CS 514: Advanced Algorithms II – Sublinear Algorithms **Rutgers: Spring 2020**

Lecture 6

February 25, 2020

Instructor: Sepehr Assadi *Scribe: Harsha Tirumala*

Disclaimer: *These notes have not been subjected to the usual scrutiny reserved for formal publications. They may be distributed outside this class only with the permission of the Instructor.*

1 Compressed Sensing

In this lecture, we are going to look at a somewhat different model of sublinear algorithms, that is recovering data via *global* queries (with applications to (i) sparse recovery in compressed sensing and (ii) cut queries in graph problems). This is will also serve as a great bridge between sublinear time algorithms we studied so far and streaming algorithms that we will cover from the next lecture.

Motivation Behind Compressed Sensing

Data analysis is most interesting when the data has structure – there is not much one can do with an unstructured (think of "random") data. But how can we exploit the structure of data? Even before that, what do we mean by "structure" in data?

Structure can mean many things. One major theme in data analysis is *sparsity* – for now, think of sparse data as something that can be compressed much further without losing most of its functionality. A good example is a digital image: one can use standard image compression algorithms (e.g., JPEG) to dramatically reduce the size of the image without changing the visibility of the picture almost at all. See the figure below for instance that shows an almost 20x compression without much difference:

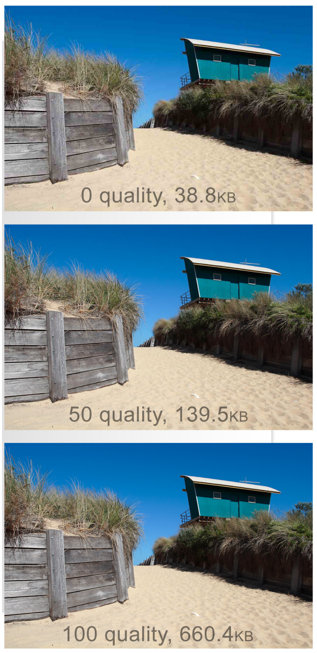

Figure 1: https://en.wikipedia.org/wiki/File:Quality_comparison_jpg_vs_saveforweb.jpg

The usual approach to compressing (approximately) sparse data is to first collect the raw data, and then to compress it in software (in the example of an image, the camera first captures an image in the standard pixel basis and only then compress it further via, say, JPEG algorithm). This two-step approach works fine in many applications, but in terms of efficiency, this seems quite counterintuitive: If most of the raw data is going to be thrown out immediately anyway, why did we bother to collect it all in the first place?

The main idea of *compressed sensing* or "compressive sensing" is to *directly* capture data in a compressed form. To put this in the context digital cameras, using compressed sensing we can now use fewer pixels while capturing the photo without degrading image quality, resulting in qualitatively less battery drain for a given number of photos. Another example is in MRI machines. In these machines, the scan time is proportional to the number of measurements (or "slices") taken, and can easily be 30 minutes or more. Compressed sensing techniques have been tried out in some hospitals, and they have sped up scan times by a constant factor.

A Concrete Example: Sparse Recovery

In this lecture, we are going to consider the problem of *sparse recovery* in compressed sensing, which we now define formally. Suppose our target data—often called the *signal*—is a real-valued vector $x \in \mathbb{R}^n$. We have "access" to this data using *linear* measurements. I.e., we can pick a vector $a \in \mathbb{R}^n$ and *observe* $\langle a, x \rangle$. Our goal is then to recover the signal x using a *minimal* number of measurements. Formally,

Step 1: **Design** m linear measurements $a_1, \ldots, a_m \in \mathbb{R}^m$ or equivalently a matrix $A \in \mathbb{R}^{m \times n}$.

Step 2: An unknown signal $x \in \mathbb{R}^n$ is now picked as input.

Step 3: Receive the measurement results $b = \langle a_1, x \rangle, \ldots, \langle a_m, x \rangle$ or equivalently $b := A \cdot x$.

Step 4: **Recover** the unknown signal x from the measurement signal b.

In the setting above, our task as an algorithm designer is to design a suitable *measurement matrix* A and a *recovery algorithm* that allows for recovering x from $A \cdot x$. Note that at this stage, this is an *information theoretic* question not a computational one; in other words, we can, *for now*, ignore the runtime of algorithms for constructing A and recovering x from $A \cdot x$, and solely focus on designing smallest number of measurements possible, i.e., minimize m.

At this level, the problem above has a straightforward solution: set $A = I_n$ to get $b = I_n x = x$ which obviously allows us to recover $x \in \mathbb{R}^n$. This gives us an upper bound of n measurements for recovering x. We also have a lower bound of n measurements: Any system of k equations $A_{k \times n} \cdot x = b$ where $k < n$ is undetermined and thus multiple x satisfy this system, making unique recovery of x from $A \cdot x$ impossible.

However, recall our earlier discussion in compressed sensing. We are typically not interested in recovering arbitrary signals but rather ones with "structure". In particular, in the **sparse recovery** problem, we are guaranteed that the vector x is sparse, namely, has a small number of non-zero entries. Formally, we say that a vector $x \in \mathbb{R}^n$ is k-sparse iff $||x||_0 = k$, i.e., there are only k non-zero entries in x. The k-sparse recovery problem is then defined as follows.

Problem 1 (**Real-Valued Sparse Recovery**)**.** Given parameters $n, k > 0$, *design* a minimal set of measurements $A \in \mathbb{R}^{m \times n}$ such that *for all* k-sparse vectors $x \in \mathbb{R}^n$, x can be uniquely *recovered* from $A \cdot x$.

Note that the lower bound of n on number of measurements no longer hold for this problem because we only need the mapping $x \mapsto A \cdot x$ to be injective for k-sparse vectors and not all vectors in $\mathbb{R}^n$. We are *not* going to consider this problem as it is somewhat beyond the scope of our course and instead we will consider a *discrete* version of it in the next section (which will also have applications to later parts of the course). The optimal solution to this problem needs only $\Theta(k \cdot \log(n/k))$ measurements (which are also known to be necessary) and was presented in [2] and [3]. We shall also note that these results extend to the more practical setting of approximately recovering an *approximately* k-sparse vector[1].

[1]Informally speaking, we say a vector y is k-sparse if $y = x + \sigma$ for a k-sparse vector x and a "noise" signal σ with small ℓ_1 or ℓ_2 (or some other) norms.

Remark. In Problem 2, we are designing the measurement matrix A *before* x is even chosen and thus A should work simultaneously for all x. This is often referred to as the **for-all** guarantee in the literature. A "weaker" version of this problem is to design A to recover a *single unknown* x which is chosen independent of A – this weaker guarantee is referred to as the **for-each** guarantee.[a]

[a]This can only make a difference when we use randomization: the for-all guarantee says that the answer is correct simultaneously for all k-sparse x with some fixed probability, while for-each guarantee says that for any one (unknown) signal the answer is correct with fixed probability.

2 Discrete Sparse Recovery

We are now going to consider a discrete version of sparse recovery. This is both to gives us some intuition about the real-valued sparse recovery in Problem 2 and also for its applications to the remainder of the course. By discrete sparse recovery, we mean recovery over finite fields and in particular field $\mathbb{F}_2$[2].

Problem 2 (Discrete Sparse Recovery on $\mathbb{F}_2$). Given parameters $n, k > 0$, *design* a minimal set of measurements $A \in \mathbb{F}_2^{m \times n}$ such that *for all* k-sparse vectors $x \in \mathbb{F}_2^n$, x can be uniquely *recovered* from $A \cdot x$.

Before getting to an upper bound for this problem, let us first see what is a natural lower bound on number of measurements. By using m measurements, the set of vectors b that can be given as answer to $A \cdot x$ is 2^m as there are only 2^m m-dimensional vectors in $\mathbb{F}_2^m$. At the same time, we should make sure that for any two different $x \neq y \in \mathbb{F}_2^n$, $A \cdot x \neq A \cdot y$ as otherwise we cannot distinguish x and y from the resulting vector b. Since the number of k-sparse vectors in $\mathbb{F}_2^n$ is $\binom{n}{k}$, we should have,

$$2^m \geq \binom{n}{k} \geq \left(\frac{n}{k}\right)^k \implies m \geq k \cdot \log\left(\frac{n}{k}\right). \qquad \text{(for all } a, b,\ \tbinom{a}{b} \geq \left(\tfrac{a}{b}\right)^b\text{)}$$

As such, $k \cdot \log\left(\frac{n}{k}\right)$ measurements are necessary for this problem.

The natural question at this point is that whether can match this lower bound by an algorithm also, i.e., design $O(k \cdot \log\left(\frac{n}{k}\right))$ measurements that allows us to recover any k-sparse vectors in $\mathbb{F}_2^n$. We will do so in the following.

Warm-Up: $k = 1$ Case

Consider the problem of recovering *1-sparse* vector $x \in \mathbb{F}_2$. We are going to design $O(\log n)$ measurements for this problem.

For simplicity, we assume n is a power of 2. Consider the set of vectors a_i for $i \in [\log n]$, where:

– a_1 has all 1's in the first $n/2$ coordinates and zero outside.

– a_2 has all 1's in the first $n/4$ and third $n/4$ coordinates and zero outside.

– $\cdots$

– a_i have $n/2^i$ 1's, followed by $n/2^i$ zeros, followed by $n/2^i$ 1's and so on and so forth.

See Figure 2 for an illustration.

This forms the measurement matrix A. We now design the recovery algorithm. Note that A is basically simulating a binary search (suppose j is the index where $x_j = 1$):

– $\langle a_1, x\rangle = 1$ implies that index j belongs to $[1 : n/2]$ and 0 means it belongs to the rest.

[2]Recall that $\mathbb{F}_2$ is the field on $\{0, 1\}$ where computation is mod 2, or equivalently addition is replaced with XOR-operation.

- $\langle a_2, x\rangle = 1$ implies that index j belongs to $[1 : n/4] \cup [n/2 + 1 : 3n/4]$ and 0 means it belongs to the rest. Combined with the answer to the above, this identifies an interval of length $n/4$ that j belongs to.
- $\cdots$
- In general, by considering first i rows, we can identify an interval of length $n/2^i$ that contains j.

This way, after $\log n$ rows, we can uniquely recover index j, giving us our recovery algorithm.

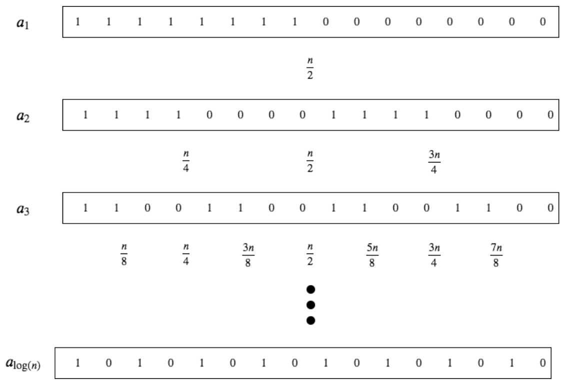

Figure 2: $\log(n)$ vectors in $\mathbb{F}_2^n$ for recovering *1-sparse* vectors

General Case

We are now going to solve the general case of the problem. Extending the binary search approach for $k = 1$ case to general k seems challenging so we will use another approach based on probabilistic arguments.

Theorem 1. *For any $n, k > 0$, there exists a set of $m = O(k \log(\frac{n}{k}))$ measurements $A \in \mathbb{F}_2^{m\times n}$ for recovering any* k-sparse *vector $x \in \mathbb{F}_2^n$.*

This proof is based on the *probabilistic method* which relies on a very basic principle: suppose we sample an object randomly and can show that with *non zero* probability this object satisfies some desired properties; this effectively means that there exists at least one object that satisfies the desired properties, hence proving its existence. In the context of our problem, we are going to pick $A \in \mathbb{F}_2^{m\times n}$ randomly and show that with non-zero probability we can recover x from $A \cdot x$ for all $x \in \mathbb{F}_2^n$. This will prove the existence of a suitable measurement matrix A.

We start with the following claim about taking inner product of a random vector $a \in \mathbb{F}_2^n$ with two *distinct* vectors $x, y \in \mathbb{F}_2^n$. In the following, we use the notation $a \in_R \mathbb{F}_2^n$ to mean that a is chosen uniformly at random from $\mathbb{F}_2^n$.

Claim 2. *For all $x \neq y \in \mathbb{F}_2^n$, and $a \in_R \mathbb{F}_2^n$, $\Pr_a(\langle a, x\rangle = \langle a, y\rangle) = 1/2$.*

Proof. For any index $i \in [n]$ and vector $z \in \mathbb{F}_2^n$, define $z_{-i} = (z_1, z_2, .., z_{i-1}, z_{i+1}, ..z_n)$. Let i be any index where $x_i \neq y_i$. Consider the following process of picking a: first pick $a_{-i} \in \mathbb{F}_2^{n-1}$ and then pick $a_i \in \mathbb{F}_2$.

<u>*Case 1*</u>: $\langle a_{-i}, x_{-i}\rangle = \langle a_{-i}, y_{-i}\rangle$: If $a_i = 1$, then $\langle a, x\rangle \neq \langle a, y\rangle$. So, conditioned on this case, with probability $\frac{1}{2}$ we have $\langle a, x\rangle \neq \langle a, y\rangle$.

<u>*Case 2*</u>: $\langle a_{-i}, x_{-i}\rangle \neq \langle a_{-i}, y_{-i}\rangle$: If $a_i = 0$, then $\langle a, x\rangle \neq \langle a, y\rangle$. So, conditioned on this case, with probability $\frac{1}{2}$ we have $\langle a, x\rangle \neq \langle a, y\rangle$.

This proves the claim. $\square$

As a corollary of Claim 2, we have that a random matrix can distinguish x and y with "high enough" probability.

Claim 3. *For all $x \neq y \in \mathbb{F}_2^n$, and $A \in_R \mathbb{F}_2^{m \times n}$, $\Pr_A(A \cdot x = A \cdot y) = 1/2^m$.*

Proof. Follows immediately from Claim 2 and independence of the m rows of A. □

Finally, we can use Claim 3 to argue that with non-zero probability, A can distinguish between all pairs of k-sparse vectors $x, y \in \mathbb{F}_2^n$.

Claim 4. *For $m = 2k \log\left(\frac{en}{k}\right)$ and $A \in_R \mathbb{F}_2^{m \times n}$,*

$$\Pr_A\left(\forall k\text{-sparse } x \neq y \in {\mathbb{F}_2}^n \quad Ax \neq Ay\right) > 0.$$

Proof. Since both x,y are *k-sparse* vectors in $\mathbb{F}_2^n$, the number of distinct pairs (x, y) is less than $\binom{n}{k}^2$. Thus, by applying union bound to the events in Claim 3,

$$\Pr_A(\exists x \neq y \in \mathbb{F}_2^n : Ax = Ay) \leq \sum_{x \neq y \in \mathbb{F}_2^n} \Pr_A(Ax = Ay) < \frac{\binom{n}{k}^2}{2^m} < \frac{\left(\frac{en}{k}\right)^{2k}}{2^m} = 1. \qquad \left(\binom{a}{b} \leq \left(\frac{ea}{b}\right)^b\right)$$

As the LHS is strictly less than 1, the probability of its complement event is non-zero as desired. □

We can now conclude the proof of Theorem 1.

Proof of Theorem 1. By probabilistic method, Claim 4 implies existence of $A \in \mathbb{F}_2^{m \times n}$ for $m = O(k \log(\frac{n}{k}))$ such that $A \cdot x \neq A \cdot y$ for all k-sparse $x \neq y \in \mathbb{F}_2^n$. We use this matrix as our measurement matrix. For recovery, given $b \in \mathbb{F}_2^m$, we simply go over all $z \in \mathbb{F}_2^n$ and check whether $Az = b$ or not; as the unique answer to this system of equations (among k-sparse vectors) is $Ax = b$, we can recover x uniquely. □

We shall note that in Theorem 1 we only focused on designing minimal number of measurements and in particular ignored the runtime of algorithms. As it is, our algorithms require exponential time to choose the measurement matrix A, and more importantly, also take exponential time to do the recovery. We revisit this issue in problem set 2.

Remark. Note that the proof of Theorem 1 used k-sparsity in a very simple way by simply bounding number of vectors that we aim to do the recovery for by $\binom{n}{k}$. It is thus easy to see that we can extend this theorem for performing recovery of any subset $\mathcal{X} \subseteq \mathbb{F}_2^n$ using $O(\log |\mathcal{X}|)$ measurements.

3 Global Queries in Graphs

We can see the discrete sparse recovery problem we studied in the previous section as an alternative query model by thinking of each linear measurement as a query to the input. While in standard query complexity model and sublinear time algorithms we studied so far, each query to the input was "local", e.g., a function of a single bit of the input or neighborhood of a single vertex, a linear measurement query is a "global" function of the input that depends on many input bits[3].

In this section, we will consider another model of global queries, this time in the context of graph problems and compare and contrast their power against local queries. The global query model we will consider is the *cut query* model. To define this model, we first need a quick graph theory refresher on notion of cuts. Consider an undirected graph $G = (V, E)$. Recall that a *cut* in G is any partition $S, \bar{S}$ of vertices and the set of *cut edges* of S is denoted by $\delta(S) := \{(u, v) \in E(G) : u \in S, v \in V \setminus S\}$.

[3]There was another important distinction between the two models. In sparse recovery, all queries to the input are *non-adaptive*, i.e., are chosen in parallel at the beginning. However, typically in query complexity and sublinear time algorithms, we also allow for adaptive querying of the input and indeed this is indeed crucial in many cases.

Cut query model: In the cut query model, we have an undirected graph $G = (V, E)$ with n known vertices (e.g. $V = \{1, \ldots, n\}$) and unknown edges. Our access to the edges of the graph is via the following query:

- *Cut query:* Given a set $S \subseteq V$, return $|\delta(S)|$.

Similar to before, we are interested in solving different graph problems in this query model. In this lecture, we will consider the problem of estimating the average degree, as well as finding a spanning forest of the input in this model.

3.1 Estimating Average Degree or Number of Edges

Recall the problem of estimating average degree of a graph introduced in Lecture 2.

Problem 3 (Estimating average degree). Given a graph G in the cut query model, approximation parameter $\epsilon \in (0, 1)$, and confidence parameter $\delta \in (0, 1)$, output an approximate average degree $\tilde{d}$ such that the following holds:

$$Pr(|\tilde{d} - \bar{d}| \le \epsilon \bar{d}) \ge 1 - \delta,$$

where $\bar{d}$ is the average degree of G.

In lecture 2, we have seen that the degree estimation problem can be solved by $O(\frac{\sqrt{n}}{\epsilon^2} \ln(\frac{1}{\delta}))$ local queries in the adjacency list model assuming $m = \Omega(n)$. We will now provide an algorithm that uses only cut queries and solves this problem with much smaller number of queries. For simplicity of exposition, we consider the equivalent problem of estimating number of edges m in G – dividing the answer by $n/2$ (known to the algorithm) will give us an estimate of average degree.

Algorithm 1: A "low" variance unbiased estimator for number of edges using cut queries.

1. Sample a uniformly random cut $S \subseteq V$ and obtain $|\delta(S)|$.
2. Return $X = 2|\delta(S)|$.

We will prove that X is an unbiased estimator of m and $\mathrm{Var}[X]$ is sufficiently small.

Claim 5. $\mathbb{E}[X] = m$.

Proof. Let Y_e be the indicator random variable that takes value 1 when $e \in \delta(S)$ and 0 otherwise. As such, $X = 2\sum_{e \in E} Y_e$. Since $\Pr(Y_e = 1) = 1/2$ as $e = (u, v) \in \delta(S)$ iff $u \in S$ and $v \notin S$ or vice versa, and by linearity of expectation, we have $\mathbb{E}[X] = 2\sum_{e \in E} \mathbb{E}[Y_e] = 2 \cdot \sum_{e \in E} 1/2 = m$. □

Claim 6. $\mathrm{Var}[X] \le 2m$.

Proof. Again note that $X = 2\sum_{e \in E} Y_e$. As such,

$$\mathrm{Var}[X] = 4\mathrm{Var}\left[\sum_{e \in E} Y_e\right] = 4 \sum_{e, f \in E} \mathrm{Cov}(Y_e, Y_f), \tag{1}$$

by what we shown in previous lectures. It is easy to see that the random variables Y_e are *not* independent[4]. However, we argue that these variables are still *pairwise independent*, meaning that for any $e \neq f$, $Y_e \perp Y_f$ (meaning Y_e is independent of Y_f).

[4]For instance, having $(u, v) \in \delta(S)$ and $(v, w) \in \delta(S)$ necessarily implies $(u, w) \notin \delta(S)$.

Let $e = (u, v)$ and $f = (w, z)$. If $|\{u, v, w, z\}| = 4$, we are done by the randomness in choice of S. So we consider the case $|\{u, v, w, z\}| = 3$ (this is the only other case since G is a simple graph with no parallel edges). Without loss of generality, let us assume $u = w$, and so u, v, z are distinct. The choice of Y_e is only a function of u, v. Even conditioned on u, v in S or not, the choice of z is independent which makes choice of Y_f independent as well. In other words,

$$\Pr(Y_f = 1 \mid Y_e) = \Pr(Y_f = 0 \mid Y_e) = \frac{1}{2},$$

hence $Y_e \perp Y_e$.

Since covariance of two independent random variables is zero (as $\mathrm{Cov}(X, Y) = \mathbb{E}[XY] - \mathbb{E}[X]\mathbb{E}[Y] = \mathbb{E}[X]\mathbb{E}[Y] - \mathbb{E}[X]\mathbb{E}[Y] = 0$ for independent X, Y), by Eq (1),

$$\begin{aligned}\mathrm{Var}[X] &= 4 \sum_{e,f \in E} \mathrm{Cov}(Y_e, Y_f) = 4 \cdot \left(\sum_{e \in E} \mathrm{Var}[Y_e] + \sum_{e \neq f \in E} \mathrm{Cov}(Y_e, Y_f) \right) = 4 \sum_{e \in E} \mathrm{Var}[Y_e] \\ &\leq 4 \sum_{e \in E} \mathbb{E}\left[Y_e^2\right] = 4 \sum_{e \in E} \mathbb{E}[Y_e] = 2 \cdot m,\end{aligned}$$

concluding the proof. □

Now that we designed an unbiased estimator for m and showed it is a "low" variance, we can use our usual technique of repeating it multiple times and taking its average to get the final algorithm.

Algorithm 2: An algorithm for estimating the number of edges using cut queries.

1. Repeat Algorithm 1 for $k := \frac{8}{\epsilon^2}$ times to get estimates $X_1, X_2, X_3, \ldots X_k$.
2. Return $Z := \frac{1}{k} \sum_{i=1}^{k} X_i$.

We prove the following claim on correctness of Algorithm 2. This is a straightforward application of Chebyshev's inequality and we have already done this multiple times in the course.

Claim 7. $\Pr(|Z - m| > \varepsilon m) \leq \frac{1}{4}$.

Proof. As we have already shown several times in the course, taking average of k independent copies of a random variable, keeps the expected value unchanged while reducing the variance by a factor of k. As such,

$$\begin{aligned}\Pr(|Z - m| > \varepsilon m) &= \Pr(|Z - \mathbb{E}[Z]| > \varepsilon m) && \text{(by Claim 5)} \\ &\leq \frac{\mathrm{Var}[Z]}{\varepsilon^2 m^2} && \text{(by Chebyshev's inequality)} \\ &\leq \frac{2m}{k \cdot \varepsilon^2 m^2} && \text{(by Claim 6)} \\ &\leq \frac{1}{4},\end{aligned}$$

by the choice of k and since $m \geq 1$. □

This algorithm achieves the "right" approximation ratio but probability of success only $3/4$ (as opposed to $1 - \delta$). We can however "boost" the probability of success of the algorithm using the median trick discussed in previous lectures: run the algorithm for $O(\ln(1/\delta))$ times and return the median answer. This solves Problem 3 now yielding the following theorem.

Theorem 8. *There is an algorithm for estimating the average degree (or number of edges) of a given graph to within a $(1 \pm \varepsilon)$-approximation with probability $1-\delta$ using $O(\frac{1}{\varepsilon^2} \cdot \ln(1/\delta))$ cut queries.*

This theorem was first proved in [1] by using even "weaker" queries than cut-queries that only output whether or not there is an edge in the cut or not as opposed to returning the actual size of the cut.

Remark. This section already demonstrate the stronger power of global queries (in particular cut queries) compared to local queries. While $\Omega(\sqrt{n})$ queries are needed for estimating average degree in the adjacency list model (and thus local queries), we can estimate the average degree using cut queries in only $O(1)$ queries for constant $\varepsilon, \delta \in (0,1)$.

3.2 Finding A Spanning Forest

In Lecture 2, we have solved the problem of estimating (with confidence $(1-\delta)$) the number of connected components of a given undirected graph $G=(V,E)$ to within a $\pm\epsilon n$ additive factor using $O(\frac{1}{\epsilon^2}\ln(\frac{1}{\delta}))$ local queries (neighbor, degree queries etc) and later in Lecture 3 proved that any algorithm for deciding whether a graph is connected or not requires $\Omega(n^2)$ queries. We will now show that in contrast to this lower bound, if we use cut queries, then we can find a spanning forest of G deterministically in $O(n \log n)$ cut queries and thus solve the connectivity problem as well.

Problem 4 (Spanning Forest Problem). Given an undirected graph $G=(V,E)$ in the cut query model, output a spanning forest of G.

Before getting to design an algorithm for this problem, let us see what is a natural lower bound on number of cut queries for this problem. Let $\mathcal{T}$ denote the set of all different spanning trees on n vertices. By Cayley's theorem in graph theory, $|\mathcal{T}| = n^{n-2}$. Note that any set of queries by a deterministic algorithm should uniquely identify which tree in $\mathcal{T}$ was the output. Considering m queries would result in $(n^2)^m$ answers in total (as the value of a cut query is a number between 0 to $< n^2$), we should have that,

$$n^{2m} \geq n^{n-2} \implies m = \Omega(n).$$

As such, we need to use at least $\Omega(n)$ queries to find a spanning tree of a connected graph G. In the following, we will show that $O(n \log n)$ queries are also sufficient for this purpose. Before that, we need to first design some primitives based on cut queries.

Necessary Primitives

Note that a cut query S will determine the value of $|\delta(S)|$. In the following, we use this to design some "stronger" primitives (for instance, a one that returns a neighbor of a vertex). In the next part, we will use these to design our spanning forest algorithm.

- **Primitive 1:** Given two subsets $S,T \subseteq V$ with $S \cap T = \emptyset$, output the size of the (S,T)-cut, i.e., the *size* of $\delta(S,T) := \{(u,v) \mid u \in S \wedge v \in T \text{ or } u \in T \wedge v \in S\}$, namely, the number of edges with one endpoint in S and another in T.

We can design Primitive 1 using exactly 3 cut queries. The idea is as follows : Let $V = S \uplus T \uplus U$[5] and $a := |\delta(S,T)|$, $b := |\delta(S,U)|$, and $c := |\delta(T,U)|$. Observe that

$$|\delta(S)| = a+b, \qquad |\delta(T)| = a+b, \qquad |\delta(U)| = b+c.$$

See Figure 3 for an illustration. So we can query G for cut queries on S,T, and U and then solve the linear system to get the value of $|\delta(S,T)| := \frac{1}{2}\left(|\delta(S)| + |\delta(T)| - |\delta(U)|\right)$.

[5] $X \uplus Y$ refers to the *disjoint union* of sets X,Y.

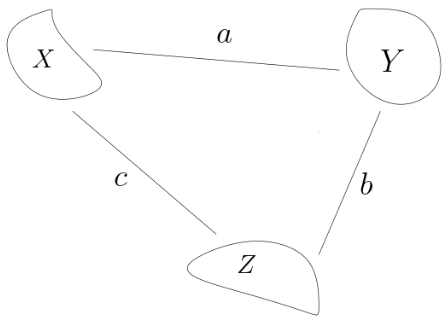

Figure 3: Edges going across subsets X,Y and Z

- **Primitive 2:** Given two subsets $S, T \subseteq V$ with $S \cap T = \emptyset$ and $|\delta(S,T)| > 0$, return an edge $e \in \delta(S,T)$.

We design Primitive 2 using $O(\log n)$ cut queries. For this, we need the following key claim.

Claim 9. *There is an algorithm using $O(\log n)$ cut queries that given a vertex $v \in V$ and a set $T \subseteq V \setminus \{v\}$, returns an edge (v,w) for some $w \in T$ if such an edge exists.*

Proof. Consider the following algorithm.

1. Let $X = T$. Use primitive 1 to check if $|\delta(\{v\}, X)| = 0$ and if so return $\perp$. Otherwise continue.
2. If $|X| = 1$, then return the edge (v,w) where w is the unique element of X. Otherwise, partition X into equal-sized sets X_1 and X_2 (± 1 elements when X is odd).
3. Use primitive 1 to check if $|\delta(\{v\}, X_1)| > 0$: if so, set $X = X_1$ and go to line (2), otherwise let $X = X_2$ and go to line (2).

The number of cut queries is $O(\log n)$ and the correctness follows because whenever we recurse on a set X, there is at least one edge in $\delta(\{v\}, X)$. ☐

To design primitive 2, we first *consolidate* all vertices in S into a single vertex v in G – this basically means that whenever we want to make a cut query, we treat all vertices of S as a singleton vertex v. By Claim 9, we can recover an edge (v,w) in the new graph. This gives us an edge with one endpoint in S (unspecified) and another endpoint $w \in T$. We then apply Claim 9 again this time by having w on one side and S on the other side to find one vertex in S that has an edge to w (which we know it should exist). This gives us an $O(\log n)$ cut query implementation of Primitive 2.

Spanning Forest via Cut Queries

We now have the tools to set up our $O(n \log n)$ cut query algorithm for finding a spanning forest for an input graph G. The high level idea is to expand a known set of connected vertices one vertex at a time (starting with a single arbitrary vertex) very similar to a DFS. At each step of this search, we can use $O(\log(n))$ cut queries to implement Primitive 2 to find one outgoing edge of this component. We now formalize this.

An Algorithm for computing a spanning forest of G.

1. Let $F = \emptyset$ be the spanning forest initially set to $\emptyset$. While an unmarked vertex exists:
   (a) Pick an arbitrary unmarked vertex v, let $S = \{v\}$, and T be the remaining unmarked vertices.
   (b) While $|\delta(S)| > 0$, use Primitive 2 to find an edge (u,v) from S to T (assume $u \in S$ and $v \in T$)

```
        and add $(u, v)$ to $F$. Update $S = S \cup \{v\}$ and $T = T \setminus \{v\}$.
    (c) Mark all vertices in $S$.
2. Return $F$ as a spanning forest of $G$.
```

Claim 10. *The above algorithm requires $O(n \log(n))$ queries to output a spanning forest for G.*

Proof. The fact that this algorithm outputs a spanning forest is identical to the proof of correctness of DFS and we omit it. As for the number of queries, each edge in the spanning forest is found using $O(\log n)$ queries and there are in total $O(n)$ other queries. As such, since there are at most $n-1$ edges in the spanning forest, the total number of queries is $O(n \log n)$. ☐

This allows us to conclude the following theorem.

Theorem 11. *There is a deterministic algorithm for finding a spanning forest of any given undirected graph using $O(n \log n)$ cut queries.*

This theorem was first proved in [4] who used this as a primitive for designing an algorithm that finds an *exact* global minimum cut using $O(n \cdot \text{poly}\log(n))$ cut queries.

References

[1] Paul Beame, Sariel Har-Peled, Sivaramakrishnan Natarajan Ramamoorthy, Cyrus Rashtchian, and Makrand Sinha. Edge estimation with independent set oracles. In *9th Innovations in Theoretical Computer Science Conference, ITCS 2018, January 11-14, 2018, Cambridge, MA, USA*, pages 38:1–38:21, 2018. 8

[2] Emmanuel J. Candès, Justin K. Romberg, and Terence Tao. Robust uncertainty principles: exact signal reconstruction from highly incomplete frequency information. *IEEE Trans. Information Theory*, 52(2):489–509, 2006. 2

[3] David L. Donoho. Compressed sensing. *IEEE Trans. Information Theory*, 52(4):1289–1306, 2006. 2

[4] Aviad Rubinstein, Tselil Schramm, and S. Matthew Weinberg. Computing exact minimum cuts without knowing the graph. In *9th Innovations in Theoretical Computer Science Conference, ITCS 2018, January 11-14, 2018, Cambridge, MA, USA*, pages 39:1–39:16, 2018. 10