Game Theory II

1 Yao's Minimax Theorem

1.1 Theorem Statement

Theorem 1. *[Yao's Minimax Theorem]. Fix an input size n. Let $\mathscr{I}$ denote the set of all inputs of size n (finite). Let $\mathscr{A}$ be a finite set of algorithms, each of which correctly solves some given problem. Let $C(A, I)$ be a cost measure for algorithm $A \in \mathscr{A}$ on input $I \in \mathscr{I}$ (e.g., running time). Let $\hat{\mathbf{p}}$ be any distribution over $\mathscr{A}$ (i.e., a randomized algorithm), and let $\hat{\mathbf{q}}$ be any distribution over inputs $\mathscr{I}$ (i.e., a randomized input). Then,*

$$\min_{A \in \mathscr{A}} \mathbb{E}_{I \sim \hat{\mathbf{q}}}[C(A, I)] \leq \max_{I \in \mathscr{I}} \mathbb{E}_{A \sim \hat{\mathbf{p}}}[C(A, I)].$$

This is a restatement of the corollary from last time.

Note that the finiteness required makes application to algorithms with possibly unbounded runtime problematic (*e.g.,* coupon collector).

So, we can establish lower bounds on randomized algorithms by arguing which algorithm does best on a certain distribution of inputs – even a single input! By choosing a clever input, we can get a good lower bounds; if we choose badly, the lower bound will be weak. If $\hat{\mathbf{p}}$ is a point distribution, then A can be "return 0" or "return 1", which runs in constant time.

1.2 Game Tree Evaluation

A good first idea is a uniform distribution over all inputs. In this case, each of the leaves is independently 0 or 1 with probability 1/2. We would like different levels of the tree to have the same probability of being true (could be useful for analysis). In particular, under the uniform distribution, each AND node would have a different probability of being true. For example, if every leave had probability 1/2, then the parent OR nodes would have a 3/4 probability of being true, and their parent AND node would have a 9/16 probability of being true.

Note that an AND/OR tree is exactly the same as a NOR tree. So what distribution on the leaves would give us equal probability on every NOR node? Let's solve for one level being the same. Let p be the probability that a leaf is set to 1. Then,

$$\begin{aligned} p &= (1-p)^2 \\ \implies p &= \frac{3-\sqrt{5}}{2}. \end{aligned}$$

To summarize, our input distribution is to set each leaf to 1 independently with probability p.

Without loss of generality, the best deterministic algorithm starts with the leftmost leaf (because all leaves look the same initially). Subsequently, it is always better to continue in a partially evaluated subtree. This is, in a sense, monotonicity: if T is isomorphic to a subtree of T', then evaluating T takes at most as many queries in expectation as evaluating T'. This property is specific to the independent input distribution, and it can be proved via induction. Thus, the optimal algorithm queries the leaves left to right, but skip any whose value is not needed

Now, we'll calculate the expected number of queries this takes. Let $X(h)$ be the expected number of queries to learn the value of a tree of height h. We always evaluate the left subtree, and we evaluate the right subtree only if the left subtree returned 0. Thus,

$$\begin{aligned} X(h+1) &= X(h) + (1-p)X(h) \\ &= (2-p)X(h). \end{aligned}$$

So,

$$X(h) = (2-p)^h.$$

Since $h = \log_2 n$ for the entire tree,

$$\begin{aligned} X(h) &= (2-p)^{\log_2 n} \\ &= n^{\log_2(2-p)} \\ &= n^{\log_2 \frac{\sqrt{5}+1}{2}} \\ &= n^{\log_2(\sqrt{5}+1)-1} \\ &\approx n^{0.694}. \end{aligned}$$

So every randomized algorithm must query at least $n^{0.694}$ leaves in expectation, by Yao's Minimax Theorem. This is a better bound than the information-theoretic $\sqrt{n}$ lower bound, but it is less than the $\mathcal{O}(n^{0.793})$ upper bound of Snir's Algorithm. At least one of the upper and lower bounds is not tight.

1.3 Improving the Bounds

The analysis of Snir's Algorithm got a 3/2 probability at each internal node because we used a bound assuming that whenever the algorithm flipped a coin, there was a "bad" outcome (both subtrees must be evaluated) and a "good" outcome (only one subtree must be evaluated). The independent leaves create instances where, at many internal nodes, there are no "bad" outcomes. This implies an easier instance of the problem.

A better input distribution would generate inputs such that, at each internal node, there exists a "bad" choice.

The distribution is defined from the root down. Make the root have value 1 with probability q. If a node has value 1, then recursively choose values below it such that both subtrees have value 0. If a node has value 0, flip a coin. Based on the outcome, recursively make one subtree have value 1 and the other have value 0.

The distribution was inspired by Snir's Algorithm, but we use it to prove lower bounds for every randomized algorithm. To do so, we must derive & prove the optimal deterministic algorithm against this distribution, then apply Yao's Minimax Theorem.

We won't derive an algorithm here, but this distribution does give a lower bound of $n^{0.793}$, proving that Snir's Algorithm is optimal.

2 Derandomization

The goal of derandomization is frequently to reduce or remove randomness, both for practical reasons and out of complexity interest (*e.g.,* ultimately proving $\boldsymbol{P} = \boldsymbol{RP}$).

Here, we will cover two concepts: first, we will see how to derandomize polynomial circuits; then, we will study the method of conditional expectations.

2.1 Derandomizing Polynomial Circuits

The input is a string in $\{0,1\}^n$ with n fixed for now. The algorithm uses m bits of randomness: $\{0,1\}^m$. We can think of the algorithm as having two inputs: the input bitstring and the randomness bitstring. If the algorithm runs in polynomial time, then $m = poly(n)$.

Recall that the definition of an ***RP*** algorithm is as follows: if the correct output is "false", then the algorithm always answers "false", and if the correct output is "true", then for at least 1/2 of the randomness strings, the algorithm answers "true".

We will assume that the algorithm is represented by a poly-sized circuit by AND, OR, and NOT gates with no feedback (*i.e.,* the graph of connections is a DAG). The input "wires" are n bits for the input and m bits for the randomness; we have one output "wire".

For a function f, a **circuit family** $C_1, C_2, \ldots$ contains a correct circuit C_n for each n, and $|C_n| = \mathcal{O}(n^k)$ for some constant k.

Theorem 2. *[Adelman]. If a function has a randomized poly-sized circuit family, then it has a non-randomized poly-sized circuit family.*

Proof. Fix an input size n and corresponding circuit C_n. Write $a(\boldsymbol{x}, \boldsymbol{r})$ for the output of C_n on input bitstring $\boldsymbol{x}$ with randomness bitstring $\boldsymbol{r}$. Correctness of C_n implies that if $f(\boldsymbol{x}) = 0$, then $a(\boldsymbol{x}, \boldsymbol{r}) = 0$ for all $\boldsymbol{r} \in \{0,1\}^m$, while if $f(\boldsymbol{x}) = 1$, then $a(\boldsymbol{x}, \boldsymbol{r}) = 1$ for at least 1/2 of the strings $\boldsymbol{r} \in \{0,1\}^m$.

The key insight is to look at the matrix $\mathbf{A} \in \{0,1\}^{2^n \times 2^m}$ of all these entries. More specifically, we look at the sub-matrix of only the rows $\boldsymbol{x}$ where $f(\boldsymbol{x}) = 1$. Our goal is to construct polynomially many copies of C_n in which the randomness is hard-wired, then take a big OR of all of them.

Note that if we allowed 2^m copies, we could take an OR of C_n with all randomness strings hard-wired. This computes $f(\boldsymbol{x})$ even for ***NP***.

[To be continued next lecture.] $\square$