CS 466/666: Algorithm Design and Analysis **University of Waterloo: Fall 2023**

Lecture 14

October 30, 2023

Instructor: Sepehr Assadi

Topics of this Lecture

1 Graph Sketching: Spanning Forests

Consider the following problem:

- We have an undirected graph $G = (V, E)$ and our goal is to find any spanning forest of G (So far, the problem is pretty easy, just run DFS or BFS, right? But, the setting is quite different actually).
- We do not have access to the whole graph. Instead, there is one player for each vertex v denoted by P_v. The player P_v can only sees the list of neighbors of the vertex v as a set $\{u \mid u \in N(v)\}$. We assume the vertices are simply $V := \{1, \ldots, n\}$ and n is known to all players.
- The goal for the players is to *simultaneously*, each send a message to a central referee/coordinator, and the referee outputs the spanning forest. The message of each player is only a function of the input of the player (Still, the problem is still pretty easy, just ask each player to send all their inputs to the referee and then the referee runs DFS/BFS; there is yet another restriction though).
- The goal of the players is to *minimize* the length of the messages they sent. Specifically, we would like each message to be of length polylog (n) bits at most (the problem is no longer that easy, no?).
- To make our life (much) easier, we are going to assume that the players and the referee have access to a *shared source of randomness*, i.e., an infinite tape of uniformly 0/1 random bits chosen independent of players' inputs. The important part however is that the players and the referee can see the *same* randomness. Moreover, we require the answer to be correct with high probability, i.e., with probability $1 - 1/\text{poly}(n)$.

See Figure 1 for an illustration of the problem.

At this point, solving this problem may actually look quite hopeless. For instance, suppose G consists of two cliques on $n/2$ vertices connected by a single edge (u, v) (see Figure 2). The referee needs to know

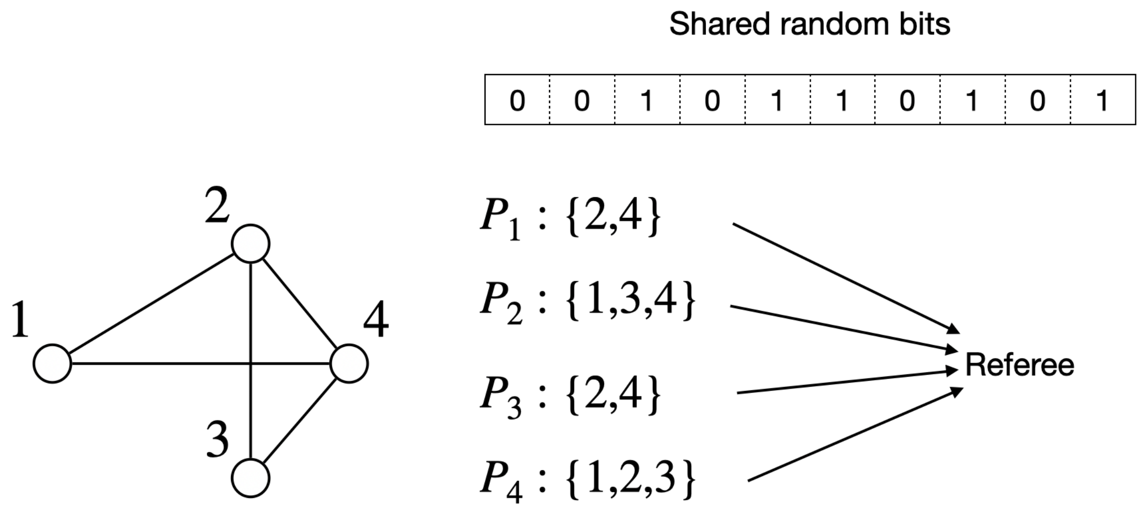


Figure 1: An illustration of the problem. We would like the messages sent by each player to be very short, i.e., polylog (n) bits.

about this edge (u, v) before it can compute any spanning forest of G, but from the perspective of players P_u and P_v, this edge is "just another" one of their edges; so, it seems impossible for them to communicate anything about this edge with messages of length $o(n)$ bits.

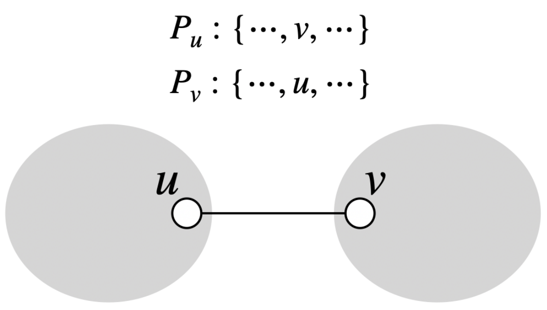


Figure 2: A "hard" example for the problem. Each gray circle denotes a clique on $n/2$ vertices. From the perspective of player P_u (resp., P_v), the vertex v (resp. u) is just another vertex in its list of neighbors so how can it inform the referee about this edge without communicating all (or at least most) of its neighbors?

The trick however, as we shall see, we can in fact get the other players also to *indirectly* convey some information about the edge (u, v) (or rather, convey something about the other edges of, say, u, so that together with the message of u, the referee can in fact recover the edge (u, v) also).

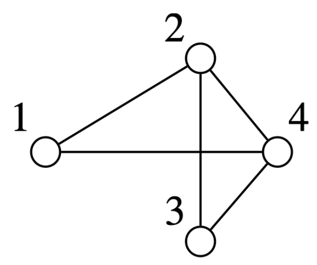
1.1 Attempt 1: A "too-good-to-be-true" approach

Let us see a first attempt on this problem that is actually not going to work, but give us a very good intuition. Fix a graph $G = (V, E)$. For every vertex $v \in V$, define the following vector $x(v) \in \{-1, 0, 1\}^{\binom{n}{2}}$:

- The entries of $x(v)$ are indexed by pairs of vertices $u < w \in V$ (recall that $V = \{1, \ldots, n\}$ so u and w are just numbers in $[n]$); we denote the set of all indices with $\binom{[n]}{2}$.
- For each index $(u, w) \in \binom{[n]}{2}$, if (u, w) is *not* incident on v (i.e., $v \notin \{u, w\}$), then $x(v)_{uw} = 0$.

Otherwise, if $v = u$, we have $x(v)_{uw} = -1$, and if $v = w$, we have $x(v)_{uw} = 1$.

See Figure 3 for an example of these vectors for the input of Figure 1.



$$x(1) := [\underbrace{-1}_{1,2}, \underbrace{0}_{1,3}, \underbrace{-1}_{1,4}, \underbrace{0}_{2,3}, \underbrace{0}_{2,4}, \underbrace{0}_{3,4}]$$

$$x(2) := [\underbrace{1}_{1,2}, \underbrace{0}_{1,3}, \underbrace{0}_{1,4}, \underbrace{-1}_{2,3}, \underbrace{-1}_{2,4}, \underbrace{0}_{3,4}]$$

$$x(3) := [\underbrace{0}_{1,2}, \underbrace{0}_{1,3}, \underbrace{0}_{1,4}, \underbrace{1}_{2,3}, \underbrace{0}_{2,4}, \underbrace{-1}_{3,4}]$$

$$x(4) := [\underbrace{0}_{1,2}, \underbrace{0}_{1,3}, \underbrace{1}_{1,4}, \underbrace{0}_{2,3}, \underbrace{1}_{2,4}, \underbrace{1}_{3,4}]$$

Figure 3: The vectors $x(1), x(2), x(3), x(4) \in \{-1, 0, 1\}^{\binom{4}{2}}$ for the graph on the left.

Note that each player P_v for $v \in V$ can create the vector $x(v)$ for its input with no communication. The following simple claim captures the main property of these vectors for our purpose.

Claim 1. *Let $S \subseteq V$ be any set of vertices and $\delta(S)$ denote the set of edges in the cut $(S, V \setminus S)$. The non-zero entries of the vector $x(S) := \sum_{v \in S} x(v)$ is precisely the set of edges in $\delta(S)$.*

Before proving Claim 1, let us check this claim for the example of Figure 3 and two arbitrarily sets S:

- For $S := \{1, 2, 3\}$, we have that $x(1) + x(2) + x(3)$ is

$$\begin{aligned} & [\underbrace{-1}_{1,2}, \underbrace{0}_{1,3}, \underbrace{-1}_{1,4}, \underbrace{0}_{2,3}, \underbrace{0}_{2,4}, \underbrace{0}_{3,4}] \\ + \; & [\underbrace{1}_{1,2}, \underbrace{0}_{1,3}, \underbrace{0}_{1,4}, \underbrace{-1}_{2,3}, \underbrace{-1}_{2,4}, \underbrace{0}_{3,4}] \\ + \; & [\underbrace{0}_{1,2}, \underbrace{0}_{1,3}, \underbrace{0}_{1,4}, \underbrace{1}_{2,3}, \underbrace{0}_{2,4}, \underbrace{-1}_{3,4}] \\ = \; & [\underbrace{0}_{1,2}, \underbrace{0}_{1,3}, \underbrace{-1}_{1,4}, \underbrace{0}_{2,3}, \underbrace{-1}_{2,4}, \underbrace{-1}_{3,4}], \end{aligned}$$

  and the non-zero entries are $\{(1, 4), (2, 4), (3, 4)\}$, precisely the edges of the cut $S = \{1, 2, 3\}$.

- For $S := \{1, 4\}$, we have that $x(1) + x(4)$ is

$$\begin{aligned} & [\underbrace{-1}_{1,2}, \underbrace{0}_{1,3}, \underbrace{-1}_{1,4}, \underbrace{0}_{2,3}, \underbrace{0}_{2,4}, \underbrace{0}_{3,4}] \\ + \; & [\underbrace{0}_{1,2}, \underbrace{0}_{1,3}, \underbrace{1}_{1,4}, \underbrace{0}_{2,3}, \underbrace{1}_{2,4}, \underbrace{1}_{3,4}] \\ = \; & [\underbrace{-1}_{1,2}, \underbrace{0}_{1,3}, \underbrace{0}_{1,4}, \underbrace{0}_{2,3}, \underbrace{1}_{2,4}, \underbrace{1}_{3,4}], \end{aligned}$$

and the non-zero entries are $\{(1,2),(2,4),(3,4)\}$, precisely the edges of the cut $S = \{1,4\}$.

Proof of Claim 1. The proof is kind of obvious by the definition and we are just going to go over it carefully for completeness.

Consider any pair $u < w$ in $\binom{[n]}{2}$. Firstly, throughout all the vectors $x(v)$ for $v \in V$, the index (u,w) is non-zero only in $x(u)$ (where $x(u)_{uw} = -1$) and $x(w)$ (where $x(w)_{uw} = 1$) and even then only if (u,w) is an edge in the graph. Now:

- If (u,w) is not an edge, then,
$$x(S)_{uw} = 0,$$
so, can consider the cases when (u,w) is an edge in the following.
- If both u, w are in S, then
$$x(S)_{uw} = \sum_{v \in S} x(v)_{uw} = x(u)_{uw} + x(w)_{uw} = -1 + 1 = 0.$$
- If neither u nor w are in S, then,
$$x(S)_{uw} = \sum_{v \in S} x(v)_{uw} = 0.$$
- If exactly one of u or w is in S, then,
$$x(S)_{uw} = \sum_{v \in S} x(v)_{uw} = \text{either } x(u)_{uw} = -1 \text{ or } x(w)_{uw} = 1.$$

So, the only non-zero entries of $x(S)$ are the edges (u,w) where exactly one of u or w is in S, precisely, the set of edges in the cut $\delta(S)$. $\square$

Okay, so now that we have these vectors how can we use them for solving our original problem? This is where we are going to make our "too-good-to-be-true" assumption:

> **Assumption 1:** *Suppose there exists a matrix A with dimensions $s \times m$ for some $s = \text{polylog}(m)$ and entries bounded by $\text{poly}(m)$ such that for all vectors $x \in \{-1,0,1\}^m$, we can recover one non-zero entry of x only given the vector $A \cdot x$ (and the original matrix A).*

As an aside, one typically refers to A as a **sketching matrix** and to $A \cdot x$ as a **linear sketch** of x or just the **sketch** of x (the term 'linear' is because we obtain the sketch via a linear projection).

Using this assumption, we can design the following algorithm.

Algorithm 1. First attempt in solving the spanning forest problem.

1. The players all pick the same matrix A of Assumption 1 for $m = \binom{n}{2}$.
2. Each player P_v for $v \in V$, sends $A \cdot x(v)$ to the referee.
3. The referee runs Boruvka's algorithm as follows:
   (a) First, the referee uses $A \cdot x(v)$ and the assumption to recover one non-zero entry from each $x(v)$, i.e., one edge incident on each vertex.
   (b) Then, the referee combine all these edges to create connected components $S_1, S_2, \ldots$.
   (c) For each connected component S now, the referee can also compute[a]

$$\sum_{v \in S} A \cdot x(v) = A \cdot \left(\sum_{v \in S} x(v) \right) = A \cdot x(S),$$

   using the linearity of A. Thus, the referee can recover one non-zero entry of each $x(S)$ for each connected component, which, by Claim 1, gives us an edge going out of each component.
   (d) This means that the referee can effectively contract each connected component into a single vertex and gets one edge out of it still. Hence, it can continue running this Boruvka-style algorithm for connectivity until it finds a spanning forest of G.

[a]This is *without* any further communication from the players and only using the sketches $\{A \cdot x(v) \mid v \in V\}$ already sent by the players.

Given that Algorithm 1 is simulating Boruvka's algorithm faithfully (as we specified how to find an outgoing edge per each contracted vertex of the graph), by the correctness of Boruvka's algorithm, we get that this algorithm also outputs a spanning forest. Moreover, by our assumption, each sketch $A \cdot x(v)$ has $s = \text{polylog}(n)$ rows with entries bounded by $\text{poly}(n)$ and so can be communicated in $\text{polylog}(n)$ bits. So, the "only" missing part is to design a proper sketching matrix that satisfies the guarantee of Assumption 1. This is where the problem happens!

Assumption 1 is too strong: Let x be any arbitrary vector in $\{-1, 0, 1\}^m$, so there are exactly 3^m choices for x. Now, suppose the matrix A in Assumption 1 exists and compute $A \cdot x$. Given *only* $A \cdot x$ (with no access to x), use the guarantee of the sketch to recover some non-zero entry x_i of x. With a slight abuse of notation, let us take x_i to be a m-dimensional vector which is all 0 except on index i which is x_i. So, now compute $A \cdot (x - x_i) = A \cdot x - A \cdot x_i$ using only the information about $A \cdot x$ (as we know A and x_i, we can compute $A \cdot x_i$ also). This allows us to recover one other non-zero entry from x (because entry x_i is now zero in $x - x_i$). Continue doing this without ever going back to x and using $A \cdot x$ to recover all of x. This means that we can recover x *fully* given only $A \cdot x$. Thus, there is a one-to-one mapping between x and $A \cdot x$ for each $x \in \{-1, 0, 1\}^m$. This implies that we need 3^m choices for $A \cdot x$ also, which means the number of bits needed to write the sketch is $m \log 3 \gg \text{polylog}(m)$.

This implies that our assumption was too strong and we cannot hope to have such a linear sketch A.

1.2 Attempt 2: A "wrong-way-of-using-randomization" approach

Let us try a somewhat different approach. What if we settle for a "weaker" sketch?

> **Assumption 2:** *Suppose there exists a distribution D over $s \times m$ matrices for some $s = \text{polylog}(m)$ and entries bounded by $\text{poly}(m)$ such that for* each *vector $x \in \{-1, 0, 1\}^m$, if we sample A from D* independent *of x, then, with high probability, we can recover one non-zero entry of x only given the vector $A \cdot x$ (and the original matrix A).*

There are two differences between this assumption and Assumption 1. Firstly, A is now chosen randomly but independent of the vector x, and secondly, the guarantee is now for each single x, if we pick A randomly independent of x, then A is going to work, i.e., we can recover one non-zero entry of x from $A \cdot x$ (although this means that for every A, there might be "many" x's that are also not good, but since we pick x and A independently, we "most likely" are not going to encounter them).

Having Assumption 2, it seems there is a natural strategy to use Algorithm 1 again. The players first use public randomness to sample A from the distribution D, and now that they all have the same matrix A, they will be able to continue running the algorithm. Moreover, since Boruvka's algorithm only needs to compute an outgoing edge from the connected components for at most

$$n + n/2 + n/4 + \cdots + 2 \leqslant 2n$$

times, we can do a union bound over all applications of the sketch A and say it did not make an error with high probability. Does this mean we are done?

The answer is unfortunately *No*, because the argument we made at the end is actually not correct. The guarantee of A is as follows: if we pick A *independent* of the input x, then, with high probability, we can recover a non-zero entry from $A \cdot x$. In our case, this is certainly true that A is independent of $x(v)$ for all $v \in V$ in the first step. But, after we compute the connected components $S_1, S_2, \cdots$, and then run $A \cdot x(S)$, we cannot actually say that A is chosen independent of $x(S)$ – the only reason we ended up with $x(S)$ was because we used A already to pick edges that connect S! So, there is really no guarantee that $A \cdot x(S)$ is actually going to work as we needed it, i.e., it can fail almost all the time, and this is in no contradiction with our assumption. Thus, the above algorithm fails also.

Remark. This is an important point that is worth repeating again. In general, whenever we work with a *randomized* algorithm, the guarantee of the algorithm only holds in the setting when the randomness of the algorithm is chosen *independent* of the input (there are some specific randomized algorithms that are often called "adversarially robust" and can handle certain scenarios when tge input is not independent of the random bits but that is a special case and beyond the scope of our course).

Before we move to a third attempt (which is actually going to work), let us mention that Assumption 2 is indeed valid, as in, we can construct such matrices. As an aside, one refers to such a sketch as an **ℓ_0-sampler**, since it can sample a non-zero entry from x, i.e., sample an entry from the support of the vector x whose size is denoted by $\ell_0(x) := \|x\|_0$. We mention the following theorem here and postpone its proof to the end of this lecture.

Theorem 2. *For every $M \geqslant 1$ and sufficiently large $m \in \mathbb{N}$, there exists a distribution D such that the following is true. For* each *vector $x \in \{-M, \ldots, 0, \ldots, M\}^m$, if we sample A from D* independent *of x, then, with high probability, using only $A \cdot x$ and A, we can recover a pair (i, x_i) such that i is chosen uniformly at random from the support of x and $x_i \neq 0$ is its value. Moreover, $A \cdot x$ has size $O(\log^4(mM))$ bits.*

1.3 Attempt 3: The final (and correct) approach

In light of Theorem 2, Assumption 2 is indeed valid. But, we need to fix the problem about the independence of input and randomness. However, this has an easy fix actually by recalling that Boruvka's algorithm only involves running $O(\log n)$ rounds of contraction in parallel. This is the new algorithm now.

Algorithm 2.

1. Let $t = \log n$. The players sample matrices $A_1, \dots, A_t$ of Theorem 2 from the distribution D independently for each $i \in [t]$ using shared randomness (for $m = \binom{n}{2}$ and $M = 1$).
2. Each player P_v for $v \in V$, sends $A_1 \cdot x(v), A_2 \cdot x(v), \cdots, A_t \cdot x(v)$ to the referee.
3. For round $i = 1$ to t of Boruvka's algorithm, the referee does the following:
   (a) Let $S_1, S_2, \cdots$ be the connected components at the beginning of this round.
   (b) For every S_j, compute
   $$\sum_{v \in S_j} A_i \cdot x(v) = A_i \cdot \left(\sum_{v \in S_j} x(v) \right) = A_i \cdot x(S_j).$$
   Use the recovery step of Theorem 2 to find one non-zero entry of $A_i \cdot x(S_j)$ which is an edge of the cut $\delta(S_j)$ by Claim 1.
   (c) Use these edges to connect as many of the connected components together as possible. First, the referee uses $A \cdot x(v)$ and the assumption to recover one non-zero entry from each $x(v)$, i.e., one edge incident on each vertex.
   (d) Once round i finishes, throw out all sketches $A_i \cdot x(v)$ and never use them again.
4. At the end, the referee outputs the spanning forest found by the Boruvka's algorithm.

Why does this algorithm work? Well, whenever we apply $A_i \cdot x(S_j)$, S_j is only a function of the matrices $A_1, \dots, A_{i-1}$ and A_i is chosen independent of them, thus, A_i is independent of $x(S_j)$ also. Given this independence, we can apply Theorem 2 to say that with high probability, we will indeed get a correct edge from $\delta(S_j)$. A union bound over the $O(n)$ cuts used by the algorithm now concludes the correctness of the algorithm with high probability. Finally, the bound on the message lengths follow because each $A_i \cdot x(v)$ requires $O(\log^3 n)$ bits and we are calculating $O(\log n)$ of them, so $O(\log^4 n)$ bits in total.

2 The ℓ_0-Sampler Construction: Proof of Theorem 2

Let us now prove Theorem 2 which is the key component of Algorithm 2. Note that this theorem is quite standalone and has nothing to do with our original graph problem any more. We will prove the theorem in a couple of steps.

Step 1: When $\|x\|_0 = 1$. In this case, there is a very simple solution. Let

$$a_1 := [1, 1, 1, \dots, 1]$$
$$a_2 := [1, 2, 3, \dots, m]$$
$$A_1 := \begin{bmatrix} a_1 \\ a_2 \end{bmatrix}.$$

Then, if the only non-zero entry of x is x_i, we will get,

$$A_1 \cdot x = \begin{bmatrix} \langle a_1 \, , \, x \rangle \\ \\ \langle a_2 \, , \, x \rangle \end{bmatrix} = \begin{bmatrix} \sum_{j=1}^{m} 1 \cdot x_j \\ \\ \sum_{j=1}^{m} j \cdot x_j \end{bmatrix} = \begin{bmatrix} x_i \\ \\ i \cdot x_i \end{bmatrix}.$$

Thus, if we denote $A_1 \cdot x = [b_1; b_2]$, we get that b_1 is the value of x_i and b_2/b_1 is the index of i. I.e., the output should be $(b_2/b_1, b_1) = (i, x_i)$.

In this case, the matrix A_1 has two rows and the vector $A_1 \cdot x$ has size $O(\log(mM))$ bits.

Step 2: How to test if $\|x\|_0 = 1$? Step 1 implies that if $\|x\|_0 = 1$, then we can solve the problem quite easily. But, can we even check if we are in this "good" case? We will do so using a simple randomized strategy.

Sample a $3 \times m$ dimensional matrix A where every column has exactly one 1 and two 0's and the position of the 1 is chosen independently and uniformly at random. Consider $A \cdot x = [b_1; b_2; b_3]$.

Suppose first that $\|x\|_0 = 1$. Then, exactly one of b_1 or b_2 or b_3 is non-zero. In particular, if i is the index of the non-zero entry of x, then, the row of A on column i that has the value 1 corresponds to the non-zero entry in $A \cdot x$ also.

Now, suppose that $\|x\|_0 \geqslant 2$. Let $i \neq j$ be two non-zero indices from x. Define the vector $y(i)$ where $y(i)_i = x_i$ and $y(i)$ is zero everywhere else. Define $y(j)$ similarly with $y(j)_j = x_j$ and zero everywhere else. And define $y = x - y(i) - y(j)$. Fix all columns of A other that i-th and j-th column, which also fixes the value of $A \cdot y = [c_1; c_2; c_3]$. We now have,

$$A \cdot x = A \cdot y + A \cdot y(i) + A \cdot y(j) = \begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix} + \begin{bmatrix} i_1 \\ i_2 \\ i_3 \end{bmatrix} + \begin{bmatrix} j_1 \\ j_2 \\ j_3 \end{bmatrix},$$

where exactly one of i_1, i_2, i_3 (resp, j_1, j_2, j_3) is non-zero depending which row of A in the column i (resp. the column j) receives the value of 1. Note that these choices are still independent and uniform at random for both i and j.

Claim 3. *In this case, with probability at least $1/9$, at least two values in b_1, b_2, b_3 are non-zero.*

Proof. We say the event 'success' has happened if at least two-values in b_1, b_2, b_3 are non-zero. We consider different cases based on c_1, c_2, c_3.

- All of c_1, c_2, c_3 are equal to zero: In this case, if 1 of column i is different than 1 of column j, then, two different values in b_1, b_2, b_3 are going to end up non-zero. Thus, the probability of success is:

$$\Pr(\text{success}) \geqslant \Pr(1 \text{ of column } i \text{ is different from } 1 \text{ of column } j) = \frac{2}{3} > \frac{1}{9}.$$

- Two of c_1, c_2, c_3 are equal to zero: by symmetry, let us assume $c_1 = c_2 = 0$ and $c_3 \neq 0$. If 1 column i is different from 1 of column j and both are different from 3, then, all three of b_1, b_2, b_3 will be non-zero and success happens. Thus,

$$\Pr(\text{success}) \geqslant \Pr(1 \text{ of column } i \text{ 1 and 1 of column } j \text{ is 2 or vice versa}) = 2 \cdot \frac{1}{9} > \frac{1}{9}.$$

- One of c_1, c_2, c_3 is equal to zero: by symmetry, let us assume $c_1 = 0$ and $c_2, c_3 \neq 0$. If 1 of column i and 1 of column j both go to index 1, then, both of b_2, b_3 remain non-zero also. Thus,

$$\Pr(\text{success}) \geqslant \Pr(1 \text{ of column } i \text{ 1 and 1 of column } j \text{ both go to index 1}) = \frac{1}{9}.$$

- Finally, none of c_1, c_2, c_3 are zero. In this case, as long as 1 of column i and 1 of column j go to the same index, then both of the remaining indices remain non-zero in b_1, b_2, b_3. Thus,

$$\Pr(\text{success}) \geqslant \Pr(1 \text{ of column } i \text{ 1 and 1 of column } j \text{ go to the same index}) = \frac{1}{3} > \frac{1}{9}.$$

This concludes the proof. □

Finally, in the case that $\|x\|_0 = 0$, we always have $A \cdot x = 0$.

Putting these results together, we have the following *distinguisher*: if $[b_1, b_2, b_3]$ has exactly one non-zero entry, we consider it a $\|x\|_0 = 1$ case, and otherwise, we consider the input not having this property. Now, suppose we pick a matrix A_2 which consists of $t = 90 \ln m$ independent copies of the matrix A. When $\|x\|_0 = 1$, *all* the copies return 'one non-zero entry' case, while when $\|x\|_0 = 1$, the probability of this even happening is, by Claim 3, at most

$$\left(1 - \frac{1}{9}\right)^{90 \ln m} \leqslant \exp\left(-10 \ln m\right) = m^{-10}.$$

So, with high probability, we can solve the problem correctly in this case.

In this case, the matrix A_2 has $270 \ln m$ rows and the vector $A_2 \cdot x$ has size $O(\log^2(mM))$ bits.

Step 3: when $2^k \leqslant \|x\|_0 \leqslant 2^{k+1}$ for some given $k \geqslant 0$. We will attempt to reduce this case to the case of step 1 while also running the step 2 *test* in parallel to make sure the reduction indeed worked correctly. First pick a *diagonal* matrix B with dimensions $m \times m$ where each entry B_{ii} for $i \in [m]$ is 1 with probability $1/2^{k+1}$ and 0 otherwise. Let $y := B \cdot x$.

Claim 4. *With probability at least* $1/8$, $\|y\|_0 = 1$.

Proof. Let $s = \|x\|_0$. For $\|y\|_0 = 1$, we need *exactly* one of the indices i in the support of x to be sampled on the diagonal of B. Thus,

$$\Pr\left(\|y\|_0 = 1\right) = s \cdot \frac{1}{2^{k+1}} \cdot \left(1 - \frac{1}{2^{k+1}}\right)^{s-1} \geqslant \frac{1}{2} \cdot \left(1 - \frac{1}{2^{k+1}}\right)^{2^{k+1}} > \frac{1}{2 \cdot e} > \frac{1}{8}.$$

□

Now, use the matrices A_1 and A_2 from the previous step to create the matrix

$$A := \begin{bmatrix} A_1 \\ A_2 \end{bmatrix} \cdot B.$$

We have

$$A \cdot x = \begin{bmatrix} A_1 \\ A_2 \end{bmatrix} \cdot B \cdot x = \begin{bmatrix} A_1 \cdot y \\ A_2 \cdot y \end{bmatrix}.$$

Thus, *if* y has exactly one non-zero entry, then, we can recover that non-zero entry from $A_1 \cdot y$ and $A_2 \cdot y$ allows us to test with high probability that $\|y\|_0 = 1$ or not.

The final matrix we create in this step is then consists of copying A $80 \ln m$ times independently on top of each other to get a matrix A_3. The probability that A_3 cannot recover an index because non of the y-vectors it creates have $\|y\|_0 = 1$ is at most (by Claim 4):

$$\left(1 - \frac{1}{8}\right)^{80 \ln m} \leqslant \exp\left(-10 \ln m\right) = m^{-10}.$$

Moreover, since there are only $O(\log m)$ 'tests' for $\|y\|_0 = 1$, with high probability, all of them succeeds and thus the algorithm does not return a wrong answer.

In this case, the matrix A_3 has $O(\log^2 m)$ rows and the vector $A_3 \cdot x$ has size $O(\log^3(mM))$ bits.

Step 4: arbitrary x. We can now solve the general problem. While we do not know the value of k, there are only $\log m$ different choices for it for choices of k in $1 \leqslant 2^k \leqslant m$. So, we simply create matrices A'_k for $k = 1$ to $\log m$, where each A'_k is the matrix we created in step 3 for the value of k. We then set A_4 to be these matrices stacked on top of each other.

For the "correct" choice of k, the matrix A'_k, by the argument in step 3 returns a correct solution with high probability. For all the other indices, we never run the matrix A_1 and its recovery phase before getting ensured by matrix A_2 that the number of non-zero entries passed to A_1 is only 1, hence, with high probability, we will never return a wrong answer.

Finally, the matrix A_4 has $O(\log^3 m)$ rows and the vector $A_3 \cdot x$ has size $O(\log^4 (mM))$ bits.

This concludes the proof of Theorem 2.