CS 7870: Seminar for Algorithms for Big Data (Spr'26) **Northeastern University**

Lecture 6&7: Lowerbounds for Sublinear Algorithms

January 29, 2026

Instructor: Soheil Behnezhad *Scribe: Soheil Behnezhad, Rose Silver*

1 Graph Connectivity

Our goal for today is to prove the following theorem.

Theorem 1. *Any (*potentially randomized*) algorithm for determining if a graph G is connected or not with probability at least $2/3$ requires $\Omega(n^2)$ queries in the general query model.*

Note that the input is deterministic. The randomization comes from the algorithm.

Also note that the $2/3$ is arbitrary — if we have an algorithm that succeeds with probability $1/2 + \Omega(1)$ then we can repeat the algorithm multiple times and return whichever outcome it produces most often. By a Chernoff bound, if we want a probability $1 - \epsilon$ of being correct, then $\Theta(\log \epsilon^{-1})$ repetitions suffices.

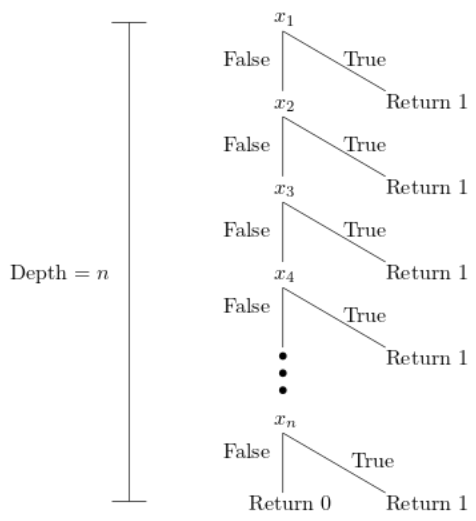
One corollary is that: we cannot distinguish between 1 vs 2 connected components. Thus, a better than 2-approximation for the number of connected components requires $\Omega(n^2)$.

Thus even a better than 2-approximation also requires $\Omega(n^2)$ queries.

Finally, note that this lower bound is *unconditional*. We are not relying on any assumptions such as $P \neq NP$.

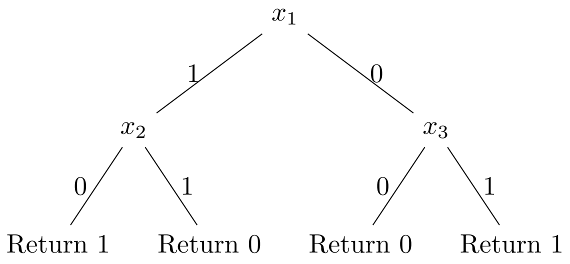
1.1 Proving lowerbounds for deterministic algorithms

Suppose we have a boolean function f. In the general query model, one way to think about any (deterministic) algorithm for evaluating f is with a decision tree. For example, here would be the decision tree for evaluating $\text{OR} = x_1 \vee x_2 \vee x_3 \vee \cdots \vee x_n$:



Each node of the decision tree performs a query, and then the edges leaving that node determine where the algorithm should go next based on that query.

Another example of a decision tree is the following, which computes the function $f(x_1x_2x_3) = (x_1 \wedge \bar{x}_2) \vee (\bar{x}_1 \wedge x_3)$



This tree demonstrates two important things: a decision tree does not need to query all elements, and a decision tree can have a depth (in this case 2) significantly smaller than the size of the input.

Definition 2 (Deterministic Query Complexity $D(f)$)**.** The ***Deterministic Query Function D(f)*** of a function f is the minimum worst-case number of queries made by the best algorithm for realizing f.

Equivalently, the best algorithm is the decision tree with the smallest depth. Thus, $D(f)$ is the depth of the best decision tree

It's worth making a few remarks. Note that query complexity is not the same as time complexity (although it is a lower bound). Also, we can define $D(f)$ on f's that operate on restricted sets of inputs — for example, f might distinguish between a binary string that has $> 2/3n$ ones vs $< 1/3n$ ones, in which case the tree could return arbitrary values or fail to have a return path for inputs that have between $1/3n$ and $2/3n$ ones.

Some more examples:

- $D(f) = 0$ for constant functions, i.e. $f(x) = 0$ and $f(x) = 1$.
- $D(f) = 1$ for "Dictatorship functions", i.e. $\{f_i(x) = x_i\}_{i \in [n]}$ and $\{f_i(x) = 1 - x_i\}_{i \in [n]}$
- $D(f) \leq n$ for all f, since there is always a trivial algorithm that queries the entire input.

Importantly, if we have a lower bound for the depth of a decision tree, we have a lower bound for (deterministic) query complexity. So what we really care about is decision-tree depth.

How do we prove lower bound? We consider a two-player game, with an **Adversary (AD)** player and an **Algorithm (ALG)** player. At all points in time, AD has a set $S \in \{0,1\}^n$ which initially equals all possible inputs (or if f is partial, then S is the set of all possible inputs), and then shrinks over time. ALG's job is to perform queries and AD's job is to determine the answer to each query. If ALG queries an entry i, then AD decides to return some x_i such that there exists possible $y \in S$ where $y_i = x_i$. Once the query is answered, all $z \in S$ where $z_i \neq x_i$ are removed from S. The game ends when $f(x)$ is the same for all $x \in S$.

A key observation is that, if both players play optimally, then the number of steps in the game will be the query complexity $D(f)$. More importantly for our purposes, if AD has an algorithm forcing the game to last for k steps, then that algorithm must work even when ALG uses the optimal decision tree, and so we have a lower bound of $D(f) \geq k$. In other words, strategies for AD imply lower bounds for $D(f)$.

As a warmup, let's prove a few simple lower bounds.

Lemma 3. *$D(OR_n) \geq n$.*

Proof. Whenever an entry x_i is queried, AD returns 0. If R is the set of entries queried so far, then S is the set of all inputs that return 0 for those queries, and $|S| = 2^{n-|R|}$. As long as $|R| < n$, then S contains both an input x satisfying $f(x) = 0$ (i.e., the 0 input) and at least one input x satisfying $f(x) = 1$. Thus we have a lower bound of n steps for the length of the game, implying that $D(OR_n) \geq n$. $\square$

Define $M_n : \{0,1\}^n \to \{0,1\}$ to be the function that returns:

$$M_n(x) = \begin{cases} 1 & \text{if } x \text{ contains } \geq \frac{2}{3}n \text{ ones} \\ 0 & \text{if } x \text{ contains } \leq \frac{1}{3}n \text{ ones} \\ \text{undefined} & \text{otherwise.} \end{cases}$$

Note that M_n is an example of a function that is defined only on a restricted set of inputs.

Lemma 4. *$D(M_n) > \frac{2}{3}n$.*

Proof. AD's strategy is as follows. For the first $n/3$ queries returns 0. For the second $n/3$ queries returns 1. If all the remaining entries are 1's, $M_n(x) = 1$. But if all the remaining entries are 0's, $M_n(x) = 0$. Thus, at least $2/3n + 1$ queries are needed for ALG, hence $D(M_n) > \frac{2}{3}n$. $\square$

1.2 Proving lowerbounds for randomized algorithms

Now onto randomized complexity.

Definition 5. We say a randomized ALG, which takes as input a random tape r, decides f if

$$\Pr[ALG(x) = f(x)] \geq 2/3 \qquad \forall x.$$

For any fixed random tape r, we use ALG_r to denote the deterministic algorithm obtained by instantiating the random tape to be r.

Definition 6. We use $R(f)$ to denote the worst-case queries of the best randomized algorithm deciding f

Remark. Note from the definition that an expected or even a high probability upper bound on the query-complexity of a randomized algorithm is not enough. That is, if $R(f) = K$ then there should be a randomized algorithm ALG that on all inputs x and all random tapes r queries at most K entries of the input.

Alternatively, A randomized algorithm can be viewed as a distribution over decision trees.

As an example of a case where randomized algorithms beat deterministic algorithms, note that $R(M_n) = 1$. Indeed, if ALG_r takes a random index i and returns x_i, then it will be correct with probability at least 2/3 on any input.

It turns out that that we can analyze randomized query complexity $R(f)$ with the same game as before (between ALG and AD), but with an important twist: AD is no longer adaptive. That is, AD must decide up front what it will return for every entry—or, to be more specific, AD must decide up front on a probability distribution for x to be drawn from.

Rather than thinking about randomized query complexity directly in terms of this game, however, we will instead introduce a principle known as Yao's minimax principle that allows for us to think about randomized algorithms in terms of probability distributions over deterministic algorithms. To do this, we first need two definitions.

Definition 7 (Distributional Query Complexity)**.** Let μ be a distribution over $\{0,1\}^n$. We say that a *deterministic* algorithm decides f if

$$\Pr_{x\sim\mu}[ALG(x) = f(x)] \geq 2/3.$$

Definition 8. Given a function $f : \{0,1\}^n \to \{0,1\}$ and input distribution μ, define $D_\mu(f)$ to be the worst-case number of queries of the best (fewest worst-case queries) deterministic algorithm deciding f on distribution μ.

With these definitions in mind, we can now state Yao's minimax principle.

Proposition 9 ([**yao**])**.** *For any function $f : \{0,1\}^n \to \{0,1\}$*

- *$D_\mu(f) \leq R(f)$ for every distribution μ.*
- *$D_{\mu^*}(f) = R(f)$ for some distribution μ^*.*

The former bound is known as the 'easy direction' of Yao's minimax principle and the latter is the 'hard direction'. For us, we only need to prove the easy direction, since it is the one that we need to obtain a lower bound on the randomized query complexity $R(r)$. Critically, what Yao's minimax principle tells us is that: to lower bound $R(f)$, all we need to do is construct a distribution μ for which we can establish a lower bound on $D_\mu(f)$.

Proof of the easy direction of Yao's minimax Theorem. Fix $f : \{0,1\}^n \to \{0,1\}$ and fix the best randomized algorithm ALG deciding f. Let

$$z(x,r) = \mathbb{1}_{ALG_r(x)=f(x)}$$

i.e., $z(x,r)$ is the indicator that the algorithm returns the right output for x with random tape r.

Fix any distribution μ. We can compute

$$\begin{aligned}
\mathbb{E}_{x\sim\mu,r}[z(x,r)] &= \sum_x \sum_r \Pr[x,r] \cdot z(x,r) \\
&= \sum_x \sum_r \Pr[x]\Pr[r] \cdot z(x,r) && \text{(Since x and r are independent.)} \\
&= \sum_x \Pr[x] \sum_r \Pr[r] z(x,r) \\
&= \sum_x \Pr[x] \mathbb{E}_r[z(x,r)] \\
&\geq \sum_x \Pr[x] \cdot \frac{2}{3} && \text{(Since ALG must succeed with probability at least $2/3$ on any input x.)} \\
&= 2/3.
\end{aligned}$$

Moreover, we have that

$$\begin{aligned}
\mathbb{E}_{x\sim\mu,r}[z(x,r)] &= \sum_x \sum_r \Pr[x]\Pr[r] \cdot z(x,r) \\
&= \sum_r \Pr[r] \sum_x \Pr[x] z(x,r) \\
&= \sum_r \Pr[r] \cdot \mathbb{E}_{x\sim\mu}[z(x,r)] \\
&\leq \max_r \mathbb{E}_{x\sim\mu}[z(x,r)]. && \text{(Since $\textstyle\sum_r \Pr[r] = 1$.)}
\end{aligned}$$

Note that in the last inequality above, $\mathbb{E}_{x\sim\mu}[z(x,r)]$ is the success probability of the deterministic algorithm ALG_r on $x \sim \mu$. Chaining our equations together, we get that there exists an r such that $\mathbb{E}_{x\sim\mu}[z(x,r)] \geq 2/3$. Since $\mathbb{E}_{x\sim\mu}[z(x,r)] \geq 2/3$, the deterministic algorithm ALG_r (i.e., the randomized algorithm using r as its random tape) satisfies $\Pr[ALG_r(x) = f(x)] \geq 2/3$ (here, the only randomness is from μ, since the random tape has been fixed). This means that ALG_r decides f on μ. It follows that $D_\mu(f)$ is at most the worst-case query complexity of ALG_r. But the worst-case query complexity of ALG_r is at most the worst-case query complexity of ALG, which is $R(f)$. This establishes that, for any distribution μ, $D_\mu(f) \leq R(f)$. □

Theorem 10. $R(OR_n) \geq n/3$

Proof. We need to come up with an input distribution, and show that any deterministic algorithm over this distribution has to have a large query complexity. We can then use Yao's minimax theorem to lowerbound the randomized query complexity $R(OR_n)$.

Consider the following distribution μ over the input: Pick $i \in [n]$ uniformly at random, and pick $b \in \{0,1\}$ independently from i and uniformly at random. For $x = x_1 \dots x_n$, let $x_i = b$ and $x_j = 0$ for all $j \neq i$.

Suppose for contradiction that there is a deterministic algorithm A that makes $k < n/3$ queries and guarantees

$$\Pr_{x\sim\mu}[A(x) = f(x)] \geq 2/3. \tag{1}$$

Suppose that A queries k indices and has not yet terminated (i.e. all of the indices returned 0) and define $i_1, \dots, i_k$ to be these indices. Define E to be the event that $i \notin \{i_1, \dots, i_k\}$. It follows that

$$\begin{aligned}\Pr[E] &= \frac{n-k}{n} \\ &> \frac{n-n/3}{n} \\ &= \frac{2}{3}.\end{aligned}$$

Since A is a deterministic algorithm, at this point it will return either 0 or 1. Thus, conditioned on E, A makes a mistake with probability 1/2. Thus, the probability that A succeeds over our distribution is

$$\begin{aligned}\Pr_{x\sim\mu}[A(x) = f(x)] &= 1 - \Pr_{x\sim\mu}[A(x) \neq f(x)] \\ &< 1 - \Pr[E] \cdot 1/2 \\ &< 1 - 1/3 \\ &= 2/3\end{aligned}$$

which is a contradiction to (1) above. Thus, any deterministic algorithm over μ must make at least $n/3$ queries, implying that the randomized query complexity $R(OR_n)$ is at least $n/3$. □

1.3 Proving upperbounds for the randomized query complexity

Let's now consider upperbounds for $R(OR_n)$. It is clear that $R(OR_n) \leq n$ if we query all indices, but there's still a gap between the lowerbound $n/3$ and upperbound n. The following claim gives a better upper bound.

Claim 11. $R(OR_n) \leq \frac{2}{3}n$

Proof. Consider a random algorithm that queries $\frac{2}{3}n$ indices at random without replacement and return the OR of these. If $OR_n(x_1, \dots, x_n) = 1$, then the algorithm succeeds with probability at least 2/3. If $OR_n(x_1, \dots, x_n) = 0$, then the algorithm succeeds with probability 1. □

Next, we show that we can go further down and solve OR_n with only $n/2$ queries.

Claim 12. *$R_n(OR_n) \leq \frac{1}{2}n$.*

Proof. Let a randomized algorithm A query $n/2$ indices at random without replacement. If a 1 is seen, A returns 1. Otherwise, A returns 1 with probability 1/3. Let E be the event that a 1 is found among the queried indices. If $OR_n(x) = 1$, the success probability of A is

$$\begin{aligned}\Pr[A(x) = OR_n(x)] &\geq \Pr[E] + (1 - \Pr[E]) \cdot \frac{1}{3} \\ &= \frac{1}{2} + \frac{1}{2} \cdot \frac{1}{3} \\ &= 2/3.\end{aligned}$$

If $OR_n(x) = 0$, the random algorithm returns 0 with probability 2/3. □

It turns out that $n/2$ is the correct answer. This can be proved by slightly modifying the input distribution we used to prove our $n/3$ lower bound.

Exercise 13. *Prove $R(OR_n) \geq \frac{1}{2}n$*

1.4 Proving Theorem 1

As a reminder, here is the theorem we want to prove:

Theorem 14. *Any (*potentially randomized*) algorithm for determining if a graph G is connected or not with probability at least 2/3 requires $\Omega(n^2)$ queries in the general query model.*

We first recall some definitions

Definition 15. The ***adjacency matrix model*** is a representation of a graph that, for any two vertices v_i and v_j, can determine if these two vertices are connected.

Definition 16. The ***adjacency list model*** is a representation of a graph that, for any vertex v_i and index j, can output the j-th neighbor of v_i.

Definition 17. The ***general query model*** is a representation of a graph that supports all adjacency matrix model queries and all adjacency list model queries

For intuition on this theorem, imagine two situations: (1) G has two connected components and (2) G has the same two connected components with the same number of edges as in (1) except there are two cross-edges between the components (replacing an internal edge in each connected component). In order to determine if this graph is connected, we would need to find one of these four possible edges, and that requires n^2 queries. More formally, we can prove the theorem by giving a reduction to the OR function. We let $N = \binom{n}{2}$ and let $x = x_{1,2}x_{1,3} \ldots x_{1,N}x_{2,3} \ldots x_{N-1,N}$. By Theorem 10, $R(OR_N(x)) \geq N/3 = \Omega(n^2)$.

Given an x, we can construct a graph G as a function of x. We create n vertices $v_1, \ldots, v_n$, n vertices $u_1, \ldots, u_n$, and two vertices a and a'. We then connect vertex a to each v_i, and connect vertex a' to each u_i. Additionally, for any i, j such that $i < j$, if $x_{i,j} = 0$, we add an edge between (v_i, v_j) and (u_i, u_j). If $x_{i,j} = 1$, add (v_i, u_j) and (v_j, u_i). Let E_1 be the set of edges created from cases where $x_{i,j} = 0$ and let E_2 be the set of edges created from cases where $x_{i,j} = 1$. Then, the vertex set V and edge set E can be written as:

- $V = \{a, a', v_1, \ldots, v_n, u_1, \ldots, u_n\}$

- $E = \{(a, v_1), \ldots, (a, v_n), (a', u_1), \ldots, (a', u_n)\} \cup E_1 \cup E_2$.

Let's formalize what adjacency list queries look like in this graph. For a vertex v_i, and for all $j \neq i$, we set its j-th neighbor to be either v_j if $x_{i,j} = 0$ or u_j if $x_{i,j} = 1$. We set the i-th neighbor of v_i to be a. We can do the analogous neighboring for all vertices u.

We begin by observing that there is no information for any algorithm to gain by learning the degree of a vertex, since in both cases all of the degrees are n.

Claim 18. *For all vertices v_i, it follows that $deg(v_i) = n$. Similarly, for all vertices u_i, it follows that $deg(u_i) = n$.*

Proof. Every vertex v_i is connected to a. Additionally, for every index $i \neq j$, v_i is connected to either v_j or u_j. Thus, v_i is connected to a total of n other vertices. The same is true for u_i. □

Next we observe that the two components $\{a, v_1, \ldots, v_n\}$ and $\{a', u_1, \ldots, u_n\}$ are connected to each other iff $OR_N(x) = 1$.

Claim 19. *$G(x)$ is connected iff $OR_N(x) = 1$.*

Finally, we observe that queries to G correspond to queries to x.

Claim 20. *Any neighbor or pair query to G can be answered with one query to x.*

Proof. Suppose we want to query the j-th neighbor of v_i. If $j = i$, return a without any queries to x. Otherwise, query $x_{i,j}$ and return v_j if $x_{i,j} = 0$ and u_j otherwise. We can do the analogous process for neighbor queries of vertices u_i. Additionally, for pair queries (v_i, u_j), (v_i, v_j), and (u_i, v_j), we can just query $x_{i,j}$ to get the answer. It is easy to see that all adjacency list and adjacency matrix queries on a and a' are trivial and do not require a query to x. □

Proof of Theorem 1. By Claim 19 and Claim 20, we can give a reduction from a graph with $2n + 2$ nodes to OR_N. Since $R(OR_N) \geq N/3 = \Omega(n^2)$, then it takes $\Omega(n^2)$ queries to determine if G is connected with probability at least 2/3. □

We note that while Theorem 1 proves the hardness of separating one connected component from two, essentially the same lower bound can be proven for separating one connected component from C for any constant C. We leave this as an exercise.

Exercise 21. *Let $C > 1$ be any constant and suppose that we are given a graph that is promised to either have C connected components or one connected component. Prove that determining the number of connected components of the graph (possibly with a randomized algorithm) with success probability at least 2/3 requires $\Omega(n^2)$ queries in the general query model.*