are initialized to 0.

- UPDATE(i, a): $c = c + a$, $C = C + a \cdot i$
- REPORT: Construct $v = c$, $i = C/v$. Return (i, v).

The correctness of the data structure follows immediately. It uses $O(\log(UM))$ bits of space.

Distinct Elements (DE). The data structure supports two operations on a vector $x : [U] \to [M]$: UPDATE (as above) and REPORT, which with probability $1 - \delta$ returns a value k such that $k \le \|x\|_0 \le k(1 + \epsilon)$; the numbers $\delta, \epsilon > 0$ are parameters. One can use a data structures from [15] to solve this problem using $O(\log^2(MU/\delta)/\epsilon^2)$ bits of space. We use them here.

Our data structure. Our data structure uses hash functions $h_j, j \in [\log U]$. Each h_j is of the form $h_j : [U] \to [2^j]$. Initially, we assume that each h_j is a fully random hash function, we relax this assumption later.

In addition, we use:

- Unique Element data structures $UE_j, j \in [\log U]$
- Distinct Elements data structures $DE_j, j \in [\log U]$, and DE, with parameters $\epsilon = 1/2$ and $\delta = \delta'/2$

The operations are implemented as follows:

UPDATE(i, a)
  for $j \in [\log U]$ **do**
    if $h_j(i) = 0$ **then** $UE_j.update(i, a); DE_j.update(i, a)$
  $DE.update(i, a)$

SAMPLE
  $j = \lceil \log(DE.report) \rceil$
  if $DE_j.report = 1$ **then return** $UE_j.report$ **else return** FAIL

Correctness. Assume that DE is correct, and the data structure DE_j is correct as well for $j = \lceil \log(DE.report) \rceil$. Note that this happens with probability at least $1 - \delta'$. In that case, we have $DE_j.report = 1$ iff $|\text{Supp}(x) \cap h_j^{-1}(0)| = 1$. Thus, the element reported by UE_j is an element chosen uniformly at random from Supp(x).

It remains to show a lower bound on the probability of $|\text{Supp}(x) \cap h_j^{-1}(0)| = 1$. Denote $S_j = h_j^{-1}(0)$ and $d = |\text{Supp}(x)|$. By correctness of DE it follows that $d \le 2^j \le 4d$. Thus, the probability that $|S_j \cap \text{Supp}(x)| = 1$ is equal to $d2^{-j}(1 - 2^{-j})^{d-1} \ge 1/4 \cdot (1/4)^4 = 1/4^5$.

Randomness. To remove the assumption that the functions h_j are fully random, we show that (as in [20]) they can be pseudo-generated using a generator of Nisan [25]. The argument is as follows. Fix the randomness used by DE and DE_j. One can verify that, at any point, the state of our data structure (including the states of DE_j, UE_j and DE) depends only on projections $x_{|S_j}, j \in [\log U]$. That is, it is invariant w.r.t. the order of updates. Thus, it suffices to consider the case when the updates (i, a) are sorted by the value of i. In this case, the values of $h_j(i)$, for $i = 1, 2, 3, \dots$ can be generated from a read-once sequence $\mathcal{R}$ of random bits of length $O(U \log M \log U)$. Since the algorithm uses space $O(\log^2(UM/\delta))$, by the properties of Nisan's PRG we can replace $\mathcal{R}$ by $PRG(\mathcal{R}')$, where $\mathcal{R}'$ is a stream of random bits of length $O((\log U + \log\log M) \cdot \log^2(UM/\delta))$.
The PRG guarantees that replacing $\mathcal{R}$ by $PRG(\mathcal{R}')$ changes the distribution by less than δ.

LEMMA 2.1. *Given a sequence of update operations on a vector* $x : [U] \to [M]$, *there is a streaming algorithm that with probability* $1 - 1/\delta$ *returns an element* $r \in$ *Supp*(x) *such that* **Pr**$[r = i] = 1/\|x\|_0 \pm \delta$ *for every* $i \in$ *Supp*(x). *The algorithm uses* $O(\log^2(MU/\delta))$ *space.* □

3. ESTIMATING THE WEIGHT OF A EUCLIDEAN MINIMUM SPANNING TREE

In this section we will show how to estimate the weight of a Euclidean Minimum Spanning Tree in a dynamic geometric data stream. In the dynamic streaming model it is assumed that algorithms have access to a sequence of m update operations either being an ADD or REMOVE operation of a point from the d-dimensional discrete geometric space $\{1, ..., \Delta\}^d$. These operations arrive in an arbitrary order and they can only be accessed one after the other, i.e. there is no random access to the input. We assume that the input sequence is valid, i.e. a point can only be removed, if it has been added to the point set before. We denote by $P = \{p_1, \dots, p_n\}$ the current point set. Further EMST denotes the weight of the Euclidean minimum spanning tree of the current set.

We impose $\log_{1+\epsilon}(\sqrt{d}\Delta)$ square grids over the point space. The side lengths of the grid cells are $\frac{\epsilon\cdot(1+\epsilon)^i}{\sqrt{d}}$ for $0 \le i \le \log_{1+\epsilon}(\sqrt{d}\Delta)$. Our algorithm maintains certain statistics of the distribution of points in the grids. We show that these statistics can be used to compute a $(1+\epsilon)$-approximation of the weight EMST of the Euclidean minimum spanning tree. Our computation is based on a formula from [11] for the value of the minimum spanning tree of an n point metric space. Let G_P denote the complete Euclidean graph of a point set P and W an upper bound on its longest edge. Further let $c_P^{((1+\epsilon)^i)}$ denote the number of connected components in $G_P^{((1+\epsilon)^i)}$, which is the subgraph of G_P containing all edges of length at most $(1+\epsilon)^i$. Under these assumptions we can write

$$\frac{1}{1+\epsilon}\cdot \text{EMST} \le n - W + \epsilon \cdot \sum_{i=0}^{\log_{1+\epsilon} W-1} (1+\epsilon)^i c_P^{((1+\epsilon)^i)} \le \text{EMST} \tag{1}$$

where n is the number of points in P. Instead of considering the number of connected components in $G_P^{(t)}$ for $t = (1+\epsilon)^i$ we first move all points of P to the centers of a grid of side length $\frac{\epsilon\cdot t}{\sqrt{d}}$. After removing multiplicities we obtain the point set $P^{(t)}$. Then we consider the graph $G^{(t)}$ whose vertex set is $P^{(t)}$ and that contains an edge between two points if their distance is at most t. Instead of counting the connected components in $G_P^{(t)}$ we count the connected components in $G^{(t)}$. This only introduces a small error. We denote by $n^{(t)} = |P^{(t)}|$ the number of non-empty grid cells of side length $\frac{\epsilon t}{\sqrt{d}}$. We denote by $c^{(t)}$ the number of connected components of $G^{(t)}$.

Our algorithm maintains approximations $\tilde{n}, \widetilde{W}, \tilde{n}^{(t)}$, and $\tilde{c}^{(t)}$ (for $t = (1+\epsilon)^0, (1+\epsilon)^1, (1+\epsilon)^2, \ldots$) of the number n of points currently in the set, the maximal pairwise distance W, the size $n^{(t)}$ of $P^{(t)}$, and the number $c^{(t)}$ of connected components in $G^{(t)}$, respectively. The approximation is derieved by inserting the maintained approximations into formula 1. In the following we discuss the data structures we need to maintain our approximations.

3.1 Number of points

We observe that we can remember the value of n exactly by increasing/decreasing $\tilde{n}$ in case of an ADD/REMOVE operation.

3.2 Extend

We show how to maintain an approximation $\widetilde{W}$ of W with $W \le \widetilde{W} \le 4\sqrt{d}W$, where W is the largest distance between two points in the current point set. To do so, we maintain an approximation $\widetilde{W}_j$ of the extend of the point set in each of the d dimensions with $W_j \le \widetilde{W} \le 4W_j$, where W_j is the extend in dimension j for $1 \le j \le d$. The maximum of the $\widetilde{W}_j$ is our approximation $\widetilde{W}$.

We will now show, how maintain the extend of the point set in dimension j. For each $i \in \{0, ..., \log\Delta\}$ we introduce two one-dimensional grids $G_{i,1}$ and $G_{i,2}$, each of them having a side length of 2^i. $G_{i,2}$ is displaced by $2^{(i-1)}$ against $G_{i,1}$. Let $g_{i,1}$ and $g_{i,2}$ be the number of occupied cells in the grid $G_{i,1}$, $G_{i,2}$, respectively. We use our Distinct Elements data structure from Section 2 to count the number of grid cells containing a point. We only want to distinguish between the case $g_{i,1} = 1$ and $g_{i,1} > 1$ (we assume that there is always at least one point in the set; otherwise the problem becomes trivial).

If there is exactly one point in the current set we have $g_{0,1} = 1$ and $g_{0,2} = 1$ and the extend is 0. Otherwise, the extend must be at least 1. Therefore in the finest grids $G_{0,1}$ and $G_{0,2}$ at least two cells are occupied, which means that $g_{0,1} > 1$ and $g_{0,2} > 1$. We now find the smallest value i such that $g_{i+1,1} = 1$ or $g_{i+1,2} = 1$. In this case we know that $W_j \le 2^{i+1}$.

Since $g_{i,1} > 1$ and $g_{i,2} > 1$, we know that in both grids $G_{i,1}$ and $G_{i,2}$ at least two cells are occupied. This means that the convex hull of the point set contains the border of a cell in both grids $G_{i,1}$ and $G_{i,2}$. Since these cell borders have a distance of at least 2^{i-1} we have $W_j \ge 2^{i-1}$. Therefore, we can output $\widetilde{W}_j = 2^{i-1}$ as a 4-approximation of the extend in dimension j.

3.3 Size of $P^{(t)}$

The problem to estimate $n^{(t)}$ is equivalent to maintaining the number of distinct elements in a data stream. This can be seen as follows. Once a point arrives we can determine its grid cell from its position. Thus we can interpret the input stream as a stream of grid cells and we are interested in the number of distinct grid cells. This can be approximated using an instance of the Distinct Elements (DE) data structure of section 2.

3.4 The Sample Set

To approximate the number of connected components we have to maintain multisets $S^{(t)}$ of points chosen uniformly at random (with repetitions) from $P^{(t)}$. For each point $p \in S^{(t)}$ we also maintain all other points from $P^{(t)}$ whose distance to p are at most some constant R. Having such a sample and the value $\tilde{n}^{(t)}$ we can use an algorithm from [6] to obtain the number of connected components with sufficiently small error. This is proven in Section 3.5 using a modified analysis that charges the error in the approximation to the weight of the minimum spanning tree. This way we get our estimation $\tilde{c}^{(t)}$ of $c^{(t)}$.

To get our random sample we use the following variant of our data structure from Section 2. When our algorithm sees a point in the data stream it can determine the cell the point is contained in. Thus we can see the stream as a stream of increase / decrease operations of the number of points in specified cells and we can use the data structure from Section 2. It remains to show that we are able to get information about the R-neighbourhood (all cells within a radius of R) of a sample cell. Let Z be the number of cells within a radius of R. Notice that Z is of polylogarithmic size. We alter the sampling algorithm in the following way: Instead of scalar numbers M representing the number of points in a cell c we store Δ^d-ary numbers with Z digits. The first digit of such a number represents the number of points in a cell c, the second the number of points in the right neighbour cell. The ith component encodes the number of points in the cell $c+\nu_i$, where $\nu_1, ..., \nu_Z$ are fixed vectors. This way we can encode the information about the whole neighbourhood of a cell in one Δ^d-ary number with Z digits. When we encounter an insert operation of a point, we first identify the cell c it is contained in. For each neighbour cell q within the neighbourhood of c fulfilling $c = q + \nu_i$ we perform UPDATE(q, Δ^{di}). For a delete operation we perform UPDATE$(q, -\Delta^{di})$. Furthermore we alter the sampling algorithm to return a sample chosen uniformly out of all cells c fulfilling $(x_c)_0 \ne 0$ (i.e. containing at least one point).

Then it follows from Lemma 2.1 (here we charge the overall deviation from the uniform distribution to the error probability).

COROLLARY 3.1. *There is a streaming algorithm that with probability* $1-\delta$ *computes a sample set* $S^{(t)}$ *of* s *points from* $P^{(t)}$ *chosen uniformly at random and of all points within a radius of* R *around these points. The algorithm uses* $O(s \cdot Z \cdot \log^2 \Delta \cdot (\log \Delta + \log^2(1/\delta)))$ *bits of memory, where* Z *denotes the number of grid cells in radius* R. $\square$

3.5 Computing $\tilde{c}^{(t)}$

In this section we show how to compute our estimator $\tilde{c}^{(t)}$. To do this we will use our sample set $S^{(t)}$. In the computation of the sample set $S^{(t)}$ we need to specify the value R. We choose $R := \log(\sqrt{d} \cdot \Delta) 2^{d+4} \sqrt{d} \epsilon^{-2} \cdot t$.

Further we need in the following the value $D = R/t$. Our algorithm for estimating $c^{(t)}$ works as follows. First, we check, if $\widetilde{W} < 4\epsilon t$. If that is the case, $W < 4\epsilon t$ follows. Therefore if we take an arbitrary sample point we know that every point of the current point set is contained in the radius R. Therefore, we know the whole graph $G^{(t)}$ and can compute $c^{(t)}$ exactly.

Thus let us assume $\widetilde{W} \geq 4\epsilon t$. In this case our algorithm is essentially similar to the one presented in [6], but our analysis is somewhat different. The difference comes from the special structure of our input graphs $G^{(t)}$. We exploit the lower bound from Lemma 3.2 below to relate the error induced by our approximation algorithm to the weight of the EMST.

LEMMA 3.2. *If* $\widetilde{W} \geq 4\epsilon t$ *then*

$$\mathit{EMST} \geq \frac{n^{(t)} \epsilon t}{\sqrt{d} 2^{d+1}}$$

Proof : We distinguish between the case $n^{(t)} \geq 2^{d+1}$ and $n^{(t)} < 2^{d+1}$. We start with the case $n^{(t)} \geq 2^{d+1}$. In this case we can color the grid cells using 2^d colors in such a way that no two adjacent cells have the same color. Since we have $n^{(t)}$ occupied cells there must be one color c which is assigned to at least $\lceil \frac{n^{(t)}}{2^d} \rceil$ occupied cells. Notice that $\lceil \frac{n^{(t)}}{2^d} \rceil \geq 2$. These occupied cells are pairwise not adjacent, therefore any pair of points that is contained in two distinct of these cells has a distance of at least $\frac{\epsilon \cdot t}{\sqrt{d}}$. We can conclude EMST $\geq \left(\frac{n^{(t)}}{2^d} - 1 \right) \frac{\epsilon \cdot t}{\sqrt{d}} \geq \frac{n^{(t)} \epsilon t}{\sqrt{d} 2^{d+1}}$.

In the second case we get $W \geq \frac{\epsilon t}{\sqrt{d}} \geq \frac{n^{(t)} \epsilon t}{\sqrt{d} 2^{d+1}}$ since $\widetilde{W} \geq 4\epsilon t$. This implies the result. $\square$

We now present a description of our method to estimate $c^{(t)}$. The idea is pick a random set of vertices (with repetition) and start a BFS with a stochastic stopping rule at each vertex v from the sample to determine the size of the connected component of v. If the BFS explores the whole connected component we set the corresponding indicator variable to 1 and else to be 0. To implement this algorithm we can use our sample set $S^{(t)}$. The sample set provides a multiset of points from $P^{(t)}$ chosen uniformly at random. It also provides every other point within a distance of at most $R = Dt$. Since we consider only edges of length at most t and since the algorithm below stops exploiting a component when it has size D or larger, the BFS cannot reach a point with distance more than R from the starting vertex. Therefore, our sample set $S^{(t)}$ is sufficient for our purposes. The algorithm we use is given below.

```
APPROXCONNECTEDCOMPONENTS(P, t, ε)
    Choose s points q_1, ..., q_s ∈ P^(t)
    uniformly at random
    for each q_i do
        Choose integer X according to distribution
        Prob[X ≥ k] = 1/k
        if X ≥ D then β_i = 0.
        else
            Check, whether the connected component
            of G^(t) containing q_i has
                (a) at most X vertices
                (b) more than X vertices
            In case (a), set β_i = 1
            In case (b), set β_i = 0
    Output: ĉ^(t) = n^(t)/s · Σ_{i=1}^{s} β_i
```

Thus, β_i is an indicator random variable for the event that the connected component containing q_i has at most X vertices. We obtain

$$\begin{aligned} \mathrm{E}[\beta_i] &= \sum_{\substack{\text{conn. comp.} \\ C \text{ in } G^{(t)}}} \mathrm{Prob}[q_i \in C] \cdot \mathrm{Prob}[X \geq |C| \wedge X < D] \\ &\leq \sum_{\substack{\text{conn. comp.} \\ C \text{ in } G^{(t)}}} \mathrm{Prob}[q_i \in C] \cdot \mathrm{Prob}[X \geq |C|] \\ &= \sum_{\substack{\text{conn. comp.} \\ C \text{ in } G^{(t)}}} \frac{|C|}{n^{(t)}} \cdot \frac{1}{|C|} = \frac{c^{(t)}}{n^{(t)}} \ . \end{aligned}$$

Therefore, we define the output value of the algorithm as

$$\hat{c}^{(t)} = \frac{\tilde{n}^{(t)}}{s} \cdot \sum_{i=1}^{s} \beta_i \ .$$

It follows immediately

$$\mathrm{E}[\hat{c}^{(t)}] \leq \frac{\tilde{n}^{(t)}}{n^{(t)}} c^{(t)} \leq (1+\epsilon) c^{(t)} \ . \tag{2}$$

From

$$\begin{aligned} \mathrm{E}[\beta_i] &= \sum_{\substack{\text{conn. comp.} \\ C \text{ in } G^{(t)}}} \mathrm{Prob}[q_i \in C] \cdot \mathrm{Prob}[X \geq |C| \wedge X < D] \\ &\geq \sum_{\substack{\text{conn. comp.} \\ C \text{ in } G^{(t)}}} \frac{|C|}{n^{(t)}} \cdot \left(\frac{1}{|C|} - \frac{1}{D} \right) = \frac{c^{(t)}}{n^{(t)}} - \frac{1}{D} \end{aligned}$$

and Lemma 3.2 we obtain

$$\begin{aligned} \mathrm{E}[\hat{c}^{(t)}] &\geq \frac{\tilde{n}^{(t)}}{n^{(t)}} \left(c^{(t)} - \frac{n^{(t)}}{D} \right) \\ &\geq (1-\epsilon) \left(c^{(t)} - \frac{\mathrm{EMST} \cdot 2^{d+1} \sqrt{d}}{\epsilon t D} \right) \\ &\geq (1-\epsilon) \left(c^{(t)} - \frac{\epsilon \cdot \mathrm{EMST}}{8t \log(\sqrt{d} \cdot \Delta)} \right) \ . \end{aligned} \tag{3}$$

Our next step is to find a bound for the variance of $\hat{c}^{(t)}$. Since the β_i are $\{0,1\}$ random variables, we get:

$$\mathbf{Var}[\beta_i] \le E[\beta_i^2] = E[\beta_i] \le \frac{c^{(t)}}{n^{(t)}}$$

By mutual independence of the β_i we obtain for the variance of $\hat{c}^{(t)}$ for fixed $\widetilde{n}^{(t)}$:

$$\mathbf{Var}[\hat{c}^{(t)}] = \mathbf{Var}[\frac{\widetilde{n}^{(t)}}{s}\sum_{i=1}^{s}\beta_i] = \frac{\left(\widetilde{n}^{(t)}\right)^2}{s^2}\cdot s \cdot \mathbf{Var}[\beta_i]$$

$$\le \frac{(\widetilde{n}^{(t)})^2}{s}\cdot\frac{c^{(t)}}{n^{(t)}} \le (1+\epsilon)^2\frac{n^{(t)}c^{(t)}}{s}$$

From (2) and (3) we know that

$$|c^{(t)} - E[\hat{c}^{(t)}]| \le \epsilon c^{(t)} + \frac{\epsilon\cdot \text{EMST}}{8\cdot t\cdot\log(\sqrt{d}\cdot\Delta)} \tag{4}$$

is satisfied.

We now choose s, the number of sample points, as

$$s = (1+\epsilon)^2\frac{2^{2d+10}\cdot d\cdot\log^2(\sqrt{d}\cdot\Delta)\cdot\log_{1+\epsilon}(\sqrt{d}\cdot\Delta)}{\epsilon^4}$$

$$= O\left(\frac{\log^3\Delta}{\epsilon^5}\right)$$

Chebyshev's inequality and Lemma 3.2 implies:

$$\begin{aligned}
&\mathbf{Pr}\Big[\big|\hat{c}^{(t)} - E[\hat{c}^{(t)}]\big| \ge \frac{\epsilon\cdot\text{EMST}}{8\cdot t\cdot\log(\sqrt{d}\cdot\Delta)}\Big]\\
&\le (1+\epsilon)^2\cdot\frac{n^{(t)}\cdot c^{(t)}}{s}\cdot\frac{64\cdot t^2\cdot\log^2(\sqrt{d}\cdot\Delta)}{\epsilon^2\cdot\text{EMST}^2}\\
&\le (1+\epsilon)^2\cdot\frac{64\cdot d\cdot 2^{2d+2}\cdot\log^2(\sqrt{d}\cdot\Delta)}{s\cdot\epsilon^4}\\
&\le \frac{1}{4\log_{1+\epsilon}(\sqrt{d}\cdot\Delta)}
\end{aligned}$$

Therefore we get together with (4):

LEMMA 3.3. *With probability* $1 - \frac{1}{4\log_{1+\epsilon}(\sqrt{d}\cdot\Delta)}$ *we have* $|\hat{c}^{(t)} - c^{(t)}| \le \epsilon c^{(t)} + \frac{\epsilon\cdot EMST}{4\cdot t\cdot\log(\sqrt{d}\cdot\Delta)}$.

It now follows that with probability at least $3/4$ all $\widetilde{c}^{(t)}$ values satisfy the inequality in Lemma 3.3. It remains to sum up the overall error taking into account that we considered connected components of the graph $G^{(t)}$ and not of the corresponding subgraph of G_P. Intuitively, the connected component of $G^{(t)}$ are sufficient because in each of the $G^{(t)}$ we moved every point by at most ϵt which is small compared to the threshold edge length of t. We can prove (see Appendix) for our output value $\widetilde{M}$

LEMMA 3.4.

$$\left|EMST - \widetilde{M}\right| \le 47\sqrt{d}\cdot\epsilon\cdot EMST\ .$$

□

From this lemma our final result follows immediately using standard amplification techniques to ensure that the estimation is correct at every point of time.

THEOREM 1. *Given a sequence of insertion/deletions of points from the discrete* d*-dimensional space* $\{1,\dots,\Delta\}^d$ *there is a streaming algorithm that uses* $O(\log(1/\delta)\cdot(\log(\Delta)/\epsilon)^{O(d)})$ *space and* $O(\log(1/\delta)\cdot(\log(\Delta)/\epsilon)^{O(d)})$ *time (for constant* d*) for each update and computes with probability* $1-\delta$ *a* $(1+\epsilon)$*-approximation of the Euclidean minimum spanning tree.* □

4. CONCLUSIONS

In this paper we showed how to maintain a random sample of a point set in a dynamic data stream. We applied our method to obtain ϵ-approximations of range spaces with constant VC dimension and to maintain a $(1+\epsilon)$-approximation of the weight of the minimum spanning tree. As random sampling and ϵ-approximations are powerful tools in Computational Geometry we believe that our techniques have many other applications.

5. 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)*, 1996.

[2] 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, pp. 144–151, 2004.*

[3] Z. Bar-Yossef, T. S. Jayram, R. Kumar, D. Sivakumar, L. Trevisan. Counting Distinct Elements in a Data Stream. Proc. RANDOM*, 2002.*

[4] T. M. Chan. Faster Core-Set Constructions and Data Stream Algorithms in Fixed Dimensions. Proc. 20th Annual Symposium on Computational Geometry, pp. 152-159, 2004.

[5] T. M. Chan., B. S. Sadjad Geometric Optimization Problems over Sliding Windows *Proc. 15th International Symposium on Algorithms and Computation, pp. 246–258, 2004.*

[6] B. Chazelle, R. Rubinfeld, and L. Trevisan. Approximating the minimum spanning tree weight in sublinear time. Proc. 28th Annual International Colloquium on Automata, Languages and Programming (ICALP)*, pages 190–200, 2001.*

[7] G. Cormode, M. Datar, P. Indyk. Comparing Data Streams using Hamming norms. Proc. International Conference on Very Large Databases (VLDB)*, 2002.*

[8] G. Cormode and S. Muthukrishnan. Radial Histograms for Spatial Streams. DIMACS Technical Report 2003-11, 2003.

[9] G. Cormode, S. Muthukrishnan, and I. Rozenbaum. Summarizing and Mining Inverse Distributions on Data Streams via Dynamic Inverse Sampling. DIMACS Technical Report 2005-11 , 2005.

[10] A. Czumaj, F. Ergün, L. Fortnow, A. Magen, I. Newman, R. Rubinfeld, and C. Sohler. Sublinear-time approximation of Euclidean minimum spanning tree. Proc. 14th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 813–822, 2003.*

[11] A. Czumaj and C. Sohler. Estimating the Weight of Metric Minimum Spanning Trees in Sublinear-Time. Proc. 36th Annual ACM Symposium on Theory of Computing (STOC)*, pp. 175–183, 2004.*

[12] *M. Fang, N. Shivakumar, H. Garcia-Molina, R. Motwani, and J. D. Ullman. Computing iceberg queries efficiently.* Proc. 1998 Intl. Conf. on Very Large Data Bases*, pp. 299-310, 1998.*

[13] *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.*

[14] *G. Frahling, C. Sohler. Coresets in dynamic geometric data streams. Manuscript, 2004.*

[15] *P. Flajolet and G. Martin. Probabilistic counting algorithms for data base applications. Journal of Computer and System Sciences, 31:182–209, 1985.*

[16] *S. Ganguly,M. Garofalakis, R. Rastogi. Processing Set Expressions over Continuous Update Streams.* Proc. SIGMOD Conference 2003*, pp. 265-276, 2003.*

[17] *T. Hagerub and C. Rüb. A Guided Tour of Chernoff Bounds. Information Processing Letters, 33:305–308, 1989/90.*

[18] *S. Har-Peled, S. Mazumdar. On Coresets for* k*-Means and* k*-Median Clustering.* Proc. 36th Annual ACM Symposium on Theory of Computing (STOC)*, pp. 291–300, 2004.*

[19] *J. Hershberger and S. Suri. Convex Hulls and Related Problems in Data Streams. Proceedings of the ACM/DIMACS Workshop on Management and Processing of Data Streams, 2003.*

[20] *P. Indyk. Stable Distributions, Pseudorandom Generators, Embeddings and Data Stream Computation.* Proc. 41st IEEE Symposium on Foundations of Computer Science (FOCS)*, 2000.*

[21] *P. Indyk. Algorithms for Dynamic Geometric Problems over Data Streams.* Proc. 36th Annual ACM Symposium on Theory of Computing (STOC)*, pp. 373–380, 2004.*

[22] *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.*

[23] *G. S. Manku, R. Motwani. Approximate Frequency Counts over Data Streams.* Proc. 2002 Intl. Conf. on Very Large Data Bases*, pp. 346–357, 2002.*

[24] *S. Muthukrishnan. Data streams: Algorithms and applications (invited talk at SODA'03). Available at http://athos.rutgers.edu/ muthu/stream-1-1.ps, 2003.*

[25] *N. Nisan. Pseudorandom generators for Space-Bounded Computation.* Proc. 22nd Annual ACM Symposium on Theory of Computing (STOC)*, 204–212, 1990.*

[26] *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

A. PROOF OF LEMMA 3.4

First we show that our output value

$$\widetilde{M} := n - \widetilde{W} + \epsilon \sum_{i=0}^{\log_{1+\epsilon} \widetilde{W}-1} (1+\epsilon)^i \widetilde{c}^{((1+\epsilon)^i)}$$

is close to

$$M_p := n - \widetilde{W} + \epsilon \sum_{i=0}^{\log_{1+\epsilon} \widetilde{W}-1} (1+\epsilon)^i c_p^{((1+\epsilon)^i)}$$

which is a $(1+\epsilon)$-approximation of the EMST value by equation (1).

CLAIM A.1.

$$c_p^{(1+\epsilon)^{i+1}} \le c^{(1+\epsilon)^i} \le c_p^{(1+\epsilon)^{i-2}}$$

Proof : Let us consider two arbitrary points p, q and the centers of their corresponding cells p', q' in the grid graph $G^{((1+\epsilon)^i)}$. Recall that the corresponding grid has side length $\frac{\epsilon\cdot(1+\epsilon)^i}{\sqrt{d}}$. Thus by moving p and q to the centers of the corresponding grid cells their distance changes by at most $\epsilon \cdot (1+\epsilon)^i$.

Now assume that p, q are in the same connected component in $G_p^{((1+\epsilon)^{i-2})}$. Then they are connected by a path of edges of length at most $(1+\epsilon)^{i-2}$. If we now consider the path of the corresponding centers of grid cells, then any edge of the path has length at most $(1+\epsilon)^{i-2} + \epsilon \cdot (1+\epsilon)^i \le (1+\epsilon)^i$. Therefore, p', q' are in the same connected component of the grid cell graph. In a similar way, one can obtain the other inequality. □

Proof : [of Lemma 3.4] From Lemma 3.3 and Claim A.1 it follows that

$$(1-\epsilon)c_p^{((1+\epsilon)^{i+1})} - \frac{\epsilon\text{EMST}}{4(1+\epsilon)^i \log(\sqrt{d}\Delta)} \le \widetilde{c}^{((1+\epsilon)^i)}$$

$$\le (1+\epsilon)c_p^{((1+\epsilon)^{i-2})} + \frac{\epsilon\text{EMST}}{4(1+\epsilon)^i \log(\sqrt{d}\Delta)}$$

holds with probability at least $1 - \frac{1}{4\log_{1+\epsilon}(\sqrt{d}\cdot\Delta)}$. By a union bound and some calculation we get with probability $3/4$:

$$\begin{aligned}
&\epsilon \sum_{i=0}^{\log_{1+\epsilon} \widetilde{W}-1} (1+\epsilon)^i \widetilde{c}^{((1+\epsilon)^i)} \\
\ge\ & \epsilon(1-\epsilon) \sum_{i=0}^{\log_{1+\epsilon} \widetilde{W}-1} (1+\epsilon)^i c_p^{((1+\epsilon)^{i+1})} - \frac{1}{2}\epsilon\text{EMST} \\
\ge\ & \epsilon(1-\epsilon) \sum_{i=1}^{\log_{1+\epsilon} \widetilde{W}} (1+\epsilon)^{i-1} c_p^{((1+\epsilon)^i)} - \epsilon\text{EMST} \\
\ge\ & \epsilon\frac{1-\epsilon}{1+\epsilon} \sum_{i=0}^{\log_{1+\epsilon} \widetilde{W}-1} (1+\epsilon)^i c_p^{((1+\epsilon)^i)} \\
& -\frac{\epsilon(1-\epsilon)}{1+\epsilon} n - \epsilon \cdot \text{EMST} \\
\ge\ & \frac{1-\epsilon}{1+\epsilon} \cdot \epsilon \sum_{i=0}^{\log_{1+\epsilon} \widetilde{W}-1} (1+\epsilon)^i c_p^{((1+\epsilon)^i)} \\
& -\epsilon n - \epsilon \cdot \text{EMST}
\end{aligned}$$

and

$$\epsilon \sum_{i=0}^{\log_{1+\epsilon}\widetilde{W}-1} (1+\epsilon)^i \widetilde{c}^{((1+\epsilon)^i)}$$
$$\leq \frac{1}{2}\epsilon \text{EMST} + \epsilon \sum_{i=-2}^{\log_{1+\epsilon}\widetilde{W}-3} (1+\epsilon)^{i+3} c_p^{((1+\epsilon)^i)}$$
$$\leq \epsilon \text{EMST} + \sum_{i=0}^{\log_{1+\epsilon}\widetilde{W}-1} (1+\epsilon)^{i+3} \widetilde{c}_p^{((1+\epsilon)^i)}$$
$$+2\epsilon(1+\epsilon)^2 n$$
$$\leq (1+\epsilon)^3 \epsilon \sum_{i=0}^{\log_{1+\epsilon}\widetilde{W}-1} (1+\epsilon)^i c_p^{((1+\epsilon)^i)}$$
$$+8\epsilon n + \epsilon \cdot \text{EMST}$$

Using these inequalities together with $\widetilde{W} \leq 4\sqrt{d} \cdot \text{EMST}$ and $n \leq \text{EMST}$ and $\epsilon \leq 1/2$ we are now able to prove that $|M_p - \tilde{M}|$ is small.

$$\widetilde{M} = n - \widetilde{W} + \epsilon \sum_{i=0}^{\log_{1+\epsilon}\widetilde{W}-1} (1+\epsilon)^i \widetilde{c}^{((1+\epsilon)^i)}$$
$$\geq n - \widetilde{W} + \frac{1-\epsilon}{1+\epsilon}\epsilon \sum_{i=0}^{\log_{1+\epsilon}\widetilde{W}-1} (1+\epsilon)^i c_p^{((1+\epsilon)^i)}$$
$$-\epsilon n - \epsilon \text{EMST}$$
$$= n - \widetilde{W} + \frac{1-\epsilon}{1+\epsilon}\left(M_p - n + \widetilde{W}\right) - \epsilon n - \epsilon \text{EMST}$$
$$= \frac{\epsilon - \epsilon^2}{1+\epsilon} n - \frac{2\epsilon}{1+\epsilon}\widetilde{W} + \frac{1-\epsilon}{1+\epsilon} M_p - \epsilon \text{EMST}$$
$$\geq (1-2\epsilon) M_p - 2\epsilon \widetilde{W} - \epsilon \text{EMST}$$
$$\geq (1-2\epsilon) M_p - 9\sqrt{d}\epsilon \text{EMST}$$

and

$$\widetilde{M} = n - \widetilde{W} + \epsilon \sum_{i=0}^{\log_{1+\epsilon}\widetilde{W}-1} (1+\epsilon)^i \widetilde{c}^{((1+\epsilon)^i)}$$
$$\leq n - \widetilde{W} + (1+\epsilon)^3 \epsilon \sum_{i=0}^{\log_{1+\epsilon}\widetilde{W}-1} (1+\epsilon)^i c_p^{((1+\epsilon)^i)}$$
$$+8\epsilon n + \epsilon \text{EMST}$$
$$= n - \widetilde{W} + (1+\epsilon)^3 \left(M_p - n + \widetilde{W}\right) + 8\epsilon n + \epsilon \text{EMST}$$
$$\leq 7\epsilon \widetilde{W} + M_p + 7\epsilon M_p + 8\epsilon n + \epsilon \text{EMST}$$
$$\leq M_p + 28\epsilon\sqrt{d}\text{EMST} + 7\epsilon M_p + 8\epsilon \text{EMST} + \epsilon \text{EMST}$$
$$= M_p + (28\epsilon\sqrt{d} + 9\epsilon)\text{EMST} + 7\epsilon M_p$$
$$\leq M_p + 37\sqrt{d}\epsilon \text{EMST} + 7\epsilon M_p$$

alltogether:

$$M_p - 2\epsilon M_p - 9\sqrt{d}\epsilon \cdot \text{EMST} \leq \tilde{M}$$
$$\leq M_p + 37\sqrt{d}\epsilon \cdot \text{EMST} + 7\epsilon \cdot M_p$$

By inequality (1) we have:

$$\frac{1}{1+\epsilon}\text{EMST} \leq M_p \leq \text{EMST}$$

and eventually obtain

$$\widetilde{M} \leq \text{EMST} + 44\sqrt{d}\epsilon \text{EMST}$$

and

$$\widetilde{M} \geq M_p - 2\epsilon M_p - 9\sqrt{d}\epsilon \text{EMST}$$
$$\geq \frac{\text{EMST}}{1+\epsilon} - 2\epsilon \text{EMST} - 9\sqrt{d}\epsilon \text{EMST}$$
$$\geq (1-\epsilon)\text{EMST} - 2\epsilon \text{EMST} - 9\sqrt{d}\epsilon \text{EMST}$$
$$\geq \text{EMST} - 11\sqrt{d}\epsilon \text{EMST}$$

□