Lecture 13: Streaming Algorithms for Geometric Problems

Overview

We have seen streaming algorithms which process a stream of updates to a vector \( x = (x_1, x_2, \ldots, x_n) \), where each update \( (i, c) \) is interpreted as adding \( c \) to the \( i \)-th coordinate of \( x \): \( x_i = x_i + c \). These algorithms maintain a linear sketch of the vector which is updated as the stream is processed. We have seen such algorithms that estimate the \( \ell_p \) norm of \( x \), compute the heavy hitters of \( x \), or find a sparse approximation to \( x \).

We are going to move away from this framework and start looking at algorithms that solve geometric problems. Many of these are classic computational geometry problems with the added difficulty that the set of points is described by a stream. Each stream update is the insertion (and possibly deletion) of a point from the current set, and we want to estimate the answer to some geometric problem on the current set (using sublinear space).

We describe a number of geometric problems. We will focus on the case when the points lie in \( \mathbb{R}^2 \). We will assume that the distance between any two points \( p, q \in \mathbb{R}^2 \) is defined using \( L_1 \) distance. However, the actual choice of the distance function does not matter much: in the plane, all \( L_p \) norms are equivalent up to a factor of 2, and our algorithms have super-constant approximation factors anyway.

1. Diameter

We are given a set of points \( P \), and we want to find the maximum distance between any two point of \( P \). This maximum distance is called the diameter of \( P \). In our streaming setting, we want an algorithm that is able to estimate the value of the diameter.

2. Minimum Weight Bi-chromatic Matching (MWBM)

We are given two equally-sized sets of points \( G \) and \( B \). A bi-chromatic matching is a pairing of each point in \( G \) with each point in \( B \) (we can imagine that points in \( G \) are green, and that points in \( B \) are blue). The cost (or weight) of such a matching is the sum of distances between matched points; we are interested in a matching that minimizes this sum. Note that the description of an actual matching takes linear space, so we are interested in estimating only the cost of the minimum matching.

A motivation for why the cost of the MWBM might be useful is that it can be used as a measure of similarity between the two sets of points; this measure is also called earth mover’s distance.

3. Minimum Weight Matching (MWM)

Similar to MWBM, except that we are given a single set of points \( P \), and \( |P| \) is even. We want to match every point in \( P \) with exactly one other point in \( P \). Again, we want to estimate the minimum cost of a matching (which is the sum of the distances between paired points).
4. Minimum Cost Spanning Tree (MST)

As before, the description of a spanning tree is linear, so we only want to estimate the cost of the MST of the current set of points.

A potential application of MST cost estimation is in sensor networks, when the sensor devices are mobile and a connection tree that changes dynamically must be maintained; the cost of the MST is a good measure of the size of such a tree.

5. Facility Location

We are given a set of points \( P \). The problem is to choose a set of points \( F \) (the points need not be points from \( P \)); these points are the positions of facilities we are building. Each point in \( P \) uses the nearest facility; we want to minimize the sum of the distances to nearest facility. In addition, building each facility costs a fixed amount \( f \). We thus define the cost of \( F \) as being the sum of distances to nearest facility plus \( f \cdot |F| \). We want to estimate the cost of the best possible set \( F \).

Streaming variants

The input stream to these algorithms describes a dynamic set of input points. There are two possibilities of what the stream can contain: either insertions and deletions of points, or only insertions. This distinction greatly affects the style of algorithms used to solve the problems.

Insertions and deletions

In this variant, each stream element is either an insertion or a deletion of a point from set \( P \); we estimate the solution to some geometric problem on the set \( P \), without actually storing the entire set. Usually, these algorithms need to assume that the coordinates are discrete, i.e. for \( \mathbb{R}^2 \) the points come from \( \{1 \ldots \Delta\}^2 \). Also, most of these algorithms are randomized and make use of reductions from geometric problems to numeric (vector) problems.

Insertions only

Historically, algorithms were first invented for the easier variant, where the stream contains only insertions to set \( P \). Most of the algorithms that fall in this class can assume arbitrary coordinate values; they are deterministic in nature and make use of a core-set technique, in which a smaller set of points with certain properties is maintained.

Today we will see algorithms that handle both insertions and deletions - they naturally utilize the techniques that we have seen earlier in the class. We will cover core-sets in later lectures.

Diameter problem

We show an algorithm that uses \( O(1/\varepsilon^{O(1)} \cdot \text{polylog}) \) space to obtain a \( (1 + O(\varepsilon)) \)-approximation to the diameter of \( P \), a set of points in the plane. The actual poly-logarithmic factors are not very important, as this algorithm is not the best known solution to this problem (in terms of these factors).
We assume the points in the point set $P$ are from \{1 \ldots \Delta\}^2. We impose square grids $G_0 \ldots G_k$ with side lengths $(1 + \varepsilon)^0$, $(1 + \varepsilon)^1$, $(1 + \varepsilon)^2$, \ldots, $(1 + \varepsilon)^k = \Delta$. We define a count vector for each grid: each element of the vector corresponds to a cell in the grid, and counts how many points are in that grid cell at a given time. Formally, for every $G_i$, and for every cell $c$ in the grid $G_i$, let $n^i_P(c)$ be the number of points from $P$ in that cell $c$.

The stream contains updates to set $P$. Each update is an insertion or deletion of a point $p$ to/from $P$; this corresponds to the increment or decrement of the values $n^i_P(c_i)$ for all $i$ and for $c_i$ being the cell in $G_i$ in which point $p$ lies. We already know how to maintain sketches of these vectors under such updates; we can maintain sketches over vectors $n^i_P$ to solve

1. the $k$-sparse recovery problem for $k = O(1/\varepsilon^2)$
2. the $\ell_0$ norm estimation problem, used to verify if a vector is $k$-sparse (and thus if the sketch in 1. results in exact recovery); alternatively, the sketch recovery algorithm used in 1. might already be able to tell us if the sparse recovery is exact or not.

Depending on the value of the diameter, some grid granularity will be small enough to give us a good “idea” of how the point set looks like (and hence allow for a good approximation), but also coarse enough that the bounding box of all the points intersects a small number of grid cells (and hence the count vector is sparse). If $D$ is the diameter of $P$, let $i$ be the grid level such that

$$(1 + \varepsilon)^i \leq \varepsilon D \leq (1 + \varepsilon)^{i+1}$$

Note that the maximum horizontal (and vertical) distance between any two points is at most $D$, and $D$ is at most $1/\varepsilon$ grid cell lengths. It follows that all points in $P$ are contained in a bounding box consisting of $k = O(1/\varepsilon^2)$ cells. Also note that the grid granularity is coarse enough so that if we move the points that form the diameter to opposing corners of their cells, the diameter only increases by a factor of $O(\varepsilon)$.

Of course, we do not know $i$, but we can choose the finest grid granularity which results in a $k$-sparse count vector. Let $i^*$ be the smallest value such that $\|n^i_P\|_0 \leq k$. We can find $i^*$ because we can estimate the $\ell_0$ norm of the count vectors\(^1\). We recover the set $S$ of non-zero cells in $n^i_P$.

---

\(^1\)the error in the $\ell_0$ estimation is not a problem if we set $k$ to be a little bigger than what we would otherwise need
from the sketch; we find the diameter $D_S$ of the cells, i.e. the longest distance between any two points in any two cells of $S$, which can be found in $O(k^2)$. We scale this diameter by the grid cell length and thus return the diameter of $P$ as

$$D = (1 + \varepsilon)^{i^*} D_S$$

Because $i^* \leq i$, the grid granularity is good enough so that the returned $D$ is at most within a factor of $1 + O(\varepsilon)$ from the real diameter.

Note that we can improve this algorithm in the insertions-only model: instead of maintaining all grid levels, we can start with grid $G_0$, and whenever the set of non-zero cells in the current grid becomes greater than $k$, we increase the grid size to the next level (we recover the $k$ cells and use them to initialize the sketch for the next grid level). This results in a better space bound, by a factor of $(\log \Delta)/\varepsilon$.

### Other Problems

A class of problems, like MST and Matchings, can be solved with algorithms based on the main idea of a probabilistic embedding of $\mathbb{R}^2$ into a quad-tree metric, resulting in $O(\log \Delta)$-approximations (note however that a better approximation of $1 + \varepsilon$ exists for MST). The algorithms presented here appeared in [3].

### Probabilistic Embedding

The idea of the quad-tree embedding is to create a tree whose leaves correspond to the original points and with weighted edges such that the path length inside the tree between any two leaves is (with high probability) approximately equal to the distance between the corresponding points in the original plane. If we can build such a tree, we can solve the problem of interest optimally on the tree, generating a result which is approximately optimal in the original plane.

A quad-tree can be generated in the following manner: start with a square box that is big enough to fit all the elements (say with side length $2\Delta$); this corresponds to the root $r$ of the tree. Divide this box into four equal sub-squares of side length $\Delta$, and recursively build the subtrees for these sub-squares (only for those that still contain points); the roots of these subtrees are the children of $r$, and the edges that connect them to $r$ have length (weight) equal to $\Delta$, the edge-length of the small squares. Stop when a square contains a single point, and create a leaf corresponding to that point. The height of the tree is $O(\log \Delta)$.

An alternative way to imagine this quad-tree is to use grids $G_0, G_1, G_2, \ldots$ with cell lengths $2^0, 2^1, 2^2, \ldots$. Start with grid $G_0$. Note that since the coordinates of the points are separated by at least 1, no two points can end up in the same cell of $G_0$. For each cell that contains a point, create a leaf of the tree. Continue with the next grid level. At each step, we are building a level in the tree. Suppose we created the level for $G_i$ and want to create the level for $G_{i+1}$. Create a node $n$ for each cell of $G_{i+1}$ that contains points. This cell contains exactly four cells of $G_i$. Some of these are non-empty and thus have corresponding nodes. Connect these nodes to $n$ with edges of length equal to $2^i$, the side length of a cell in $G_i$. Continue this procedure until we reach a single root.

Let the tree-distance between two points be the path length in the tree between the leaves corresponding to the points. It turns out that if the positioning of the grids (or of the initial square in the previous method) is shifted by a random vector $v \in \{1, \ldots, \Delta\}^2$, the tree-distance between
any two points is WHP similar to the real distance. First, notice that we chose the weights of tree edges so that, for two points \( p, q \)

\[ \| p - q \| \leq D_{\text{tree}}(p, q) \]

The more surprising fact is that

\[ E[D_{\text{tree}}(p, q)] \leq \| p - q \| \cdot O(\log \Delta) \]

The intuition behind this is that the tree-distance is much bigger than the real distance when two points that are close to each other happen to be separated by a high-level grid line, but this can only happen with a very small probability if the grids are shifted randomly.

More precisely, if \( p, q \) are separated in a grid of side length \( 2^i \), the tree distance is \( O(2^i) \); the probability that the points are separated in this grid is \( \frac{\| p - q \|}{2^i} \). Thus, the expected length of the part of \( D_{\text{tree}}(p, q) \) at level \( i \) is

\[ O(2^i) \cdot \frac{\| p - q \|}{2^i} = O(1) \cdot \| p - q \| \]

This holds for all \( O(\log \Delta) \) levels, thus

\[ E[D_{\text{tree}}(p, q)] \leq \| p - q \| \cdot O(\log \Delta) \]

See [2], slides 34+, for more details on this embedding. Note that the original idea of such “probabilistic embedding” is due to [1].

**MST**

Let \( T \) be the quad-tree embedding of set \( P \). Let \( T' \) be the MST of \( P \). Let \( T'' \) be the image of \( T' \) in \( T \) (each edge in \( T' \) corresponds to a path in \( T \), so \( T'' \) is the union of all these paths, removing duplicates). Note that

\[ E[\text{Cost}(T'')] = O(\log \Delta) \text{Cost}(T') \]

from the properties of the embedding. Also, if we consider that each internal node of \( T \) corresponds to the point in plane that is the center of the respective grid cell, then \( T'' \) corresponds to a Steiner

![Figure 2: A set of points and the corresponding quad-tree](image)
tree in the plane of cost at most $Cost(T'')$. Because a MST costs at most twice as much as the minimal Steiner tree,

$$Cost(T'') \geq Cost(T')/2$$

But $T''$ is a subtree of $T$; let $L$ be the smallest level of $T$ such that every node above it has exactly one child. Then $T''$ is the same as $T$ up to level $L$:

$$Cost(T'') = Cost(T \text{ up to level } L)$$

It follows that 2 times the cost of $T$ up to level $L$ is an $O(\log \Delta)$-approximation to the cost of MST.

Now recall the grid approach to the diameter problem, but replace the grids from that solution with the grids $G_0, G_1 \ldots$ from the quad-tree embedding. Similarly define the count vectors $n_i^P$ such that $n_i^P(c)$ is the number of points in set $P$ that fall in the cell $c$ of the grid $G_i$. Since each cell corresponds to a (potential) node in the corresponding quad-tree, $n_i^P(c)$ tells how many points are in the quad-subtree rooted at $c$. Thus, the level $L$ in the above formula is the smallest grid level with exactly one non-zero entry in the $n_i^P$ count vector. With this in mind, the cost of $T$ up to level $L$ is $^2$

$$\sum_{i}^{L-1} 2^i \sum_{c \in G_i} [n_i^P(c) > 0] = \sum_{i}^{L-1} 2^i \|n_i^P\|_0$$

We can maintain the $\ell_0$ norm of the count vectors and simply compute this value to find an $O(\log \Delta)$-approximation to the cost of the MST of $P$. Note that while the quad-tree embedding is the central idea, we are not using it directly in the algorithm (we do not explicitly build the tree).

**Minimum Weight Matching**

We can also solve the matching problem on the quad-tree. Note that a tree edge at some level is always twice as long as any edge at the level below. Because of this, a simple greedy solution for the matching problem is optimal: start from the bottom up and always match what is possible at the current level; odd leftovers wait to be matched at the next level. Because the edge lengths double as we go up, it is never better to leave more than one unmatched node in any subtree at the current level.

Translating the cost of this greedy matching to our count vector representation, we get that

$$Cost = \sum_{i} 2^i \sum_{c \in G_i} [n_i^P(c) \text{ is odd}]$$

**Summary**

We can use similar reasoning to solve the Min-Weight Bi-chromatic Matching and Facility Location problems. We summarize the approximation formulas below.

- **MST:**
  $$\sum_{i}^{L-1} 2^i \sum_{c \in G_i} [n_i^P(c) > 0] = \sum_{i}^{L-1} 2^i \|n_i^P\|_0$$

$^2$note that $[expr]$ is 0 when $expr$ is false and 1 when $expr$ is true
where $L$ is the smallest level with exactly one non-zero entry in $n^L_P$.

- **MWM:**
  \[
  \sum_i 2^i \sum_{c \in G_i} [n^i_P(c) \text{ is odd}]
  \]

- **MWBM:**
  \[
  \sum_i 2^i \sum_{c \in G_i} |n^i_G(c) - n^i_B(c)| = \sum_i 2^i \|n^i_G - n^i_B\|_1
  \]
  We can solve this by maintaining $\ell_1$ sketches for $n^i_G - n^i_B$.

- **Facility Location:**
  \[
  \sum_i 2^i \sum_{c \in G_i} \min[n^i_P(c), f]
  \]
  where $f$ is the cost of building a facility.

**References**

