1 Introduction

A data stream is an ordered sequence of points that can be read only once or a small number of times. Formally, a data stream is a sequence of points $x_1, x_2, \ldots, x_i, \ldots, x_n$ read in increasing order of the indices $i$. The performance of the algorithm is measured on the number of passes the algorithm must make over the data under limited memory constraints. The data streaming model is motivated by emerging application involving massive data sets, e.g., customer click streams, telephone records, large sets of web pages, multimedia data.

A dynamic geometric data stream can be viewed as a stream of $m$ ADD/REMOVE operations on points from a discrete geometric space $\{1, \ldots, \Delta\}$. ADD($p$) adds a point to the current pool of points. DELETE($P$) removes a point from the current pool of points. The challenge is to maintain low space data structures which solve a certain geometric problem on the current pool of points. This concept was first introduced in

In this project, we summarize some of the geometric data streaming algorithms and techniques. We look at problems like the $k$-median, range searching and frequency moments in the insertion-only model. We also look at estimating the weight of euclidean minimum spanning tree using the technique of dynamic sampling which is used to develop low space solutions for maintaining $\epsilon$-approximations on range spaces with constant VC-dimensions on the point set.

We apply some of the techniques from the papers read to show that approximating the length of the closest pair of points doesn't have a sub-linear space solution which runs in a constant number of passes on the input.

2 Insertion-only streaming algorithms

2.1 Counting frequency streaming moments

Classical example of a streaming algorithm is the algorithm developed for counting the number of distinct elements in a stream. In general, if $n$ is the
stream size, $m$ is the size of the universe and $f_i$ is the number of occurrences of item $i$, then the $k^{th}$ frequency moment

$$F_k = \sum_{1}^{m} f_i^k$$

$F_0$ = the number of distinct elements,
$F_1$ = the size of the stream,
$F_2$ = self-join size.

In general, the value $F_k$ measures data-skewness.

**Theorem 1.** [1] For any fixed $k > 5$ and $\gamma < 1/2$, any randomized algorithm that outputs, given an input sequence $A$ of at most $n$ elements of $N = \{1, 2, \ldots, n\}$, a number $Z_k$ such that $\Pr(|Z_k - F_k| > 0.1F_k) \leq \gamma$ uses at least $\Omega(n^{1-5/k})$ memory bits.

**Theorem 2.** For every $k \geq 1$, every $\lambda > 0$ and every $\epsilon > 0$ there exists a randomized algorithm that computes, given a sequence $A$ of $m$ numbers of $N = \{1, 2, \ldots, n\}$, in one pass and using

$$O\left(\frac{k\log(1/\epsilon)}{\lambda^2}n^{1-1/k}(\log n + \log m)\right)$$

memory bits, a number $Y$ so that the probability that $Y$ deviates from $F_k$ by more than $\lambda F_k$ is at most $\epsilon$.

The algorithm to compute frequency moments essentially constructs a low space estimator $X$ such that, $E[X] = F_k$, $Var(X)$ is small and Chebyshev’s inequality is used to prove the desired results.

We will discuss an interesting result from the above paper. We then show how the same proof can be used to show hardness of certain geometric proximity problems.

**Lemma 1.** Any randomized algorithm that outputs given a sequence $A$ of at most $2n$ elements of $N = \{1, 2, \ldots, n\}$ a number $Y$ such that the probability that $Y$ deviates from $F_\infty$ by at least $F_\infty/3$ is less than $\epsilon$, for some fixed $\epsilon < 1/2$ must use $\Omega(n)$ bits of memory.

**Proof.** Consider the following communication problem.

Let $DIS_n : \{0, 1\}^n \times \{0, 1\}^n \rightarrow \{0, 1\}$ denote the boolean function (called the disjointness function) where $DIS_n(x, y)$ is 1 iff the subsets of $\{1, 2, \ldots, n\}$ whose characteristic vectors $x$ and $y$ intersect. In [2] it has been shown that for any fixed $\epsilon < 1/2$, any approximation communication algorithm requires $\Omega(n)$ bits of communication.

We prove the above algorithm by contradiction. Let there be a randomized approximation algorithm for $F_\infty$ which uses $o(n)$ bits of memory. We can use that to solve the $DIS_n$ in space $o(n)$ which would result in a contradiction.
Here is how we construct a protocol. The first party knowing \( x \) runs the approximation algorithm on the first \(|x|\) bits of \( A \). It then sends the content of the memory (which is \( o(n) \)) to the second party. The second party essentially continues the algorithm. The second party outputs disjoint iff the output of the approximation algorithm is smaller than \( 4/3 \); else it outputs 1. It is obvious that this is correct value with probability at least \( 1 - \epsilon \), since the precise value of \( F_\infty \) is 1 if the sets are disjoint, and otherwise it is 2.

Now consider another problem called the Element Uniqueness problem. The problem is to detect duplicates in a stream of values. The proof mentioned above can be used to show that any randomized approximation algorithm for the element uniqueness problem has an \( \Omega(n) \) space lower bound.

**Lemma 2.** Element Uniqueness reduces to the 1-dimensional closest pair problem.

**Proof.** Suppose you have a randomized approximation algorithm for the closest pair problem in 1-dimension. Clearly, if there are duplicate elements, then the closest pair algorithm will return a value 0 with high probability. On the other hand if there are no duplicates, the closest pair algorithm will give an approximate and positive value. Hence, if we have a solution for the closest pair problem, we can solve the element uniqueness problem.

Now since, we have claimed that any randomized approximation algorithm for element uniqueness should take \( \Omega(n) \) bits of memory, it follows that the closest pair problem also should take \( \Omega(n) \) bits of memory. In fact, we can go ahead and show that there doesn’t exist a multiple pass algorithm which uses \( o(n) \) space for the closest pair problem.

Further, if under some error model, we can solve the element uniqueness problem in \( S(n) \) space, then a \((1+\epsilon)\) approximation of the 2 dimensional closest pair problem in discrete geometric space \( \{1, 2, \ldots \Delta\}^d \) can be found in space \( \log_{1+\epsilon} \Delta S(n) \), where \( \Delta \) the range from which the points are coming.

We briefly sketch the technique. This is essentially the technique used in finding the approximate weight of the Euclidean Minimum Spanning Tree Consider \( \log_{1+\epsilon} \Delta \) grids with squares of side length \((1+\epsilon)\). We treat the stream of points as \( \log_{1+\epsilon} \Delta \) streams of cells. In other words, we map every point to the center of the grid-cell within which it lies. Hence, a duplicate detection implies that there are two points in the same grid cell. We can ensure, by laying multiple grids of the same size, that the closest pair of points lie in some common grid cell such that the side length of the grid cell is \(1/(1+\epsilon)\) times the length of the closest pair. Thus we have shown that the closest pair problem and the element uniqueness problem are equivalent problems.

In [3] an algorithm for computing frequency moment which supports deletion is presented. The space complexity asymptotically matches the known lower bounds for the problem.
2.2 Metric Clustering

The objective of clustering under data streaming model is to essentially maintain a good clustering of the sequence observed so-far, using small amount of memory and time.

In, [4] a randomized algorithm for constructing $k$ medians is given.

<table>
<thead>
<tr>
<th>Algorithm 1 RANDOM-K-MEDIAN(S)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: Draw a sample of size $s = \sqrt{nk}$</td>
</tr>
<tr>
<td>2: Find $k$ medians from these $s$ points using primal dual method</td>
</tr>
<tr>
<td>3: Assign each of the $n$ points to its closest median</td>
</tr>
<tr>
<td>4: Collect the $n/s$ points with the largest assignment distance</td>
</tr>
<tr>
<td>5: Find $k$ medians among these $n/s$ points.</td>
</tr>
<tr>
<td>6: return $2k$ medians thus found</td>
</tr>
</tbody>
</table>

**Lemma 3.** Algorithm 1 gives an $O(1)$ approximation with $2k$ medians with constant probability.

To be able to find the medians with high probability we need multiple passes over the data. The space requirement of the above algorithm can be shown to be $O(\sqrt{nk})$.

Charikar, Chekuri, Feder and Motwani [5] gave a constant factor algorithm for incremental $k$-center problem which is a single pass algorithm requiring $O(nk \log k)$ time and $O(k)$ space.

In [6] a streaming algorithm to find the $k$-median is given. A simple divide and conquer approach is built. The general approach is to examine the data in a piecemeal fashion. In particular, an algorithm which divides the data into pieces and then again clusters the centers obtained. It can be shown that if there is a constant factor approximation algorithm, running the divide and conquer approach would still yield a slightly-worse constant factor approximation. This approach is to essentially maintain sketch of sketches. They use a well-known off-line approximation algorithm for $k$-median and extend that to the streaming model.

The essential divide and conquer strategy has been defined below

**Lemma 4.** For any constant $i$, Algorithm 3 would give a constant factor approximation to the $k$-median problem.

**Theorem 3.** The $k$-median problem has a constant factor approximation algorithm running in time $O(nk \log n)$, in one pass over the data set, using $n^\epsilon$ memory for small $k$. 

Algorithm 2 SMALL-SPACE(S)
1: Divide $S$ into $l$ disjoint pieces $X_1, X_2 \ldots X_l$
2: for each $i$ do
3:  Find $O(k)$ centers in $X_i$.
4:  Assign each point in $X_i$ to its closest center.
5: end for
6: Let $X'$ be the $O(lk)$ centers obtained where each center $c$ is weighted by the number of points assigned to it.
7: Cluster $X'$ to find $k$ centers.

Algorithm 3 SMALLER-SPACE($S, i$)
1: Divide $S$ into $l$ disjoint pieces $X_1, X_2 \ldots X_l$
2: for each $i$ do
3:  Find $O(k)$ centers in $X_i$.
4:  Assign each point in $X_i$ to its closest center.
5: end for
6: Let $X'$ be the $O(lk)$ centers obtained where each center $c$ is weighted by the number of points assigned to it.
7: Call Algorithm SMALLER-SPACE($X', i-1$)

2.3 Streaming algorithms for range counting

The one dimensional case In [8], Greenwald and Khanna present a deterministic algorithm for building a data structure of size $O\left(\frac{1}{\epsilon} \log \epsilon n\right)$ that can report the rank of a given query point $q \in \mathbb{R}$ among the $n$ points seen so far with an error within the interval $[-n\epsilon, 0]$. Therefore it can also answer the range counting queries with the same error.

For a point set $P \subset \mathbb{R}^d$, we call a system $F = \{F_i : i \in I\}, \{r_i : i \in I\}$, a representative system if $F = \{F_i : i \in I\}$ is a partition of $P$ and $r_i \in \mathbb{R}^d, i \in I$.

Also a representative system is $\epsilon$-deficient for $Q$, if for any range $q' \in Q$ the total number of points separated from their representative by $Q$ is at most $\epsilon|P|$.

In the paper they maintain a representative system $F = \{F_i \{r_i\} | i = 1, \ldots, k\}$ of size $k = O\left(\frac{1}{\epsilon} \log \epsilon n\right)$ and store only the pairs $\{|F_i|, r_i\}$ in memory. Additionally, $r_i$ is the rightmost point of $F_i$ and for half-space queries $Q^- = \{(-\infty, q), q \in \mathbb{R}\}$, $F$ is $\epsilon$-deficient. This immediately implies that the data structure can answer grounded range queries. In particular it can answer interval queries with error $\epsilon n$.

Using the above technique on each of the dimensions independently, the following theorem was developed in [?]
Theorem 4. For a stream of points in \( \mathbb{R}^d \), we can deterministically main-
tain a weighted \( \epsilon \)-approximation of size \( O(\frac{1}{\epsilon d} \log^d \text{en}) \) for axis-aligned rectangle
ranges.

3 Dynamic Data Streaming Algorithms

In this section, we introduce dynamic sampling and \( \epsilon \)-approximation in dynamic
data streams and show its application to estimating the weight of euclidean
minimum spanning tree. The solution is in [9]

3.1 Dynamic Sampling

Consider a sequence of \( n \) insertions and deletions from a finite universe \( U = \{1 \ldots u - 1\} \). Each element can be inserted and deleted multiple number of
times. Thus the stream represents a multiset. Dynamic Sampling would be a
low-space data structure which maintains a random element chosen uniformly
at random from \( P \). There are a few assumptions without which this problem
would be trivially impossible. We state the assumptions made below

1. It might appear that any algorithm maintaining a random element from
dynamic point set might require \( \Omega(U) \) space. This is because, you can
retrieve all elements of \( P \) by picking a random element and deleting it
from \( P \) repeatedly. To avoid this problem, it is assumed that the stream of
INSERT/ DELETE operations is chosen before the algorithm chooses its
random bits.

2. The points are assumed to be from discrete geometric space. Alternatively
we could have assumed that the inter-point distances are integers between
1 and \( \Delta \). Such assumptions are fair when bounded precision arithmetic is
used. In streaming algorithms, the assumption of bounded precision is com-
mon because otherwise the notion of storage is not well defined. (Example:
We cannot use the tools of lower bound proofs from discrete communication
complexity)

Under these assumptions, we show that one can maintain a random element
with probability \( 1 - \delta \) using only \( O(\log^2(U M/\delta)) \) space. (\( U \) is the size of the
universe and \( M \) is the maximum multiplicity of an element in \( P \)).

The data structure Consider a data structure problem: Build a low space data
structure that enables certain operations on a vector \( x : [U] \mapsto [M] \). Initially
the vector \( x \) is 0 Let \( \text{Supp}(x) \) be the set of indices \( i \) for which \( x_i > 0 \). The data
structure is parameterized by two values \( \delta' > 0 \) and \( \delta'' > 0 \). The operations
supported by this data structure are:

1. UPDATE(\( i, a \)): Changes \( x_i = x_i + a \). The assumption here is that \( a \) is such
that \( x_i \) remains in the range defined by the universe \( U \).
2. **SAMPLE**: This operation returns a pair \((r, v)\). \(r \in U\) and \(v = x_r\) or a flag `FAIL`. If a pair \((r, v)\) is returned, then \(r\) is chosen uniformly at random from \(\text{Supp}(x)\) with probability \(\Pr[r = i] = \frac{1}{|\text{supp}(x)|} \pm \delta'\). The probability of a `FAIL` being returned is \(\delta''\).

We construct a data structure with \(\delta''\) strictly separated from 1.

To implement the above data structure the following data structures are used.

**Unique Element** This data structure supports two operations, namely, `UPDATE` (defined as above) and `REPORT`. `REPORT` essentially returns a pair \((i, x_i)\) such that \(x_i > 0\) only if \(|\text{Supp}(x)| = 1\). Else, it can return any pair. This can be easily implemented using counters. The space requirement for the data structure is \(O(\log U M)\) bits.

**Distinct Elements** This data structure supports two operations, namely, `UPDATE` (defined as above) and `REPORT`. `REPORT` essentially returns a value \(k\) such that \(k \approx |\text{Supp}(x)|\). In other words, `REPORT` returns a value \(k\) such that \(k \leq |\text{supp}(x)| \leq k(1 + \epsilon)\). This problem is essentially maintaining frequency moments solved in [1]. The space complexity of this data structure is \(O(\log^2(MU/\delta)/\epsilon^2)\).

The data structure proposed in [9] uses the data structures mentioned above and fully-random hash functions \(h_j, j \in [\log U]\). Each \(h_j\) is of the form \(h_j : [u] \mapsto [2^j]\). It also uses \(\log U\) instances of both unique element data structure and Distinct element data structure. We call the Unique element data structure \(UE_j\) for \(j \in [\log U]\) and distinct element data structure \(DE_j\) for \(j \in [\log U]\), with parameters \(\epsilon = 1/2\) and \(\delta = \delta'/2\).

The operations \(\epsilon = 1/2\) and \(DE_j\) reports 1

**Algorithm 4 UPDATE\((P)\)**

1: for \(j \in [\log U]\) do
2: if \(h_j(i) = 0\) then
3: \(UE_j\).UPDATE\((i, a)\)
4: \(DE_j\).UPDATE\((i, a)\)
5: end if
6: end for
7: end for

**Correctness proof** With at least probability \(1 - \delta'\), \(DE_j\) reports the correct approximation of the number of distinct elements. Since, \(\epsilon = 1/2\), \(DE_j\) reports 1
Algorithm 5 SAMPLE($P$)

1: $j = \log(DE.\text{REPORT})$
2: if $DE_j.\text{REPORT} = 1$ then
3: $\quad$ return $UE_j.\text{REPORT}$
4: else
5: $\quad$ return $\text{FAIL}$
6: end if

iff $|\text{Supp}(x) \cap H_j^{-1}(0)| = 1$ with probability $1 - \delta'$. Thus the element reported by $UE_j$ is an element chosen uniformly at random from $|\text{Supp}(x)|$ with probability $1 - \delta'$. We also 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 $d = |\text{Supp}(x)|$. From correctness of $DE$ it follows that $d \leq 2^j \leq 4d$. Thus probability of $|S_j \cap \text{Supp}(x)| = 1$ is equal to $(d/2^j)(1 - 2^{-j}d^{-1}) \geq (1/4)\delta'$.

Further, it is shown in [10] that the assumption of hash functions being fully random can be removed by replacing them with pseudorandom hash functions.

Lemma 5. Given a sequence of update operations on a vector $x : [U] \mapsto [M]$, there is a streaming algorithm with probability $1 - 1/\delta$ returns an element $t \in \text{Supp}(x)$ such that $\Pr[r = i] = 1/|\text{Supp}(x)| \pm \delta$ for every $i \in \text{Supp}(x)$. The algorithm uses $O(\log^2(MU/\delta))$ space.

3.2 Estimating the weight of Euclidean Minimum Spanning Tree

In this section, we will show how to estimate the weight of a Euclidean minimum spanning tree in a dynamic data stream. Let the current pool of points be denoted by $\{p_1, p_2, \ldots, p_n\}$. Further let EMST denote the Euclidean Minimum Spanning Tree.

We impose $\log_{1+\epsilon}(\sqrt{d}\Delta)$ square grids over the point space. The side length of the grid cells are $\frac{\epsilon(1+\epsilon)^i}{\sqrt{d}}$ for $0 \leq i \leq \log_{1+\epsilon}(\sqrt{d}\Delta)$. The essential idea is to maintain statistics of the distribution of points on the grid. These statistics can be used to compute a $(1+\epsilon)$-approximation of the point set. Let $G_P$ denote the complete Euclidean graph of a point set $P$ and $W$ be the upperbound on the longest edge (diameter). Further, let $e_p^{((1+\epsilon)^i)}$ denote the number of connected components in $G_p^{((1+\epsilon)^i)}$, which is the subgraph containing all edges of length at most $(1+\epsilon)^i$. Under these assumptions we can use the formula from [11]

$$\frac{1}{1+\epsilon}\text{EMST} \leq n - W + \epsilon\sum_{i=0}^{\log_{1+\epsilon}W-1} (1+\epsilon)^ie_p^{((1+\epsilon)^i)} \leq \text{EMST}$$

where $n$ is the number of points in $P$. 
Instead of considering the number of connected components in $G_P(t)$ for $t = (1 + \epsilon)^t$, we first move all the points of $P$ to the centers of a grid of side length $\sqrt[2d]{\epsilon t}$. After removing multiplicities we obtain the point set $P'(t)$ we consider the graph $G(t)$ whose vertex set is $P'(t)$ and that contains an edge between two vertices if their distance is at most $t$. Instead of counting the connected components in $G_P(t)$ we count the number of connected components in $G(t)$.

It turns out that,

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

This suggests that the total error introduced by approximating $G(t)$ with $G_P(t)$ is very small.

Now let us describe the data structures required to approximate the minimum spanning tree. From the equation for the minimum spanning tree, it is sufficient if we have constant factor approximations for the values $n, W$ and $c_P((1+\epsilon)^t)$. This would result in a $1+\epsilon$ approximation for the Euclidean Minimum Spanning Tree.

**Approximating the number of points** We can maintain a counter which is incremented when a point is added and decremented when a point is deleted from the pool of points.

**Approximating the diameter** The essential idea is to estimate the diameter in each of the $d$ dimensions. The paper gives a procedure to obtain a 4 approximation in each of the $d$ dimensions. Thus we get a $4\sqrt{d}$ approximation of the diameter, which is an $O(1)$ approximation for a given dimension $d$.

**Approximating the number of components in $G(t)$** The essential idea is to use the algorithm for finding connected components from [12]. The algorithm requires us to maintain a multiset of points chosen uniformly at random. We use the dynamic sampling introduced earlier for this task.

Using these approximation techniques we obtain the following result.

**Theorem 5.** Given a sequence of insertion/deletion of points from discrete geometric space, there is a streaming algorithm that uses $O(\log(1/\delta)(\log(\Delta)/\epsilon)^{O(d)})$ space and $O(\log(1/\delta)(\log(\Delta)/\epsilon)^{O(d)})$ time for a constant $d$ for each update and computes with probability $1-\delta$, a $1+\epsilon$ approximation of the Euclidean Minimum Spanning Tree.

4 Conclusion

In this project, we have seen various standard techniques for both the insertion only and the dynamic data streaming model. Some topics which were not
Algorithm 6 APPROX-CONNECTED-COMPONENTS(P)

1: Choose s points $q_1, q_2 \ldots q_s \in P^{(t)}$ uniformly at random
2: for each $q_i$ do
3:     Choose integer $X$ according to distribution $\text{Prob}[X \geq k] = 1/k$
4:     if $X \geq D$ then
5:         $\beta_i = 0$
6:     else
7:         if Connected component of $G^{(t)}$ containing $q_i$ has at most $X$ vertices
8:             set $\beta_i = 1$
9:         else
10:            set $\beta_i = 0$
11:        end if
12:    end if
13: end for
14: return $n^t/s \sum_{i=1}^{s} \beta_i$

covered in the write-up include coresets. Coresets were covered in the course. A dynamic algorithm for maintaining coresets would readily give streaming algorithms for many problems in the insertion only model. There are coresets developed for various problems in the dynamic geometric data streaming model and has been summarized in [13]

References