

International Journal of Computational Geometry & Applications
 © World Scientific Publishing Company

Sampling in Dynamic Data Streams and Applications

Gereon Frahling ^{*†}

Piotr Indyk [‡]

Christian Sohler ^{§¶}

Received 11/09/05

Revised 07/15/06

Communicated by Alon Efrat

A dynamic geometric data stream is a sequence of m ADD/REMOVE operations of points from a discrete geometric space $\{1, \dots, \Delta\}^d$. $\text{ADD}(p)$ inserts a point p from $\{1, \dots, \Delta\}^d$ into the current point set P , $\text{REMOVE}(p)$ deletes p from P . We develop low-storage data structures to (i) maintain ϵ -nets and ϵ -approximations of range spaces of P with small VC-dimension and (ii) maintain a $(1 + \epsilon)$ -approximation of the weight of the Euclidean minimum spanning tree of P . Our data structure for ϵ -nets uses $O\left(\frac{\log(1/\delta)}{\epsilon} + \frac{D}{\epsilon} \log \frac{D}{\epsilon}\right) \cdot d^2 \cdot \log^2(\Delta/\delta)$ bits of memory and returns with probability $1 - \delta$ a set of $\tilde{O}\left(\frac{D + \log(1/\delta)}{\epsilon}\right)$ points that is an ϵ -net for an arbitrary fixed finite range space with VC-dimension D . Our data structure for ϵ -approximations uses $\tilde{O}\left(\frac{D + \log(1/\delta)}{\epsilon^2} \cdot d^2 \cdot (\log^2 \Delta \cdot \log(1/\delta) + \log^2(1/\delta))\right)$ bits of memory and returns with probability $1 - \delta$ a set of $\tilde{O}\left(\frac{D + \log(1/\delta)}{\epsilon^2}\right)$ points that is an ϵ -approximation for an arbitrary fixed finite range space with VC-dimension D . The data structure for the approximation of the weight of a Euclidean minimum spanning tree uses $O(\log(1/\delta)(\log \Delta/\epsilon)^{O(d)})$ space and is correct with probability at least $1 - \delta$.

Our results are based on a new data structure that maintains a set of elements chosen (almost) uniformly at random from P .

Keywords: Randomized algorithms, data streaming, random sampling, minimum spanning trees, ϵ -nets, ϵ -approximations

1. Introduction

The prevalence of high-rate information sources in today's society results in 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

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

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

‡Laboratory for Computer Science and Artificial Intelligence, Massachusetts Institute of Technology, indyk@theory.lcs.mit.edu

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

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

^aWe use the \tilde{O} -notation to suppress logarithms in the largest occurrence of a (function of a) variable, i.e. for any function $f(x)$ we have $f(x) \cdot \log(f(x)) = \tilde{O}(f(x))$.

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 a 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 space. 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 three well-studied geometric problems over dynamic geometric streams:

- (1) Maintaining an ϵ -net of P ; that is, a subset $N \subset P$ such that for any *range* R from a fixed family of ranges of VC dimension \mathcal{D} (e.g., set of all rectangles), we have $|N \cap R| > 0$, if $\frac{|R \cap P|}{|P|} \geq \epsilon$. We show how to maintain a set of $\tilde{O}\left(\frac{\mathcal{D} + \log(1/\delta)}{\epsilon}\right)$ points that with probability $1 - \delta$ is an ϵ -net of P .

Our data structure uses $O\left(\left(\frac{\log(1/\delta)}{\epsilon} + \frac{\mathcal{D}}{\epsilon} \log \frac{\mathcal{D}}{\epsilon}\right) \cdot d^2 \cdot \log^2(\Delta/\delta)\right)$ space.

- (2) Maintaining an ϵ -approximation of P ; that is, a subset $A \subset P$, such that for any *range* R from a fixed family of ranges of bounded VC dimension \mathcal{D} , we have $\frac{|A \cap R|}{|A|} = \frac{|R \cap P|}{|P|} \pm \epsilon$. In this case our algorithm maintains a set of $\tilde{O}\left(\frac{\mathcal{D} + \log(1/\delta)}{\epsilon}\right)$ points that with probability $1 - \delta$ is an ϵ -approximation.

Our data structure uses $\tilde{O}\left(\frac{\mathcal{D} + \log(1/\delta)}{\epsilon^2} \cdot d^2 \cdot (\log^2 \Delta \cdot \log(1/\delta) + \log^2(1/\delta))\right)$ space.

The ϵ -approximations have applications to many other problems, including Tukey depth, simplicial depth, regression depth, the Thiel-Sen estimator, and the least median of squares ⁴.

- (3) Maintaining a $(1 + \epsilon)$ -approximation of the cost of minimum weight tree spanning the points in P . This quantity in turn enables to achieve constant factor approximations 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 $1 - \delta$.

We note that our first two results require that, at the time when the ϵ -net or ϵ -approximation is requested, the maintained set P does not contain duplicate points, i.e., is *not* a multi-set. See the discussion in the next paragraph for more details.

The above results are obtained by using a subroutine "Dynamic Sampling", which is formally defined in Section 3 and does the following. Consider a sequence of insertions and deletions of elements of a finite universe $[U] = \{0, \dots, U - 1\}$. In the geometric setting $[U]$ corresponds to the set of all possible point coordinates. In this case, the stream represents a multi-set P of elements from $[U]$. The subroutine maintains a random element, chosen (almost) uniformly at random from the set of elements occurring in P . Note that each element has (approximately) *the same* probability of being selected, independently of its multiplicity in P .

By maintaining several instances of the sampling data structure, we obtain a random sample representing P . Such a procedure yields an ϵ -approximation of desired size as long as P is a *set* (without multiplicities)^b, at the time step when the sample was generated.

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^{9,14}. We use an algorithm from⁹ 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 also be 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 a 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 $(1/4)^6 - \delta$ a random element chosen almost uniformly at random using only $O(\log^2(U\Delta/\delta))$ space (where Δ 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.

We mention that the idea of random sampling under insertions and deletions appeared earlier in¹⁷. 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.

1.1. Previous work

One of the first geometric problems studied in the streaming model was to approximate the diameter of a point set in the plane¹⁶ using $O(1/\epsilon)$ space. Later this problem has also been considered in higher dimensions²⁶, 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 model⁸. In this model one considers an infinite input streams but only the last

^bTo obtain an ϵ -approximation of a multi-set, we would need a procedure which samples elements from P with probability proportional to their multiplicity in P . However, our procedure does not support this function.

n elements are relevant (the window). They gave a $(1 + \epsilon)$ -approximation algorithm for fixed dimension, which stores $O((\frac{1}{\epsilon})^{(d+1)/2} \log \frac{R}{\epsilon})$ points and where R denotes the ratio between the diameter and the smallest pairwise distance between two points in the window. Cormode and Muthukrishnan introduced the *radial histogram* ¹⁰ to approximate different geometric problems in the plane. A radial histogram is a subdivision of the plane given by concentric circles around a center point and halflines starting at the center. This way we can assign every point in the stream to a cell of the radial histogram. Problems that can be approximated via radial histograms include the diameter, convex hull, and the furthest neighbor problem. 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 ²³. Agarwal et al. introduced a framework for approximating various extent measures of point sets for constant dimension ². Their technique can be used to obtain streaming algorithm for problems like the diameter, width, smallest bounding box, ball and cylinder of the point set. Chan used this framework to develop improved streaming algorithms (using less space than previous ones) for a number of geometric problems with constant dimension including diameter, width, minimum-radius enclosing cylinder, minimum width enclosing annulus, etc.

Har-Peled and Mazumdar gave $(1 + \epsilon)$ -approximation algorithms for the k -median and k -means problem ²¹. Their algorithm is based on maintaining a small *weighted* set of points that approximates the original point set with respect to the k -median or k -means problem.

Bagchi et al. gave a deterministic streaming algorithm that maintains ϵ -nets and ϵ -approximations ⁴. 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. Suri et al. gave both deterministic and randomized algorithms to compute a (weighted) ϵ -approximation for ranges that are axis-aligned boxes ³¹.

The model of dynamic geometric data streams has been introduced in ²⁵. In ²⁵ $O(d \cdot \log \Delta)$ -approximation algorithms for (the weight of) minimum weighted matching, minimum bichromatic matching and minimum spanning tree are given. Further results were given for the facility location problem and the k -median problem. In ¹⁷ $(1 + \epsilon)$ -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 ²⁹.

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 ⁹. It has a running time of $\tilde{O}(D \cdot W/\epsilon^2)$ 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 $\tilde{O}(\sqrt{n}/\epsilon^{O(1)})$ time $(1 + \epsilon)$ -approximation algorithm was given, if the point set can be accessed using certain complex data structures ¹³. In the metric case one can compute a $(1 + \epsilon)$ -approximation of the minimum spanning tree weight in $O(n/\epsilon^{O(1)})$ time ¹⁴.

2. Preliminaries

We will use $[U]$ to denote the set $\{0, \dots, U - 1\}$. In the geometric setting we will assume that points are from the discrete d -dimensional space $[\Delta]^d$. Although such 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).

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 $[\Delta]^d$. 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.

3. Dynamic Sampling

In this section we first consider a non-geometric streaming problem. We are interested in maintaining a random sample from the set of non-zero entries of a vector $x = (x_0, \dots, x_{U-1})$ from $[M]^U$. The distribution of this sample should be (almost) uniform. We consider this problem in a dynamic model, which is sometimes called *the turnstyle model*²⁹. In this model, we are given a sequence of update operations on x , which is initially 0. Each UPDATE(i, a) adds a to the value x_i , where a is a number in the range $\{-M, \dots, M\}$ and will always lead to an x_i value within $\{0, \dots, M\}$. Our structure does not verify this assumption. Thus, at any moment, $0 \leq x_i \leq M$ for $i \in [U]$. Let $\text{Supp}(x)$ 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 $\epsilon, \delta > 0$. The operations are as follows:

- UPDATE(i, a): performs $x_i \leftarrow x_i + a$, where $i \in [U]$, $a \in \{-M \dots M\}$.
- SAMPLE: returns either a pair (i, x_i) for some $i \in [U]$ or a flag FAIL. The procedure satisfies the following constraints:
 - If a pair (i, x_i) is returned, then i is chosen at random from $\text{Supp}(x)$ such that for any $j \in \text{Supp}(x)$,

$$\Pr[i = j] = \frac{1}{\|x\|_0} \pm \delta.$$
 - The probability of returning FAIL is at most some constant.

Keeping $O(s)$ instances of this data structure it is possible to choose a sample set S of size s (almost) uniformly at random from the non-zero entries of v . In the setting of dynamic geometric data streams we will set $U = \Delta^d$ and identify $[U]$ with $[\Delta]^d$ using a bijection $\tau : [\Delta]^d \rightarrow [U]$. An ADD(p) operation translates to $\tau(p) \leftarrow \tau(p) + 1$ and a REMOVE(p) operation to $\tau(p) \leftarrow \tau(p) - 1$. This way it is possible to select a random sample from a dynamic geometric data stream. But our structure is not limited to this functionality. In Section 3.5.1 we show how to get a uniformly distributed sample and all input points that are close to the sample points.

6 Gereon Frahling, Piotr Indyk, Christian Sohler

Our sample data structure uses two elementary data structures. The first such structure checks whether there is exactly one non-zero entry in x . If this is the case the index of the entry and its value it returned. The second data structure approximates the number of non-zero entries in x .

3.1. The Unique Element (UE) Data Structure

The first elementary data structure called UE checks whether there is exactly one non-zero entry in x . The data structure is deterministic. It supports the following operations on a vector $x = (x_0, \dots, x_{U-1})$ with entries from $[M]$.

- UPDATE(i, a): as above
- REPORT: if $\|x\|_0 \neq 1$, then it returns FAIL. If $\|x\|_0 = 1$, then it returns the unique pair (i, x_i) such that $x_i > 0$.

The data structure keeps three counters c_0, c_1 , and c_2 which are initialized to 0. Our UPDATE operation will ensure that $c_j = \sum_{i \in [U]} x_i \cdot i^j$ at any point of time. The operations UPDATE and REPORT are implemented as follows:

| UPDATE(i, a) | REPORT |
|---------------------------|---|
| $c_0 = c_0 + a$ | if $c_0 \cdot c_2 - c_1^2 \neq 0$ or $c_0 = 0$ then return FAIL |
| $c_1 = c_1 + a \cdot i$ | else $i \leftarrow c_1/c_0$ |
| $c_2 = c_2 + a \cdot i^2$ | $x_i \leftarrow c_0$ |
| | return (i, x_i) |

Claim 3.1. *Data structure UE uses $O(\log(UM))$ bits of space. It returns FAIL if and only if $\|x\|_0 \neq 1$. Otherwise, it correctly returns the unique pair (i, x_i) with $x_i \neq 0$.*

Proof : The maximum value of the counters c_0, c_1, c_2 is $O(U^3 \cdot M)$ and so the counters need $O(\log(UM))$ bits. It remains to prove that the data structure correctly recognizes the case $\|x\|_0 = 1$ and returns the correct value. From $x_i \geq 0$ for all $i \in [U]$ it follows that

$c_0 = 0$, if and only if $\|x\|_0 = 0$. It follows

$$\begin{aligned}
c_0 \cdot c_2 - c_1^2 &= \left(\sum_{i \in [U]} x_i \right) \cdot \sum_{i \in [U]} x_i \cdot i^2 - \left(\sum_{i \in [U]} x_i \cdot i \right)^2 \\
&= \left(\sum_{i \in [U]} x_i \right) \cdot \sum_{i \in [U]} x_i \cdot i^2 - \sum_{i, j \in [U]} x_i \cdot i \cdot x_j \cdot j \\
&= \sum_{i, j \in [U]} x_i x_j \cdot j^2 - \sum_{i, j \in [U]} x_i \cdot i \cdot x_j \cdot j \\
&= \frac{1}{2} \sum_{i, j \in [U]} x_i x_j \cdot j^2 + \frac{1}{2} \sum_{i, j \in [U]} x_i x_j \cdot i^2 - \sum_{i, j \in [U]} x_i x_j \cdot i \cdot j \\
&= \frac{1}{2} \sum_{i, j \in [U]} x_i x_j (j^2 - 2ij + i^2) \\
&= \frac{1}{2} \sum_{i, j \in [U]} x_i x_j (j - i)^2 .
\end{aligned}$$

Hence, $c_0 \cdot c_2 - c_1 > 0$ iff $\|x\|_0 > 1$. The correctness of the data structure follows immediately. \square

3.2. The Distinct Elements (DE) Data Structure

The data structure supports two operations on a vector $x = (x_0, \dots, x_{U-1})$ with entries from $[M]$: UPDATE (as above) and REPORT, which with probability $1 - \delta$ returns a value $k \in [0, U]$ such that $\|x\|_0 \leq k \leq (1 + \psi) \cdot \|x\|_0$; δ and ψ are parameters. The value δ gives an upper bound on the error probability and ψ is related to the approximation guarantee of the data structure. One can use a data structure from ¹⁹ to solve this problem using $O(1/\psi^2 \cdot \log(1/\psi) \log(1/\delta) \log(U) \log(UM))$ bits of space.

3.3. The Sample Data Structure

The basic idea behind our data structure is to use a hash function that maps the universe to a smaller space $[2^j]$. The value 2^j corresponds to a guess of the number of non-zero entries currently present in vector x . Assuming a fully random hash function every non-zero entry is mapped to 0 with the same probability. Further, the probability that exactly one non-zero entry is mapped to 0 can be checked using our unique elements data structure. If this is the case, our sample data structure returns the corresponding entry (it returns the index and the value of the entry). We now give the procedure in detail.

Our data structure uses hash functions $h_j, j \in [\log U]$. Each h_j is of the form $h_j : [U] \rightarrow [2^j]$. Initially, we assume that each h_j is a fully random hash function, we relax this assumption later. The value 2^j corresponds to the guess that, currently, there are roughly 2^j non-zero entries in x .

In addition, we use:

- Unique Element data structures $UE_j, j \in [\log U]$, and
- A Distinct Elements data structure DE , with parameters $\psi = 1/2$ and $\delta = 1/(8e)$.

We write $UE_j.UPDATE$ to denote an `UPDATE` operation for data structure UE_j . 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.UPDATE(i, a)
  SAMPLE
  j = [log(DE.REPORT)]
  return UE_j.REPORT

```

Correctness. Assume that DE is correct. Note that this happens with probability at least $1 - 1/(8e)$. We have $UE_j.REPORT \neq FAIL$, iff $|\text{Supp}(x) \cap h_j^{-1}(0)| = 1$. Since we assume fully random hash functions, the element reported by UE_j is an element chosen uniformly at random from $\text{Supp}(x)$.

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 $\ell = |\text{Supp}(x)|$. By correctness of DE it follows that $\ell \leq 2^j \leq 4\ell$. Thus,

$$\Pr[|S_j \cap \text{Supp}(x)| = 1] = \ell \cdot 2^{-j} \cdot (1 - 2^{-j})^{\ell-1} \geq 1/4 \cdot 1/e = 1/(4e) .$$

Since the error probability in our distinct elements structure is at most $1/(8e)$ we obtain that our algorithm returns a random element with probability $1/(4e) - 1/(8e) = 1/(8e)$.

3.4. Randomness

We introduce two different methods to overcome the assumption of totally random hash-functions and achieve space complexity polylogarithmic in M and U . First we will present a general approach based on a lemma from ²⁴. The method introduces a small additive error in the probability to sample a fixed element. This error can be translated into a small multiplicative error of ϵ . However, the method is difficult to implement and for many applications a multiplicative error is sufficient. In Section 3.4 we will present a method based on pairwise independent hashfunctions. It is easy to implement but introduces a multiplicative error in the probability to sample a fixed element. One can use our second method to obtain similar results for sampling in data streams, ϵ -nets, ϵ -approximations and approximations of the minimum spanning tree weight, etc. . However, due to the multiplicative error these results are slightly worse than those obtained using the first method.

We use a lemma from ²⁴ to replace the assumption of fully random h_j .

Lemma 1. ²⁴ Consider an algorithm A which, given a stream S' of pairs (i, α) , and a function $f : [n'] \times \{0, 1\}^{R'} \times \{-M' \dots M'\} \rightarrow \{-M'^{O(1)} \dots M'^{O(1)}\}$, does the following:

- Set $O = 0$; Initialize length- R' chunks $R_0 \dots R_{[n']}$ of independent random bits.
- For each new pair (i, α) : perform $O = O + f(i, R_i, \alpha)$.
- Output $A(S') = O$.

Assume that the function $f(\cdot, \cdot, \cdot)$ is supported with an evaluation algorithm using $O(C + R')$ space and $O(T)$ time. Then there is an algorithm A' producing output $A'(S)$, that uses only $O(C + R' + \log(M'n'))$ bits of storage and $O([C + R' + \log(M'n')] \log(n'R'))$ random bits, such that

$$\Pr[A(S) \neq A'(S)] \leq 1/n'$$

over some joint probability space of randomness of A and A' . The algorithm A' uses $O(T + \log(n'R'))$ arithmetic operations per each pair (i, α) . \square

For each fixed j our algorithm uses the hash function h_j to select a subset $S_j := \{i \in [U] : h_j(i) = 0\}$ of indices. Each index is in the set S_j with probability $1/2^j$. The UE data structure maintains the values $c_0 = \sum_{i \in S_j} x_i$, $c_1 = \sum_{i \in S_j} x_i \cdot i$, and $c_2 = \sum_{i \in S_j} x_i \cdot i^2$.

Instead of using a hash function h_j to select the set S_j we can use chunks R_0, \dots, R_{U-1} of $\log U$ random bits each. We select S_j as

$$S_j := \{i \in [U] : \text{first } j \text{ bits of } R_i \text{ are all } 0\}.$$

Again each index $i \in [U]$ is selected to be in the set S_j with probability $1/2^j$. An update (i, α) on the UE data structure then simply adds the functions f_0 to c_0 , f_1 to c_1 , and f_2 to c_2 , where

- $f_1(i, R_i, \alpha) := \begin{cases} 0 & \text{iff first } j \text{ bits of } R_i \neq 0 \\ \alpha & \text{iff first } j \text{ bits of } R_i = 0 \end{cases}$
- $f_2(i, R_i, \alpha) := \begin{cases} 0 & \text{iff first } j \text{ bits of } R_i \neq 0 \\ \alpha \cdot i & \text{iff first } j \text{ bits of } R_i = 0 \end{cases}$
- $f_3(i, R_i, \alpha) := \begin{cases} 0 & \text{iff first } j \text{ bits of } R_i \neq 0 \\ \alpha \cdot i^2 & \text{iff first } j \text{ bits of } R_i = 0 \end{cases}$

Using Lemma 1 with values $n' = 1/\delta + U$, $M' = U \cdot M$, $R' = \log(U)$, and $C = \log(UM)$ we can replace the random bits R_i by $O([C + R' + \log(M'n')] \log(n'R')) = O(\log(UM/\delta) \log(U/\delta))$ random bits. We use the same random bits for all j and get an algorithm having the desired properties and using just $O(\log^2(UM/\delta))$ bits of storage. The distribution changes by less than δ .

Lemma 2. Let $\delta > 0$ be an error probability parameter. Given a sequence of update operations on a vector $x = (x_0, \dots, x_{U-1})$ with entries from $[M]$, there is a data structure that with probability at least $(1/(8e)) - \delta$ returns a pair (i, x_i) with $i \in \text{Supp}(x)$ such that $\Pr[i = j] = 1/\|x\|_0 \pm \delta$ for every $j \in \text{Supp}(x)$ and returns a flag FAIL otherwise. The algorithm uses $O(\log^2(UM/\delta))$ space. \square

A sample data structure using pairwise independent hash functions. In this section we focus on the development of a sample data structure that only uses pairwise independent hash functions⁶ instead of the approach using a pseudo-random generator. For technical reasons we will consider a relative error instead of an additive error.

The basic idea behind the second sample data structure is simple. Assume we knew the number n of non-zero entries in our vector x . Then we could use a hash function h that maps $[U]$ to a space somewhat larger than n , say to $\lceil n/\epsilon \rceil$. It is easy to see that such a hash function has ϵn collisions in expectation. This means that typically most of the non-zero entries in x do not collide. We now choose a value $\alpha \in [n/\epsilon]$ uniformly at random and check using the UE data structure whether exactly one non-zero entry was mapped to α . If this is the case we return the corresponding unique pair (i, x_i) with $h(i) = \alpha$ and $x_i \neq 0$. Since there are only few collisions this probability is close to $n/(n/\epsilon) = \epsilon$. If we keep $O(1/\epsilon)$ instances of the data structure, one of them is likely to return a pair. We will show that the index of the returned pair is almost uniformly distributed among the non-zero entries in x . It remains to deal with the fact that we do not know the value n and that it changes over time. We will keep $\log U$ hash functions h_j , $1 \leq j \leq \lceil \log U \rceil$ each corresponding to a guess $n \approx 2^j$. It will suffice to work with a good guess, which can be identified using our DE data structure.

We now give a detailed description of our sample data structure. Let $\delta > 0$ be a parameter for the error probability of our data structure. Our data structure uses $O(\log U \cdot \log(1/\delta)/\epsilon)$ pairwise independent hash functions $h_{j,k}$ with $j \in [\lceil \log U \rceil]$, $k \in [I]$, and number of instances $I := \lceil \log_{1-\epsilon/16}(\delta/2) \rceil = O(\log(1/\delta)/\epsilon)$. Each $h_{j,k}$ is of the form $h_{j,k} : [U] \rightarrow [T]$ with $T := \lceil 2^{j+1}/\epsilon \rceil$ and corresponds to the k -th hash function for the j -th guess, $n \approx 2^j$. For each hash function we choose a value $\alpha_{j,k}$, $j \in [\lceil \log U \rceil]$, $k \in [I]$, uniformly at random from $[T]$. Additionally, we need UE data structures $UE_{j,k}$ for $j \in [\lceil \log U \rceil]$ and $k \in [I]$. Each of these data structure is supposed to handle a subset of the $UPDATE(i, a)$ operations in the input stream. Namely, data structure $UE_{j,k}$ will process all updates with $h_{j,k}(i) = \alpha_{j,k}$. We also need one instance of the DE data structure with $\psi = 1$ and error probability at most $\delta/2$.

All hash functions, $\alpha_{j,k}$ and $UE_{j,k}$ are initialized during a separate initialization step. We write $UE_{j,k}.UPDATE$ to denote an $UPDATE$ operation for data structure $UE_{j,k}$.

The operations $UPDATE$ and $SAMPLE$ are implemented as follows:

```

UPDATE(i, a)
  for j ∈ [log U] do
    for k ∈ [I] do
      if hj,k(i) = αj,k then UEj,k.UPDATE(i, a)
  DE.UPDATE(i, a)

```

```

SAMPLE
  j = ⌈log(DE.REPORT)⌉
  if  $\forall_k \cup E_{j,k}.REPORT = FAIL$  then return FAIL
   $k_0 \leftarrow \min\{k \mid \cup E_{j,k}.REPORT \neq FAIL\}$ 
  return  $\cup E_{j,k_0}.REPORT$ 

```

Lemma 3. Let $\delta > 0$ be an error probability parameter. Given a sequence of update operations on a vector $x = (x_0, \dots, x_{\mathcal{U}-1})$ with entries from $[M]$, there is a data structure that with probability $1 - \delta$ returns a pair (i, x_i) with $i \in \text{Supp}(x)$ such that $\Pr[i = j] = (1 \pm \epsilon) / \|x\|_0$ for every $j \in \text{Supp}(x)$. The algorithm uses $O(\log(\mathcal{U}M/\epsilon) \cdot \log \mathcal{U} \cdot \log(1/\delta)/\epsilon)$ space.

Proof : Denote $n := |\text{Supp}(x)| = \|x\|_0$ and assume that DE returns a 2-approximation, which happens with probability at least $1 - \delta/2$. Then we can choose j such that $n \leq 2^j \leq 4n$. Let $k \in [I]$ be fixed. For simplicity of notation we define $h = h_{j,k}$ and $\alpha = \alpha_{j,k}$. We say that $l \in [U]$ is *chosen*, if it is the only entry x_l from x with $x_l \neq 0$ and $h(l) = \alpha$.

For fixed l, m the probability over the choice of h for the event $h(l) = h(m)$ is $1/T$ by pairwise independence. Let us fix an l with $x_l \neq 0$. Then we get

$$\Pr[\exists_m x_m \neq 0 \wedge h(l) = h(m)] \leq (n-1)/T \leq \epsilon \cdot (n-1)/2^{j+1} \leq \epsilon/2$$

Since α is chosen uniformly at random from the target space $[T]$ and independently of the choice of h , the events $h(l) = \alpha$ and $\exists_m (x_m \neq 0 \wedge h(l) = h(m))$ are independent. Therefore,

$$\frac{1}{T} \geq \Pr[l \text{ is chosen}] \geq \frac{1}{T} \cdot (1 - \epsilon/2) .$$

Since these events are disjoint for all l we get

$$\frac{n}{T} \geq \Pr[\text{any element is chosen}] \geq \frac{n}{T} \cdot (1 - \epsilon/2) .$$

It follows for each l with $x_l \neq 0$

$$\Pr[l \text{ is chosen} \mid \text{any element is chosen}] \geq \frac{1}{T} \cdot (1 - \epsilon/2) / \frac{n}{T} = (1 - \epsilon/2) \cdot 1/n$$

and

$$\Pr[l \text{ is chosen} \mid \text{any element is chosen}] \leq \frac{1/T}{n/T \cdot (1 - \epsilon/2)} = 1/(n \cdot (1 - \epsilon/2)) \leq \frac{1}{n} (1 + \epsilon) .$$

Thus, when an element is chosen, it is chosen almost uniformly at random from $\text{Supp}(x)$. It remains to show that for at least one k an element is chosen. From the argumentation above it follows:

$$\Pr[\text{any element is chosen}] \geq \frac{n}{T} (1 - \epsilon/2) = \frac{\epsilon n}{2^{j+1}} (1 - \frac{\epsilon}{2}) \geq \frac{\epsilon}{8} (1 - \frac{\epsilon}{2}) = (2 - \epsilon) \frac{\epsilon}{16} \geq \frac{\epsilon}{16}$$

Note that our data structure fails exactly when the data structure DE does not work (probability $\delta/2$) or when no element is chosen for all k . No element is chosen, when for all k

the data structure $UE_{j,k}$ fails. Since the number of instances I is $\lceil \log_{(1-\epsilon/16)} (\delta/2) \rceil$ we know that the probability that no element is chosen for any k is at most $(1-\epsilon/16)^I \leq \delta/2$. Hence the overall probability of error is at most δ and our data structure is correct with probability at least $1 - \delta$.

The algorithm uses $O(\log U \cdot \log(1/\delta)/\epsilon)$ hash functions, $\alpha_{j,k}$ values and UE data structures. Each hash function uses $O(\log U)$ space, each $\alpha_{j,k}$ value uses $O(\log(U/\epsilon))$ space and each UE data structures uses $O(\log(UM))$ space. The DE data structure uses $O(\log(1/\delta) \log(U) \log(UM))$ space.

Hence the overall space complexity is $O(\log(UM/\epsilon) \cdot \log U \cdot \log(1/\delta)/\epsilon)$. \square

3.5. Sampling in Geometric Data Streams

In this section we will discuss some direct consequences of Lemma 2. First, we observe that we can apply our data structure to the setting of dynamic geometric data streams in the following way. We will use $U = \Delta^d$ and $M = \{0, 1\}$. An $\text{ADD}(p)$ operation with $p = (p_0, \dots, p_{d-1})$ is implemented as an $\text{UPDATE}(P, 1)$ operation with $P = \sum_j p_j \cdot \Delta^j$, i.e. by interpreting p as a Δ -ary number with d digits. In a similar way, a $\text{REMOVE}(p)$ operation translates to $\text{UPDATE}(P, -1)$. Using the SAMPLE procedure we can get a pair (i, x_i) having $x_i = 1$, which can be also be re-interpreted as the corresponding unique point $p = (p_0, \dots, p_{d-1})$ with $i = \sum_j p_j \cdot \Delta^j$. Thus we can sample a point from the current point set. Using $O(s + \log(1/\delta))$ instances of our data structure we obtain

Theorem 1. (Sampling in geometric data streams.) *Let $\delta > 0$ be an error probability parameter. Given a sequence of ADD and REMOVE operations of points from the discrete space $[\Delta]^d$, there is a data structure that with probability $1 - \delta$ returns s points r_0, \dots, r_{s-1} from the current point set $P = \{p_0, \dots, p_{n-1}\}$ such that $\Pr[r_i = p_j] = \frac{1}{n} \pm \frac{\delta}{\Delta^d}$ for every $j \in [n]$ and returns a flag FAIL otherwise. The points are chosen independently and may contain duplicates. The algorithm uses $O\left((s + \log(1/\delta)) \cdot d^2 \cdot \log^2(\Delta/\delta)\right)$ space.*

Proof: We apply Lemma 2 with error probability parameter δ/Δ^d and invoke $2^{13} \cdot (s + \ln(1/\delta))$ instances of the data structure, each working with independent random choices. Let Y denote the random variable for the number of instances that return a random element. We assume that $1/(8e) - \delta/\Delta^d \geq 2^{-10}$ since otherwise we can simply store all elements in constant space. Hence, $\mathbf{E}[Y] \geq 8 \cdot (s + \ln(1/\delta))$ and

$$\Pr[Y < s] \leq \Pr[Y \leq (1 - (1/2))\mathbf{E}[Y]] \leq e^{(1/2)^2 \cdot \mathbf{E}[Y]/2} \leq \delta .$$

\square

We remark that Theorem 1 requires that no point in P occurs more than once, i.e. P is not a multiset. Before we apply this theorem to get ϵ -nets and ϵ -approximations of the current point set, we will derive two simple consequences. First we will show that we can recover the current point set P , if we spend space slightly larger than the size of P . This will be useful in situations when there are many insertions in the data stream followed by almost the same number of deletions, i.e. when the current point set is very small.

Corollary 1. *Let $\delta > 0$ be an error probability parameter. Given a sequence of ADD and REMOVE operations of points from the discrete space $[\Delta]^d$, there is a data structure that with probability $1 - \delta$ returns the current point set $P = \{p_0, \dots, p_{n-1}\}$. The algorithm uses $O(n(\log n + \log(1/\delta)) \cdot d^2 \cdot \log^2(\Delta))$ space.*

Proof : Let $\delta > 0$ be given. We apply Theorem 1 with parameters $s = n(\ln n + \ln(2/\delta))$ and error probability parameter $1/2$. Following the proof of Theorem 1 we get with probability at least $1 - \delta/2$ a random sample of size s . Let us fix an arbitrary point $p \in P$. We have

$$\Pr^{\forall i \in [s] : p \neq r_i} \leq \left(1 - \frac{1}{n} + \frac{1/2}{\Delta^d}\right)^s \leq \left(1 - \frac{1/2}{n}\right)^s \leq \left(1 - \frac{1}{2n}\right)^s \leq e^{-\ln n - \ln(2/\delta)} = \frac{\delta}{2n} .$$

It follows

$$\Pr^{\forall p \in P \exists r_i : r_i = p} \leq n \cdot \frac{\delta}{2n} \leq \delta/2 .$$

Therefore, the overall probability of failure is at most δ and the corollary follows. \square

The second consequence can be obtained by translating Theorem 1 (which uses independent draws and thus the sample set may contain multiple copies of the same point) to the case of sampling of random subsets.

Corollary 2. *Let $s \leq n/2$ and let $\delta > 0$ be an error probability parameter. Given a sequence of ADD and REMOVE operations of points from the discrete space $[\Delta]^d$, there is a data structure that with probability $1 - \delta$ returns a subset $S = \{r_0, \dots, r_{s-1}\}$ of s points from the current point set $P = \{p_0, \dots, p_{n-1}\}$ such that $\Pr[p_j \in S] = \frac{s}{n} \pm \frac{\delta}{\Delta^d}$ for every $j \in [n]$. The algorithm uses $O\left((s + d \cdot \log(\Delta/\delta)) \cdot d^2 \cdot \log^2(\Delta/\delta)\right)$ space.*

Proof : Let us first assume that we have a data structure that returns a point distributed exactly uniformly among the current point set. Then we can select $s' = c \cdot (s + \log(1/\delta))$ points $r'_0, \dots, r'_{s'}$ from P uniformly at random with repetitions, where c is a suitable constant. Since $s \leq n/2$ we know that for each r'_i we have with probability at least $1/2$ a point that is not among the previously chosen points. And so we get that with probability at least $1 - \delta/2$ we have at least s distinct points among $r'_0, \dots, r'_{s'}$ for c large enough. If there are more than s distinct points among $r'_0, \dots, r'_{s'}$ for c we choose s points uniformly at random from the distinct points. Clearly, the computed point set is distributed uniformly at random.

It remains to deal with the fact that Theorem 1 does not provide us with an exactly uniform distribution. We will use error probability parameter $\delta/(2 \cdot s' \cdot \log(1/\delta) \cdot \Delta^{2d}) \leq \delta/(2 \cdot s' \cdot \Delta^d)$ when we apply the theorem. Each r_i is chosen according to a distribution that has a statistical difference to the uniform distribution of at most $(\delta/(2 \cdot s' \cdot \Delta^d))/\Delta^d \cdot n \leq \delta/(2 \cdot s' \cdot \Delta^d)$ (for each possible outcome of r_i we deviate at most $\delta/(2 \cdot s' \cdot \Delta^d)/\Delta^d$ from the uniform distribution). Since we choose s' elements the overall statistical difference of our process to the ideal one described above is at most $\delta/(2\Delta^d) \leq \delta/2$. Therefore, the overall probability of error is at most δ and $\Pr[p_j \in S] = \frac{s}{n} \pm \frac{\delta}{\Delta^d}$ for every $j \in [n]$. \square

3.5.1. Random Sampling with Neighborhood Information

We now want to develop a more sophisticated sampling strategy. We would like to draw a set of points (almost) uniformly at random and for each point we also would like to know its neighborhood, for example all points within a distance of at most z or all points in a square with side length z centered at the random point. Formally, we define a neighborhood in the following way.

Definition 1. Let $V = \{v_0, \dots, v_{Z-1}\}$ denote a set of grid vectors with $v_0 = (0, \dots, 0)$. We define the V -neighborhood of a point p to be the set $\mathcal{N}(V, p) = \bigcup_{i \in Z} \{p + v_i\}$. We call $Z = |V|$ the size of the V -neighborhood.

We will typically assume that the size Z of a V -neighborhood is small, i.e. polylogarithmic. It remains to show that we are able to get information about the V -neighborhood of a sample point. This can be achieved in the following way. We use Lemma 2 with $U = \Delta^d$ and $M = \Delta^{Zd}$ to map the problem from the discrete Euclidean space to a vector problem. We identify each element from $[U]$ with a vector $x_i = (q_0, \dots, q_{Z-1})$ with entries q_i from $[\Delta]^d$. Hence, every entry of our vector x will itself be a vector representing the neighborhood of a point. An $\text{ADD}(p)$ operation then translates to Z $\text{UPDATE}(q_i, \Delta^{di})$ operations, where $q_i = p - v_i$ for $v_i \in V$. In a similar way, a $\text{REMOVE}(p)$ translates to Z $\text{UPDATE}(q_i, -\Delta^{di})$ operations.

By Lemma 2 we have a data structure that returns a random element from the support of x . However, we would like to have a sample from those entries $x_i = (q_0, \dots, q_{Z-1})$ that have $q_0 \neq 0$. Let $I = \{(i, x_i) : x_i = (q_0, \dots, q_{Z-1}) \text{ with } q_0 \neq 0\}$. We observe that $|I| \cdot Z \geq \|x\|_0$. Let us first assume that the element returned by the data structure from Lemma 2 is distributed exactly uniformly. We proceed similarly as in the proof of Theorem 1. If we pick a sample of size $s^* = \lceil c \cdot Z \cdot \ln(1/\delta) \rceil$ for suitably chosen constant c we get with probability $1 - (1 - 1/Z)^{\lceil Z \ln(1/\delta) \rceil} \geq 1 - \delta$ at least one random element from I . If we have more than one element we pick one of them uniformly at random. This way, we get a single random element from I .

Finally, we want to take into account that the distribution in Lemma 2 is not exactly uniform and that there is a probability of failure. Using error probability parameter $\delta/(2 \cdot s^* \cdot \Delta^d)$ we get that the statistical difference from the same process with exactly uniform distribution is at most δ/Δ^d . Hence, we can conclude

Corollary 3. *Let the set $V = \{v_0, \dots, v_{Z-1}\}$ be fixed. Let $\delta > 0$ be an error probability parameter. Given a sequence of ADD and REMOVE operations of points from the discrete space $[\Delta]^d$, there is a data structure that with probability $1 - \delta$ returns a point r from the current point set $P = \{p_0, \dots, p_{n-1}\}$ such that $\Pr[r = p_j] = \frac{1}{n} \pm \frac{\delta}{\Delta}$ for every $j \in [n]$. Additionally, the algorithm returns the set $P \cap \mathcal{N}(V, r)$. The algorithm uses $O\left(Z^3 \cdot \log(1/\delta) \cdot d^2 \cdot \log^2(\Delta/\delta)\right)$ space. \square*

Using error probability parameter δ/Δ^d and running s instances of the data structure from Corollary 3 we get

Corollary 4. Let the set $V = \{v_0, \dots, v_{Z-1}\}$ be fixed. Let $\delta > 0$ be an error probability parameter. Given a sequence of ADD and REMOVE operations of points from the discrete space $[\Delta]^d$, there is a data structure that with probability $1 - \delta$ returns s points r_0, \dots, r_{s-1} from the current point set $P = \{p_0, \dots, p_{n-1}\}$ such that $\Pr[r_i = p_j] = \frac{1}{n} \pm \frac{\delta}{\Delta}$ for every $j \in [n]$. The points are chosen independently and may contain duplicates. Additionally, the algorithm returns the sets $P \cap \mathcal{N}(V, r_i)$ for every $i \in [s]$. The algorithm uses $O\left(s \cdot Z^3 \cdot d^2 \cdot \log^3(\Delta/\delta)\right)$ space. \square

3.6. ϵ -Nets and ϵ -Approximations in Data Streams

A consequence of Theorem 1 is that we can get ϵ -nets and ϵ -approximations of the current point set. We briefly recapitulate the definitions required for ϵ -nets and ϵ -approximations, which can, for example, be found in ¹.

Definition 2. Let X be a set of objects and R be a family of subsets of X . Then we call the set system $\Sigma = (X, R)$ a *range space*. The elements of R are the *ranges* of Σ . If X is a finite set, then Σ is called a *finite range space*.

Definition 3. The *Vapnik-Chervonenkis dimension (VC-dimension)* of a range space $\Sigma = (X, R)$ is the size of the largest subset of X that is *shattered* by R . We say that a set A is *shattered* by R , if $\{A \cap r \mid r \in R\} = 2^A$.

Definition 4. Let $\Sigma = (X, R)$ be a finite range space. A subset $N \subset X$ is called ϵ -net, if $N \cap r \neq \emptyset$ for every $r \in R$ with $|r| \geq \epsilon|X|$. A subset $A \subseteq X$ is called ϵ -approximation, if for every $r \in R$ we have $\left| \frac{|A \cap r|}{|A|} - \frac{|r|}{|X|} \right| \leq \epsilon$.

Obviously, an ϵ -approximation is always an ϵ -net, while the contrary is not necessarily true.

A Data Streaming Algorithm for ϵ -Approximations. The following theorem by Vapnik and Chervonenkis shows that for any finite range space with constant VC-dimension \mathcal{D} a random sample of size $\tilde{O}\left(\frac{\mathcal{D} + \log(1/\delta)}{\epsilon^2}\right)$ is an ϵ -approximation with probability at least $1 - \delta$.

Theorem 2. ³² There is a positive constant c such that if (X, R) is any range space of VC-dimension at most \mathcal{D} , $A \subset X$ is a finite subset and $\epsilon, \delta > 0$, then a random subset B of cardinality s of A where s is at least the minimum between $|A|$ and

$$\frac{c}{\epsilon^2} \cdot \left(\mathcal{D} \cdot \log \frac{\mathcal{D}}{\epsilon} + \log \frac{1}{\delta} \right)$$

is an ϵ -approximation for A with probability at least $1 - \delta$.

We can now combine Corollary 2 (or Corollary 1 in the case that the current point set is small) with Theorem 2 to obtain a data structure that with probability $1 - \delta$ returns an ϵ -approximation of the current point set.

Theorem 3. Let $\delta > 0$ be an error probability parameter. Given a sequence of ADD and REMOVE operations of points from the discrete space $[\Delta]^d$, there is a data structure that with probability $1 - \delta$ returns a set A of $O\left(\frac{\mathcal{D} \cdot \log(\mathcal{D}/\epsilon) + \log(1/\delta)}{\epsilon^2}\right)$ points that is

an ϵ -approximation of a range space (X, R) with VC-dimension \mathcal{D} . The algorithm uses $O\left(\frac{1}{\epsilon^2}(\mathcal{D} \log \frac{\mathcal{D}}{\epsilon} + \log \frac{1}{\delta})d^3 \log^3 \frac{\Delta}{\delta}\right)$ space.

Proof : By Theorem 2 a sample set of size $\alpha = \frac{c}{\epsilon^2}(\mathcal{D} \cdot \log \frac{\mathcal{D}}{\epsilon} + \log \frac{1}{\delta})$ is an ϵ -approximation with probability at least $1 - \delta/3$. Let $P = \{p_0, \dots, p_{n-1}\}$ denote the current point set. We can easily track the size n of P by increasing a counter with every ADD operation and decreasing it with every REMOVE operation. If $n \leq \alpha$ we use the data structure from Corollary 1 to recover P completely.

If $n > \alpha$ we will use our data structure from Corollary 2 to obtain a random sample of size α . We will use error probability parameter $\delta/3$. This guarantees that the overall statistical difference from the same process using the uniform distribution is at most $\delta/(3\Delta^d) \cdot n \leq \delta/3$. Similarly, the data structure fails with probability at most $\delta/3$. And a set of size α is with probability $1 - \delta/3$ an ϵ -approximation. Summing up the errors we get an ϵ -approximation with probability $1 - \delta$.

The space requirement follows immediately from Corollaries 2 and 1 and Theorem 2.

□

A Data Streaming Algorithm for ϵ -Nets. Haussler and Welzl showed that a random sample of size $\tilde{O}\left(\frac{\mathcal{D} + \log(1/\delta)}{\epsilon}\right)$ is an ϵ -net with probability at least $1 - \delta$.

Theorem 4. ²² Let (X, R) be a range space of VC-dimension \mathcal{D} , let A be a finite subset of X and suppose $0 < \epsilon, \delta < 1$. Let N be a set obtained by m random independent draws from A , where

$$m \geq \max\left(\frac{4}{\epsilon} \log \frac{2}{\delta}, \frac{8 \cdot \mathcal{D}}{\epsilon} \log \frac{8 \cdot \mathcal{D}}{\epsilon}\right)$$

then N is an ϵ -net for A with probability at least $1 - \delta$.

Combining Theorem 1 with Theorem 4 we obtain

Theorem 5. Let $\delta > 0$ be an error probability parameter. Given a sequence of ADD and REMOVE operations of points from the discrete space $[\Delta]^d$, there is a data structure that with probability $1 - \delta$ returns a set N of $O\left(\frac{\log(1/\delta)}{\epsilon} + \frac{\mathcal{D}}{\epsilon} \log \frac{\mathcal{D}}{\epsilon}\right)$ points that with probability at least $1 - \delta$ is an ϵ -net of a range space (X, R) with VC-dimension \mathcal{D} . The algorithm uses $O\left(\left(\frac{\log(1/\delta)}{\epsilon} + \frac{\mathcal{D}}{\epsilon} \log \frac{\mathcal{D}}{\epsilon}\right) \cdot d^2 \cdot \log^2(\Delta/\delta)\right)$ space.

Proof : We use a random sample (with repetitions) as given by the data structure from Theorem 1. We choose error probability parameter $\delta/3$. Since the statistical difference from the exact uniform distribution is at most $\delta/3$, the error probability is at most $\delta/3$, and a set of size $\max\left(\frac{4}{\epsilon} \log \frac{2}{\delta}, \frac{8 \cdot \mathcal{D}}{\epsilon} \log \frac{8 \cdot \mathcal{D}}{\epsilon}\right)$ is an ϵ -net with probability at least $1 - \delta/3$, we get that our sample is an ϵ -net with probability at least $1 - \delta$. □

4. 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. 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 $0 \leq i \leq \log_{1+\epsilon}(\sqrt{d}\Delta)$. 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 EMST . Our computation is based on a formula from ¹⁴ 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

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

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 $G^{(t)}$. It follows from Claim 4.1 that this only introduces a small error. We denote by $c^{(t)}$ the number of connected components of $G^{(t)}$. Then we get

Claim 4.1.

$$c_P^{(1+\epsilon)^{i+1}} \leq c^{(1+\epsilon)^i} \leq c_P^{(1+\epsilon)^{i-2}}$$

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 $(1+\epsilon)^{i-2} + \epsilon \cdot (1+\epsilon)^i \leq (1+\epsilon)^i$. Therefore, p', q' are in the same connected component of the grid cell graph.

Assume that p and q are in the same connected component of the grid graph $G^{((1+\epsilon)^i)}$. They are connected by a path of edges of length at most $(1+\epsilon)^i$ in the grid graph $G^{((1+\epsilon)^i)}$. After switching to the point graph G_P any edge of the corresponding path has a length of at most $(1+\epsilon)^i + \epsilon(1+\epsilon)^i = (1+\epsilon)^{i+1}$. Therefore p and q are in the same connected component of $G_P^{((1+\epsilon)^{i+1})}$. \square

We denote by $n^{(t)} = |P^{(t)}|$ the number of non-empty grid cells of side length $\frac{\epsilon t}{\sqrt{d}}$. Our algorithm maintains approximations $\tilde{n}, \tilde{W}, \tilde{n}^{(t)}$, and $\tilde{c}^{(t)}$ (for $t = (1 + \epsilon)^0, (1 + \epsilon)^1, (1 + \epsilon)^2, \dots$) of the number n of points currently in the set, the diameter W , the size $n^{(t)}$ of $P^{(t)}$, and the number $c^{(t)}$ of connected components in $G^{(t)}$, respectively. The approximation is derived by inserting the maintained approximations into formula 1.

4.1. Required Data Structures

In the following we discuss the data structures we need to maintain our approximations.

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.

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

We will now show, how to maintain the diameter of the point set in dimension j . For each $i \in \{0, \dots, \log \Delta\}$ 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 3 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 diameter is 0. Otherwise, the diameter 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 $W_j \leq 2^{i+1}$.

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 $W_j \geq 2^{i-1}$. Therefore, we can output $\tilde{W}_j = 2^{i-1}$ as a 4-approximation of the diameter in dimension j .

Size of $P^{(t)}$. The problem to find an estimation $\tilde{n}^{(t)}$ of $n^{(t)}$ satisfying $(1 - \epsilon) \cdot n^{(t)} \leq \tilde{n}^{(t)} \leq n^{(t)}$ 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 3.

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 $P^{(t)}$. We will use V_R to denote the set of grid vectors of length at most R . For each point $p \in S^{(t)}$ we maintain all points in the V_R -neighborhood of p for some suitably chosen value R . Since our input stream contains ADD and REMOVE operations of points from P rather than $P^{(t)}$ we have to map every point from P to the corresponding point from $P^{(t)}$. This may have the effect that $P^{(t)}$ becomes a multiset although P is not. This is no problem because our procedure from Lemma 2 samples from the support of the vector (or, in this case from the support of the multiset). Straightforward modifications of Corollary 4 shows that we can also maintain the required sets $S^{(t)}$.

Having such a sample and the value $\tilde{n}^{(t)}$ we can use an algorithm from ⁹ to obtain the number of connected components with sufficiently small error. This is proven in Section 4.2 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 $c^{(t)}$.

4.2. 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 $\tilde{W} < 4\epsilon t$. 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 $\tilde{W} \geq 4\epsilon t$. In this case our algorithm is essentially similar to the one presented in ⁹, 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 4 below to relate the error induced by our approximation algorithm to the weight of the EMST.

Lemma 4. *If $\tilde{W} \geq 4\epsilon t$ then*

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

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 a corresponding indicator variable β 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 all other points within a distance of at most $R = Dt$. Since we consider only edges of length at most t and since the algorithm below stops exploring 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. We remark that the random points from $S^{(t)}$ are not chosen *exactly* uniformly. We will choose the error probability parameter in the sampling data structure in such a way that the deviation from the uniform distribution is between $(1 - \epsilon)$ and $(1 + \epsilon)$ times its probability in the uniform distribution (this means, we choose $\epsilon \cdot \delta / \Delta^d$). We take care of this fact in the analysis below. The algorithm we use is given below.

```

APPROXCONNECTEDCOMPONENTS( $P, t, \epsilon$ )
  Choose  $s$  points  $q_1, \dots, q_s \in P^{(t)}$  uniformly at random
  for each  $q_i$  do
    Choose integer  $X$  according to distribution  $\text{Prob}[X \geq k] = 1/k$ 
    if  $X \geq D$  then  $\beta_i = 0$ 
    else
      if Connected component of  $G^{(t)}$  containing  $q_i$  has at most  $X$  vertices
      then set  $\beta_i = 1$ 
      else set  $\beta_i = 0$ 
  Output:  $\hat{c}^{(t)} = \frac{\tilde{n}^{(t)}}{s} \cdot \sum_{i=1}^s \beta_i$ 

```

Thus, β_i is an indicator random variable for the event that the connected component containing q_i has at most X vertices. We first show upper and lower bounds on the expected output value of the algorithm. Then we compute the variance and use it to show that the

output is concentrated around its expectation. We obtain

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

For the output value

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

of our algorithm we obtain

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

From

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

and Lemma 4 we obtain

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

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

$$\mathbf{Var}[\beta_i] \leq \mathbf{E}[\beta_i^2] = \mathbf{E}[\beta_i] \leq (1 + \epsilon) \cdot \frac{c^{(t)}}{n^{(t)}} .$$

By mutual independence of the β_i 's we obtain for the variance of $\hat{c}^{(t)}$ for fixed $\tilde{n}^{(t)}$:

$$\begin{aligned} \mathbf{Var}[\hat{c}^{(t)}] &= \mathbf{Var}\left[\frac{\tilde{n}^{(t)}}{s} \sum_{i=1}^s \beta_i\right] \\ &= \frac{(\tilde{n}^{(t)})^2}{s^2} \cdot s \cdot \mathbf{Var}[\beta_i] \\ &\leq (1 + \epsilon) \cdot \frac{(\tilde{n}^{(t)})^2}{s} \cdot \frac{c^{(t)}}{n^{(t)}} \\ &\leq (1 + \epsilon) \cdot \frac{n^{(t)} c^{(t)}}{s} . \end{aligned}$$

Using (2) and (3) we obtain

$$|c^{(t)} - \mathbb{E}[\hat{c}^{(t)}]| \leq 2\epsilon c^{(t)} + \frac{3\epsilon \cdot \text{EMST}}{8 \cdot t \cdot \log(\sqrt{d} \cdot \Delta)} . \quad (4)$$

We choose s , the number of sample points, as

$$s = (1 + \epsilon) \frac{2^{2d+10} \cdot d \cdot \log^2(\sqrt{d} \cdot \Delta) \cdot \log_{1+\epsilon}(\sqrt{d} \cdot \Delta)}{\epsilon^4} = O\left(\frac{\log^3 \Delta}{\epsilon^5}\right) .$$

Chebyshev's inequality and Lemma 4 imply:

$$\begin{aligned} \Pr\left[|\hat{c}^{(t)} - \mathbb{E}[\hat{c}^{(t)}]| \geq \frac{\epsilon \cdot \text{EMST}}{8 \cdot t \cdot \log(\sqrt{d} \cdot \Delta)}\right] &\leq (1 + \epsilon) \cdot \frac{n^{(t)} \cdot c^{(t)}}{s} \cdot \frac{64 \cdot t^2 \cdot \log^2(\sqrt{d} \cdot \Delta)}{\epsilon^2 \cdot \text{EMST}^2} \\ &\leq (1 + \epsilon) \cdot \frac{64 \cdot d \cdot 2^{2d+2} \cdot \log^2(\sqrt{d} \cdot \Delta)}{s \cdot \epsilon^4} \\ &\leq \frac{1}{4 \log_{1+\epsilon}(\sqrt{d} \cdot \Delta)} . \end{aligned}$$

Therefore we get together with (4):

Lemma 5. *With probability $1 - \frac{1}{4 \log_{1+\epsilon}(\sqrt{d} \cdot \Delta)}$ we have*

$$|\hat{c}^{(t)} - c^{(t)}| \leq 2\epsilon c^{(t)} + \frac{\epsilon \cdot \text{EMST}}{2 \cdot t \cdot \log(\sqrt{d} \cdot \Delta)} .$$

It now follows that with probability at least $3/4$ all $\tilde{c}^{(t)}$ values satisfy the inequality in Lemma 5. 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 .

Lemma 6. *Let \tilde{M} be the output of our algorithm. Then*

$$|\text{EMST} - \tilde{M}| \leq 69\sqrt{d} \cdot \epsilon \cdot \text{EMST} .$$

Proof: We will first 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). From Lemma 5 and Claim 4.1 it follows that

$$\begin{aligned} (1 - 2\epsilon)c_p^{((1+\epsilon)^{i+1})} - \frac{\epsilon \cdot \text{EMST}}{2(1 + \epsilon)^i \log(\sqrt{d}\Delta)} &\leq \widetilde{c}^{((1+\epsilon)^i)} \\ &\leq (1 + 2\epsilon)c_p^{((1+\epsilon)^{i-2})} + \frac{\epsilon \cdot \text{EMST}}{2(1 + \epsilon)^i \log(\sqrt{d}\Delta)} \end{aligned}$$

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

$$\begin{aligned} &\epsilon \cdot \sum_{i=0}^{\log_{1+\epsilon} \widetilde{W} - 1} (1 + \epsilon)^i \widetilde{c}^{((1+\epsilon)^i)} \\ &\geq \epsilon \cdot (1 - 2\epsilon) \sum_{i=0}^{\log_{1+\epsilon} \widetilde{W} - 1} (1 + \epsilon)^i c_p^{((1+\epsilon)^{i+1})} - \frac{1}{2} \epsilon \cdot \text{EMST} \\ &\geq \epsilon \cdot (1 - 2\epsilon) \sum_{i=1}^{\log_{1+\epsilon} \widetilde{W}} (1 + \epsilon)^{i-1} c_p^{((1+\epsilon)^i)} - \epsilon \cdot \text{EMST} \\ &\geq \epsilon \cdot \frac{1 - 2\epsilon}{1 + \epsilon} \sum_{i=0}^{\log_{1+\epsilon} \widetilde{W} - 1} (1 + \epsilon)^i c_p^{((1+\epsilon)^i)} - \frac{\epsilon(1 - 2\epsilon)}{1 + \epsilon} n - \epsilon \cdot \text{EMST} \\ &\geq (1 - 4\epsilon) \cdot \epsilon \sum_{i=0}^{\log_{1+\epsilon} \widetilde{W} - 1} (1 + \epsilon)^i c_p^{((1+\epsilon)^i)} - \epsilon n - \epsilon \cdot \text{EMST} \end{aligned}$$

and

$$\begin{aligned}
 & \epsilon \cdot \sum_{i=0}^{\log_{1+\epsilon} \widetilde{W}-1} (1+\epsilon)^i \widetilde{c}^{((1+\epsilon)^i)} \\
 & \leq \frac{1}{2} \epsilon \cdot \text{EMST} + \epsilon \sum_{i=-2}^{\log_{1+\epsilon} \widetilde{W}-3} (1+\epsilon)^{i+2} (1+2\epsilon) c_p^{((1+\epsilon)^i)} \\
 & \leq \epsilon \cdot \text{EMST} + \epsilon (1+\epsilon)^2 (1+2\epsilon) \sum_{i=0}^{\log_{1+\epsilon} \widetilde{W}-1} (1+\epsilon)^i \widetilde{c}_p^{((1+\epsilon)^i)} + 2\epsilon (1+\epsilon) (1+2\epsilon) n \\
 & \leq (1+11\epsilon) \cdot \epsilon \sum_{i=0}^{\log_{1+\epsilon} \widetilde{W}-1} (1+\epsilon)^i c_p^{((1+\epsilon)^i)} + 12\epsilon n + \epsilon \cdot \text{EMST}
 \end{aligned}$$

which gives us (together with $\widetilde{W} \leq 4\sqrt{d} \cdot \text{EMST}$ and $n \leq \text{EMST}$) a bound on the difference of M_P and \widetilde{M} :

$$\begin{aligned}
 |M_P - \widetilde{M}| & \leq 11\epsilon^2 \sum_{i=0}^{\log_{1+\epsilon} \widetilde{W}-1} (1+\epsilon)^i c_p^{((1+\epsilon)^i)} + 12\epsilon n + \epsilon \cdot \text{EMST} \\
 & = 11\epsilon (M_P - n + \widetilde{W}) + 12\epsilon n + \epsilon \cdot \text{EMST} \\
 & \leq 24\epsilon \cdot \text{EMST} + 44\epsilon \sqrt{d} \cdot \text{EMST}
 \end{aligned}$$

By the triangle inequality and (1) we get the final result:

$$\begin{aligned}
 |\widetilde{M} - \text{EMST}| & \leq |\widetilde{M} - M_P| + |M_P - \text{EMST}| \\
 & \leq 24\epsilon \cdot \text{EMST} + 44\epsilon \sqrt{d} \cdot \text{EMST} + \epsilon \cdot \text{EMST} \\
 & \leq 69\epsilon \sqrt{d} \cdot \text{EMST}
 \end{aligned}$$

□

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 6. *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.* □

5. 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 -nets and ϵ -approximations of range spaces with small 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.

The space used by our algorithm for maintaining ϵ -approximations, i.e., roughly $O(1/\epsilon^2)$, is essentially optimal as a function of ϵ . This is because it is known that, for some range spaces, the size of ϵ -approximations tends to $1/\epsilon^2$ as the VC dimension tends to infinity. However, for some range spaces smaller ϵ -approximations can be constructed, even for points delivered in an insertions-only stream. For example ³¹ showed how to compute ϵ -approximations, for ranges defined by halfspaces in d dimensions, of size roughly $O(1/\epsilon^{2-2/(d+1)})$. We do not know how to extend this result to dynamic data streams.

Another intriguing question is as follows. Suppose that the goal of the data stream algorithm is not to compute an actual ϵ -approximation, but simply to construct an “approximation oracle” which, given any range R , returns a value $\frac{|P \cap R|}{|P|} \pm \epsilon$, with probability $1 - \delta$ for some $\delta > 0$. Note that maintaining an ϵ -approximation is a particular way of implementing such an approximation oracle, but other methods for its construction might conceivably exist. Is it possible to construct an approximation oracle for some family of ranges, whose space requirement, as a function of ϵ , is asymptotically smaller than the size of an ϵ -approximation? E.g., is it possible to construct an oracle for halfspaces in d dimensions which uses $1/\epsilon^{2-2/(d+1)-\alpha}$ space for some $\alpha > 0$? We conjecture that this is *not* possible; however, we do not know how to show this.

Finally, we remark that our sampling approach provides an approximation oracle using space that is roughly quadratic in $1/\epsilon$, even for range spaces with *arbitrary* VC dimension ^c. Without limiting the VC dimension of the range space, the quadratic dependence on $1/\epsilon$ is tight. Specifically, if the ranges are allowed to be *arbitrary* subsets of the universe X (i.e., if the VC dimension of the range space is equal to $|X|$), then the quantity $|R \cap P|$ is simply equal to $v(R) \cdot v(P)$, where $v(R)$ (and $v(P)$, resp.) is a characteristic vector of R (and P , resp.). It is known ²⁷ that, for $|P| = \Theta(|X|)$, any streaming algorithm which estimates such a dot product with error $\pm \epsilon|X|$ requires $\Omega(1/\epsilon^2)$ space. Thus, for unbounded VC dimension, our $O(1/\epsilon^2)$ bound is tight.

References

1. P. Agarwal. Range Searching. Handbook of Discrete and Computational Geometry. J. Goodman and J. O’Rourke (eds.). Second edition, CRC Press, pp. 809–838, 2004.
2. P. Agarwal, S. Har-Peled, and K. Varadarajan. Approximating Extent Measures of Points. *Journal of the ACM*, 51(4):606–635, 2004.
3. N. Alon, Y. Matias, and M. Szegedy. The Space Complexity of Approximating the Frequency Moments. *J. Comput. Syst. Sci.*, 58(1): 137–147, 1999.
4. 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.

^cNote that the definition of approximation oracle, unlike the definition of ϵ -approximation, allows each answer to be incorrect with some small probability $\delta > 0$. Thus, a random sample of points in P provides a good estimation for any given R .

26 Gereon Frahling, Piotr Indyk, Christian Sohler

5. Z. Bar-Yossef, T. S. Jayram, R. Kumar, D. Sivakumar, and L. Trevisan. Counting Distinct Elements in a Data Stream. *Proc. RANDOM*, pp. 1–10, 2002.
6. J. Carter and M. Wegman. Universal Classes of Hash Functions. *Journal of Computer and System Sciences*, 18(2):143-154, 1979.
7. 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.
8. T. M. Chan. and B. S. Sadjad. Geometric Optimization Problems over Sliding Windows. *Proc. 15th International Symposium on Algorithms and Computation*, pp. 246–258, 2004.
9. 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.
10. G. Cormode and M. Muthukrishnan. Radial Histograms for Spatial Streams. DIMACS Technical Report 2003-11, 2003.
11. G. Cormode, M. Muthukrishnan, and I. Rozenbaum. Summarizing and Mining Inverse Distributions on Data Streams via Dynamic Inverse Sampling. *Proc. 31st VLDB Conference*, pp. 25–36, 2005.
12. G. Cormode, M. Datar, and P. Indyk. Comparing Data Streams using Hamming norms. *Proc. International Conference on Very Large Databases (VLDB)*, pp. 335–345, 2002.
13. A. Czumaj, F. Ergün, L. Fortnow, A. Magen, I. Newman, R. Rubinfeld, and C. Sohler. Sublinear-time approximation of Euclidean minimum spanning tree. *SIAM Journal on Computing*, 35(1): 91-109, 2005.
14. 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.
15. 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.
16. 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.
17. G. Frahling and C. Sohler. Coresets in Dynamic Geometric Data Streams. *Proc. 37th Annual ACM Symposium on Theory of Computing (STOC)*, pp. 209–217, 2004.
18. P. Flajolet and G. Martin. Probabilistic Counting Algorithms for Data Base Applications. *Journal of Computer and System Sciences*, 31:182–209, 1985.
19. S. Ganguly, M. Garofalakis, and R. Rastogi. Tracking Set-Expression Cardinalities over Continuous Update Streams. *The VLDB Journal*, 13(4), pp. 354–369, 2004.
20. T. Hagerub and C. Rüb. A Guided Tour of Chernoff Bounds. *Information Processing Letters*, 33:305–308, 1989/90.
21. S. Har-Peled and 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.
22. D. Haussler and E. Welzl. ϵ -Nets and Simplex Range Queries. *Discrete and Computational Geometry*, 2:127–151, 1987.
23. 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.
24. P. Indyk. Stable Distributions, Pseudorandom Generators, Embeddings and Data Stream Computation. *Proc. 41st IEEE Symposium on Foundations of Computer Science (FOCS)*, pp. 189–197, 2000.
25. P. Indyk. Algorithms for Dynamic Geometric Problems over Data Streams. *Proc. 36th Annual ACM Symposium on Theory of Computing (STOC)*, pp. 373–380, 2004.
26. 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.

27. P. Indyk and D. Woodruff. Tight Lower Bounds for the Distinct Elements Problem. *Annual Symposium on Foundations of Computer Science*, pages 283–290, 2003.
28. G. S. Manku and R. Motwani. Approximate Frequency Counts over Data Streams. *Proc. 2002 Intl. Conf. on Very Large Data Bases*, pp. 346–357, 2002.
29. M. Muthukrishnan. Data streams: Algorithms and Applications (invited talk at SODA'03). Available at <http://athos.rutgers.edu/muthu/stream-1-1.ps>, 2003.
30. N. Nisan. Pseudorandom generators for Space-Bounded Computation. *Proc. 22nd Annual ACM Symposium on Theory of Computing (STOC)*, 204–212, 1990.
31. 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.
32. V. Vapnik and A. Chervonenkis. On the Uniform Convergence of Relative Frequencies of Events to their Probabilities. *Theory Probab. Appl.*, 16:264–280, 1971.