Algorithms for Dynamic Geometric Problems over Data Streams

Piotr Indyk*
CSAIL MIT

1. INTRODUCTION

Computing over data streams is a recent phenomenon that is of growing interest in many areas of computer science, including databases, computer networks and theory of algorithms. In this scenario, it is assumed that the algorithm sees the elements of the input one-by-one in arbitrary order, and needs to compute a certain function of the input. However, it does not have enough memory to store the whole input. Therefore, it must maintain a "sketch" of the data. Designing a sketching method for a given problem is a novel and exciting challenge for algorithm design.

The initial research in streaming algorithms has focused on computing simple numerical statistics of the input, like median [23], number of distinct elements [11] or frequency moments [1]. More recently, the researchers showed that one can use those algorithms as subroutines to solve more complex problems (e.g., cf. [13]); see the survey [24] for detailed description of the past and recent developments. Still, the scope of algorithmic problems for which stream algorithms exist is not well understood. It is therefore of importance to identify new classes of problems that can be solved in this restricted settings.

In this paper we investigate stream algorithms for *dynamic geometric problems*. Specifically, we present low-storage data structures that maintain approximate solutions to geometric problems, under insertions and deletions of points (this is called a *turnstile model* in [24]). From the data stream perspective, the stream consists of m operations, each of them is either ADD(p) (that adds p to the current set) or REMOVE(p) (which removes p from the current set); the set is initially empty.

The turnstile model has been quite challenging as far as geometric streaming problems are concerned, e.g., see [24], p. 23. In this paper we present approximation algorithms for the following fundamental geometric problems defined over a set of points P in the discrete d-dimensional space $\{1 \dots \Delta\}^d$:

- Minimum Spanning Tree (MST): find a tree spanning the points in P with minimum weight[1],
- Minimum Weight Matching (MWM): assuming $|P|$ is even, find a set of $|P|/2$ node-disjoint edges that minimize the sum of the edge weights
- Facility location[2] : for a parameter $f > 0$, find a facility set $F \subset P$ that minimizes the objective $f|F| + C(F, P)$, where
$$C(F, P) = \sum_{p \in P} \min_{q \in F} \|p - q\|$$
- k-median: find a set $Q \subset \{1 \dots \Delta\}^d$ of size k to minimize $C(Q, P)$

We also point out that the result of Charikar [4] can be used to obtain an algorithm for the bi-chromatic minimum weight matching problem, where the $P = R \cup B$, and each of $|P|/2$ node-disjoint edges are required to have exactly one node in each of B and R.

For all problems but k-median, we focus on estimating the *cost* of the solution, rather than reporting the solution itself. This is because for those problems, the solution size is or could be $\Omega(|P|)$ which makes it impossible to design algorithms with sublinear storage. For the case of k-median, we output the set of medians as well as the (approximate) solution cost.

An example motivation for these problems in the streaming model is as follows (we focus on the MST problem, since the arguments hold for other problems as well). Consider a scenario in which many cheap[3] sensors are distributed over a wide area (e.g., floating in a lake, attached to wild animals

*Supported in part by NSF ITR grant CCR-0220280.

STOC'04, June 13–15, 2004, Chicago, Illinois, USA.

[1]Here and in what follows, the weight of an edge between two points $p, q \in P$ is equal to the norm of the vector $p - q$. Our algorithm works for any Minkowski norm l_p.
[2]For the facility location problem, we need to assume that the input consists of *distinct* points; i.e., no two points overlap. To avoid notational complications, we will make this assumptions for all problems.
[3]It is natural to ask why can't the sensors be equipped with at least a few MB of RAM, to be able to store the coordinates of all sensors. After all, the memory is quite inexpensive. The authors' answer to this question is: if the sensors are equipped with anything that can be reused (like large memory, fast processor, etc), then someone is likely going to reuse (i.e., steal) them. Thus, it is beneficial if the sensors have only the minimum capability needed to perform the designed task.

etc). We assume that a sensor is able to measure and transmit its position. Periodically, each sensor updates its position, by announcing its old as well as its new coordinates. To perform any kind of computation or information aggregation, the sensors must maintain an ad-hoc communication network. To keep the communication cost low, it is natural if the network topology is based on the MST of the set of sensors. However, computing MST in a distributed manner is a fairly complex operation. Thus, instead of recomputing it after every sensor move, it might be advantageous to keep a sub-optimal communication tree for some time, and recompute it only if its cost is much higher than the current MST cost. For this purpose, however, one needs to maintain the cost of MST, using only the limited resources available to the sensors.

See [24, 17] and the references therein for further motivation for problems over geometric data streams, as well as other models of geometric data streams. For more information about sensor networks, see [16].

Our results. For simplicity of exposition, we assume that the dimension d is constant. Together with the assumption of the distinctness of all elements, it implies that $\log n = O(\log \Delta)$, where n is the maximum number of elements in the set P. All of the algorithms use space that is polynomial in $\log \Delta$ (and k, for k-median) All algorithms are randomized. Other parameters of the algorithms are showed in the following table.

Although the approximation factor of $\log \Delta$ can appear fairly large (especially for problems that are solvable exactly in polynomial time), it is likely that in practice the actual quality of the reported answers will be much better. In particular, the experimental evaluation of a variant of the algorithm for the bi-chromatic matching [19] shows that the estimation provided by the algorithm is, for natural data sets, within 10% of the actual value. We also mention that the approximation factor for facility location can be improved to $O(\log \Delta)$, but then the analysis is no longer elegant, so the details are deferred to the final version.

Our techniques. On the high-level, our approach as follows. Since the dynamic geometric problems over streams are poorly understood, we show how to reduce them to problems over *high-dimensional vectors*. The latter problems have been extensively studied and many solutions are known. This yields solutions to the geometric problems.

This general idea is implemented as follows. In the first step, we impose $\log \Delta$ randomly shifted, nested square grids over the point space; the grid cells have side lengths $2^0, 2^1, 2^2 \ldots$ etc. For each grid, we compute a certain statistic of the distribution of points in the grid. Let $n_S(c)$ be the number of points in set S falling into cell c. Then:

- To estimate the cost of minimum bi-chromatic matching, we compute $\sum_c |n_B(c) - n_R(c)|$ (the guarantees for this estimator were essentially proved in [4])

- To estimate the cost of MST, we compute the *number* of cells c such that $n_P(c) > 0$

- To estimate the cost of MWM, we compute the *number* of cells c such that $n_P(c)$ is *odd*

- To estimate the cost of facility location, we compute $\sum_c \min(n_P(c), T)$, for a parameter T

- To estimate the cost of a *given* set of medians Q, we compute $\sum_{c \cap B(Q,r)=\emptyset} n_P(c)$, where $B(Q,r)$ is a union of balls of radius r around points in Q. Then we *find* an approximately optimal set of medians via greedy algorithm or local search[4].

From the above, it suffices to give algorithms for estimation of the respective statistics. Estimating $\sum_c |n_B(c) - n_R(c)|$ is equivalent to maintaining the l_1 norm of the difference vector $n_B - n_R$, a problem that has been solved in [18]. Estimating of the number of c's such that $n_P(c) > 0$ is equivalent to maintaining the number of distinct elements in a stream [11, 1] (see [8] for a more recent description of that algorithm). The problem of estimating the number of odd values of $n(c)$ (we call it ODD-COUNT) was not investigated earlier in the literature. We show how to solve it by adapting a method for dimensionality reduction in a hypercube due to [20]. The next problem (that we call BOUNDED-COUNT) was not addressed in the literature either, although it generalizes the problem of counting the non-zero entries (for $T = 1$) or counting the number of elements (for $T = \infty$). We provide a novel algorithm for this problem. Finally, the quantity needed for the k-median problem can be computed using a variant of an algorithm for the heavy-hitters problem as in [9].

Except for the last case, the approximation guarantees of our algorithms are showed by viewing the algorithms as implementations of the following two stage process:

- Perform low-distortion embedding of the metric defined over the input set P into a *probabilistic tree metric* (see Preliminaries for the definitions)

- Compute statistics over the tree metric

Although low-distortion embedding have been used earlier for the purpose of stream computation (e.g., in [18]), to our knowledge this is the first time that embeddings into probabilistic tree metrics has been used for this purpose.

Discrete geometric space. As mentioned earlier, we assume that the input points live in the discrete space $\{1 \ldots \Delta\}^d$. Such assumption is not very common in computational geometry, where typically the coordinates of points are assumed to be real. However, in practice our assumption is always satisfied, due to the bounded precision of real-life computation. Moreover, finite precision of the input is a common assumption in the area of streaming computation, since otherwise the notion of storage is not well defined (e.g., one cannot use discrete communication complexity tools to prove lower bounds). Finally, we mention that our algorithms can be easily adapted to work for data sets in $\Re^d$ in which the minimum inter-point distance is at least 1, and the diameter is bounded by Δ.

Previous work. We are not aware of any prior results on MST, MWM or bi-chromatic matching in a stream, even in the insertions-only settings. The facility location problem was considered earlier in the metric setting [21]. However, the algorithms developed in that paper were using $\Omega(|F|)$ space, which is linear in n in the worst case. Very recently, Frahling and Sohler [12] (building on the techniques of [7, 10]) gave an algorithm for $(1+\epsilon)$-approximation of the MST cost in the insertions-only model.

[4] We mention that the idea of using greedy algorithms in the data stream setting first appeared in [13].

Problem	Approximation	Update	Est.
Bi-chromatic matching	$O(d \log \Delta)$	$(\log \Delta + d)^{O(1)}$	$O(1)$
MST	$O(d \log \Delta)$	$(\log \Delta + d)^{O(1)}$	$O(1)$
Matching	$O(d \log \Delta)$	$(\log \Delta + d)^{O(1)}$	$O(1)$
Facility Location	$O(d \log^2 \Delta)$	$(\log \Delta + d)^{O(1)}$	$O(1)$
k-median (local search)	$O(1)$	$O(\log \Delta + 2^{O(d)})^{O(1)}$	$O(\Delta^d \cdot (\log \Delta + 2^{O(d)})^{O(1)})$
k-median (greedy)	$[1+\epsilon, O(\log \Delta(\log \Delta + \log(1/\epsilon)/\epsilon))]$	$O(\log \Delta + 1/\epsilon + 2^{O(d)})^{O(1)}$	$O(\log \Delta + 1/\epsilon + k)^{O(d)}$

Figure 1: Our results. The last two columns show the bounds for the update (i.e., processing the insertions or deletion of a point) and estimation (i.e., returning the desired solution or quantity). The last algorithm provides an $[a, b]$-approximation guarantee, that is the algorithm returns bk medians whose cost is at most a times the minimum cost of any k medians. The estimation time $O(1)$ means that the bound is subsumed by the update times.

The k-median problem for streaming data has been extensively investigated for the insertions-only case. In particular, low-storage $O(1)$-approximation algorithms are known [14, 6]. Very recently, Har-Peled and Mazumdar [15] discovered a low-storage $(1+\epsilon)$-approximation algorithm for this problem, for the case where the input points live in a low-dimensional $\Re^d$ space.

For a survey on the previous work on streaming problems, see [24].

2. PRELIMINARIES

To simplify expressions, we assume that $\log n = O(\log \Delta)$. As mentioned earlier, since all input points are distinct, this assumption is automatically satisfied if d is a constant.

Embedding into trees. Let D, D' be metrics on the same set X. Then D' dominates D if $D(x,y) \geq D'(x,y)$ for all $x, y \in X$. Let X be a set, $T_1, T_2, \ldots, T_l$ be metrics over that set, and $\alpha_1, \ldots, \alpha_k$ be non-negative reals summing to 1. The convex combination of the T_i (with the coefficients α_i) is the metric D given by $D(x,y) = \sum_{i=1}^{l} \alpha_i T_i(x,y)$, $x, y \in X$. A convex combination of metrics $T_1, \ldots, T_k$ on X can be thought of as a *probabilistic metric* (this concept was suggested by Karp). Namely, $D(x,y)$ is the expectation of $T_i(x,y)$ for $i \in \{1, 2, \ldots, k\}$ chosen at random according to the distribution given by the α_i.

We need also to define a hierarchically well-separated tree (k-HST). Let T be a rooted tree with nonnegative weights of edges such that (a) the distances from any node to all of its children are the same, and (b) on each path from the root to a leaf, the lengths of the consecutive edges decrease by at least a certain factor $k > 1$. An HST (with parameter k) is the metric space with the leaves of T as points and with the shortest-path metrics over T.

Building on [3], Charikar et al [5] showed the following

FACT 1. *Let P be a set of points from $\{1 \ldots \Delta\}^d$. Then the metric over P induced by the l_p norm can be embedded into a convex combination of dominating 2-HST's with distortion $O(d \log \Delta)$.*

In the following we briefly sketch how the above result was obtained[5]. We will do it by presenting a randomized procedure for generating a tree T; the metric is then defined by taking convex combination of all trees that can be generated by the procedure, with coefficients equal to the proper probabilities.

Let G_i, $i = 1 \ldots \log \Delta$, be nested square grids over $\Re^d$ with side length 2^{i-1}, shifted by a vector chosen uniformly at random from $[0, \Delta]^d$. The nodes at the zero level, i.e., leaves, correspond to the points in P. The internal nodes at any level $i \geq 1$ correspond to the non-empty grid cells in G_i. We also add a root node at level $\log \Delta + 1$. For any node v at level $i \leq \log \Delta - 1$, its parent w in T is the node corresponding to a grid cell in G_{i+1} that contains the cell in G_i (or the point in P) corresponding to v. For the nodes at level $\log \Delta$, the parent w is the root node. The weight of the edge $\{v, w\}$ is equal to 2^i. One can verify that the metric induced by T dominates the metric over P induced by any l_p norm, and that the distortion is as claimed.

3. MINIMUM SPANNING TREE

In this section we describe a $O(d \log \Delta)$-approximation stream algorithm for maintaining the weight of MST, that uses only $O(\log^{O(1)}(n + \Delta))$ bits of storage. This is done in two parts. First, we observe that the weight of a 2-HST T generated via the procedure in preliminaries approximates the weight of MST up to an (expected) factor of $O(d \log \Delta)$. Then, we show how to maintain (approximately) the weight of T in a stream fashion.

Approximating MST by an 2-HST. It is not difficult to see that (by increasing the total weight by at most a factor of 2) one can convert a 2-HST T into a tree T', such that (a) the only nodes in T' are the leaves of T, i.e., the points in P, and (b) the metric induced by T' dominates the metric induced by T (when restricted to points in P). If we replace the cost of all edges $\{v, w\}$ in T' by the actual distance $\|v - w\|_p$ (forming T'') the weight of the tree is only going to decrease, while the metric induced by T'' still dominates the metric over P induced by $\|\cdot\|_p$.

The tree T'' is a spanning tree of P. It suffices to bound its weight. Since the weight of T'' is larger than the weight of T by only a constant factor, it suffices to show the following:

CLAIM 1. *The expected weight of T is at most $C = O(d \log \Delta)$*

[5]Strictly speaking, the latter paper employed a somewhat different approach. We spare the reader the detailed account of the differences.

times the weight of the MST T^*.

Proof: The proof follows through the following sequence of observations:

- For each edge $\{v, w\} \in T^*$, the expected cost of the corresponding path $P(v, w)$ from v to w in T is at most C times its cost in T^*
- Thus, the expected total weight of all edges in $E = \cup_{\{v,w\}\in T^*} P(v, w)$ is at most C times the cost of T^*
- Since the graph (P, E) is connected, E contains all edges in T.

The claim follows.

□

Approximating the weight of the HST. By the above argument, it suffices to maintain (approximately) the weight of the 2-HST T. Since we cannot maintain T explicitly in small memory, we employ the following approach. Let n_i, $i \geq 1$, be the number of non-empty grid cells in G_i (or equivalently, the number of level-i nodes in T); we set $n_0 = |P|$.

CLAIM 2. *The weight of T is equal to*

$$\sum_{i\geq 0} 2^i n_i$$

Therefore, it suffices to estimate the quantities n_i for $i = 1 \dots \log \Delta$. This is done as follows. For any $p \in P$, let $G_i(p)$ be the cell in G_i containing p. The problem of estimating n_i is now equivalent to the problem of estimating (up to a constant factor) the number of distinct elements in the stream consisting of all cells $G_i(p)$, $p \in P$. By the result of [11, 1], the latter task can be performed using $O(\log \Delta^d)$ bits. Therefore, the total space used by our algorithm is $O(d \log^2 \Delta)$.

THEOREM 1. *There is a $O(d \log \Delta)$ -approximation algorithm for computing the weight of MST of a stream of points in $\Re^d$ under any l_p norm. The algorithm uses $O(d \log^2 \Delta)$ bits of storage.*

4. MATCHING

In this section we describe a $O(d \log \Delta)$-approximation stream algorithm for maintaining the weight of the minimum weight matching (MWM) of points in P, that uses only $O(\log^{O(1)}(n + \Delta))$ bits of storage.

Consider a 2-HST T over P. By a similar argument as in the case of MST, it suffices to estimate the cost of MWM of all points in P with respect to the metric induced by T. To this end, let m_i be the number of grid cells in G_i that contain an odd number of points in P.

CLAIM 3. *The cost of MWM of P with respect to T is equal to*

$$n + \sum_{i\geq 1} 2^i m_i$$

Proof: We start from the lower bound. The term n comes from the cost of connecting leaves to the nodes at level 1. Let v be any internal vertex in T with an odd number of leaves in the subtree $T(v)$ rooted at v. Let $i \geq 1$ be the level of v. One of the leaves of $T(v)$ must be matched with a node that does not belong to the subtree $T(v)$. This will add 2^i to the total matching cost. Moreover, the costs induced by different nodes v do not overlap. The fact follows.

The upper bound is showed using a similar idea. Consider a matching of points in P created using the following greedy approach. First, we match (as many as we can) points p, q, such that the shortest path between p and q in T passes only through levels 0 and 1. The matched nodes are removed. Then we match points such that the shortest path between them passes through levels up to 2. We continue by increasing the upper bound on the level to 3, 4 etc. The total number of yet-unmatched points at level i is equal to m_i. The bound follows. □

Thus, it suffices to estimate the values m_i, for $i = 1 \dots \log \Delta$. For each i, this problem reduces to the following one: design a data structure maintaining vector $x[1 \dots M]$, that supports the following operations:

- UPDATE(i, c): perform $x[i] = x[i] + c$
- OC: report the number $OC(x)$ of positions i such that $x[i]$ is odd

We call this an ODD-COUNT problem. In the following, we give an $O(1)$-approximation[6] algorithm for this problem that uses $O(\log M) = O(\log \Delta)$ bits, and has constant probability of success. By running $O(\log \log \Delta)$ copies of the procedure in parallel, we can reduce the probability of failure to $\frac{1}{2 \log \Delta}$. This will ensure that the estimation of all m_i's is correct with constant probability. Overall, this will result in a $O(d \log \Delta)$-approximation algorithm for the MWM estimation that uses $O(d \log^2 \Delta \cdot \log \log \Delta)$ bits.

In order to solve ODD-COUNT, we show how to solve its (approximate) *decision* version. The latter problem is parametrized by $T \in \{1 \dots n\}$. An algorithm solves that problem if the following two conditions are satisfied:

1. If $OC(x) > T$, then the algorithm outputs YES with probability $> 1/3$
2. If $OC(x) < T/10$, then the algorithm outputs YES with probability $< 1/10$

Below show how to solve this problem using only *one* bit of storage. To be more exact, we will also use $O(\log M) = O(d \log \Delta)$ bits of randomness that needs to be stored. However, the same randomness can be reused for each of the $O(\log^2 \Delta)$ instances of problem, and thus the total storage bound remains unchanged.

The algorithm that solves the decision version of ODD-COUNT is as follows:

- To initiate, choose a random set $R \subset \{1 \dots M\}$, such that each $i \in \{1 \dots M\}$ belongs to R with probability $1/T$. Also, set $s = 0$
- To perform UPDATE(i, c): if $i \in R$, then $s = s + c \bmod 2$
- To determine $OC(x)$: if $s = 1$, answer YES; otherwise answer NO

[6] We mention that by using more careful analysis, one can obtain a $(1 + \epsilon)$-approximation algorithm for the problem. The storage would increase by a factor of $O(1/\epsilon^2)$.

Clearly, the above algorithm uses only one bit of random access storage. Although it appears to use M bits of randomness as well, they can be generating via pseudorandom generator that uses only $O(\log M)$ truly random bits, as described in [18].

CLAIM 4. *The above algorithm satisfies the conditions (1) and (2) of the decision version of the* ODD-COUNT *problem.*

Proof: Let $m = OC(x)$. Firstly, we consider condition (2). If $m < T/10$, then the probability that any of the m odd-count elements belongs to R is at most $T/10 \cdot 1/T = 1/10$. In the complementary case, the algorithm will output NO. Thus condition (2) is satisfied.

To analyze (1), observe that, without loss of generality, we can assume that each element in $\{1 \ldots M\}$ occurs either 1 or 0 times in S; this is due to the fact that the behavior of the algorithm depends only on the parity of the element occurrences, not the actual number. The analysis then can be completed by performing calculations identical to those given in [20] in proof of Lemma 1, where they were used to analyze the dimensionality reduction algorithm in a Hamming cube. □

THEOREM 2. *There is a $O(d \log \Delta)$ -approximation algorithm for computing the weight of minimum weight matching of a stream of points in $\Re^d$ under any l_p norm. The algorithm uses $O(d \log^2 \Delta \log\log \Delta)$ bits of storage.*

5. FACILITY LOCATION

In this section we describe a $O(d \log^2 \Delta)$-approximation stream algorithm for maintaining the cost an optimal facility location solution to points in P, that uses only $O(\log^{O(1)}(n+\Delta))$ bits of storage.

As before, we focus on showing how to solve this problem in metrics induced by 2-HST's, and then describe how to implement that algorithm in a stream fashion. Let T be a 2-HST. We will assume that F can contain not only the leaves but also the internal nodes of T; this changes the cost by at most a factor of 2.

Let T_i be the set of nodes of T at level i. Let $|T(v)|$ be the number of leaves of the subtree $T(v)$ of T rooted at v. Our algorithm relies on the following claim.

CLAIM 5. *Let C be the minimum facility location cost, and let*

$$Q = \sum_{i=1}^{\log \Delta} \sum_{v \in T_i} \min(|T(v)| 2^i, f)$$

Then $C \le Q + f \le C \log \Delta + f$

Proof: We start from proving the first inequality, by constructing a set F that has the cost at most $Q + f$. The facilities are placed as follows:

- One facility is placed at the root
- For each node $v \in T_i$, if $|T(v)| 2^i \ge f$, then place one facility at v

The cost of the solution is as claimed.

To show the second inequality, it suffices to show that for each level i

$$\sum_{v \in T_i} \min(|T(v)| 2^i, f) \le C$$

Let F be the facility set inducing cost C. Consider any node $v \in T_i$. If $T(v)$ contains an element from F, then it adds f to the total cost. Otherwise, if $T(v)$ does not contain a facility, then all $|T(v)|$ nodes must connect to a facility outside of $T(v)$, which induces a cost of at least $2^i |T(v)|$.

□.

Thus, in order to estimate the optimal cost C, it suffices to estimate the quantity Q. This can be done by solving $\log \Delta$ instances of the following BOUNDED-COUNT problem. The goal of the problem is to design a data structure that maintains a set S of pairs (i, j), where $i \in \{1 \ldots M\}, j \in \{1 \ldots M'\}$, under insertions and deletions. It is assumed that at any time, there can be at most one pair $(i, j) \in S$ with a given value of j. Let $c_S(i)$ be the number of pairs $(i, j) \in S$. The data structure needs to support the following operations:

- ADD(i, j): adds (i, j) to S
- REMOVE(i, j): removes (i, j) from S
- BC: report the number $BC(S) = \sum_i \min(c_S(i), T)$, for an input parameter T

The data structure for facility location invokes the BOUNDED COUNT data structure by performing addition and removal of pairs (c, p), where $p \in \{1 \ldots \Delta^d\}$ is a point added to or removed from P, and c is the grid cell (at a proper level) that p belongs to. Note that, to satisfy the constraints of BOUNDED COUNT, it is crucial that all points in P are unique, i.e., there are no duplicates in P.

We will give an approximation algorithm for this problem. To this end, observe first that we can assume without loss of generality that $BC(S) \ge T$. The reason for that is as follows. As long as $n \le T$, we have $BC(S) = n$, and thus the problem can be solved exactly. Once $n > T$, then $BC(S) \ge T$.

In what follows we focus on the case $BC(S) \ge T$. The problem is then solved (approximately) by the following algorithm. Let $h : \{1 \ldots M'\} \to \{1 \ldots T\}$ be a random hash function, i.e., such that all values of $h(i)$ are independently chosen from $\{1 \ldots T\}$. As in the previous section, the randomness (and storage) needed to create h can be reduced by using pseudorandom generator, as in [18].

Our data structure uses another data structure (that we call DE) for maintaining the number of distinct elements in a stream, under insertions and deletions. Our implementation is as follows:

- To perform ADD(i, j): if $h(j) = 0$, add i to DE
- To perform REMOVE(i, j): if $h(j) = 0$, remove i from DE
- To estimate $BC(S)$: invoke DE to estimate the number K of distinct elements, and output $T \cdot K$ as an estimation of $BC(S)$.

The algorithm clearly can be implemented to use only $O(\log M)$ space, if we do not count the random bits required by h The pseudorandom generator needed to construct h

requires additional $O(\log M \cdot \log M)$ bits of space. It remains to show that $T \cdot K$ is a good estimator of $BC(S)$. For this purpose, we show the following.

CLAIM 6. *Let $p(k) = 1-(1-1/T)^k$ be the probability that, given k elements, if we remove each of them independently with probability $1-1/T$, then at least one element remains. Then*

$$E[K] = \sum_i p(c_S(i))$$

CLAIM 7. *For any $k \geq 0$, we have*

$$\min(k,T)/2 \leq Tp(k) \leq \min(k,T)$$

Proof: The second inequality follows from that $p(k) \leq 1$, and an easy "union bound" inequality $p(k) \leq k/T$. For the first inequality, we first observe that $p(k)$ is concave for $k \geq 0$. Moreover, $p(0) = 0$ and $p(T) = 1-(1-1/T)^T > 1-1/e > 1/2$. Thus, the graph of $p(k)$ lies above the line from $(0,0)$ to $(T,1/2)$. □.

From the last two claims, it follows that it suffices to show that K is a good estimator of $E[K]$, i.e., that with constant probability K is within constant factor away from $E[K]$. For this purpose, we will bound the variance $V(K)$. As long as $\sqrt{V(K)} = O(E[K])$, using standard sampling techniques (e.g., see [1]) guarantees that with constant probability K will estimate $E[K]$ up to a constant factor. To this end, observe that

$$V(K) = \sum_i p(c_S(i)) - p(c_S(i))^2 \leq \sum_a p(c_S(i)) = E[K]$$

If $BC(S) \geq T$, then $V(K) = \Omega(1)$. In this case, $\sqrt{V(K)} = O(E[K])$, and thus the above algorithm estimates $E[K]$ (and thus $BC(S)$) up to a constant factor with constant probability.

THEOREM 3. *There is a $O(d \log^2 \Delta)$ -approximation algorithm for computing the minimum facility location cost for a stream of points in $\Re^d$ under any l_p norm. The algorithm uses $O(d^2 \log^2 \Delta)$ bits of storage.*

6. K-MEDIAN

For simplicity we focus on the case $d = 2$, that is, the input points live in a discrete plane. The algorithms easily generalize to higher dimensions.

6.1 Tools

Exclusive Count. Our first tool is a low-space data structure that solve the $(1+\epsilon)$-approximate EXCLUSIVE-COUNT (XCOUNT) problem. The data structure maintains a vector $x[1 \dots M]$, which is initially set to 0. It is required to support the following operations:

- UPDATE(i,c): perform $x[i] = x[i] + c$
- XCount(Q): return a value R such that $(1-\epsilon)XC \leq R \leq XC$, where $XC = \|x\|_1 - \|x_{|Q}\|_1$.

We will parametrize the data structure by t; the resulting data structure *XCount*(t) assumes that $|Q| \leq t$ for all XCount queries. In addition, we require that $x \geq 0$ at all times.

Our solution to this problem uses hashing technique akin to *Min-Count sketches of* [9]. It consists of:

- A pair-wise independent hash function $h : \{1 \dots M\} \rightarrow \{1 \dots l\}$, for $l = 2t(1+1/\epsilon)$
- An array $A[1 \dots l]$, initialized to 0

The operations are implemented as follows:

- Update(i,c): we perform $A[h(i)] = A[h(i)] + c$
- XCount(Q): return $R = \sum_{i \notin h(Q)} A[i]$

LEMMA 1. *Any XCount query reports a correct (approximate) value with probability at least $1/2$.*

Proof: Clearly $R \leq XC$. Moreover, $XC = R + b$, where $b = \sum_{p \notin Q, h(p) \in h(Q)} x_p$. We have $E[b] \leq \epsilon/2 \cdot XC$. By Markov inequality we have $\Pr[b > 2E[b] = \epsilon XC] \leq 1/2$. □.

Median Cost Evaluation. Our main tool is the *Median Cost Evaluation (MediEval)* data structure. The data structure maintains a set $P \subset U = \{1 \dots \Delta\}^2$, under addition and deletion of points (as for the k-median). In addition, it supports an operation EVAL(Q), that returns C such that $C = (1 \pm O(\epsilon))C(Q,P)$. The data structure is parametrized by l; the resulting data structure MEDIEVAL(l) assumes that $|Q| \leq l$ for all EVAL queries.

The data structure is implemented as follows. We define G_i to be a square grid imposed on U, with side length $\epsilon/\sqrt{2} \cdot (1+\epsilon)^i$. We use $G_i(p)$ to denote a grid cell that contains p, and C_i to denote the set of all cells of G_i. For any cell $c \in C_i$, define $n_i(c)$ to be the number of points in P that fall into c.

Our data structure maintains data structures XC_i that solve the XCount(t) problem for vectors n_i. We set $t = \Theta(l/\epsilon^2)$. Clearly, the Add/Delete operations on MediEval structure naturally translate into operations on XC_i's.

We now show how to implement the EVAL(Q) operation. Let $B(p,r)$ be the set of points in U with distance less than r from p. Let $B(Q,r) = \cup_{q \in Q} B(q,r)$. Let $r_i = (1+\epsilon)^i$, for $-\infty < i < \infty$. Let i_0 be such that $r_{-i_0} < \epsilon$. Note that $i_0 = O(\log(1/\epsilon)/\epsilon)$.

Observe that for any i

$$B(Q,r_i) \subset G_i(B(Q,r_i)) \subset B(Q,r_{i+1})$$

Note that $|G_i(B(Q,r_i))| = O(l/\epsilon^2) \leq t$. Let $\hat{R}_i(Q)$ be the value of XCount for $G_i(B(Q,r_i))$, and let $R_i(Q)$ be the approximation of $\hat{R}_i(Q)$ returned by XC_i. Recall that $R_i(Q) \geq (1-\epsilon)\hat{R}_i(Q)$, and that

$$|P - B(Q, r_i(1+\epsilon))| \leq \hat{R}_i(Q) \leq |P - B(Q,r_i)|$$

Define $\hat{C}(Q,P) = \sum_{-i_0 \leq i \leq \log(2\Delta)} (r_i - r_{i-1})\hat{R}_i(Q)$.

LEMMA 2. *The quantity $\hat{C}(Q,P)$ provides a good approximation of $C(Q,P)$. That is*

$$C(Q,P) = (1 \pm O(\epsilon))\hat{C}(Q,P)$$

Proof: We will show the $\leq$ inequality, the other direction is similar.

$$
\begin{aligned}
C(Q,P) &= \int_0^\infty |P - B(Q,r)| dr \\
&\le \sum_i |P - B(Q, r_{i+1})|(r_{i+2} - r_{i+1}) \\
&\le 1/(1-\epsilon) \cdot \sum_{i \ge -i_0} |P - B(Q, r_{i+1})|(r_{i+2} - r_{i+1}) \\
&\le \frac{(1+\epsilon)^2}{1-\epsilon} \cdot \sum_{i \ge -i_0} |P - B(Q, r_{i+1})|(r_i - r_{i-1}) \\
&\le \frac{(1+\epsilon)^2}{(1-\epsilon)} \sum_{i \ge -i_0} \hat{R}_i(Q)(r_i - r_{i-1}) \\
&= \frac{(1+\epsilon)^2}{(1-\epsilon)} \sum_{-i_0 \le i \le \log(2\Delta)} \hat{R}_i(Q)(r_i - r_{i-1})
\end{aligned}
$$

□

6.2 Algorithms

The MEDIEVAL data structure can be used in several ways to construct a data structure for bi-criterion approximation of the k-median.

Exhaustive Search. If we set the probability of failure of the data structure to $\frac{1}{2\Delta^{2k}}$, then, with probability 1/2, the data structure will correctly estimate the value of $C(Q,P)$ for any $Q \subset \{1\ldots\Delta\}^2$. Thus, one can retrieve a $(1+\epsilon)$-approximate solution to the k-median problem for P by enumerating all sets Q and choosing the best one. This takes time $\Delta^{O(k)}$. Although the query time is prohibitively expensive, this shows that, in principle, it is possible to get $(1+\epsilon)$-approximate solution to the k-median problem in small space.

Local search. In the paper [2], the authors showed the following fact. Consider an algorithm that starts with arbitrary set Q of size k. The algorithm proceeds via a sequence of steps. At any step, it enumerates all sets $Q' = Q - \{q\} \cup \{p\}$ for $q \in Q, p \notin Q$. If $C(Q',P) < (1-\alpha)C(Q,P)$, then Q becomes Q'. The algorithm ends when there is no Q' satisfying the above condition. Arya et al show that for the case $\alpha = 0$, the algorithm provides a 5-approximation. It is not difficult to observe that even if $\alpha > 0$, then the algorithm provides a $(5+\delta)$-approximation for $\delta = \delta(\alpha, k)$ [22].

The above algorithm requires only an oracle that maintains the approximate value of $C(Q,P)$ for given Q. This is precisely what MEDIEVAL does. Thus, we obtain an algorithm for $(5+\delta)$-approximate k-median. Note that the time needed to compute the k medians is proportional to $|U|$, i.e., is $\Omega(\Delta^2)$.

THEOREM 4. *There is an $O(1)$-approximation algorithm for the dynamic k-median problem on a stream, where the input points live in $\{1\ldots\Delta\}^2$. The algorithm uses space $O(k \log^{O(1)} \Delta)$. The update time given a new point is $O(k(\log\Delta + 1/\epsilon)^{O(1)})$. The time needed to report the solution is $O(\Delta^2 k(\log\Delta + 1/\epsilon)^{O(1)})$.*

Greedy algorithm. Instead of the local search, one can apply the greedy approach. The greedy algorithm proceeds in the following way. Initially, $Q = \emptyset$. Then, in each step, it chooses $q \notin Q$ such that $C(Q \cup q, P)$ is the smallest, and adds q to Q. Since the function $C(Q) = C(Q,P)$ is supermodular, the result of [25] implies that the resulting algorithm constructs a set Q' consisting of $O(k\log\Delta)$ medians such that $C(Q',P) \le C(Q,P)$.

Again, we can use MEDIEVAL to implement the greedy procedure. If we use $\hat{C}(\cdot,\cdot)$ instead of $C(\cdot,\cdot)$, by the same argument we are guaranteed that $C(Q',P) \le (1+O(\epsilon))C(Q,P)$. This gives us a bicriterion $[O(\log\Delta), 1+\epsilon]$-approximate algorithm for the k-median problem. The time to construct the medians is still $\Omega(\Delta^2)$.

To obtain a faster algorithm, consider the formula for $\hat{C}(P,Q)$. It consists of $s = \log\Delta + O(\log(1/\epsilon)/\epsilon)$ terms of the form $a_i \hat{R}_i(Q)$ where $a_i = (r_i - r_{i-1})$. By properly setting the accuracy of the XC_i data structure, we can assume that $\hat{R}(Q) \approx R_i(Q)$, i.e., that we can interchange these terms without loss of correctness (note that accuracy inversely polynomial in $1/\epsilon + \log\Delta$ suffices). Assume that there is a $q^* \notin Q$ such that $\hat{C}(P, Q \cup \{q^*\}) < (1-\alpha)\hat{C}(P,Q)$. This implies that there exists i such that $t_i(Q) - t_i(Q \cup \{q^*\}) > \alpha/s \cdot \hat{C}(P,Q)$. Our algorithm will search for q such that $t_i(Q) - t_i(Q \cup \{q\}) \ge \frac{1}{2}(t_i(Q) - t_i(Q \cup \{q^*\}))$. $\hat{C}(P, Q \cup \{q^*\}) \le (1-\alpha/s)\hat{C}(P,Q)$. This leads to a bicriterion $[1+\epsilon, O(\log\Delta(\log\Delta + \log(1/\epsilon)/\epsilon))]$-approximation algorithm.

The point q can be found as follows. For a cell $c \in G_i$, let $n_i(c)$ be the number of points in P that belong to c. Let B be a set of cells in $G_i(B(q^*, r_i)) - G_i(B(Q, r_i))$. We have $\sum_{c\in B} n_i(c) = t_i(Q) - t_i(Q \cup \{q^*\}) = T$. Let $B' \subset B$ be the set of cells c such that $n_i(c) \ge \frac{T}{2|B|}$. Clearly, $\sum_{c\in B'} n_i(c) \ge T/2$.

The algorithm proceeds by finding a set S of *all* cells c for which $n_i(c) \ge \frac{T}{2|B|}$. Then, it enumerates all points q such that $G_i(B(q, r_i))$ intersects S, and returns q that maximizes $\sum_{c \in S \cap [G_i(B(q,r_i)) - G_i(B(Q,r_i))]} n_i(c)$. It follows that the resulting point q satisfies the aforementioned constraints.

It remains to bound the space and time complexity of the above procedure. The set S can be found in time and space polynomial in $(|S| + 1/\epsilon + k + \log\Delta)$ - it is a variant of a *heavy hitters problem*, investigated e.g., in [9]. The size of $|S|$ is at most $O(s/\alpha)$ times $O(1/\epsilon^2)$. Finally, the number of possible sets $G_i(B(q, r_i))$ that intersects S is polynomial as well.

THEOREM 5. *There is a $[1+\epsilon, O(\log\Delta(\log\Delta + \log(1/\epsilon)/\epsilon))]$-approximation algorithm for the dynamic k-median problem on a stream, where the input points live in $\{1\ldots\Delta\}^2$. The algorithm uses space $O((k + \log\Delta + 1/\epsilon)^{O(1)})$. The update time given a new point is $O(k(\log\Delta + 1/\epsilon)^{O(1)})$. The time needed to report the solution is $O(k + \log\Delta + 1/\epsilon)^{O(1)}$*

Acknowledgments: The author would like to thank Graham Cormode, Sudipto Guha and Muthu Muthukrishnan, for helpful discussions.

7. REFERENCES

[1] N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. *Proceedings of the Symposium on Theory of Computing*, pages 20–29, 1996.

[2] V. Arya, N. Garg, R. Khandekar, A. Meyerson, K. Munagala, and V. Pandit. Local search heuristics for k-median and facility location problems. *Proceedings of the Symposium on Theory of Computing*, 2002.

[3] Y. Bartal. Probabilistic approximation of metric spaces and its algorithmic applications. *Proceedings of the Symposium on Foundations of Computer Science*, 1996.

[4] M. Charikar. Similarity estimation techniques from rounding. *Proceedings of the Symposium on Theory of Computing*, 2002.

[5] M. Charikar, C. Chekuri, A. Goel, S. Guha, and S. Plotkin. Approximating a finite metric by a small number of tree metrics. *Proceedings of the Symposium on Foundations of Computer Science*, 1998.

[6] M. Charikar, L. O'Callaghan, and R. Panigrahy. Better streaming algorithms for clustering problems. *Proceedings of the Symposium on Theory of Computing*, pages 30–39, 2003.

[7] B. Chazelle, R. Rubinfeld, and L. Trevisan. Approximating the minimum spanning tree weight in sublinear time. *ICALP*, 2001.

[8] G. Cormode, M. Datar, P. Indyk, and S. Muthukrishnan. Comparing data streams using hamming norms. *Proceedings of the International Conference on Very Large Databases (VLDB)*, 2002.

[9] G. Cormode and S. Muthukrishnan. Improved data stream summaries: The count-min sketch and its applications. *DIMACS Tech Report*, 2003.

[10] A. Czumaj and C. Sohler. Estimating the weight of metric minimum spanning trees in sublinear-time. *Proceedings of the Symposium on Theory of Computing*, 2004.

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

[12] G. Frahling and C. Sohler. Estimating the weight of euclidean minimum spanning trees in data streams. *Manuscript*, 2004.

[13] A. Gilbert, S. Guha, Y. Kotidis, P. Indyk, M. Muthukrishnan, and M. Strauss. Fast, small-space algorithms for approximate histogram maintenance. *Proceedings of the Symposium on Theory of Computing*, 2002.

[14] S. Guha, N. Mishra, R. Motwani, and L. O'Callaghan. Clustering data streams. *Proceedings of the Symposium on Theory of Computing*, 2001.

[15] S. Har-Peled and S. Mazumdar. Coresets for k-means and k-medians and their applications. *Proceedings of the Symposium on Theory of Computing*, 2004.

[16] J. M. Hellerstein, S. Madden, and W. Hong. The sensor spectrum: technology, trends and requirements. *SIGMOD Record*, December, 2003.

[17] M. Hoffman, S. Muthukrishnan, and R. Raman. Location streams: Models and algorithms. *manuscript*, 2004.

[18] P. Indyk. Stable distributions, pseudorandom generators, embeddings and data stream computation. *Proceedings of the Symposium on Foundations of Computer Science*, 2000.

[19] P. Indyk and N. Thaper. Fast color image retrieval via embeddings. *Workshop on Statistical and Computational Theories of Vision (at ICCV)*, 2003.

[20] E. Kushilevitz, R. Ostrovsky, and Y. Rabani. Efficient search for approximate nearest neighbor in high dimensional spaces. *Proceedings of the Thirtieth ACM Symposium on Theory of Computing*, pages 614–623, 1998.

[21] Adam Meyerson. Online facility location. *Proceedings of the Symposium on Foundations of Computer Science*, pages 426–431, 2001.

[22] K. Munagala. Personal communication. 2003.

[23] J. I. Munro and M. S. Paterson. Selection and sorting with limited storage. *TCS*, 12, 1980.

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

[25] L. A. Wolsey. An analysis of the greedy algorithm for the submodular set covering problem. *Combinatorica*, 2(4):385–393, 1982.