15-859(M): Randomized Algorithms **Lecturer:** Anupam Gupta
Topic: Lower Bounds on Randomized Algorithms **Date:** September 22, 2004
Scribe: Srinath Sridhar

4.1 Introduction

In this lecture, we will first consider the problem of boolean function evaluation and prove that randomized algorithms (in the expected case) perform better than deterministic algorithms. The boolean circuits that we consider, can be naturally extended to obtain a game tree. We will then state Von Neumann's minimax theorem and its extension - Yao's principle. Finally, we show how Yao's principle can be used to obtain lower bounds on the expected running time of randomized algorithms.

4.2 Boolean Function Evalutation

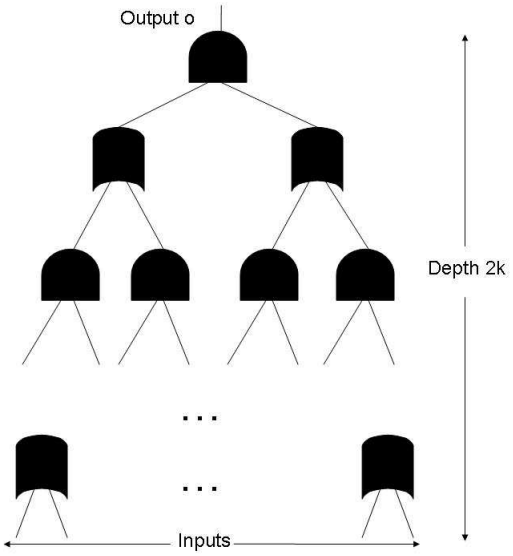

Figure 4.2.1: A depth $2k$ AND-OR circuit

Boolean functions in general can be formulated using various forms. For the purposes of this lecture we shall focus on functions that have only two kinds of gates: AND and OR. The number of inputs to each gate is restricted to two. The functions are nested by alternating AND and OR gates, so that the circuit corresponding to the function is similar to Figure 4.2.

We assume that the circuit forms a full binary tree (each internal node being an AND or OR gate) of height $2k$. The number of inputs of the circuit is 2^{2k}. The root node is an AND gate and all

the nodes at height $2k-1$ are OR gates. Note that, the functions considered are monotone and therefore complementation is not permitted and also the wires are not allowed to overlap or branch out.

The problem of boolean function evaluation is to compute the result of the function (circuit) while minimizing the number of queries. An algorithm makes a query if it looks at one of the input values. We make a quick digression to game trees, so that it is easier to compare games with boolean circuits.

Game trees are used extensively by computer programs to play games. In a game tree each node corresponds to a position and each adjacent out-edge a legal move that can be played from the position. The root of the game tree is the current position from which the analysis is being performed. The even nodes correspond to positions from which player 1 (or computer) picks the move and the odd nodes position from which player 2 (opponent) moves. To decide on the best move from the current position, player 1 picks the move that maximizes an evaluation function. In doing so he assumes that the opponent plays optimally and therefore picks the move that minimizes the evaluation function at the next step.

The boolean circuit can be naturally viewed as a game tree, where the edges are directed from the root to the leaves. The OR gates correspond to the positions from which player 1 moves (MAX nodes) and the AND gates the positions from which the opponent moves (MIN nodes). The number of rounds of a game tree is defined as the number of moves performed by a single player. Therefore a round of the boolean circuit can be thought of as a pair of AND followed by an OR gate. In the game tree (boolean circuit) considered above, there are k rounds.

It is well known that an AND-OR circuit can be converted to a NOR circuit. Specifically, a circuit where the AND gates and the OR gates alternate can be converted to a NOR circuit, by replacing each gate of the AND-OR circuit with a NOR gate. Consider the the root (AND gate) and the two children (OR gates), assume that the wires of the two children are A, B, C and D. Then,

$$\begin{aligned}(A \vee B) \wedge (C \vee D) &= \neg(\neg(A \vee B) \vee \neg(C \vee D)) \\ &= (A \text{ NOR } B) \text{ NOR } (C \text{ NOR } D)\end{aligned}$$

Therefore, by induction (on depth) the AND-OR circuit can be converted to a NOR circuit by simple replacement. For the rest of the lecture, we will use an equivalent NOR circuit. We now return back to the question of computing the value of the circuit for a given set of inputs while minimizing the number of inputs examined (number of queries performed). Assume that the number of input values $n = 2^{2k}$

Exercise 4.2.1 *The number of input values examined by any* deterministic *boolean function evaluation algorithm is* $n = 2^{2k}$.

We make the following important observation. Consider any NOR gate, and assume that its inputs are A and B. If a deterministic algorithm computes that $A = 0$, then it needs the value of B to compute A NOR B. If a deterministic algorithm computes that $A = 1$, then we can assume that it does not compute/examine the value of B. To perform better than the deterministic algorithm, we pick one out of A and B randomly and evaluate/examine its value. If exactly one of A and B

is 1, then the randomized algorithm has a 0.5 probability of computing/examining the wire with value 1 first.

This leads us to the following algorithm. Assume that the tree used by the recursive function is T, rooted at r. Further assume that the two children of r are L and R.

Eval-Rand (Tree T)

1. pick $x \in \{L, R\}$ uniformly at random
2. a := Eval-Rand (subtree rooted at x)
3. if $a = 1$ then
   (a) return 0
4. else
   (a) Eval-Rand (subtree rooted at $y \in \{L, R\} \backslash x$)

Theorem 4.2.2 *The expected number of queries performed by function Eval-Rand:*

$$cost(\textit{Eval-Rand}) \leq 3^k = n^{\log_4 3}$$

Proof: By induction on the number of rounds of the circuit. Let X_k be the random variable that denotes the number of queries performed by the algorithm on a circuit with k rounds. We want to compute $\mathbf{E}[X_k]$. Let o denote the output of the circuit. Assume that the values carried by the two wires to r are i_1^1 and i_2^1 and the values carried by the wires to the left gate at depth one are i_1^2 and i_2^2 and that of the right gate being i_3^2 and i_4^2. We break the analysis of the expected cost into two cases. The two cases are shown in Figure 4.2.

Case 1: Output $o = 1$. In this case, though it might be necessary to examine both inputs of depth 1, randomization helps improve the cost at depth 2. We have,

$$(o = 1) \implies (i_1^1 = 0 \text{ and } i_1^2 = 0)$$

$$(i_1^1 = 0) \implies (i_1^2 = 1 \text{ or } i_2^2 = 1), (i_2^1 = 0) \implies (i_3^2 = 1 \text{ or } i_4^2 = 1)$$

In this case, both the recursive calls Eval-Rand (Tree rooted at L) and Eval-Rand (Tree rooted at R) are performed. Consider either of the two calls. The child x is chosen at random and evaluated with cost $\mathbf{E}[X_{k-1}]$. The value $a = 0$ with probability 0.5 and the function has to perform an additional recursive call with cost $\mathbf{E}[X_{k-1}]$. The value $a = 1$ with probability 0.5 in which case we incur no additional cost besides $\mathbf{E}[X_{k-1}]$ used to compute the value of a.

Therefore in total,

$$\begin{aligned} \mathbf{E}[X_k | o = 1] &= 2\mathbf{E}[X_{k-1}]\mathbf{Pr}[x = 0] + \mathbf{E}[X_{k-1}]\mathbf{Pr}[x = 1] \\ &\leq \mathbf{E}[X_{k-1}] + 0.5\mathbf{E}[X_{k-1}] < 3\mathbf{E}[X_{k-1}] \end{aligned}$$

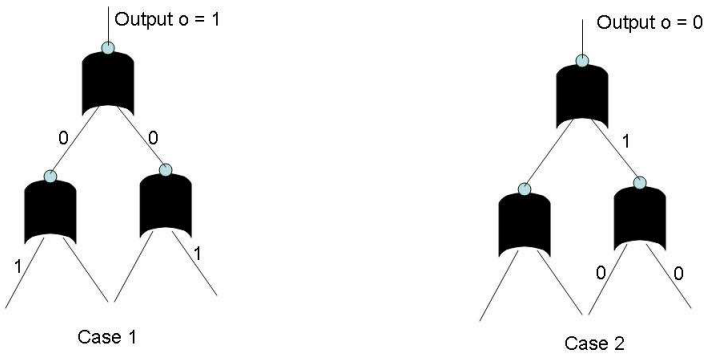

Figure 4.2.2: Two cases in the analysis of the number of queries performed by Eval-Rand

Case 2: Output $o = 0$**.** Using the same notations, we have,

$$(o = 0) \implies (i_1^1 = 1 \text{ or } i_2^1 = 1)$$

In the worst case, exactly one of $i_i^1 = 1$ and w.l.o.g assume $i_1^1 = 0$ and $i_2^1 = 1$.

$$i_1^1 = 0 \implies (i_1^2 = 1 \text{ or } i_2^2 = 1) \text{ and } i_2^1 = 1 \implies (i_3^2 = 0 \text{ and } i_4^2 = 0)$$

Once again, w.l.o.g assume $i_1^2 = 1$ and $i_2^2 = 0$. In the function call Eval-Rand(T), $x = L$ and therefore $a = 0$ with probability 0.5. In this case, the entire of right subtree has to be queried with cost $2\mathbf{E}[X_{k-1}]$. Furthermore, in the function call Eval-Rand(Tree rooted at L), $a = 1$ with probability 0.5 in which case the expected number of queries is $\mathbf{E}[X_{k-1}]$; $a = 0$ with probability 0.5 in which case the expected number of queries is $\mathbf{E}[X_{k-1}]$.

Now consider the function call Eval-Rand(T), $x = R$ and therefore $a = 1$ with probability 0.5, and here we incur a cost of $2\mathbf{E}[X_{k-1}]$ to compute the value of a. Using the notation, a_1, a_l and a_r to denote the values of a in the function calls Eval-Rand(T), Eval-Rand(L) and Eval-Rand(R) respectively we have,

$$\begin{aligned}
\mathbf{E}[X_k|o = 0] &\leq \mathbf{E}[X_k|a_1 = 0]\mathbf{Pr}[a_1 = 0] + \mathbf{E}[X_k|a_1 = 1]\mathbf{Pr}[a_1 = 1] \\
&= 0.5(4\mathbf{E}[X_{k-1}]\mathbf{Pr}[a_l = 0] + 3\mathbf{E}[X_{k-1}]\mathbf{Pr}[a_l = 1]) + 0.5(2\mathbf{E}[X_{k-1}]) \\
&= 0.5(4 + 1.5)\mathbf{E}[X_{k-1}] < 3\mathbf{E}[X_{k-1}]
\end{aligned}$$

The proof follows induction (on the number of rounds) with a trivial base case. ■

Moves	H	T
H	1	-1
T	-1	1

Figure 4.3.3: Payoff Matrix for matching pennies game. Row Player: Roberta and Column Player: Charles

4.3 Games

In this section we will focus on two player zero sum games. Zero sum games have the property that one player's loss is exactly the other player's gain. A standard example is the game stone, paper and scissors. We will assume that the two players are Roberta and Charles. The payoff (rules) of a game is usually expressed in a payoff matrix M, where the rows correspond to the possible moves for Roberta and the columns the possible moves for Charles. An entry $M_{i,j}$ in the matrix corresponds to the payoff awarded to Roberta if she plays i and Charles plays j. In general, if the game supports n moves for Roberta and m for Charles, the payoff matrix is of size $n \times m$.

We will consider the game of matching pennies. Roberta and Charles simultaneously show one side of their penny. If the displayed sides match then Roberta wins one dollar and if the sides don't match then Charles wins a dollar. The payoff matrix for the game is shown in Figure 4.3.3.

If Roberta picks move i, then she is guaranteed a payoff $\min_j M_{i,j}$. The lower bounds for the gain of Roberta $\hat{V}_R$ and Charles $\hat{V}_C$ is given by

$$\hat{V}_R = \max_i \min_j M_{i,j}$$

and

$$\hat{V}_C = \min_j \max_i M_{i,j}.$$

In the game of matching pennies, $\hat{V}_R = -1$ and $\hat{V}_C = 1$.

Claim 4.3.1 *For any payoff matrix $\hat{V}_R \leq \hat{V}_C$*

Proof: Let $\hat{V}_R = M_{i,k}$ and $\hat{V}_C = M_{l,j}$. We have for all m, $\hat{V}_R \leq M_{i,m}$, and specifically, $\hat{V}_R \leq M_{i,j}$. Similarly, for all m, $\hat{V}_C \geq M_{m,j}$, and specifically $\hat{V}_C \geq M_{i,j}$ and therefore $\hat{V}_R \leq \hat{V}_C$.

■

In games where $V = \hat{V}_R = \hat{V}_C$, the game is said to have a solution and V is called the value of the game. In such games, the two players can advertise what they are going to play without any harm.

Definition 4.3.2 *A pure strategy is defined as a deterministic choice to play in a game.*

Definition 4.3.3 *A mixed strategy is defined by a probability distribution on deterministic strategies.*

We will now focus on mixed strategies. Let p_i be the probability that Roberto picks move i and let q_j be the probability that Charles picks move j. Therefore,

$$\text{Expected payoff} = \sum_{i=1}^{n}\sum_{j=1}^{m} p_i q_j M_{i,j} = p^T M q,$$

where vectors p and q contain the probabilities of selecting moves. Similar to the previous definition for pure strategies, we can obtain lower bounds on the payoffs for Roberta and Charles as

$$\hat{V}_R = \max_p \min_q p^T M q$$

and

$$\hat{V}_C = \min_q \max_p p^T M q$$

Note that the min and max are computed over all possible distributions p and q over deterministic choices.

Theorem 4.3.4 *(Von Neumann's Minimax Theorem [Ne28]) For any two person zero sum game with game matrix M,*

$$\max_p \min_q p^T M q = \min_q \max_p p^T M q$$

Note that in the above theorem, if Roberta announces the probability distribution that she is using, Charles picks a single move (corresponding to the maximum term in vector $p^T M$) and his strategy becomes deterministic. Therefore an equivalent definition is

$$\hat{V}_R = \max_p \min_j p^T M e_j,$$

where e_j is a vector with 0 in all positions except j which is set to 1. This equivalent definition of $\hat{V}_R$ leads to Loomis' theorem.

Theorem 4.3.5 *(Loomis' Theorem [Lo46]) For any two player zero sum game specified by game matrix M,*

$$\max_p \min_j p^T M e_j = \min_p \max_i e_i^T M q$$

4.4 Yao's Principle

We can use the above ideas from game theory to lower bound the running time of randomized algorithms. This is the only known technique currently used to establish lower bounds. Consider any randomized algorithm that uses a finite number of random bits. Fixing the values of the random bits specifies a deterministic algorithm. The values for the random bits can themselves be used to label each such deterministic algorithm. Informally a randomized algorithm can be viewed as specifying a probability distribution over several deterministic algorithms.

Consider a matrix M similar to the game matrix. For any randomized algorithm, the rows of M correspond to different inputs and the columns correspond to different deterministic algorithms

(obtained by fixing the random bits of a randomized algorithm). The payoff $M_{i,j}$ of the matrix corresponds to the running time of algorithm A_j on input I_i. Let vector p denote the probability distribution on the inputs and vector q the probability distribution on the algorithms and e_j the (0, 1)-vector with 1 in exactly position j.

Using the same ideas from the previous section, we have,

$$\begin{aligned} M_{i,j} &= C(A_j, I_i), \\ \mathbf{E}[C(A_j, I_p)] &= p^T M e_j, \\ \mathbf{E}[C(A_q, I_i)] &= e_i^T M q \end{aligned}$$

Using 4.3.5 we have,

$$\max_p \min_q \mathbf{E}[C(I_p, A_q)] = \min_q \max_p \mathbf{E}[C(I_p, A_q)]$$

Dropping the max term on the left side and the min term on the right side from the above equation gives us Yao's principle.

Proposition 4.4.1 *(Yao's Principle [Ya77]) For all distributions p over $\mathcal{I}$ and q over $\mathcal{A}$,*

$$\min_{A \in \mathcal{A}} \mathbf{E}[C(I_p, A)] \leq \max_{I \in \mathcal{I}} \mathbf{E}[C(I, A_q)]$$

We will now show a simple example and illustrate how Yao's principle can be applied to establish a lower bound on randomized algorithms.

The find-bill problem can be stated as follows. There are n boxes and exactly one box contains a dollar bill, and the rest of the boxes are empty. A probe is defined as opening a box to see if it contains the dollar bill. The objective is to locate the box containing the dollar bill while minimizing the number of probes performed. Consider the following randomized algorithm:

1. select $x \in \{H, T\}$ uniformly at random
2. if $x = H$ then
   (a) probe boxes in order from 1 through n and stop if bill is located
3. else
   (a) probe boxes in order from n through 1 and stop if bill is located

The expected number of probes made by the algorithm is $(n+1)/2$. Since, if the dollar bill is in the i^{th} box, then with probability 0.5, i probes are made and with probability 0.5, $(n - i + 1)$ probes are needed.

Lemma 4.4.2 *A lower bound on the expected number of probes required by any randomized algorithm to solve the find-bill problem is $(n + 1)/2$.*

Proof: To use Yao's principle we need to specify a distribution p on the input. In this case, we assume that the bill is located in any one of the n boxes uniformly at random. We only consider deterministic algorithms that does not probe the same box twice. By symmetry we can assume that the probe order for the deterministic algorithm is 1 through n. We therefore have,

$$\begin{aligned} \min_{A \in \mathcal{A}} \mathbf{E}[C(A, I_p)] &= \sum_{i=1}^{n} \frac{i}{n} = \frac{1}{n} \sum_{i=1}^{n} i = \frac{n+1}{2} \\ &\leq \max_{I \in \mathcal{I}} \mathbf{E}[C(I, A_q)], \end{aligned}$$

by using Yao's principle. Therefore any randomized algorithm A_q requires at least $(n+1)/2$ probes in expectation. ■

References

[Lo46] L. H. Loomis. On a theorem of von Neumann *Proceedings of the National Academy of Sciences of the U.S.A.*, 32:213-215, 1946.

[Ne28] J. von Neumann. Zur Theorie der Gesellschaftsspiele. *Mathematische Annalen*, 100:295-320, 1928.

[Sn85] M. Snir. Lower bounds on probabilistic linear decision trees. *Theoretical Computer Science*, 38:69-82, 1985.

[Ya77] A. C-C. Yao. Probabilistic computations: Towards a unified measure of complexity. *Proceedings of the 17th Annual Symposium on Foundations of Computer Science*, 222-227, 1977.