Data Stream Algorithms

Notes from a series of lectures by

S. Muthu Muthukrishnan

Guest Lecturer: Andrew McGregor

The 2009 Barbados Workshop on Computational Complexity
March 1st – March 8th, 2009

Organizer:
Denis Thérien

Scribes:
Anıl Ada, Eric Allender, Arkadev Chattopadhyay, Matei David, Laszlo Egri, Faith Ellen, Ricard Gavaldà, Valentine Kabanets, Antonina Kolokolova, Michal Koucký, Franois Lemieux, Pierre McKenzie, Phuong Nguyen, Toniann Pitassi, Kenneth Regan, Nicole Schweikardt, Luc Segoufin, Pascal Tesson, Thomas Thierauf, Jacobo Torán.

Lecture 1. Data Streams

Lecturer: S. Muthu Muthukrishnan Scribes: Anıl Ada and Jacobo Torán

We start with a puzzle.
Puzzle 1: Given an array $A[1..n]$ of $\log n$ bit integers, sort them in place in $O(n)$ time.

1.1 Motivation

The algorithms we are going to describe act on massive data that arrive rapidly and cannot be stored. These algorithms work in few passes over the data and use limited space (less than linear in the input size). We start with three real life scenarios motivating the use of such algorithms.

Example 1: Telephone call. Every time a cell-phone makes a call to another phone, several calls between switches are being made until the connection can be established. Every switch writes a record for the call over approx. 1000 Bytes. Since a switch can receive up to 500 million calls a day, this adds up to something like 1 Terabyte per month information. This is a massive amount of information but has to be analyzed for different purposes. An example is searching for drop calls trying to find out under what circumstances such drop calls happen. It is clear that for dealing with this problem we do not want to work with all the data, but just want to filter with a few passes the useful information.

Example 2: The Internet. The Internet is made of a network of routers connected to each other, receiving and sending IP packets. Each IP packet contains a packet log including its source and destination addresses as well as other information that is used by the router to decide which link to take for sending it. The packet headers have to be processed at the rate at which they flow through the router. Each package takes about 8 nanoseconds to go through a router and modern routers can handle a few million packets per second. Keeping the whole information would need more than one Terabyte information per day and router. Statistical analysis of the traffic through the router can be done, but this has to be performed on line at nearly real time.

Example 3: Web Search. Consider a company for placing publicity in the Web. Such a company has to analyze different possibilities trying to maximize for example the number of clicks they would get by placing an add for a certain price. For this they would have to analyze large amounts of data including information on web pages, numbers of page visitors, add prices and so on. Even if the company keeps a copy of the whole net, the analysis has to be done very rapidly since this information is continuously changing.

Before we move on, here is another puzzle.
Puzzle 2: Suppose there are n chairs around a circular table that are labelled from 0 to $n-1$ in order. So chair i is in between chairs $i-1$ and $i+1$ mod n. There are two infinitely smart players

that play the following game. Initially Player 1 is sitting on chair 0. The game proceeds in rounds. In each round Player 1 chooses a number i from $\{1, 2, \ldots, n-1\}$, and then Player 2 chooses a direction left or right. Player 1 moves in that direction i steps and sits on the corresponding chair. Player 1's goal is to sit on as many different chairs as possible while Player 2 is trying to minimize this quantity. Let $f(n)$ denote the maximum number of different chairs that Player 1 can sit on. What is $f(n)$?

Here are the solutions for some special cases.

$$\begin{aligned} f(2) &= 2 \\ f(3) &= 2 \\ f(4) &= 4 \\ f(5) &= 4 \\ f(7) &= 6 \\ f(8) &= 8 \\ f(p) &= p-1 \quad \text{for } p \text{ prime} \\ f(2^k) &= 2^k \end{aligned}$$

1.2 Count-Min Sketch

In this section we study a concrete data streaming question. Suppose there are n items and let $F[1..n]$ be an array of size n. Index i of the array will correspond to item i. Initially all entries of F are 0. At each point in time, either an item i is added, in which case we increment $F[i]$ by one, or an item is deleted, in which case we decrement $F[i]$ by one. Thus, $F[i]$ equals the number of copies of i in the data, or in other words, the frequency of i. We assume $F[i] \geq 0$ at all times.

As items are being added and deleted, we only have $O(\log n)$ space to work with, i.e. logarithmic in the space required to represent F explicitly. Here we think of the entries of F as words and we count space in terms of number of words.

We would like to estimate $F[i]$ at any given time. Our algorithm will be in terms of two parameters ϵ and δ. With $1-\delta$ probability, we want the error to be within a factor of ϵ.

The algorithm is as follows. Pick $\log(\frac{1}{\delta})$ hash functions $h_j : [n] \to [e/\epsilon]$ chosen uniformly at random from a family of pair-wise independent hash functions. We think of $h_j(i)$ as a bucket for i corresponding to the jth hash function. We keep a counter for each bucket, count$(j, h_j(i))$. Initially all buckets are empty, or all counters are set to 0. Whenever an item i is inserted, we increment count$(j, h_j(i))$ by 1 for all j. Whenever an item i is deleted, we decrement count$(j, h_j(i))$ by 1 for all j (see Figure 1.1). Our estimation for $F[i]$, denoted by $\tilde{F}[i]$, will be $\min_j$ count$(j, h_j(i))$.

Claim 1. *Let* $\|F\| = \sum_i F[i]$.

1. $\tilde{F}[i] \geq F[i]$.
2. $\tilde{F}[i] \leq F[i] + \epsilon\|F\|$ *with probability at least* $1-\delta$.

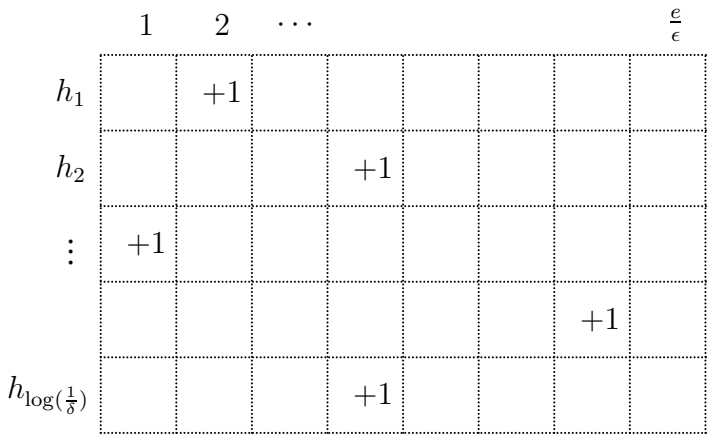

Figure 1.1: Each item is hashed to one cell in each row.

Proof. The first part is clear. For the second part, denote by X_{ji} the contribution of items other than i to the $(j, h_j(i))$th cell (bucket) of Figure 1.2. It can be easily shown that

$$\mathbf{E}[X_{ji}] = \frac{\epsilon}{e}\|F\|.$$

Then by Markov's inequality,

$$\begin{aligned}\Pr\left\{\tilde{F} > F[i] + \epsilon\|F\|\right\} &= \Pr\left\{\forall j\ F[i] + X_{ji} > F[i] + \epsilon\|F\|\right\} \\ &= \Pr\left\{\forall j\ X_{ji} > e\mathbf{E}[X_{ji}]\right\} \\ &\leq \left(\frac{1}{2}\right)^{\log 1/\delta}\end{aligned}$$

□

Thus, we conclude that we can estimate $F[i]$ within an error of $\epsilon\|F\|$ with probability at least $1 - \delta$ using $O((1/\epsilon)\log(1/\delta))$ space. Observe that this method is not effective for estimating $F[i]$ for small $F[i]$. On the other hand, in most applications, one is interested in estimating the frequency of high frequency objects.

It is possible to show that the above is tight, with respect to the space requirements, using a reduction from the communication complexity problem Index. In this problem, Alice holds an n bit string $x \in \{0,1\}^n$ and Bob holds a $\log n$ bit string $y \in \{0,1\}^{\log n}$. We view Bob's input as an integer $i \in [n]$. We consider the one-way probabilistic communication model. Therefore only Alice is allowed to send Bob information. Given the information from Alice, Bob needs to determine the value x_i. In this model, it is well known that Alice needs to send $\Omega(n)$ bits in order for Bob to determine x_i with constant probability greater than 1/2.

Lemma 1. *In order to estimate $F[i]$ within an error of $\epsilon\|F\|$ with constant probability, one needs to use $\Omega(1/\epsilon)$ space.*

Proof. Given an instance of the Index problem (x, y), where x denotes Alice's input, y denotes Bob's input and $|x| = n$, choose ϵ such that $n = \frac{1}{2\epsilon}$. Construct the array $F[0..\frac{1}{2\epsilon}]$ as follows. If $x_i = 1$ then set $F[i] = 2$ and if $x_i = 0$ then set $F[i] = 0$ and increment $F[0]$ by 2 (initially $F[0] = 0$). With this construction, clearly we have $\|F\| = 1/\epsilon$. Suppose we can estimate $F[i]$ within error $\epsilon\|F\| = 1$ with constant probability and s space. This means we can determine the value of x_i: if the estimate for $F[i]$ is above 1 then $x_i = 1$ and $x_i = 0$ otherwise. Now the $\Omega(n) = \Omega(1/\epsilon)$ lower bound on the communication complexity of Index implies a lower bound of $\Omega(1/\epsilon)$ for s. □

Homework: Given a data stream as an array $A[1..n]$, how can we estimate $\sum_i A[i]^2$? If we are given another data stream $B[1..n]$, how can we estimate $\sum_i A[i]B[i]$?

References: [CM04], [Mut09]

Lecture 2. Streaming Algorithms via Sampling

Lecturer: S. Muthu Muthukrishnan Scribes: Faith Ellen and Toniann Pitassi

2.1 Estimating the number of distinct elements

This lecture presents another technique for streaming algorithms, based on sampling.

Definition 1. *Let $a_1, a_2, \ldots, a_n$ denote a stream of items from some finite universe $[1..m]$. Let $D_n = |\{a_1, a_2, \ldots, a_n\}|$ be the number of distinct elements in the stream, and let U_n be the number of unique items in the stream, i.e. the number of items that occur exactly once.*

Let F be the frequency vector, where $F[i]$ is the number of times that item i occurs in the stream, for each $i \in [1..m]$. Then D_n is the number of nonzero entries in the frequency vector F and U_n is the number of entries of F with value 1.

Our goal is to get good estimates for U_n/D_n and D_n.

2.1.1 Estimating U_n/D_n

First we will try to estimate U_n/D_n.

We assume that n is known. We can easily choose an item uniformly at random from the stream, by choosing each item with probability $1/n$. Doing this k times in parallel gives us a sample of size k. The problem with this approach (uniform sampling from the data stream) is that heavy items, i.e. those with high frequency, are likely to appear many times. Since each such item doesn't contribute to U_n and only contributes to D_n once, such a sample is not helpful for estimating U_n/D_n.

Instead, we would like to be able to sample nearly uniformly from the set of (distinct) items in the stream, i.e. element x is chosen with probability close to $1/D_n$.

To do this, let h be a permutation of the universe chosen uniformly at random from among all such permutations. The idea is to remember the item s in the stream with the smallest value of $h(s)$ seen so far and the number of times this item has occurred. Specifically, as we see each item a_i in the stream, we compute $h(a_i)$. If $h(a_i) < h(s)$ (or $i = 1$), then s is set to a_i and $c(s)$ is set to 1. If $h(a_i) = h(s)$, then increment $c(s)$. If $h(a_i) > h(s)$, then do nothing. Note that, for any subset S of the universe, each item in S is equally likely to be mapped to the smallest value by h among all the elements in S. In particular, each element in the set of items in the stream has probability $1/D_n$ of being chosen (i.e. mapped to the smallest value by h) and, thus, being the value of s at the end of the stream. At any point in the stream, $c(s)$ is the number of times s has occurred so far in the stream, since we start counting it from its first occurrence.

Doing this k times independently in parallel gives us a collection of samples $s_1, \ldots, s_k$ of size k. We will choose $k = O(\log(1/\delta)/\epsilon^2)$. Let $c_1, \ldots, c_k$ be the number of times each of these

items occurs in the stream. Our estimate for U_n/D_n will be $\#\{i \mid c_i = 1\}/k$. Since Prob$[c_i = 1] = U_n/D_n$, using Chernoff bounds it can be shown that with probability at least $(1-\delta)$, $(1-\epsilon)U_n/D_n \leq \#\{i \mid c_i = 1\}/k \leq (1+\epsilon)U_n/D_n$. Thus, $\#\{i \mid c_i = 1\}/k$ is a good estimator for U_n/D_n.

It's not necessary to have chosen a random permutation from the set of all permutations. In fact, simply storing the chosen permutation takes too much space. It suffices to randomly choose a hash function from a family of functions such that, for any subset of the universe, every element in the subset S has the smallest hashed value for the same fraction $(1/|S|)$ of the functions in the family. This is called minwise hashing and it was defined in a paper by Broder, et al. They proved that any family of hash functions with the minwise property must be very large.

Indyk observed that an approximate version of the minwise property is sufficient. Specifically, for any subset S of the universe, each element in the subset has the smallest hashed value for at least a fraction $1/((1+\epsilon)|S|)$ of the functions in the family. There is a family of approximately minwise hash functions of size $n^{O(\log n)}$, so $(\log n)^2$ bits are sufficient to specify a function from this family.

An application for estimating U_n/D_n comes from identifying distributed denial of service attacks. One way these occur is when an adversary opens many connections in a network, but only sends a small number of packets on each. At any point in time, there are legitimately some connections on which only a small number of packets have been sent, for example, for newly opened connections. However, if the connections on which only a small number of packets have been sent is a large fraction of all connections, it is likely a distributed denial of service attack has occurred.

2.1.2 Estimating D_n

Now we want to estimate D_n, the number of distinct elements in $a_1, \ldots, a_n \in [1..m]$. Suppose we could determine, for any number t, whether $D_n < t$. To get an approximation to within a factor of 2, we could estimate D_n by determining whether $D_n < 2^i$ for all $i = 1, \ldots, \log_2 m$. Specifically, we estimate D_n by 2^k, where $k = \min\{i | c_i = 0\}$. If we do these tests in parallel, the time and space both increase by a factor of $\log_2 m$.

To determine whether $D_n < t$, randomly pick a hash function h from $[1..m]$ to $[1..t]$. Let c be the number of items that hash to bucket 1. We'll say that $D_n < t$ if $c = 0$ and say that $D_n \geq t$ if $c > 0$. To record this as we process the stream requires a single bit that tells us whether c is 0 or greater than 0. Specifically, for each item a_i, if $h(a_i) = 1$, then we set this bit to 1.

If $D_n < t$, then the probability that no items in the stream hash to bucket 1 (i.e. that $c = 0$) is $(1-1/t)^{D_n} > (1-1/t)^t \approx 1/e$. If $D_n > 2t$, then the probability no items in the stream hash to bucket 1 (i.e. that $c = 0$) is $(1-1/t)^{D_n} < (1-1/t)^{2t} \approx 1/e^2$. More precisely, using a Taylor series approximation, $Pr[c = 0 | D_n \geq (1+\epsilon)t] \leq 1/e - \epsilon/3$ and $Pr[c = 0 | D_n < (1-\epsilon)t] \geq 1/e + \epsilon/3$.

To improve the probability of being correct, repeat this several times in parallel and take majority answer. This give the following result.

Theorem 1. *It is possible to get an estimate t for D_n using $O[(1/\epsilon^2)\log(1/\delta)\log m]$ words of space such that Prob$[(1-\epsilon)t \leq D_n < (1+\epsilon)t] \geq 1-\delta$.*

2.2 Extensions for inserts and deletes

Exercise Extend the algorithm for approximating the number of distinct items to allow the stream to include item deletions as well as item insertions.

The algorithm described above for sampling nearly uniformly from the set of (distinct) items in the stream doesn't extend as easily to allow deletions. The problem is that if all occurrences of the item with the minimum hash value are deleted at some point in the stream, we need to replace that item with another item. However, information about other items that have appeared in the stream and the number of times each has occurred has been thrown away. For example, suppose in our sampling procedure all of the samples that we obtain happen to be items that are inserted but then later deleted. These samples will clearly be useless for estimating the quantities of interest.

We'll use a new trick that uses sums in addition to counts. Choose $\log_2 m$ hash functions $h_j : [1..m]$ to $[1..2^j]$, for $j = 1, \ldots, \log_2 m$. For the multiset of items described by the current prefix of the stream, we will maintain the following information, for each $j \in [1.. \log_2 m]$:

1. D'_j, which is an approximation to the number of distinct items that are mapped to location 1 by h_j,
2. S_j, which is the exact sum of all items that are mapped to location 1 by h_j, and
3. C_j, which is the exact number of items that are mapped to location 1 by h_j.

For each item a_i in the stream, if $h_j(a_i) = 1$, then C_j is incremented or decremented and a_i is added to or subtracted from S_j, depending on whether a_i is being inserted or deleted.

The number of distinct elements is dynamic: at some point in the stream it could be large and then later on it could be small. Thus, we have $\log_2 m$ hash functions and maintain the information for all of them.

If there is a single distinct item in the current multiset that is mapped to location 1 by h_j, then S_j/C_j is the identity of this item. Notice that, because S_j and C_j are maintained exactly, this works even if the number of distinct items in the current multiset is very large and later becomes 1.

Suppose that D'_j is always bounded below and above by $(1 - \epsilon)$ and $(1 + \epsilon)$ times the number of distinct items hashed to location 1 by h_j, respectively, for some constant $\epsilon < 1$. Then there is only 1 distinct item hashed to location 1 by h_j, if and only if $D'_j = 1$.

If $D'_j = 1$, then S_j/C_j can be returned as the sample. If there is no j such that $D_j = 1$, then no sample is output. If the hash functions are chosen randomly (from a good set of hash functions), then each distinct item is output with approximately equal probability.

Instead of getting just one sample, for many applications, it is better to repeat this $(1/\epsilon^2) \log(1/\delta)$ times in parallel, using independently chosen hash functions. We'll call this the sampling data structure.

Yesterday, we had an array $F[1..m]$ keeping track of the number of occurrences of each of the possible items in the universe $[1..m]$. We calculated the heavy hitters (i.e. items i whose number of occurrences, $F[i]$, is at least some constant fraction of the total number of occurrences, $\sum_{i=1}^{m} F[i]$) and estimated $F[i]$, $\sum_{i=1}^{m} F[i]$, $\sum_{i=1}^{m} F[i]^2$, and quantiles. Today, we estimated the number of distinct elements, i.e., $\#\{i \mid F(i) > 0\}$. The following definition gives a more succinct array

for answering many of the questions that we've looked at so far (i.e., distinct elements, quantiles, number of heavy hitters.)

Definition 2. *Let $I[1..k]$ be an array, where $I[j]$ is the number of items that appear j times, i.e. the number of items with frequency j, and $k \leq n$ is the maximum number of times an item can occur. For example, $I[1]$ is the number of unique items, items that appear exactly once. Heavy hitters are items that have frequency at least $\phi \sum_{i=1}^{k} I[i]$, for some constant ϕ.*

We'd like to apply the CM sketch directly to the I array. The problem is how to update I as we see each successive item in the stream. If we know how many times this item has previously been seen, we could decrement that entry of I and increment the following entry. However, we don't know how to compute this directly from I.

The sampling data structure as described above, which can be maintained as items are added and deleted, allows the entries of the I array to be approximated.

2.3 Homework Problems

1. A (directed or undirected) graph with n vertices and $m < n^2$ distinct edges is presented as a stream. Each item of the stream is an edge, i.e. a pair of vertices (i, j). Each edge may occur any number of times in the stream. Edge deletions do not occur. Let d_i be the number of distinct neighbors of vertex i. The goal is to approximate $M_2 = \sum_i d_i^2$. It is called M_2 since it is analogous to F_2 from yesterday. The key difference is that M_2 only counts a new item if it is distinct, i.e. it hasn't appeared before.

   The best known algorithm for this problem uses space $O((1/\epsilon^4)\sqrt{n}\log n)$. It can be obtained by combining two sketches, for example, the CM sketch and minwise hashing. (In general, the mixing and matching of different data structures can be useful.) The solution to this problem doesn't depend on the input being a graph. The problem can be viewed as an array of values, where each input increments two array entries.

   Although the space bound is sublinear in n, we would like to use only $(\log n)^{O(1)}$ space. This is open.

2. **Sliding window version of sampling**:
   Input a sequence of items, with no deletions. Maintain a sample uniformly chosen from among the set of distinct items in the last w items. The space used should be $O(\log w)$.

   Note that if minwise hashing is used and the last copy of the current item with minimum hashed value is about to leave the window, a new item will need to be chosen.

Lecture 3. Some Applications of CM-Sketch

Lecturer: S. Muthu Muthukrishnan Scribes: Arkadev Chattopadhyay and Michal Koucký

3.1 Count-Min Sketch

Prelude: *Muthu is a big fan of movies. What we will see today is like the movie "The Usual Suspects" with Kevin Spacey: 12 years of research fit into one sketch. It will also take some characteristics of another great movie "Fireworks" by Takashi Beat Kitano. That movie has three threads which in the end meet. This lecture will have three threads.*

Problem 1 (from yesterday): Sort an array $A[1,\ldots,n]$ of $\log_2 n$-bit integers in place in linear time.

Solution idea: With a bit of extra space, say $O(\sqrt{n})$, one could run $\sqrt{n}$-way radix sort to sort the array in $O(n)$ time. Where do we get this extra space? Sort/re-arrange the elements according to the highest order bit. Now, we can save a bit per element by representing the highest order bit implicitly. This yields $O(n/\log n)$ space to run the radix sort. The details are left to the reader. There are also other solutions.

Problem 2: We have a stream of items from the universe $\{1,\ldots,n\}$ and we want to keep a count $F[x]$ of every single item x. We relax the problem so that we do not have to provide a precise count but only some approximation $\tilde{F}[x]$:

$$F[x] \leq \tilde{F}[x] \leq F[x] + \epsilon \sum_{i=1}^{n} F[i].$$

Solution: For t that will be picked later, let $q_1,\ldots,q_t$ are the first t primes. Hence, $q_t \approx t \ln t$. We will keep t arrays of counters $F_j[1,\ldots,q_j]$, $j=1,\ldots,t$. All the counters will be set to zero at beginning and whenever an item x arrives we will increment all counters $F_j[x \mod q_j]$ by one. Define $\tilde{F}[x] = \min_{j=1,\ldots,t} F_j[x \mod q_j]$.

Claim 2. *For any $x \in \{1,\ldots,n\}$,*

$$F[x] \leq \tilde{F}[x] \leq F[x] + \frac{\log_2 n}{t} \sum_{i=1}^{n} F[i].$$

Proof. The first inequality is trivial. For the second one note that for any $x' \in \{1,\ldots,n\}$, $x' \neq x$, $x' \mod q_j = x \mod q_j$ for at most $\log_2 n$ different j's. This is implied by Chinese Reminder Theorem. Hence, at most $\log_2 n$ counters corresponding to x may get incremented as a result of an arrival of x'. Since this is true for all $x' \neq x$, the counters corresponding to x may get over-counted

by at most $\log_2 n \cdot \sum_{x' \in \{1,\ldots,n\} \setminus \{x\}} F[x']$ in total. On average they get over-counted by at most $\frac{\log_2 n}{t} \cdot \sum_{x' \in \{1,\ldots,n\} \setminus \{x\}} F[x']$, so there must be at least one of the counters corresponding to x that gets over-counted by no more than this number. □

We choose $t = \frac{\log_2 n}{\epsilon}$. This implies that we will use space $O(t^2 \log t) = O(\frac{\log^2 n}{\epsilon^2} \log\log n)$, where we measure the space in counters. This data structure is called Count-Min Sketch (CM sketch) and was introduced in *G. Cormode, S. Muthukrishnan: An improved data stream summary: the count-min sketch and its applications. J. Algorithms 55(1):58-75 (2005).* It is actually used in Sprinter routers.

Intermezzo: *Al Pacino's second movie: Godfather (depending on how you count).*

Problem 3: We have two independent streams of elements from $\{1,\ldots,n\}$. Call the frequency (count) of items in one of them $A[1,\ldots,n]$ and $B[1,\ldots,n]$ in the other one. Estimate $X = \sum_{i=1}^{n} A[i] \cdot B[i]$ with additive error $\epsilon \cdot ||A||_1 \cdot ||B||_1$.

Solution: Again we use CM sketch for each of the streams: $T^A = (T_j^A[1,\ldots,q_j])_{j=1,\ldots,t}$ and $T^B = (T_j^B[1,\ldots,q_j])_{j=1,\ldots,t}$, and we output estimate

$$\tilde{X} = \min_{j=1,\ldots,t} \sum_{k=1} T_j^A[k] \cdot T_j^B[k].$$

Claim 3.

$$X \leq \tilde{X} \leq X + \frac{\log_2 n}{t} ||A||_1 \cdot ||B||_1.$$

Proof. The first inequality is trivial. For the second one note again that for any $x, x' \in \{1,\ldots,n\}$, $x' \neq x$, $x' \mod q_j = x \mod q_j$ for at most $\log_2 n$ different j's. This means that the term $A[x] \cdot B[x']$ contributes only to at most $\log_2 n$ of the sums $\sum_{k=1} T_j^A[k] \cdot T_j^B[k]$. Hence again, the total over-estimate is bounded by $\log_2 n \cdot ||A||_1 \cdot ||B||_1$ and the average one by $\frac{\log_2 n}{t} ||A||_1 \cdot ||B||_1$. Clearly, there must be some j for which the over-estimate is at most the average. □

Choosing $t = \frac{\log_2 n}{\epsilon}$ gives the required accuracy of the estimate and requires space $O(\frac{\log^2 n}{\epsilon^2} \log\log n)$.

Intermezzo: *Mario is not happy: for vectors $A = B = (1/n, 1/n, \ldots, 1/n)$ the error is really large compare to the actual value. Well, the sketch works well for vectors concentrated on few elements.*

Problem 4: A single stream $A[1,\ldots,n]$ of elements from $\{1,\ldots,n\}$. Estimate $F_2 = \sum_{i=1}^{n} (A[i])^2$.

Solution: The previous problem provides a solution with an additive error $\epsilon ||A||_1^2$. We can do better. So far, our CM sketch was deterministic, based on arithmetic modulo primes. In general one can take hash functions $h_1,\ldots,h_t : \{1,\ldots,n\} \to \{1,\ldots,w\}$ and keep a set of counters $T_j[1,\ldots,w], j = 1,\ldots,t$. On arrival of item x one increments counters $T_j[h_j(x)]$, $j = 1,\ldots,t$. The hash functions h_j are picked at random from some suitable family of hash functions. In such a case one wants to guarantee that for a given stream of data, the estimates derived from this CM sketch are good with high probability over the random choice of the hash functions. For the problem of estimating F_2 we will use a family of four-wise independent hash functions. Our sketch

will consists of counters $T_j[1,\ldots,w]$, $j = 1,\ldots,t$, for even w. To estimate F_2 we calculate for each j

$$Y_j = \sum_{k=1}^{w/2} (T_j[2k-1] - T_j[2k])^2,$$

and we output the median $\tilde{X}$ of Y_j's.

Claim 4. *For $t = O(\ln \frac{1}{\delta})$ and $w = \frac{1}{\epsilon^2}$,*

$$|\tilde{X} - F_2| \leq \epsilon F_2$$

with probability at least $1 - \delta$.

Proof. Fix $j \in \{1,\ldots,t\}$. First observe that

$$E[Y_j] = F_2.$$

To see this let us look at the contribution of terms $A[x] \cdot A[y]$ for $x, y \in \{1,\ldots,n\}$ to the expected value of Y_j. Let us define a random variable $f_{x,y}$ so that for $x \neq y$, $f_{x,y} = 2$ if $h_j(x) = h_j(y)$, $f_{x,y} = -2$ if $h_j(x) = 2k = h_j(y) + 1$ or $h_j(y) = 2k = h_j(x) + 1$ for some k, and $f_{x,y} = 0$ otherwise. For $x = y$, $f_{x,y} = 1$ always. Notice for $x \neq y$, $f_{x,y} = 2$ with probability $1/w$ and also $f_{x,y} = -2$ with probability $1/w$. It is straightforward to verify that $Y_j = \sum_{x,y} f_{x,y} A[x] \cdot A[y]$. Clearly, if $x \neq y$ then $E[f_{x,y}] = 0$. By linearity of expectation,

$$E[Y_j] = \sum_{x,y} E[f_{x,y}] \cdot A[x] \cdot A[y] = F_2.$$

Now we show that

$$Var[Y_j] \leq \frac{8}{w} F_2^2.$$

$$\begin{aligned} Var[Y_j] &= E\left[\left(\sum_{x,y} f_{x,y} A[x] \cdot A[y] - \sum_x A[x] \cdot A[x]\right)^2\right] \\ &= E\left[\left(\sum_{x \neq y} f_{x,y} A[x] \cdot A[y]\right)^2\right] \\ &= E\left[\sum_{x \neq y, x' \neq y'} f_{x,y} \cdot f_{x',y'} \cdot A[x] \cdot A[y] \cdot A[x'] \cdot A[y']\right]. \end{aligned}$$

For $(x,y) \neq (x',y'), x \neq y, x' \neq y'$

$$E\left[f_{x,y} \cdot f_{x',y'} \cdot A[x] \cdot A[y] \cdot A[x'] \cdot A[y']\right] = 0$$

because of the four-wise independence of h_j. For $(x, y) = (x', y'), x \neq y, x' \neq y'$

$$\begin{aligned} E\left[f_{x,y} \cdot f_{x',y'} \cdot A[x] \cdot A[y] \cdot A[x'] \cdot A[y']\right] &= \left(\frac{1}{w} \cdot 4 + \frac{1}{w} \cdot 4\right) A[x]^2 \cdot A[y]^2 \\ &= \frac{8}{w} \cdot A[x]^2 \cdot A[y]^2. \end{aligned}$$

Hence,

$$Var[Y_j] \leq \frac{8}{w}\left(\cdot \sum_x A[x]^2\right)^2 = \frac{8}{w} F_2^2.$$

Applying Chebyshev's inequality, and the fact that $w = \frac{1}{\epsilon^2}$ we get,

$$\Pr\left[|Y_j - F_2| \geq \epsilon F_2\right] \leq \frac{1}{8}$$

Since each hash function is chosen independently, we can apply Chernoff bound to conclude that taking $O(\log(1/\delta))$ hash functions is enough to guarantee that the median of the Y_j's gives an approximation of F_2 with additive error less than ϵF_2 with probability at least $1 - \delta$.

□

Definition: Let $A[1, \ldots, n]$ be the count of items in a stream. For a constant $\phi < 1$, item i is called a *ϕ-heavy hitter* if $A[i] \geq \phi \sum_{j=1}^{n} A[j]$.

Problem 5: Find all ϕ-heavy hitters of a stream.

Solution: First we describe a procedure that finds all ϕ-heavy hitters given access to any sketching method. In this method, we form $\log n$ streams $B_0, \ldots B_{\log n - 1}$ in the following way:

$$B_i[j] = \sum_{k=(j-1)2^i+1}^{j2^i} A[k]$$

This means that $B_i[j] = B_{i-1}[2j-1] + B_{i-1}[2j]$. When a new element arrives in stream A, we update simultaneously the sketch of each B_i. Finally, in order to find ϕ-heavy hitters of A, we do a binary search by making hierarchical point queries on the $\log n$ streams that we created, in the following way: we start at $B_{\log n - 1}$. We query $B_{\log n-1}[1]$ and $B_{\log n-1}[2]$. If $B_{\log n-1}[1] \geq \phi \sum_{k=1}^{n} A[k] = T$ (say), then we recursively check the two next level nodes $B_{\log n-2}[1]$ and $B_{\log n-2}[2]$ and so on.

In other words, the recursive procedure is simply the following: if $B_i[j] \geq T$, then descend into $B_i[2j-1]$ and $B_i[2j]$. If $B_i[j] < T$, then this path of recursion is terminated. If $i = 0$, and $B_i[j] \geq T$, then we have found a heavy hitter.

Clearly, this procedure finds all heavy hitters if the point queries worked correctly. The number of queries it makes can be calculated in the following way: for each i, B_i can have at most $1/\phi$ heavy hitters and the algorithm queries at most twice the number of heavy-hitters of a stream. Thus, at most $(2 \log n/\phi)$ point queries are made.

If we implement the probabilistic version of CM-sketch, as described in the solution to Problem 4 above, it is not hard to see that each point-query can be made to return an answer with positive additive error bounded by ϵ, with probability $1-\delta$, by using roughly $\log(1/\delta)$ pairwise[1] independent hash functions, where each hash function has about $O(1/\epsilon)$ hash values. Such a sketch uses $O\big(\frac{1}{\epsilon}\log(1/\delta)\big)$ space.

For the application to our recursive scheme here for finding heavy hitters, we want that with probability at least $(1-\delta)$, none of the at most $(2\log n/\phi)$ queries fail[2]. Thus, using probabilistic CM-sketch with space $O\big(\frac{1}{\epsilon}\log n \log\big(\frac{2\log n}{\phi\delta}\big)\big)$ and probability $(1-\delta)$, we identify all ϕ-heavy hitters and not return any element whose count is less than $(\phi-\epsilon)$-fraction of the total count.

Reference: [CM04]

[1]Note for making point queries we just need pairwise independence as opposed to 4-wise independence used for estimating the second moment in the solution to Problem 4 before.

[2]A query fails if it returns a value with additive error more than an ϵ-fraction of the total count.

Lecture 4. Graph and Geometry Problems in the Stream Model

Lecturer: Andrew McGregor Scribes: Matei David and François Lemieux

In the lectures so far, we considered numerical data streams. In this lecture, we consider streams describing graphs and metric spaces. A *graph stream* is a stream of edges $E = \{e_1, e_2, \ldots, e_m\}$ describing a graph G on n vertices. A *geometric stream* is a stream of points $X = \{p_1, p_2, \ldots, p_m\}$ from some metric space (χ, d). We're now interested in estimating properties of G or X, e.g., diameter, shortest paths, maximal matchings, convex hulls.

This study is motivated by practical questions, e.g., edges in the graph can be pairs of IP addresses or phone numbers that communicate. In general, m is the number of edges. Unless stated otherwise, we'll assume each edge appears only once in the stream. We're interested in both directed and undirected graphs. We're using $\tilde{\mathcal{O}}$ in our bounds, hiding dependence on polylogarithmic terms in m and n. Further, we assume single points can be stored in $\tilde{\mathcal{O}}(1)$ space and that the distance $d(p_1, p_2)$ can be computed easily if both p_1 and p_2 are stored in memory.

The specific problems we consider are *counting the number of triangles in a graph* (Section 4.1), *computing a matching in a graph* (Section 4.2), *clustering points* (Section 4.3), and *computing graph distances* (Section 4.4).

4.1 Counting Triangles

The Problem. Let T_3 denote the number of triangles in a graph G. When G is presented as a stream of edges, we're interested in estimating T_3 up to a factor of $(1+\epsilon)$ with probability $1-\delta$, given the promise that $T_3 > t$ for some t.

Warmup. We start with a simple algorithm using $\tilde{\mathcal{O}}(\epsilon^{-2}(n^3/t)\log\delta^{-1})$ space. Note, this only improves on keeping all edges in $O(n^2)$ space when $t = \omega(n)$.

1. pick $1/\epsilon^2$ triples $(u_1, v_1, w_1), (u_2, v_2, w_2), \ldots$
2. as edges stream by, check that all 3 edges in every triple are present
3. estimate T_3 by the number of triples for which we found all 3 edges
4. repeat steps 1-3 for $\log\delta^{-1}$ times (in parallel), output the average of the estimates

Note that the probability (u_i, v_i, w_i) is a triangle in G is precisely $T_3/\binom{n}{3}$. Standard Chernoff bounds yield the desired correctness bounds.

Theorem 2. *To determine whether $T_3 > 0$, $\tilde{\Omega}(n^2)$ space is required, even for randomized algorithms.*

Proof. We give a reduction from 2-player Set-Disjointness: Alice and Bob have a $n \times n$ matrices A, B, and they are trying to determine if $\exists i, j$ such that $A(i,j) = B(i,j) = 1$. By [Raz92], this requires $\Omega(n^2)$ bits even for protocols that are correct with probability $3/4$.

Suppose there is a space s algorithm for determining if $T_3 > 0$. Let G be a graph on $3n$ vertices, with $V = \{u_1, \ldots, u_n, v_1, \ldots, v_n, w_1, \ldots, w_n\}$, and initial edges $E = \{(u_i, v_i) : i \in [n]\}$. Alice adds edges $\{(u_i, w_j) : A(i,j) = 1\}$, and Bob adds edges $\{(v_i, w_j) : B(i,j) = 1\}$. Alice starts simulating the algorithm until it processes the initial edges and her own, then communicates the memory of the algorithm to Bob, using s bits. He continues the simulation, eventually obtains the output of the algorithm, and announces it using one more bit.

For correctness, observe that G contains a triangle (i.e., $T_3 > 0$) iff the inputs to the protocol intersect. □

Observe that the lower bound works even for algorithms that are allowed several passes over the input stream.

Theorem 3 (Sivakumar et. al)**.** *There is an algorithm using space $\tilde{\mathcal{O}}(\epsilon^{-2}(nm/t)^2 \log \delta^{-1})$.*

Proof. The algorithm reduces this problem to that of computing frequency moments of a related stream. Given the graph stream σ, construct a new stream σ' as follows: for every edge (u, v), generate all triples (u, v, w) for $w \in V \setminus \{u, v\}$.

Denote by T_i the number of triples in V for which exactly i edges are present in G. Observe that the k-th frequency moment of σ' is

$$F_k(\sigma') = \sum_{(u,v,w)} (\#(u,v,w))^k = 1 \cdot T_1 + 2^k \cdot T_2 + 3^k \cdot T_3,$$

and that

$$T_3 = F_0 - \frac{3}{2} \cdot F_1 + \frac{1}{2} \cdot F_2.$$

Hence, good approximations for F_0, F_1, F_2 suffice to give an approximation for T_3. □

Theorem 4 (Buriol et. al)**.** *There is an algorithm using space $\tilde{\mathcal{O}}(\epsilon^{-2}(nm/t) \log \delta^{-1})$.*

Proof. We can obtain a better algorithm using the following idea.

1. pick an edge $e_i = (u, v)$ uniformly at random from the stream
2. pick w uniformly at random from $V \setminus \{u, v\}$
3. if $e_j = (u, w)$ and $e_k = (v, w)$ for $j, k > i$ exist, return 1, else return 0

To obtain an algorithm, we run this basic test many times in parallel, and we output the average of these runs, scaled by a certain amount. □

4.2 Maximum Weight Matching

The Problem. We now consider the Maximum Weight Matching problem: Given a stream of weighted edges (e, w_e), find $M \subset E$ that maximizes $\sum_{e \in M} w_e$ such that no two edes in M share an endpoint.

Warmup. Let's us first find a 2-approximation for the unweighted case using only $\tilde{O}(n)$ space. Given each edge (e, w_e) from the stream, we must decide if we add it to our current matching. For , we consider all previously choosen edges that share an end point with (e, w_e) and we compute the sum v of their weights. If $w_e > v$ then we remove these edges from M and replace them with (e, w_e). It is a simple exercice to show that the weight OPT of the optimal solution is at most twice the weight of any maximal matching.

We will sketch the proof of the following result from [McG05]:

Theorem 5. *There is a $3+\sqrt{2}$-approximation algorithm for the Maximal Weight Matching problem that uses $\tilde{O}(n)$ space .*

Before giving the algorithm, we mention that result has been improved by a series of recent results: 5.59... [Zel08] and 5.24... [Sar09] and that it is an open question to prove a lower bound or a much better result. Let γ be some parameter. The algorithm is the following:

- At all time, we maintain a matching M
- On seeing an edge (e, w_e), suppose that $e' \in M$ and (maybe) $e'' \in M$ have a common end point with e
- If $w_e \geq (1+\gamma)(w_{e'} + w_{e''})$ then replace e' and e'' by e in M.

For the analysis, we use the following (macabre) definitions to describe the execution of the algorithm:

- An edge e *kills* and edge e' if e' was removed from current matching when e arrived.
- We say an edge is a *survivor* if it is in the final matching.
- For survivor e, the *trail of the deads* is $T(e) = C_1 \cup C_1 \cup \cdots$, where $C_0 = e$ and

$$C_i = \bigcup_{e' \in C_{i-1}} \{\text{edges killed by } e'\}$$

For any set of edges S we define $w(S) = \sum_{e \in S} w_e$, where w_e is the weight of the edge e

Lemma 2. *Let S be the set of survivors and $w(S)$ be the weight of the final matching.*

1. $w(T(S)) \leq w(S)/\gamma$
2. *OPT* $\leq (1+\gamma)(w(T(S)) + 2w(S))$

Put together this give OPT $\leq (1/\gamma + 3 + 2\gamma)w(S)$ and $\gamma = 1/\sqrt{2}$ gives Theorem 5.

Proof. 1. Observe first that the $T(e)$ are disjoints. Hence, it suffices to observe that for each $e \in S$ we have:

$$(1+\gamma)w(T(e)) = \sum_{i \geq 1} (1+\gamma)w(C_i) = w(T(e) + w_e)$$

2. We can charge the weights of edges in OPT to $S \cup T(S)$ such that each edge $e \in T(S)$ is charged at most $(1+\gamma)w(e)$ and each edge $e \in S$ is charged at most $2(1+\gamma)w(e)$. More details are given in [FKM$^+$05].

□

4.3 K-Center Clustering

Due to the lack of time, the topics discussed in this section and the next one have only been sketched.

The Problem. We are given an integer k, and a stream of n distinct points $X = (p_1, p_2, \ldots, p_n)$ from a metric space (χ, d). We need to find a set of k points $Y \subseteq X$ that minimizes $\max_i \min_{y \in Y} d(p_i, y)$. Since we need to output k points, we consider the case where we have $\Omega(k)$ memory to store them.

Warmup. The standard Greedy algorithm for this problem works in small space, and it obtains a 2-approximation if given the optimal value, OPT: set radius to $2 \cdot OPT$, then pick as a centre any node which is not covered by the previous centres.

If only given bounds $a \leq OPT \leq b$ on the optimal radius, one can obtain a $(2+\epsilon)$ approximation algorithm by running the original algorithm in parallel with several values for OPT: $a, (1+\epsilon)a, (1+\epsilon)^2 a, \ldots, b$. This requires space $\tilde{\mathcal{O}}(k \log_{(1+\epsilon)}(b/a))$, which is not good when b/a is large.

Theorem 6. *[MK08a, Guh09] There exists a $(2+\epsilon)$-approximation algorithm using space $\tilde{\mathcal{O}}(k\epsilon^{-1} \log \epsilon^{-1})$.*

4.4 Distance Estimation in a Graph

The Problem. We are given a stream with the (unweighted) edges from a graph G. This defines the shortest path metric $d_G : V \times V$ (where $d_G(u, v)$ is the shortest path between u and v in G.) The problem is to estimate $d_G(u, v)$ for some vertices u, v. We can consider the problem where u, v are known in advance of seeing the graph stream, and also when they are not known in advance.

A common method for approximating graph distance is via the construction of a spanner.

Definition 3. *Given a graph $G = (V, E)$,* a t-spanner of G *is a graph $H = (V, E')$ such that for all u, v, $d_G(u, v) \leq d_H(u, v) \leq t \cdot d_G(u, v)$.*

Theorem 7. *[Elk07, Bas08] There is an algorithm that accept as input a stream of edges from a graph and that computes a $2t-1$ spanner using $\tilde{\mathcal{O}}(n^{1+1/t})$ space.*

Lecture 5. Functional Approximation

Lecturer: S. Muthu Muthukrishnan Scribes: Laszlo Egri and Phuong Nguyen

5.1 Setting

Let $\mathcal{D}$ be a subset of $\mathbb{R}^N$, called a *dictionary*. Given an input vector $A \in \mathbb{R}^N$ and an natural number $b \geq 1$, we wish to find b vectors $D_1, D_2, \ldots, D_b$ from $\mathcal{D}$ so that

$$\min_{\alpha_1,\alpha_2,\ldots,\alpha_b} \{\|A - \sum_{i=1}^{b} \alpha_i D_i\|_2 \ : \ \alpha_i \in \mathbb{R} \text{ for } 1 \leq i \leq b\} \tag{5.1}$$

is minimal. For each subset $\{D_1, D_2, \ldots, D_b\}$ of D, the quantity (5.1) always exists (it is the distance from the vector A to the subspace generated by $\{D_1, D_2, \ldots, D_b\}$) and is also called the error of the subset $\{D_1, D_2, \ldots, D_b\}$. So here we are asking for the subset of size b with smallest error.

For example, if $b = N$ and $\mathcal{D}$ contains a basis $D_1, D_2, \ldots, D_N$ for $\mathbb{R}^N$, then we can take this basis as our output. The minimal value for (5.1) for this output is 0 and is achieved by taking α_i so that $\alpha_i D_i$ is the projection of A along the corresponding basis vector D_i.

For another example, suppose that $\mathcal{D}$ consists of an orthonormal basis for $\mathbb{R}^N$, and $b < N$. Then the error (5.1) is minimal when we take $D_1, D_2, \ldots, D_b$ to be the b unit vectors with largest projections of A, and $\alpha_i = A \cdot D_i$.

In the second example, computing a projection of A takes time $\mathcal{O}(\mathcal{N})$. So the naive algorithm that computes all projections of A and then chooses b largest among them takes time $\mathcal{O}(N^2)$. The basic question is whether we can improve on this running time. We will show later that if $\mathcal{D}$ is some special basis (e.g., the Haar wavelet) then we need only linear time.

In practice, for an application (e.g., audio, video, etc.) it is important to find the "right" basis that is suitable to the common kind of queries (i.e., A and b).

5.2 Arbitrary Dictionary

We show that the general setting, where the dictionary is arbitrary, it is NP-hard even to estimate whether the minimal error (5.1) is 0. We do this by reducing the estimation problem to the *exact set cover* problem. Let $U = \{1, 2, \ldots, n\}$, U is called the set of *ground elements*. Given a collection $S_1, S_2, \ldots, S_m$ of subsets of U and a number $d \leq n$, the exact set cover problem is to find an *exact cover* of size $\leq d$, i.e., a collection of d pairwise disjoint subsets $S_{k_1}, S_{k_2}, \ldots, S_{k_d}$ such that

$$\bigcup_{i=1}^{d} S_{k_i} = U$$

(The *set cover* problem is defined in the same way but the subsets S_{k_i} are not required to be pairwise disjoint.) The problem is NP-hard to approximate, in the sense that for any given constant $\eta < 1$, there is a reduction from SAT to exact set cover so that

- a YES instance of SAT results in an instance of exact set cover with a cover of size $\leq \eta d$,
- a NO instance of SAT produces an instance of exact set cover with no cover of size $\geq d$.

We represent the inputs $S_1, S_2, \ldots, S_m$ to the exact set cover problem by an $n \times m$ matrix M, where

$$M_{i,j} = 1 \qquad \text{iff} \qquad i \in S_j$$

(Thus the j-th column of M is the characteristic vector of S_j.)

For the reduction, consider the dictionary consisting of the characteristic vectors (also denoted by S_j) of S_j (for $1 \leq j \leq m$), $A = \vec{1}$ (the all-1 vector of length n) and $b = \eta d$. It is easy to see that an exact cover of size $s \leq b = \eta d$ gives rise to a subset

$$\{D_1, D_2, \ldots, D_b\}$$

such that

$$A = \sum_{i=1}^{s} D_i$$

(Here we take

$$\alpha_1 = \alpha_2 = \ldots = \alpha_s = 1, \qquad \alpha_{s+1} = \ldots = \alpha_b = 0$$

The vectors D_i are precisely those S_{k_i} that belong to the exact cover.) Consequently, if there is an exact cover of size ηd, then the minimal value of (5.1) is 0. Otherwise, if there are no exact cover of size $\geq d$, then the error (5.1) of each subset $\{D_1, D_2, \ldots, D_b\}$ is always at least the distance from A to the subspace generated by $\{D_1, D_2, \ldots, D_b\}$. Let h be the smallest distance from A to any subspace generated by b vectors in S, then $h > 0$. The error (5.1) is at least h, and hence is strictly greater than 0.

5.3 An orthogonal basis: Haar wavelet

Suppose that $N = 2^k$. An orthogonal basis for $\mathbb{R}^N$ based on the Haar wavelet can be described as follows. Consider a fully balanced binary tree T of depth k with $N = 2^k$ leaves and $N - 1$ inner nodes. Each inner node in T is labeled with an integer n, $1 \leq n < N$ in the following way: the root is labeled with 1, and for $2 \leq n < N$ with binary notation

$$n = n_{t-1} n_{t-2} \ldots n_0$$

(where $2 \leq t \leq k$) the node labeled by n is the other endpoint of the path that starts at the root and follows the direction specified by the bits $n_{t-2}, \ldots, n_0$. (Here if n_{t-2} is 0 then we follow the left child of the root, otherwise if n_{t-2} is 1 then we follow the right child of the root, etc.)

Number the leaves of T from left to right with $1, 2, \ldots, N$. For each n ($1 \leq n < N$) the basic vector w_n:

$$w_n = (u_1, u_2, \ldots u_N)$$

is defined so that for all index i:

$$u_i = \begin{cases} 0 & \text{if } i \text{ is not a descendant of } n \\ 1 & \text{if } i \text{ is a descendant of the left child of } n \\ -1 & \text{if } i \text{ is a descendant of the right child of } n \end{cases}$$

More precisely, For a node n, $1 \leq n < N$, let $\ell(n)$ (resp. $r(n)$) be the left-most (resp. right-most) leaf descendant of n, and $t(n)$ be the number of the right-most leaf descendant of the left child of n. (Note that $\ell(n) \leq t(n) < r(n)$.) Now define the basic vector

$$w_n = (0, 0, \ldots, 0, 1, 1, \ldots, 1, -1, -1, \ldots, -1, 0, 0, \ldots, 0) \tag{5.2}$$

where the 1's are from position $\ell(n)$ to $t(n)$, and the -1's are from position $t(n) + 1$ to $r(n)$. For example, for the root:

$$w_1 = (1, 1, \ldots, 1, -1, -1, \ldots, -1)$$

where the first half of the coordinates are 1 and the other half are -1. For another example,

$$w_{2^{k-1}} = (1, -1, 0, 0, \ldots, 0), \qquad w_{N-1} = (0, 0, \ldots, 0, 1, -1)$$

Finally, define

$$w_N = (1, 1, \ldots, 1)$$

It is easy to verify that $\langle w_{n_1}, w_{n_2} \rangle = 0$ for all $n_1 \neq n_2$. So we have a set of N vectors that are orthogonal, and hence they form a basis for $\mathbb{R}^N$. We also assume that each vector w_n is normalized so we get an orthonormal basis.

5.4 An efficient way of finding the b Haar wavelets with the largest projections

Now, given a query A and $b < N$, we need to find a subset of basis vectors $D_1, D_2, \ldots, D_b$ so that (5.1) is minimal. This is equivalent to finding such a subset $D_1, D_2, \ldots, D_b$ that the projection of A on the subspace generated by $D_1, D_2, \ldots, D_b$ is maximum. We will show that this can be done in time $O(N \log N)$ (better than the obvious algorithm that takes time $\Omega(N^2)$). Indeed, the running time can be improved to linear in N, but we will not prove it here.

The inner nodes in the tree T above consist of k layers. For example, the root alone makes up the first layer, and all nodes n where $n \geq 2^{k-1}$ make up the k-th layer.

Because the Haar basis is an orthogonal basis, we can solve the problem stated in the Section 5.1 by finding the b largest projections. We can do this by calculating the inner product with each of these N vectors. This would take $O(N^2)$ time. The question is whether we can do it

faster. So far we have been looking at items that have been either inserted or deleted. We can now look at a simpler streaming model where there is a vector and you are looking at the vector left to right, just reading one character at a time. In other words, we are inserting the i-th component, the $i+1$-th component, and so on. So we are going to take a signal A with N components, read it left to right, keep computing something which in the end will give us the b largest wavelet coefficients.

More precisely, the idea comes from the following observation. For $1 \leq i \leq N$ let A_i be the vector A at time i, i.e. the vector whose first i coordinates are the same as that of A, and the remaining coordinates are 0. In other words, if $A = (A[1], A[2], \ldots, A[N])$, then

$$A_i = (A[1], \ldots, A[i], 0, 0, \ldots)$$

Consider the path p_i from the root to leaf i in the tree described above. Observe that if n is to the left of this path (i.e. $r(n) < i$), then the projection of A on w_n is determined already by A_i:

$$\langle A, w_n \rangle = \langle A_i, w_n \rangle$$

Thus, the high level idea of the algorithm is to compute recursively for $i = 1, 2, \ldots, N$ the b basis vectors w_n, where n is to the left of the path p_i, that give the largest value of $\langle A_i, w_n \rangle$. For this, we will also have to maintain the dot products $\langle A_i, w_m \rangle$ for every node m that lie on the current path p_i. Observe that to keep track of this information we need $O(b + log(N))$ space.

Consider now inserting the $(i+1)$-th element $A[i+1]$ (i.e. the $(i+1)$-st step in the algorithm). Let p_{i+1} be the path from the root to the $(i+1)$-th leaf node. We want to compute the inner product of the partial input vector (i.e. when only the first $i+1$ components have been inserted) with each vector w corresponding to a node on the path p_{i+1}. For simplicity, we assume that the entries of the Haar basis vectors are $0, -1, 1$, but note that actually, the Haar wavelets are normalized. There are three things to observe:

1. Observe that by the definition of the Haar wavelet basis, if n is on the path p_{i+1} then the $(i+1)$-th component of w_n is either 1 or -1. Assume that w is a node of both p_i and p_{i+1}. In this case, if the $(i+1)$-th element of w_n is a 1 (-1), then

   $$\langle A_{i+1}, w_n \rangle = \langle A_i, w_n \rangle \pm A[i+1]$$

   So to update $\langle A_i, w_n \rangle$ we simply need to add (subtract) $A[i+1]$ to (from) the current value. If w is a "new" node, i.e. it does not appear in p_i, then $\langle A_{i+1}, w_n \rangle = A[i+1]$.

2. Intuitively, the path we consider in the binary tree at each step is "moving" left to right. Consider a wavelet vector w' that corresponds to a node of the binary tree that is to the left of p_{i+1}. More formally, assume that for some $j < i+1$, w' corresponds to a node n of p_j, but n is not in p_{i+1}. Then the $(i+1)$-th component of w' is 0 by the definition of the Haar wavelet basis and therefore the inner product of w' with A is not affected by $A[i+1]$.

3. The inner product of A at time i with wavelet vectors that correspond to nodes which did not yet appear in any path is 0.

To keep track of the b largest coefficients, we need space $O(b)$. We can use a heap to store this information. Observe that the time required is $O(N \log N)$ since N elements are inserted and whenever an element is inserted, we need to update $O(log(N))$ inner products along the path from the root to the i-th leaf. We note that there are other ways to do this in linear time.

We consider now a slightly different problem. We want to place the previous problem into a real streaming context. In other words, we consider the input vector A and we allow insertions and deletions at any point. We are looking at the frequency of items, i.e. $A[i]$ represents the frequency of item i. The query is to find the best b-term representation using the Haar wavelet basis.

We discuss some informal ideas Muthu gave about this problem. Clearly, if we get information about A from left to right, we can just do what we did before. But that is not very useful. So observe that any time we update a particular element, it corresponds to updating the coefficients of $log(N)$ wavelet vectors. We have N wavelet basis vectors, so consider the vector W that stores the coefficients of the basis vectors when the signal A is expressed in the Haar wavelet basis. Now you can think of updating an element in A as updating $log(N)$ elements in W. In a sense, now we are facing the heavy hitter problem, i.e. we need the b largest elements of W. Using techniques we have seen before, it is possible to find b wavelet vectors whose linear combination (where the coefficients are the inner products of the wavelet vectors with A) is $\widetilde{R}$, such that the following holds: $\| A - \widetilde{R} \| \leq \| A - R_b^{OPT} \| + \epsilon \| A \|_2$, where R_b^{OPT} is the best b-term representation of A.

There is also another algorithm that guarantees that $\| A - \widetilde{R} \| \leq (1+\epsilon) \| A - R_b^{OPT} \|$. This algorithm is more complicated and Muthu gave only the high-level intuition. The difficulty with directly getting the top k elements as A gets updated is the following. We can get the large ones using a CM sketch. But if there are a lot of "medium" ones, we will not be able to get them in a small space. But you can get them if you estimate the large ones, subtract it out from the signal (use linearity of CM sketch), look at the remaining signal. (You are not estimating the large ones exactly, so the rest of the signal has some error.) Try to find the heavy hitters again. You repeat and the residual error keeps on going down. With a reasonable amount of iteration, you will get the estimation. At a high level, it is a greedy algorithm.

5.5 A variation on the previous problem

Let $\mathcal{D}$ be a dictionary consisting of Haar wavelets. Let A be a signal with N components. Let b be the number of vectors we want to combine:

$$R_b = \sum_{D_1,\ldots,D_b \in \mathcal{D}} \alpha_i D_i.$$

In the previous sections, we wanted to minimize the error

$$\| A - R_b \|^2 = \sum_{i=1}^{N} (A[i] - R_b[i])^2.$$

The following problem is open (in 2009 March). As before, the basis is the Haar wavelet basis. There is another vector π with positive entries that is also part of the input. The vector π has the

same number N of entries as the signal vector A and it is normalized to 1, i.e. $\sum_{i=1}^{N} \pi[i] = 1$. The problem is to *minimize* the "weighted error":

$$\sum_{i=1}^{N} \pi(i)(A[i] - R_b[i])^2.$$

(Note that if all the entries of π are equal, then this is just the previous problem.) The problem is well-motivated: these representations are used to approximate signals. The question is that when you approximate a signal what do you do with it? In the database context, for example, people often look at queries for individual items. So usually databases keep record which items are asked more often than others, and this is what the vector π corresponds to.

Some informal aspects of the problem: Recall that in the original problem, the coefficients were the inner products of the Haar wavelets with the signal A. It is no longer the case when we have weighted norms. When we do sparse approximation we don't just have to come up with which vectors to choose but also we have to come up with the right choice of coefficients. It is harder to work with this, but we could make the problem easier as follows. We could assume that once we picked the vectors we use the coefficients only along that direction, i.e. we assume that the coefficients are inner products with the signal vector. If we do this then there is an interesting $O(N^2b^2)$ dynamic programming algorithm. (This algorithm uses the binary tree we used before and benefits from the fact that each node in the binary tree has at most $log(N)$ ancestors. This makes it possible to take a look at all possible subsets of the $log(N)$ ancestors of a node in linear time.)

Sparse approximation people use Haar wavelets because Haar wavelets work well for the signals they work with. But if you put arbitrary weights as we done above, then the Haar basis might not be the best basis. One question is: if we know the class of weights, which dictionary should we use? Another question would be: what weights would be good for those signals for which Haar wavelets give a good basis? Muthu mentioned that they can get good approximations when they use piecewise linear weights. You can also ask the same questions about a Fourier-dictionary.

Lecture 6. Compressed Sensing

Lecturer: S. Muthu Muthukrishnan Scribes: Nicole Schweikardt and Luc Segoufin

6.1 The problem

Assume a *signal* $A \in \mathbb{R}^n$, for n large. We want to reconstruct A from linear *measurements* $\langle A, \psi_i \rangle$, where each ψ_i is a vector in $\mathbb{R}^n$, and $\langle A, \psi_i \rangle$ denotes the inner product of A and ψ_i. For suitably chosen ψ_i, n measurements suffice to fully reconstruct A (if the set of all ψ_i forms a basis of $\mathbb{R}^n$). However, we would like to do only k measurements for $k \ll n$. The question is which measurements should be done in order to minimize the error between what we measure and the actual value of A.

We fix some notation necessary for describing the problem precisely: We assume that an orthonormal basis $\psi_1, \ldots, \psi_n$ of $\mathbb{R}^n$ is given. The *dictionary* Ψ is the $n \times n$ matrix the i-th row of which consists of the vector ψ_i. The measurements $\langle A, \psi_i \rangle$, for $i = 1, \ldots, n$, form the vector $\theta(A) := \Psi A$, the vector of coordinates of A with respect to the basis $\psi_1, \ldots, \psi_n$. Note that by the orthonormality of Ψ one obtains that $A = \sum_{i=1}^{n} \theta_i(A)\psi_i$, where $\theta_i(A)$ denotes the i-th component of $\theta(A)$.

In the area of *sparse approximation theory* one seeks for a representation of A that is sparse in the sense that it uses few coefficients. Formally, one looks for a set $K \subseteq \{1, \ldots, n\}$ of coefficients such that $k = |K| \ll n$ such that for the vector

$$R(A, K) := \sum_{i \in K} \theta_i(A)\psi_i$$

the error $||A - R(A, K)||_2^2 = \sum_{i=1}^{n} (A_i - R_i(A, K))^2$ is as small as possible. Since Ψ is orthonormal,

$$||A - R(A, K)||_2^2 = \sum_{i \notin K} \theta_i(A)^2.$$

Thus, the error is minimized if K consists of the k coordinates of highest absolute value in the vector $\theta(A)$. In the following, we write $\theta_{j_1}, \theta_{j_2}, \ldots, \theta_{j_n}$ to denote the components of the vector $\theta(A)$, ordered in descending absolute value, i.e., ordered such that $|\theta_{j_1}| \geq |\theta_{j_2}| \geq \cdots \geq |\theta_{j_n}|$. Furthermore, we write $R^k_{opt}(A)$ to denote the vector $R(A, K)$ where K is a set of size k for which the error is minimized, i.e.,

$$R^k_{opt}(A) \;=\; \sum_{i=1}^{k} \theta_{j_i}\psi_{j_i}. \tag{6.3}$$

Of course, the optimal choice of K depends on the signal A which is not known in advance.

The ultimate goal in *compressed sensing* can be described as follows: Identify a large class of signals A and a dictionary Ψ', described by a $k \times n$ matrix, such that instead of performing the n

measurements ΨA, already the k measurements $\Psi' A$ suffice for reconstructing a vector $R(A)$ such that the error $||A - R(A)||_2^2$ is provably small on all signals A in the considered class.

A particluar class of signals for which results have been achieved in that direction is the class of *p-compressible signals* described in the next section.

6.2 p-compressible signals

We assume that a dictionary Ψ is given. Furthermore, let us fix p to be a real number with $0 < p < 1$.

Definition 4. *A signal A is called* p-compressible (with respect to Ψ) *iff for each $i \in \{1, \ldots, n\}$, $|\theta_{j_i}| = O(i^{-1/p})$.*

Obviously, if A is p-compressible, then

$$||A - R^k_{opt}(A)||_2^2 \quad = \quad \sum_{i=k+1}^{n} \theta_{j_i}^2 \quad \leq \quad \int_{k+1}^{n} O((i^{-1/p})^2) \quad \leq \quad C_p \cdot k^{1-2/p}$$

for a suitable number C_p. Thus, if we assume p to be fixed, the optimal error $||A - R^k_{opt}(A)||_2^2$ is of size at most $C^k_{opt} = O(k^{1-2/p})$ (for $C^k_{opt} := C_p \cdot k^{1-2/p}$).

The following result shows that for reconstructing a vector R such that the error $||A - R||_2^2$ is of size $O(C^k_{opt})$, already $k \log n$ measurements suffice.

Theorem 8 (Donoho 2006; Candès and Tao 2006)**.** *There exists a $(k \log n) \times n$ matrix Ψ' such that the following is true for all p-compressible signals A: when given the vector $\Psi' A \in \mathbb{R}^{k \log n}$, one can reconstruct (in time polynomial in n) a vector $R \in \mathbb{R}^n$ such that $||A - R||_2^2 = O(C_k^{opt})$.*

The proof details are beyond the scope of this lecture; the overall structure of the proof is by showing the existence of Ψ' by proving the following: if Ψ' is chosen to be the matrix $T\Psi$, where T is a random $(k \log n) \times n$ matrix with entries in $\{-1, 1\}$, then the probability that Ψ' satisfies the theorem will be nonzero. A crucial step in the proof is to use "the L_1 trick", i.e., to consider the L_1-norm $|| \cdot ||_1$ instead of the L_2-norm $|| \cdot ||_2$ and solve a suitable linear program.

Note that in lecture #5 we already considered the particular case where Ψ is a Haar wavelet basis, and solved similar questions as that of Donoho and Candès and Tao for that particular case.

6.3 An explicit construction of Ψ'

Theorem 8 states that a matrix Ψ' exists, and the proof of Theorem 8 shows that Ψ' can be chosen as $T\Psi$, for a suitable $(k \log n) \times n$ matrix T. The goal in this section is to give an explicit, deterministic construction of T.

Recall from Section 6.1 that $\theta(A) = \Psi A$. Our goal is to approximate the vector $R^k_{opt}(A)$ from equation (6.3) by a vector R that can be found using only the measurements $\Psi' A$ instead of using all the measurements ΨA. Clearly, if $\Psi' = T\Psi$, then $\Psi' A = T\Psi A = T\theta(A)$. Since we want to

use $\Psi' A = T\theta(A)$ to find $R^k_{opt}(A)$, we clearly should choose T in such a way that it picks up the k largest components (w.r.t. the absolute value) of $\theta(A)$.

Note the striking similarity between this problem and the following *combinatorial group testing* problem: We have a set $U = \{1, \ldots, n\}$ of items and a set D of distinguished items, $|D| \leq k$. We identify the items in D by performing "group tests" on subsets $S_i \subseteq U$. The output of each group test is 0 or 1, revealing whether the subset S_i contains at least one distinguished item, i.e., $|S_i \cap D| \geq 1$. Collections of $O(k \log n)^2)$ nonadaptive tests are known which identify each of the distinguished items precisely.

For the special case where only k-*support signals* are considered (i.e., signals A where at most k of the components in $\theta(A)$ are nonzero), a solution of the combinatorial group testing problem almost immediately gives us a matrix T with the desired properties.

For the more general case of p-compressible signals, the following is known.

Theorem 9 (Cormode and Muthukrishnan, 2006)**.** *We can construct a poly$(k, \varepsilon, \log n) \times n$ matrix T in time polynomial in k and n such that the following is true for the matrix $\Psi' := T\Psi$ and for all p-compressible signals A: when given the vector $\Psi' A$, one can reconstruct a vector $R \in \mathbb{R}^n$ such that $||A - R||_2^2 \leq ||A - R^k_{opt}(A)||_2^2 + \varepsilon C_k^{opt}$.*

The construction of T in the proof of Theorem 9 is based on the following two facts (where $[n] := \{1, \ldots, n\}$).

Fact 1 (k**-separative strong set).** *Given n and k, for $l = k^2 \log^2 n$, one can find l sets $S_1, \cdots, S_l$ included in $[n]$ such that for all $X \subseteq [n]$ with $|X| \leq k$ we have*

$$\forall x \in X, \exists i \text{ such that } S_i \cap X = \{x\}.$$

Fact 2 (k**-separative set).** *Given n and k, for $m = k \log^2 n$, one can find m sets $S_1, \cdots, S_m$ included in $[n]$ such that for all $X \subseteq [n]$ with $|X| \leq k$ we have*

$$\exists i \text{ such that } |S_i \cap X| = 1.$$

Furthermore, we need the following notations for describing the matrix T:

1. Given a $u \times n$ matrix M and a $v \times n$ matrix N, $M \oplus N$ denotes the $(u + v) \times n$ matrix consisting of the rows of M followed by the rows of N.

2. Given a vector $B \in \mathbb{R}^n$ and a $u \times n$ matrix M, $B \otimes M$ denotes the $u \times n$ matrix whose element (i, j) is $B_j * M[i, j]$. If N is a $v \times n$ matrix then $N \otimes M$ is a $uv \times n$ matrix obtained by applying the vector operation on each row of N, using $\oplus$ to merge the results.

3. The *Hamming matrix* H is the $\log n \times n$ matrix such that column i is the binary coding of i. We add an extra row to H with 1 everywhere. With $n = 8$ this yields :

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}.$$

   This basically corresponds to a binary search strategy for a set of size n.

Now set k' as a suitable function of k and p and let m be the number corresponding to n and k' as given by Fact 2 and let S' be the corresponding k'-separating sets. Set k'' as a $(k' \log n)^2$ and let l be the number corresponding to n and k'' as given by Fact 1 and let S'' be the corresponding strong k''-separative sets. From S' form the characteristic matrix M' and from S'' form the characteristic matrix M''. Let T be the matrix $(M' \otimes H) \oplus M''$. This matrix has a number of rows that is: $m * \log n + l$ which is poly-log in n by construction.

The fact that this matrix has the desired properties can be found in [Cormode and Muthukrishnan, SIROCCO 2006].

6.4 Literature

A bibliography on compressed sensing can be found at `http://dsp.rice.edu/cs`.

In particular the following references were mentioned during the lecture:

- David Donoho: Compressed sensing. IEEE Trans. on Information Theory, 52(4), pp. 1289–1306, April 2006.
- Emmanuel Candès and Terence Tao: Near optimal signal recovery from random projections: Universal encoding strategies? IEEE Trans. on Information Theory, 52(12), pp. 5406–5425, December 2006.
- Graham Cormode and S. Muthukrishnan: Combinatorial Algorithms for Compressed Sensing. SIROCCO 2006, LNCS volume 4056, pp. 280–294, Springer-Verlag, 2006.
- S. Muthukrishnan: Some Algorithmic Problems and Results in Compressed Sensing. Forty-Fourth Annual Allerton Conference. (The article is available on Muthu's webpage at `http://www.cs.rutgers.edu/~muthu/resrch_chrono.html`)

Lecture 7. The Matador's Fate and the Mad Sofa Taster: Random Order and Multiple Passes

Lecturer: Andrew McGregor Scribes: Pierre McKenzie and Pascal Tesson

7.1 Introduction

The previous lectures have focused on algorithms that get a single look at the input stream of data. This is the correct model in many applications but in other contexts a (typically small) number of passes through input may be allowed. For example, it can be a reasonable model for massive distributed data. We want to understand the inherent trade-offs between the number of passes and the space complexity of algorithms for problems considered in previous lectures.

We have also considered the space complexity of algorithms in a "doubly-worst-case" sense. We are of course assuming worst-case data but, implicitly, we have also assumed that the *order of presentation* of the data is chosen adversarially. This simplifies the analysis and provides guarantees on the space required for the computation but average-case analysis is often more appropriate: in many real-world applications, such as space-efficiently sampling salaries from a database of employees for example, data streams are relatively unstructured. We thus consider the complexity of algorithms on random-order streams and again seek to identify cases where this point of view provides significant gains in space-complexity. These gains sometimes come simply from a sharper analysis of algorithms discussed earlier but, more interestingly, we can also tweak existing algorithms to take full advantage of the random order model. Lower bounds for this model are obviously trickier but more meaningful from a practical point of view.

7.2 Basic Examples

Smallest value not in the stream

Consider first the task of identifying the smallest value x that is *not* occurring in a stream of length m consisting of values in $[n]$. Let us look at variants of the problem where some promise on the input is provided.

Version 1: Promised that $m = n - 1$ and that all elements but x occur (i.e. all elements but x occur exactly once)

In this case, the obvious solution is to keep a running sum S of the elements of the stream and get $x = m(m + 1/2) - S$. The space required is $\widetilde{\Theta}(1)$

Version 2: Promised that all elements less than x occur exactly once.

In this case, the complexity is $\widetilde{\Theta}(m^{1/p})$ where p is the number of passes.

Version 3: No promise.

In this case, the complexity is $\widetilde{\Theta}(m/p)$ where p is the number of passes.

Increasing subsequence

As a second example, consider the problem of finding an increasing subsequence of length k in the stream, given that such a subsequence exists. Liben-Nowell et al. [LNVZ06] gave an upper bound of space complexity $O(k^{1+1/2^p-1})$ which was later shown to be tight [GM09].

Medians and approximate medians

Obviously, the assumption that the data arrives in a random order is often of little help. But there are classical examples for which the gains are significant. Consider the problem of finding the median of m elements in $[n]$ using only $\mathrm{polylog}(m,n)$ space. We also consider the easier problem of finding a *t-approximate median*, i.e. an element x whose rank in the m elements is $m/2 \pm t$.

If we assume that the stream is given in an adversarial order, and if we impose a $\mathrm{polylog}(m,n)$ space bound, we can find a t-approximate median if $t = \widetilde{\Omega}(m/\mathrm{polylog}(m)$ and that bound is tight. However, if we assume that the stream is random-order, we can get t down to $\Omega(\sqrt{m})$. This bound is not known to be optimal but t-approximate medians cannot be computed for $t = \omega(\sqrt[3]{m})$.

Suppose instead that we want to compute the exact median. The bounds above show that this is not possible in space $\mathrm{polylog}(m,n)$ even assuming random order. However the bounds are for 1-pass algorithms and given sufficiently many passes, we can identify the exact median without exceeding our space bound. Specifically, the number of passes needed/sufficient for this task it $\widetilde{\Theta}(\log m/\log\log m)$ in the adversarial model and only $\widetilde{\Theta}(\log\log m)$ in the random-order model.

Theorem 10. *In the adversarial order model, one can find an element of rank $m/2 \pm \epsilon m$ in a single pass and using $\widetilde{O}(1/\epsilon)$ space. Moreover, one can find the exact median in $O(\log m/\log\log m)$ passes using $\widetilde{O}(1)$ space.*

Proof. We have already discussed the one-pass result. In fact, we even showed that it is possible to find quantiles, i.e. find for any $i \in [\epsilon^{-1}]$ an element of rank $i\epsilon m \pm \epsilon m$

The multi-pass algorithm is built through repeated applications of this idea. In a first pass, we set $\epsilon = 1/\log m$ and find a and b with

$$\mathrm{rank}(a) = m/2 - 2/\log m \pm m/\log m \quad \text{and} \quad \mathrm{rank}(b) = m/2 + 2/\log m \pm m/\log m$$

Now in pass 2, we can find the precise rank of a and b and from there recurse on elements within the range $[a,b]$. Note that this range is of size at most $m/\log m$ and every pair of passes similarly shrinks the range by a factor of $\log m$. Hence $O(\log m/\log\log m)$ passes are sufficient to find the median. □

Let us now focus on lower bound arguments for this same problem.

Theorem 11. *Finding an element of rank $m/2 \pm m^\delta$ in a single pass requires $\Omega(m^{1-\delta})$ space.*

Proof. Once again the proof relies on a reduction from communication complexity. Specifically we look at the communication complexity of the INDEX function: Alice is given $x \in \{0,1\}^t$, Bob is given $j \in [t]$ and the function is defined as $\text{INDEX}(x,j) = x_j$. Note that the communication complexity of this problem is obviously $O(\log n)$ if Bob is allowed to speak first. However, if we consider one-way protocols in which Alice speaks first, then $\Omega(t)$ bits of communication are necessary (this is easy to show using a simple information-theoretic argument).

We want to show that Alice and Bob can transform a single pass algorithm for the approximate median into a protocol for INDEX. Given x, Alice creates the stream elements $\{2i + x_i : i \in [t]\}$ while Bob appends to the stream $t - j$ copies of 0 and $j - 1$ copies of $2t + 2$. Clearly, the median element in the resulting stream is $2j + x_j$ and Alice can send to Bob the state of the computation after passing through her half of the stream. Hence, finding the exact median requires $\Omega(m)$ space. To get the more precise result about approximate medians, it suffices to generate $2m^\delta + 1$ copies of each of the elements: any m^δ-approximate median still has to be $2j + x_j$ and since the resulting stream is of length $O(tm^\delta)$, the communication complexity lower bound translates into an $\Omega(m/m^\delta)$ lower bound on the space complexity of the streaming algorithm. □

If we hope to obtain lower bounds in the multi-pass model using the same approach, we cannot simply rely on the lower bound for INDEX. Intuitively, each pass through the stream forces Bob to send to Alice the state of the computation after the completion of the first pass and forces Alice to send to Bob the state of the computation after the completion of the first half of the second pass. Instead, we consider a three-party[3] communication game in the "pointer-jumping" family. Alice is given a $t \times t$ matrix X, Bob is given $y \in [t]^t$ and Charlie is given $i \in [t]$. Let $j \in [t]$ be defined as $j = y_i$: the players' goal is to compute X_{ij} with an $A \to B \to C \to A \to B \to C$ protocol. (Alice speaks first, followed by Bob, ...) Any ABCABC protocol for this function requires $\Omega(t)$ communication [NW93].

Theorem 12. *Finding the exact median in two passes requires space $\Omega(\sqrt{m})$.*

Proof. The reduction from the pointer-jumping problem works as follows. Again, we think of Alice, Bob and Charlie as creators of, respectively, the first, second and last third of a stream. Therefore, a space-efficient 2-pass algorithm can be translated into a cheap ABCABC communication protocol.

Let $T > 2t + 2$ and let $o_k = T(k-1)$. Alice creates for each $k \in [t]$, the elements $A_k = \{2\ell + X_{\ell k} : \ell \in [t]\} + o_k$. Bob creates for each $k \in [t]$, the elements $B_k = \{t - y_k$ copies of 0 and $y_k - 1$ copies of $B\} + o_k$. In other words, Alice and Bob's stream elements form t non-overlapping blocks. Each such block has $2t - 1$ elements and follows the pattern of the 1-pass reduction. Charlie on the other hand adds $t(t-i)$ copies of 0 and $t(i-1)$ copies of Bo_t. It is convenient to also think of these as t blocks of $(t-i)$ copies of 0s and t blocks of $(i-1)$ copies of

[3]Note that the multi-party model considered here is the "number in hand model" and not the "number on the forehead" model.

Bo_t. The total number of blocks in our construction is $2t-1$ and the median element in the stream is the median element of the median block.

The elements generated by Charlie guarantee that the median block is the ith one and by construction of the A_k and B_k, the median of this block is $o_i + 2j + X_{ij}$ where $j = y_i$.

A 2-pass algorithm using space S yields an ABCABC protocol of cost $5S$ for the pointer-chasing problem and since the stream above is of length $O(t^2)$, the communication complexity lower bound translates into an $\Omega(\sqrt{m})$ space lower bound for two-pass algorithms. □

Throughout our discussion on medians, we assumed that the length m of the stream is known in advance. This is crucial for the upper bounds: one can in fact show that when m is not known a priori, an algorithm for the exact median requires $\Omega(m)$ space even if the data is presented in sorted order. (Left as an exercise)

7.3 Taking advantage of random streams

Theorem 13. *In the random order model, one can find an element of rank $m/2 \pm \widetilde{O}(\sqrt{m})$ in a stream of elements from $[n]$ in a single pass using $\widetilde{O}(1)$ space.*

Proof. We split the stream into $O(\log m)$ segments of length $O(m/\log m)$. We set $a_1 = -\infty$, $b_1 = +\infty$. At the ith step we process the ith segment. We enter the ith step thinking that we have a_i and b_i fulfilling rank(a_i) $< m/2 <$ rank(b_i), and then

- we pick in the ith segment the first c fulfilling $a_i < c < b_i$
- we use the rest of the ith segment to compute an estimate $\tilde{r}$ of rank(c), by setting $\tilde{r}$ to $O(\log m) \times$ the rank of c within the ith segment
- if $\tilde{r}$ is within $\widetilde{O}(\sqrt{m})$ of $m/2$ then we output c, otherwise we proceed to step $i+1$ after setting

$$(a_{i+1}, b_{i+1}) = \begin{cases} (a_i, c) & \text{if } \tilde{r} > m/2 \text{ (i.e. the median is likely below } c) \\ (c, b_i) & \text{if } \tilde{r} < m/2. \end{cases}$$

This algorithm finds an approximation to the median with high probability. The probability analysis uses a variant of Chernoff-Hoeffding bounds applied to sampling without replacement [GM09]. The algorithm manipulates a constant number of $(\log mn)$-bit numbers so uses $\widetilde{O}(1)$ space. □

The above algorithm is more accurate than the CM sketch but the latter yields all the quantiles. By increasing the number of passes, yet a better approximation to the median can be obtained. A second pass can improve the approximation from $\pm\widetilde{O}(\sqrt{m})$ to $\pm\widetilde{O}(m^{1/4})$, a third pass to $\pm\widetilde{O}(m^{1/8})$, and so on. One needs care to account for the fact that the input is not rerandomized at each pass [GK01]. The exact median can be found in space $\widetilde{O}(1)$ using $O(\log\log m)$ passes [GM09].

Turning to lower bounds to match the above theorem, the difficulty in extending the communication complexity technique to the random order model is to account for randomisation when partitioning the player inputs. It can be shown that approximating the median to $\pm\widetilde{O}(m^{\delta})$ in one pass requires $\Omega(m^{\frac{1-3\delta}{2}})$ space. This is known to be tight when $\delta = 0$ but an open problem is to prove $\Omega(m^{\frac{1}{2}-\delta})$. See [GM09] for a refined statement that takes the number of passes into account.

The bearing of the communication complexity of the Hamming distance problem (in which Alice and Bob want to estimate the Hamming distance between their respective n-bit strings) on the data stream complexity of computing frequency moments was not treated in these lectures. The 2-line story here is:

- see [GM09],
- see [CCM08].

Lecture 8. Random order, one pass. Linear algebraic problems.

Lecturer: Andrew McGregor and S. Muthu Muthukrishnan Scribes: Valentine Kabanets and Antonina Kolokolova

8.1 Random order

Theorem 14. *Finding the exact median requires $\Omega(\sqrt{m})$ space.*

Theorem 15. *To find m^δ-approximation of the median in one pass, in random order setting, requires space $\Omega(m^{\frac{1-3\delta}{2}})$.*

The lower bound is tight for $\delta = 0$, but not known to be tight for $\delta = 1/2$.

Problem 1. *Improve this to $\Omega(m^{\frac{1}{2}-\delta})$.*

Proof of theorem 14. The proof is by reduction from communication problem INDEX. Suppose Alice has $x \in \{0,1\}^t$ and Bob has $j \in [t]$.

Claim 5. *Even when $x \in_R \{0,1\}^t$ (that is, picked uniformly at random from $\{0,1\}^t$), any one-pass one way (Alice to Bob) protocol requires $\Omega(t)$ communication.*

Let Alice have $A = \{2i + x_i \mid i \in [t]\}$, and let Bob have $t - j$ copies of 0 (set B_1) and $j - 1$ copies of $2t + 2$ (set B_2). Then, finding a median requires Alice to know j.

This is the case of adversarial stream. For the random stream case, how can Alice and Bob simulate an algorithm on a random permutation of A, B_1, B_2? They cannot do such a simulation, but they do an "almost random" stream.

Start by adding to A, B_1, B_2 t^2 copies of B_1 and B_2 items. So the size of the set of 0s B_1 becomes $t^2 + t - j$, and of new B_2, $|B_2| = t^2 + j - 1$. Using public randomness, decide ahead of time where elements of A appear. To Alice's elements on Bob's side, add random values y_i. Alice guesses $j = t/2$, and randomly fills Bob's places in her part of the stream with values 0 (small) and $2t + 2$ (large) so that it is balanced by the end of her part of the stream. Bob knows the balance by the start of his stream, and fills in the rest of 0 and $2t + 2$ to make the balance exact. Since t^2 is large in comparison to $t - j$, $j - 1$, equal balance is ok. Finally, although Bob guesses his $2i + x_i$ mostly incorrectly, he can recover. □

Reference: Guha, McGregor SiCOMP'09, and Chakrabarti, Cormode, McGregor STOC'08.

Gap hamming: given two length n binary string, approximate hamming distance. There is a one-pass lower bound.

8.2 Linear algebraic problems

1. Compressed sensing: many parameters, time to reconstruct the signal, etc. There are tables of parameters in various paper. Pyotr Indyk has the largest table.

2. Algorithmic puzzle: given an array of n elements, find *in-place* a leftmost item that has a duplicate, if any. $O(n^2)$ is trivial, $O(n \log^2 n)$ more interesting, if allowed to lose info (overwrite elements) can get $O(n)$.

3. Different sparse approximation problem: given an array of size n and a number k, partition the array in k pieces. Approximate items in every interval by the average of the interval a_i. Want to minimize the quantity: $\sum_i \sum_{j \in i^{th} \text{ interval}} (a_i - A[j])^2$; here, full memory is allowed; no streaming.

   Dynamic programming solution gives $O(n^2 k)$. But suppose the problem is generalized to two dimensions. What kind of partition can it be? There is hierarchical partition (every new line splits an existing block), simple partition (grid), or arbitrary partition.

   The problem is NP-hard for arbitrary partition, and is in P for hierarchical partition (using Dynamic Programming). Approximation algorithm converts the problem of arbitrary partition into a hierarchical partition..

   Sparse approximation: intervals correspond to a dictionary. How to get streaming algorithms for this case? This problem is related to histogram representation problem. It is also related to the wavelets.

8.2.1 Linear algebraic problems

Here we consider three linear algebraic problems:

1. Matrix multiplication.

2. L_2 regression (least squares)

3. Low rank approximation.

Matrix multiplication

Problem: given $A : R^{m \times n}$, $B : R^{n \times p}$, compute $A \cdot B$. Frobenius norm $||x||_F = \sum_{i,j} x_{i,j}^2$: work in streaming world, only look at the space to keep the sketch what we track as the matrices are updated. Take a projection matrix S of dimensions $(1/\epsilon^2) \log 1/\delta$ by n.

Now $AB = (AS^T)(SB)$. Keep track of of (AS^T) (size $\frac{1}{\epsilon^2} \log \frac{1}{\delta} \times m$, $(SB) : \frac{1}{\epsilon^2} \log \frac{1}{\delta} \times p$. The probability $Pr(||AB - AS^T SB||_F \leq \epsilon ||A||_F ||B||_F) \geq 1 - \delta$. This is similar to the approximation of additive error in wavelets, inner product of vectors.

Expectation of the inner product is $E[\langle Sx, Sy \rangle] = \langle x, y \rangle$, variance $Var[\langle Sx, Sy \rangle] \leq 2\epsilon^2 ||x||_2^2 \, ||y||_2^2$. This was proved earlier. From this, get $E[AS^T SB] = AB$, and

$$Var(||AB - AS^T SB||_F^2) \leq 2\epsilon^2 ||A||_F ||B||_F.$$

$$Pr(\min_{1...t} ||AB - AS_i^T S_i B||_F \leq \epsilon ||A||_F ||B||_F) > 1 - \delta,$$

where $t \approx \frac{1}{log \frac{1}{\delta}}$ – need at least $\frac{1}{\epsilon^2}$ bits to solve with this level of accuracy.

L_2 regression (least squares)

Now take $A : R^{n \times d}$, $n > d$ (often $n >> d$), $b \in R^n$. The problem is to solve $Ax \stackrel{?}{=} b$. That is, given points (observational) draw a line (fit to explain all points) minimizing the sum of squares of distances of points to the line. $Z = \min_{x \in R^d} ||Ax - b||_2$. Best known algorithm is $O(nd^2)$; this algorithm is based on numerical analysis. How do we solve this problem in the streaming world? Here, A and b are updated as new points come along. Want guarantees on x and Z. Can get the result:

1. $\tilde{z} \leq (1 + \epsilon)z$
2. Can find x'_{opt} such that $||SAx_{opt} - Sb||_2 \leq (1 + \epsilon)z$. Take CM sketch, projection of A and a projection of b, and solve the problem on them. For S, take $\frac{d \log d}{\epsilon^2} \times n$ CM sketch vectors. Solve $\min_x ||SAx - Sb||_2$. Size of SA is $\frac{d \log d}{\epsilon^2} \times d$, of Sb is $\frac{d \log d}{\epsilon^2}$. This gives the d^3 term in the expression. Use fast Johnson-Lindenstrauss transform. In streaming world, assume more SA.
3. $||x_{opt} - x'_{opt}||_2 \leq \frac{\epsilon^2}{\sigma^2_{min}(A)}$, the smallest eigenvalue.

Low rank approximation

There is a simple algorithm for the low rank approximation; surprising, knowing the history of the problem.

Think of sites collecting information. There is a central site and k other sites. Let $S_i(t)$ be the number of items seen by a (non-central) cite i by the time t. We are interested in $\sum_i |S_i(t)|$. The central site gets information from all the rest of sites, and outputs a 0 if $\sum_i |S_i(t)| < (1 - \epsilon)\tau$ and output 1 if $\sum_i |S_i(t)| > \tau$, where τ is the central site's threshold parameter. We are interested in minimizing the number of bits sent to the central site by the others. There is no communication from the central site back or between non-central sites. All sites know τ and k. In case of 2 players, send 1 bit when seen $\tau/2$ items: gives 2 bits of communication. Can it be generalized to k players (would that give k bits of communication)?

Lecture 9. Massive Unordered Distributed Data and Functional Monitoring

Lecturer: S. Muthu Muthukrishnan Scribes: Eric Allender and Thomas Thierauf

9.1 Distributed Functional Monitoring

In the distributed functional monitoring problem [CMY08] we have k *sites* each tracking their input. The sites can communicate with a designated *central site*, but not among each other. Let $s_i(t)$ be the number of bits that site i has seen up to time t. The task of the central site is to monitor a given function f over the inputs $s_1(t), \ldots, s_k(t)$ for all times t. The goal is to minimize the number of bits that are communicated between the sites and the central site.

We consider the example where the function f is the sum of the values $s_i(t)$. Define

$$F_1 = \sum_{i=1}^{k} s_i(t).$$

The central site is required to detect when F_1 exceeds a certain threshold τ, i.e. we consider the function

$$c(t) \;=\; \begin{cases} 1, & \text{if } F_1 > \tau, \\ 0, & \text{otherwise.} \end{cases}$$

The interesting case is to compute the approximate version c_A of c: for some given $0 < \varepsilon \leq 1/4$ the output of the central site at time t is defined as

$$c_A(t) \;=\; \begin{cases} 1, & \text{if } F_1 > \tau, \\ 0, & \text{if } F_1 \leq (1-\varepsilon)\tau. \end{cases}$$

We do not care about the output if $(1-\varepsilon)\tau < F_1 \leq \tau$. The problem of computing c_A with these parameters is called the $(k, F_1, \tau, \varepsilon)$ *functional monitoring problem*.

There are two trivial algorithms for it:

1. Each site communicates the central site every bit it sees.
2. Site i sends a bit to the central site each time that $s_i(t)$ increases by $\tau(1-\epsilon)/k$.

The first solution would even allow the central site to compute the exact function c. However, the amount of communication is extremely high (τ bits). The second algorithm needs only about k bits of communication, but the error made in computing c_A can be very large.

In this lecture we show

Theorem 16. *There is a randomized algorithm for $(k, F_1, \tau, \varepsilon)$ monitoring with error probability $\leq \delta$ and $O(\frac{1}{\varepsilon^2} \log \frac{1}{\delta} \log\log \frac{1}{\delta}))$ bits of communication.*

It is important to note that the number of bits of communication is *independent* of τ and k. Thus it scales as well as one could hope for.

Consider the following algorithm, where we use a constant c to be determined later.

Site i: Send a bit to the central cite with probability $1/k$ each time that $s_i(t)$ increases by

$$\frac{\varepsilon^2}{ck}\tau.$$

Central site: Output 1 if the total number of bits received from the sites is

$$\geq c\left(\frac{1}{\varepsilon^2} - \frac{1}{2\varepsilon}\right),$$

otherwise output 0.

Define the random variable

$$X = \#\text{ bits the central site has received at some point of time.}$$

For the expected value of X we have the following upper and lower bound:

$$\mathrm{E}(X) \leq \frac{1}{k}\frac{F_1}{\varepsilon^2\tau/ck} = \frac{cF_1}{\varepsilon^2\tau}, \tag{9.4}$$

$$\mathrm{E}(X) \geq \frac{1}{k}\frac{F_1 - \varepsilon^2\tau}{\varepsilon^2\tau/ck} = \frac{cF_1}{\varepsilon^2\tau} - c. \tag{9.5}$$

For the variance of X we have

$$\mathrm{Var}(X) \leq \frac{ckF_1}{\varepsilon^2\tau}\left(\frac{1}{k} - \frac{1}{k^2}\right) \leq \frac{cF_1}{\varepsilon^2\tau}. \tag{9.6}$$

Case 1: $F_1 \leq (1-\varepsilon)\tau$. By equation (9.4) we have

$$\mathrm{E}(X) \leq \frac{cF_1}{\varepsilon^2\tau} \leq \frac{c}{\varepsilon^2}(1-\varepsilon) \leq \frac{c}{\varepsilon^2}. \tag{9.7}$$

The probability that the central site outputs 1 is

$$\begin{aligned}
\Pr[X \geq c\left(\frac{1}{\varepsilon^2} - \frac{1}{2\varepsilon}\right)] &\leq \Pr[X \geq E(X) - \frac{c}{2\varepsilon}] \qquad \text{by equation (9.7)} \\
&= \Pr[X - \mathrm{E}(X) \geq -\frac{c}{2\varepsilon}] \\
&\leq \frac{cF_1}{\varepsilon^2\tau}\,\frac{(2\varepsilon)^2}{c^2} \qquad \text{by Chebyshev inequality and equation (9.6)} \\
&= \frac{4F_1}{\tau c} \\
&\leq \frac{4(1-\varepsilon)\tau}{\tau c} \qquad \text{by assumption in case 1} \\
&\leq \frac{4}{c}
\end{aligned}$$

Case 2: $F_1 > \tau$. By equation (9.5) we have

$$\mathrm{E}(X) \;\geq\; \frac{cF_1}{\varepsilon^2\tau} - c \;>\; \frac{c}{\varepsilon^2} - c. \tag{9.8}$$

Then the probability that the central site does not output 1 is

$$\begin{aligned}
\Pr[X < c\left(\frac{1}{\varepsilon^2} - \frac{1}{2\varepsilon}\right)] &\leq \Pr[X < \mathrm{E}(X) + c - \frac{c}{2\varepsilon}] \qquad \text{by equation (9.8)} \\
&= \Pr[X - \mathrm{E}(X) < c - \frac{c}{2\varepsilon}] \\
&\leq \frac{cF_1}{\varepsilon^2\tau}\,\frac{1}{(c - \frac{c}{2\varepsilon})^2} \qquad \text{by Chebyshev inequality and equation (9.6)} \\
&\leq \frac{1}{c(\varepsilon - \frac{1}{2})^2} \qquad \text{by assumption in case 2} \\
&\leq \frac{16}{c} \qquad \text{for } \varepsilon \leq \frac{1}{4}.
\end{aligned}$$

Choosing $c = 64$ makes the error probability $\leq 1/16$ in case 1 and $\leq 1/4$ in case 2. Hence the total the error probability is $\leq 1/3$. The number of bits communicated is $O(1/\varepsilon^2)$.

The error probability can be decreased to δ by running $O(\log \frac{1}{\delta})$ independent instances of the algorithm. That is, we modify the protocol as follows:

Site i: Each time that $s_i(t)$ increases by $\frac{\varepsilon^2}{ck}\,\tau$, the site makes $t = O(\log \frac{1}{\delta})$ independent trials, indexed $1, \ldots, t$, to send a bit to the central site with probability $1/k$ for each trial. However, instead of just one bit, it sends the index j of each successful trial to the central site.

Central site: The central site maintains a counter for each of the t trials, where it adds 1 to counter j whenever it receives message j from one of the sites.

It outputs 1 if the majority of the t counters has a value $\geq c\left(\frac{1}{\varepsilon^2} - \frac{1}{2\varepsilon}\right)$, otherwise it outputs 0.

The number of bits communicated is thus $O(\frac{1}{\varepsilon^2}\,\log\frac{1}{\delta}\,\log\log\frac{1}{\delta})$ as claimed.

9.2 Massive Unordered Distributed Data

We consider truly massive data sets, such as those that are generated by data sources as IP traffic logs, web page repositories, search query logs, retail and financial transactions, and other sources that consist of billions of items per day, and are accumulated over many days. The amount of data in these examples is so large that no single computer can make even a single pass over the data in a reasonable amount of time. Therefore the data is distributed in pieces over many machines. For example, Google's MapReduce and Apache's Hadoop are successful large scale distributed platforms that can process many terabytes of data at a time, distributed over hundreds of even thousands of machines. The machines process the pieces of data in parallel. Then they send their results to a central machine. Clearly, the amount of data that is sent to the central machine should be small, i.e. only poly-logarithmic in the input size.

Since the distribution of the data pieces on the machines is unordered, order should have no impact on the result. Hence, in this lecture we consider a model for algorithms which is called *massive, unordered, distributed (*short: *mud)* algorithms. Mud algorithms consist of three functions:

1. a local function *map* that maps a single input data item x to a list of pairs of the form (k, v), where k is a key and v is a value.

2. an aggregate function *reduce* that gets as input the set S of all *map*(x) (over all data items x), and computes, for each k, some function of the pairs (k, v) that appear in S. Because the input data for *reduce* can be distributed on several machines, the function should be commutative and associative.

3. a function for a final post-processing step. This is not always needed.

Examples

As an example, we want to compute the number of links to a web-page. The data items x are web-pages and *map*(x) is defined to consist of all pairs (k, v), where key k is a URL that occurs

as a link in x and v is the number of times k occurs in x. The *reduce*-function simply computes, for each URL k, the sum of the values v such that (k, v) is produced during the *map* phase. This example does not use the post-processing step.

As another example (using the post-processing step), consider the problem of computing the number of triangles in a graph x on n vertices, where the data is presented as a set of edges (u, w). Applying *map*(u, w) produces pairs (k, v) where the keys k are triples of nodes (i, j, k) with $i < j < k$, where $\{a, b\} \subseteq \{i, j, k\}$, and the values v are triples $(b_{i,j}, b_{i,k}, b_{j,k})$, where $b_{i,j} = 1$, if (i, j) is an edge, and $b_{i,j} = 0$ otherwise. The *reduce*-function computes the bitwise or of the values for each key. In the post-processing step, we output the number of keys k for which $(k, (1, 1, 1))$ is produced during the *reduce* phase.

The number of keys that are used has a significant effect on the efficiency of the resulting algorithm. We will examine the computational power of the extreme case, where we have only *one* key.

Mud algorithms with one key

In the following we consider the special case where there is only one key, i.e. we can omit the key. Thus, in the *map* phase, each data item x produces *map*(x) which is communicated to the *reduce* phase. We call these "messages". In the *reduce* phase, we apply an operator to the messages (in some order).

More formally, the three functions of a mud-algorithm simplify as follows: The local function $\Phi : \Sigma \to Q$ maps an input item to a message, the aggregator $\oplus : Q \times Q \to Q$ maps two messages to a single message, and the post-processing operator $\eta : Q \to \Sigma$ produces the final output.

The output can depend on the order in which $\oplus$ is applied. Let $\mathcal{T}$ be an arbitrary binary tree circuit with n leaves. We use $m_{\mathcal{T}}(\boldsymbol{x})$ to denote the $q \in Q$ that results from applying $\oplus$ to the sequence $\Phi(x_1), \ldots, \Phi(x_n)$ along the topology of $\mathcal{T}$ with an arbitrary permutation of these inputs as its leaves. The overall output of the mud algorithm is $\eta(m_{\mathcal{T}}(\boldsymbol{x}))$, which is a function $\Sigma^n \to \Sigma$. Notice that $\mathcal{T}$ is *not* part of the algorithm, but rather, the algorithm designer needs to make sure that $\eta(m_{\mathcal{T}}(\boldsymbol{x}))$ is independent of $\mathcal{T}$. We say that a mud algorithm computes a function f if $\eta \circ m_{\mathcal{T}} = f$, for all trees $\mathcal{T}$.

The communication complexity of a mud algorithm is $\log |Q|$, the number of bits needed to represent a message from one component to the next. The time, resp. space complexity of a mud algorithm is the maximum time resp. space complexity of its component functions.

Let us compare mud algorithms with streaming algorithms. Formally, a streaming algorithm is given by a pair $s = (\sigma, \eta)$, where $\sigma : Q \times \Sigma \to Q$ maps a state and an input to a state. σ is an operator applied repeatedly to the input stream. $\eta : Q \to \Sigma$ converts the final state to the output. $s^q(\boldsymbol{x})$ denotes the state of the streaming algorithm after starting at state q and operating on the sequence $\boldsymbol{x} = x_1, \ldots x_n$ *in that order*, that is

$$s^q(\boldsymbol{x}) = \sigma(\sigma(\ldots \sigma(\sigma(q, x_1), x_2) \ldots, x_{k-1}), x_n).$$

On input $\boldsymbol{x} \in \Sigma^n$, the streaming algorithm computes $\eta(s^0(\boldsymbol{x}))$, where 0 is the starting state. The communication complexity of a streaming algorithm is $\log |Q|$, and the time, resp. space complexity is the maximum time resp. space complexity of σ and η.

Clearly, for every mud algorithm $m = (\Phi, \oplus, \eta)$ there is a equivalent streaming algorithm $s = (\sigma, \eta)$) of the same complexity by setting $\sigma(q, x) = \oplus(q, \Phi(x))$ and maintaining η. The central question is whether the converse direction also holds. The problem is that a streaming algorithm gets its input sequentially, whereas for a mud algorithm, the input is unordered. Consider the problem of computing the number of occurrences of the first element in the input. This is trivial for a streaming algorithm. However, no mud algorithm can accomplish this because a mud algorithm cannot determine the first element in the input. Therefore we restrict our attention to *symmetric* functions. Here one can show that the models are equivalent in the following sense:

Theorem 17. [FMS$^+$08] *For any symmetric function $f : \Sigma^n \to \Sigma$ computed by a $g(n)$-space and $c(n)$-communication streaming algorithm there exists a mud algorithm that computes f within space $O(g^2(n))$ and $O(c(n))$ communication.*

The proof of this theorem has much the same flavor of Savitch's theorem and can be found in [FMS$^+$08].

Lecture 10. A Few Pending Items and Summary

Lecturer: Andrew McGregor and S. Muthu Muthukrishnan Scribes: Ricard Gavaldà and Ken Regan

10.1 Four Things Andrew Still Wants to Say

Andrew wanted to tell us about 14 things, which were eventually 4 because of time constraints: Computing spanners, estimating the entropy, a lower bound for F_0, and solving the k-center problem.

10.1.1 Computing Spanners

Recall the definition of *t-spanner* of a graph G (alluded to in Lecture 4): It is a subgraph of G obtained by deleting edges such that no distance among any two vertices increases by a factor of more than t.

Here is a simple algorithm for bulding a $(t-1)$-spanner in one pass, over a stream of edges: When a new edge arrives, check whether it completes a cycle of length at most t. If it does,ignore it; otherwise, include it in the spanner. The resulting graph G' is a t-spanner of the original graph G because for every edge (u,v) in $G - G'$ there must be a path from u to v in G' of length $t-1$, namely, the rest of the cycle that prevented us from adding (u,v) to G'. Therefore, for every path of length k in G there is a path of length at most $(t-1)k$ in G' with the same endpoints.

Elkin [Elk07] proposed an algorithm using this idea that computes a $(2t-1)$-spanner in one pass using memory and total time $\tilde{O}\left(n^{1+1/t}\right)$. Furthermore, the expected time per item is $O(1)$.

10.1.2 Estimating the Entropy

Given a stream of elements in $\{1,\ldots,m\}$, let m_i be the frequency of element i in the stream. We would like to estimate the entropy of the stream,

$$S = \sum_i \frac{m_i}{m} \log \frac{m}{m_i}.$$

A first solution to this problem is simply to pick a random element in the stream, call it x, then count the occurrences of x from that point in the stream on, call this number R. Then output

$$\hat{S} = R \log \frac{m}{R} - (R-1) \log \frac{m}{R-1}.$$

We claim that $\hat{S}$ is an estimator of the entropy S. Indeed, if we define $f(r) = r\log(m/r)$,

$$\begin{aligned}
E[\hat{S}] &= \sum_r \Pr[R=r]\cdot\,(f(r)-f(r-1)) \\
&= \sum_r \sum_i \Pr[x=i]\,\cdot \Pr[R=r|x=i]\,(f(r)-f(r-1)) \\
&= \sum_i \frac{m_i}{m} \sum_{r=1}^{m_i} \frac{1}{m_i}\,(f(r)-f(r-1)) \qquad \text{(and the sum telescopes, so)} \\
&= \sum_i \frac{m_i}{m}\,\frac{1}{m_i}\,(f(m_i)-f(1)) = \sum_i \frac{m_i}{m}\,\log\frac{m}{m_i}
\end{aligned}$$

One can show that the variance is also small *when the entropy is large*. A solution that works also when the entropy is small was given by Chakrabarti, Cormode, and McGregor [CCM07].

10.1.3 A Lower Bound for $(1+\epsilon)$-approximation of F_0

One can give a $\Omega(\epsilon^{-2})$ lower bound, hence matching the algorithm that Muthu presented. The bound is a reduction from the communication complexity of the problem of estimating the Hamming distance among two streams.

Let x, y be two n-bit strings, and say that Alice has x and Bob has y. Let S_x and S_y be the set of elements with characteristic vectors x and y. Then

$$F_0(S_x \cup S_y) = |S_x| + |S_y| - |S_x \cap S_y|.$$

Jayram, Kumar, and Sivakumar [JKS35] showed that estimating the Hamming distance up to an additive $\sqrt{n}$ requires $\Omega(n)$ bits of one-way communication. From here, one can see that a one-pass algorithm that approximates F_0 within multiplicative ϵ must use $\Omega(\epsilon^{-2})$ bits of memory.

Brody and Chakrabarti [BC09] have recently shown lower bounds for the multiround communication complexity of the gap hamming distance problem, which implies lower bounds for F_0, F_1, F_2, etc. in the multipass streaming model.

10.1.4 Solving the k-center problem

Let us sketch the main trick for solving the k-center problem, discussed already in Lecture 4. Recall that the problem is: given n points $p_1, \ldots, p_n$ from some metric space, we want to take k of them, $y_1, \ldots, y_k$, such the maximum distance of any p_i to its closest y_j is minimized. That is, so that each initial point is d-away from its closest y_j, for minimal d.

We observe first that if we are told the optimal distance OPT in advance, we can give a 2-approximation algorithm easily: We get the first point p_1. We ignore all subsequent points within radius $2OPT$ of it, and keep the first one that is not as a new center. We keep opening centers as necessary, and ignore all points already $2OPT$ close to one center. If OPT is achieved by some k points, we give a $2OPT$ solution with no more than k points.

Similarly, if we only have a guess g with $OPT \leq g \leq (1+\epsilon)OPT$. we can give a $2(1+\epsilon)$-approximation. When we have no guess at all, we could of course try the algorithm above in parallel all possible guesses (spacing them out by about $(1+\epsilon)$). The problem is that instances with too large a guess will use too much memory by themselves, so we have to be more cautious. We proceed as follows:

1. Look at the first $k+1$ points, and find its best k-clustering; this gives a lower bound a on OPT.

2. Run the algorithm above with $g = (1+\epsilon)^i\, a$, for $i = 0 \ldots a/\epsilon$.

If one of these $O(1/\epsilon)$ distances goes well, take the first one that goes well and we have a $2(1+\epsilon)$ approximation.

If none goes well, this means that after examining j points $p_1, \ldots, p_j$ the algorithm is trying to open a $(k+1)$-th center besides the points $y_1, \ldots, y_k$ it has already picked. We realize now that we should have picked a guess $g > a/\epsilon$.

But observe that all points $p_1, \ldots, p_j$ are within $2g$ of some y_i. The crucial claim is that by keeping only these y_i and ignoring the previous points we can still compute a reasonable approximation to the best k-clustering:

Claim 6. *If the cheapest clustering of $p_1, \ldots, p_j, p_{j+1}, \ldots, p_n$ has cost OPT, then the cheapest clustering of $y_1, \ldots, y_k, p_{j+1}, \ldots, p_n$ has cost $OPT + 2g$.*

Therefore, if we cluster $y_1, \ldots, y_k, p_{j+1}, \ldots, p_n$ we get a (roughly) $(1+\epsilon)$-approximation to the best clustering of $p_1, \ldots, p_n$.

We therefore use $y_1, \ldots y_k$ as seeds for the next iterations, using larger guesses, of the form $(g + a/\epsilon) \cdot (1+\epsilon)^i$. We can do this by recycling the space already used, rather than using new space.

This is due to McClutchin and Khuller [MK08b] and Guha [Guh09].

10.2 Summary of the Course

[Muthu's summary actually came before Andrew's items, but we thought it better to put it last—or not quite last—and what actually came dead last was the solution to the k-center problem, which we've put first. RG+KWR]

Lectures 1 and 2 were on *CM sketches*, applied to:

- point queries, such as the number m_x of times an item x appeared;
- heavy hitters, i.e. which items appear markedly frequently;
- medians and other quantiles;
- dot products;

- quantities such as $F_2 = \sum_x m_x^2$.

The primary objective was to use $O(\frac{1}{\epsilon} \log \frac{1}{\delta})$ space, where ϵ quantifies the target accuracy and δ is the probability of missing the target accuracy. Usually the accuracy involves approximating one of these statistics to within an additive term, $\pm \epsilon n$, where n is the size of the stream—or for some, with relative error within a factor of $(1+\epsilon)$. Having an extra poly-$\log(n)$ factor was generally fine, but having something like ϵ^2 in the denominator was less welcome. We saw one example later involving sampling for distinct out-neighbors in graph streams where terms like $O(\frac{1}{\epsilon^4}\sqrt{n}\, \log \frac{1}{\delta})$ (needing improvement!) entered the analysis.

Lecture 3 was on "Inverse Problems," meaning that instead of the "forward distribution" $f(x) = m_x$, one works with its "inverse" $f^{-1}(i) =$ the number of items that appear i times. A core task here is to maintain a sample S that approximates a sample S' taken uniformly over the set of *distinct* elements in the stream, even in the presence of deletions as well as insertions. Then we can estimate the same statistics as in lecture 1, though with ϵ^2 in the denominator of space usage—coming from fully storing samples of size involving that factor. A prime ingredient here is *minwise hashing*, and calculations involve estimating the number of distinct elements, represented as the zeroth moment $F_0 = \sum_x f(x)^0$.

Lecture 4 (by Andrew) focused on *graph streams*, usually presenting a (multi-)graph G as a streamed list of its edges. Estimating the number of triangles in G was niftily reduced to estimating the moments F_0, F_1, and F_2. This lecture introduced techniques for obtaining matching lower bounds via communication complexity, here by reduction from set-disjointness for the case of telling apart when G has 0 triangles from 1-or-more. Bounds did not care about polynomials in $\log n$: "$\tilde{O}$ is your friend." Another core task is retaining a set M of edges, no two sharing an endpoint (i.e. a partial matching), that maximizes some statistic, such as the number of such edges e or a weighted sum $\sum_{e \in M} w_e$. Given a new edge e' from the stream that conflicts with some edge $e \in M$, the nub is when and whether to discard e in favor of e'.

Then *geometric streams* were introduced, meaning streams of points from a general metric space, such as higher-dimensional Euclidean space. The *k-center problem* is to retain k points from the stream so as to minimize the distance from any other point to some chosen point. (Technically this means minimizing $\max_p \min_y d(p, y)$, where d is the metric and y ranges over the set of k chosen points—can we coin the word "minimaximin"?) The shortest-path metric on graphs led to merging the two ideas of streams. A key problem was finding an edge-induced subgraph H on the same vertex set V such that the distances restricted to H are within a multiplicative α factor of the distances in G. This was also a case where allowing multiple passes over the stream gives notably better results.

Lectures 5 and 6 covered sparse approximation problems and the interesting, historically central notion of *Haar wavelets*. The goal is to approximate a signal (represented as a vector) with high fidelity using as few terms as possible over the Haar basis. This connected to the hot topic of *compressed sensing*, in which we are still trying to represent a vector by a small number of terms over a basis, but the goal is to do so with as few (physical) measurements as possible. Compressed sensing is more than just a theory of linear projections, because not all projections are the same—they have different costs. In a digital camera, each pixel performs a measurement, but generally each measurement involves some cost to the system—hence ideas from compressed sensing en-

gendered the technological goal of a "Single-Pixel Camera." It is cool that considerations from streaming played a role in the development of this field.

Lecture 7 (by Andrew) presented the random-order stream model and multi-pass algorithms. In the random-order model, the underlying input (say, edges in a graph) is chosen adversarily but items are presented in a (uniformly) random order. In both models (random order and multipass) some problems such as the median become slightly easier, but they don't help much for the frequency moments $F_0, F_1, F-2, \ldots$. We presented some lower bounds in the multipass model by reducing to the Index Problem and the Pointer Chasing technique.

In Lecture 8, Linear Algebra Problems, we discussed mainly the Matrix Product and the Least Squares problems. For the matrix product, we saw that keeping CM-sketches for the matrix rows, and multiplying the sketches when needed, gave us an approximation with memory linear rather than quadratic in n. For the least squares problem, we saw that keeping CM-vectors led to a good approximation algorithm not only in the streaming model but also in the standard sense.

Lecture 9 presented Map-Reduce and discussed distributed streaming computations. The leading example was the "continuous problem" in which we have many sites and a central site that must output 1 when the number of items collected among all sites exceeds a given threshold. This led to the definition of the MUD (Massive Unordered Data) model and its relation to streaming.

10.3 Some Topics Not Covered

Finally, let us conclude by mentioning some of the many topics not covered. [Muthu only listed the following bulleted ones, but I've supplied descriptive details for each (so any bugs are on me not him), and have also added one more example at the end from a very recent paper. KWR]

- *(More) Results on Geometric Streams:* In geometric streams, the items are not single values but rather vectors in k-dimensional space. Per their main mention on p37 of [Mut09], the main effect is that instead of having a linear spectrum of values, one must also specify and maintain a partitioning of space into geometrically-determined grids, according to some metric. The resulting notion of "bucket" requires one "to maintain quite sophisticated statistics on points within each of the buckets in small space. This entails using various (non-)standard arguments and norm estimation methods within each of the buckets."

  Topics here include: *core sets*, which are relatively small subsets C of the stream point set S such that the convex hull of C is a good approximation to the convex hull of S. Again, one must specify a spatial metric to quantify the goodness of the approximation. Another is where one wishes the median in each dimension of the points in C to be a good approximation to the respective median in S; this is called the problem of *k-medians*. *Bi-chromatic matching*, where one is given equal-sized point sets A, B and needs to find a bijection $f : A \longrightarrow B$ that optimizes some metric-dependent cost function on the pairs $(a, f(a))$, also depends delicately on the space and the metric. A 2006 presentation by S. Suri [Sur06] covers some of the technical issues.

- *String Streams.* A stream can be viewed as a string for properties that are order-dependent, such as the length of the longest common subsequence between two strings. One can also

picture streams of strings—distinguishing them from streams of k-vectors by considering problems in which the ordering of vector components matter or string-edit operations may be applied. Geometric issues also come into play here.

- *Probabilistic Streams*. This does not merely mean random presentation of the items in a stream according to some distribution, but refers to situations in which the items themselves are not determined in advance but rather drawn from a distribution D, where D may or may not be known to the algorithm. One motivation is representing streams of sensor measurements that are subject to uncertainties. One can regard D as an ensemble over possible input streams. [Our two presenters wrote a major paper on this, "Estimating Aggregate Properties on Probabilistic Streams," `http://arxiv.org/abs/cs.DS/0612031`, and then joined forces with T.S. Jayram (who initiated the topic) and E. Vee for [JMMV07].]

- *Sliding Window Models*. This can be treated as a "dynamical complexity" version of the standard streaming model. The issue is not so much what storage and other privileges may be granted to the algorithm for the last N items it sees (for some N), but more the necessity to maintain statistics on the last N items seen and update them quickly when new items are presented and older ones slide outside the window. A detailed survey is [DM07].

- *Machine Learning Applications*. Many data-mining and other learning applications must operate within the parameters of streaming: a few looks at large data, no place to store it locally. Another avenue considers not just the quality of the statistical estimates obtained, as we have mostly done here, but also their robustness when the estimates are used inside a *statistical inferencing* application. João Gama of Portugal has written and edited some papers and books on this field.

The considerations of streaming can also be applied to other computational models, for instance various kinds of protocols and proof systems. For one example, the last topic above can include analyzing the standard PAC learning model under streaming restrictions. For another, the new paper "Best-Order Streaming Model" by Atish Das Sarma, Richard Lipton, and Dampon Nahongkai [SLN09], which is publicly available from the first author's site, pictures a prover having control over the order in which the input stream is presented to the verifier. This resembles best-case models discussed above, except that the requirement that the prover cannot cheat on "no"-instances and the full dependence of the ordering on details of the input can differ from particulars of the others and their associated communication models. Consider the task of proving that a big undirected graph G with vertices labeled $1, \ldots, n$ has a perfect matching. In the "yes" case, the prover orders the stream to begin with the $n/2$ edges of a perfect matching, then sends a separator symbol, and then sends the rest of the graph. The verifier still needs to check that the n-many vertex labels seen before the separator are all distinct, indeed fill out $1, \ldots, n$. They give a randomized protocol needing only $O(\log n)$ space, but show by reduction from a lower bound for set-disjointness in a variant-of-best-case communication model that any deterministic verifier needs $\Omega(n)$ space (for graphs presented as streams of edges). For graph connectivity, which has $\Omega(n)$ space bounds even for randomized algorithms in worst-case and random-case streaming models, they give an $O(\log^2 n)$-space best-order proof system. For non-bipartiteness, the simple idea is to begin with an

odd cycle, but proving bipartiteness space-efficiently remains an open problem in their model. The motivation comes from "cloud-computing" situations in which it is reasonable to suppose that the server has the knowledge and computational resources needed to optimize the order of presentation of the data to best advantage for the verifier or learner. Whether we have optimized our notes stream for learning the material is left for you to decide!

Bibliography

[Bas08] Surender Baswana. Streaming algorithm for graph spanners - single pass and constant processing time per edge. *Inf. Process. Lett.*, 106(3):110–114, 2008.

[BC09] Joshua Brody and Amit Chakrabarti. A multi-round communication lower bound for gap hamming and some consequences. *CoRR abs/0902.2399:*, 2009.

[CCM07] Amit Chakrabarti, Graham Cormode, and Andrew McGregor. A near-optimal algorithm for computing the entropy of a stream. In *Proc. SODA'07*, volume SIAM Press, pages 328–335, 2007.

[CCM08] Amit Chakrabarti, Graham Cormode, and Andrew McGregor. Robust lower bounds for communication and stream computation. In *STOC*, pages 641–650, 2008.

[CM04] Graham Cormode and S. Muthukrishnan. An improved data stream summary: The count-min sketch and its applications. *J. Algorithms*, 55:29–38, 2004.

[CMY08] G. Cormode, S. Muthukrishnan, and K. Yi. Algorithms for distributed functional monitoring. In *19th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1076–1085, 2008.

[DM07] Mayur Datar and Rajeev Motwani. The sliding-window computation model and results. In *Advances in Database Systems*. Springer US, 2007. Chapter 8.

[Elk07] Michael Elkin. Streaming and fully dynamic centralized algorithms for constructing and maintaining sparse spanners. In *ICALP*, pages 716–727, 2007.

[FKM+05] Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian Zhang. On graph problems in a semi-streaming model. *Theor. Comput. Sci.*, 348(2-3):207–216, 2005.

[FMS+08] J. Feldman, S. Muthukrishnan, A. Sidiropoulos, C. Stein, and Z. Svitkina. On the complexity of processing massive, unordered, distributed data. In *19th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 710–719, 2008.

[GK01] Michael Greenwald and Sanjeev Khanna. Space-efficient online computation of quantile summaries. In *SIGMOD Conference*, pages 58–66, 2001.

[GM09] Sudipto Guha and Andrew McGregor. Stream order and order statistics: Quantile estimation in random-order streams. *SIAM Journal on Computing*, 38(5):2044–2059, 2009.

[Guh09] Sudipto Guha. Tight results for summarizing data streams. In *Proc. ICDT'09*, 2009.

[JKS35] T. S. Jayram, Ravi Kumar, and D. Sivakumar. The one-way communication complexity of hamming distance. *4*, 2008:Theory of Computing, 129–135.

[JMMV07] T.S. Jayram, A. McGregor, S. Muthukrishnan, and E. Vee. Estimating statistical aggregates on probabilistic data streams. In *Proceedings of PODS'07*, 2007.

[LNVZ06] David Liben-Nowell, Erik Vee, and An Zhu. Finding longest increasing and common subsequences in streaming data. *J. Comb. Optim.*, 11(2):155–175, 2006.

[McG05] A. McGregor. Finding graph matchings in data stream. In *Proceedings of the 8th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems*, pages 170–181, 2005.

[MK08a] Richard Matthew McCutchen and Samir Khuller. Streaming algorithms for k-center clustering with outliers and with anonymity. In *APPROX-RANDOM*, pages 165–178, 2008.

[MK08b] Richard Matthew McCutchen and Samir Khuller. Streaming algorithms for k-center clustering with outliers and with anonymity. In *Proc. APPROX-RANDOM 2008*, pages 165–178, 2008.

[Mut09] S. Muthu Muthukrishnan. Data Streams: Algorithms and Applications, 2009.

[NW93] Noam Nisan and Avi Wigderson. Rounds in communication complexity revisited. *SIAM J. Comput.*, 22(1):211–219, 1993.

[Raz92] Alexander A. Razborov. On the distributional complexity of disjointness. *Theor. Comput. Sci.*, 106(2):385–390, 1992.

[Sar09] Artish Das Sarma. Distributed streaming: The power of communication. *Manuscript*, 2009.

[SLN09] Atish Das Sarma, Richard J. Lipton, and Danupon Nanongkai. Best-order streaming model. In *Proceedings of TAMC'09*, 2009. to appear.

[Sur06] S. Suri. Fishing for patterns in (shallow) geometric streams, 2006. Presentation, 2006 IIT Kanpur Workshop on Algorithms for Data Streams.

[Zel08] Mariano Zelke. Weighted matching in the semi-streaming model. In *25th International Symposium on Theoretical Aspects of Computer Science (STACS 2008)*, pages 669–680, 2008.