

Lecture 2

Lecturer: Noam Nisan

Scribe: Ariel Procaccia

1 Introduction

Last week we discussed the basics of zero-sum games in strategic form. We characterized games as matrices, and associated the utility for the strategy profile (i, j) with entry a_{ij} in the game matrix. We also considered mixed strategies (denoted by x and y), and defined the utility for the strategy profile (x, y) as $\sum_{i,j} x_i a_{ij} y_j$.¹ We proved that

$$\max_i \min_j a_{ij} \leq \min_j \max_i a_{ij}. \tag{1}$$

Moreover, we noted that any pure strategy in the support of a best-response mixed strategy is itself a best-response. Finally, we formulated the row player's optimization problem as a mathematical programming problem:

$$\begin{aligned} & \max_x \min_j \sum_i x_i a_{ij} \\ \text{s.t.} & \sum_i x_i = 1 \\ & \forall i : x_i \geq 0 \end{aligned}$$

We observed that this is equivalent to the following linear programming formulation:

$$\begin{aligned} & \max c \\ \text{s.t.} & \forall j : \sum_i x_i a_{ij} \geq c \\ & \sum_i x_i = 1 \\ & \forall i : x_i \geq 0 \end{aligned} \tag{2}$$

Similarly, the column player faces the problem:

¹Under the implicit assumption that the players are risk-neutral.

$$\begin{aligned}
& \min d \\
& \text{s.t. } \forall i : \sum_j a_{ij} y_j \leq d \\
& \sum_j y_j = 1 \\
& \forall j : y_j \geq 0
\end{aligned} \tag{3}$$

In this lecture, we prove the Minimax Theorem using the LP Duality Theorem. We next present Yao's principle, a method to derive lower bounds for randomized algorithms. Finally, we define games in extensive form, and analyze the effectiveness of the $\alpha - \beta$ pruning algorithm and its randomized version.

2 The Minimax Theorem via LP

In this section, we wish to prove the following theorem:

Theorem 1 (Minimax) *For any zero-sum game it holds that:*

$$\max_x \min_y \sum_{i,j} x_i a_{ij} y_j = \min_y \max_x \sum_{i,j} x_i a_{ij} y_j. \tag{4}$$

Remark The inequality

$$\max_x \min_y \sum_{i,j} x_i a_{ij} y_j \leq \min_y \max_x \sum_{i,j} x_i a_{ij} y_j$$

can be obtained as a special case of Equation (1).

Remark The two expressions in Equation 4 are exactly the solutions of the LPs. Indeed, recall that a best-response mixed strategy has the same utility as the pure strategies in its support.

Remark Given a zero-sum game, $\max_x \min_y \sum_{i,j} x_i a_{ij} y_j (= \min_y \max_x \sum_{i,j} x_i a_{ij} y_j)$ is known as the *value* of the game.

In order to prove the theorem, we wish to transform the LPs (2) and (3) into a more canonical form. We present a new LP:

$$\begin{aligned}
& \max_x \sum_i x_i \\
& \text{s.t. } \forall j : \sum_i x_i a_{ij} \geq 1 \\
& \quad \forall i : x_i \geq 0
\end{aligned} \tag{5}$$

Lemma 2 *The optimum of (5) is the inverse of the optimum of (2).*

Proof: Let $x = \langle x_1, \dots, x_n \rangle$ be a feasible solution for (2) with value c . The vector $x/c = \langle x_1/c, \dots, x_n/c \rangle$ is a feasible solution for (5), with value $1/c$. Therefore, the optimum of (5) is at least the inverse of the optimum of (2).

In the other direction, if x is a feasible solution for (5) with value $1/c$, then the vector cx is a feasible solution for (2), with value c . Hence, it holds that the inverse of the optimum of (2) is at least the optimum of (5). ■

Remark We can assume the optima are positive; otherwise, we may transform the game into a positive, equivalent game, by adding a constant to all entries in the game matrix.

Symmetrically, the optimum of the problem (3) is equal to the inverse of the optimum of the problem:

$$\begin{aligned}
& \min_y \sum_j y_j \\
& \text{s.t. } \forall i : \sum_j a_{ij} y_j \leq 1 \\
& \quad \forall j : y_j \geq 0
\end{aligned} \tag{6}$$

The Minimax Theorem now follows directly from the linear programming Duality Theorem [1, Theorem 29.10], as by the latter the optima of (5) and (6) are equal.

Remark Randomizing the player's strategies has enabled us to abandon the setting where players stand to benefit by playing first. In other words, by "tossing coins" only after both players have decided on a strategy, we have created a situation in which rational players secure the same utility whether they play first or second.

3 Randomized Algorithms as Games

Yao [3] sought a method to derive lower bounds for randomized algorithms. In this section we present his application of game theory to this end.

For inputs of fixed size, we model algorithms' performance as a zero-sum game in strategic form. Let \mathcal{A} denote the set of all possible algorithms, and \mathcal{I} be the set of all possible inputs. The rows are associated with elements of \mathcal{I} , while the columns correspond to elements of \mathcal{A} . Entry a_{ij} is the running time of algorithm A_j on input I_i .

Computer-scientists traditionally adopt a worst-case approach to the analysis of algorithms. The worst-case performance of an algorithm A_j is the outcome of the game where the column player chooses A_j , and the row player responds with $\operatorname{argmax}_i(a_{ij})$. In this respect, the row player is considered to be a sinister adversary.

In other fields (including economy), researchers assume there is some distribution on the input, and analyze accordingly. However, in many cases this distribution is unjustified. For example, the QUICKSORT algorithm achieves a running time of $O(n \log n)$ for random inputs, but many real-world inputs are nearly sorted — slowing the algorithm's performance to $O(n^2)$. This approach is also consistent with our model: the row player plays some strategy, and the column player responds with the most suitable algorithm (in terms of expected running time).

For our purposes, a randomized algorithm is a distribution over deterministic algorithms. Denote the running time of the algorithm A_j on input I_i by $R(A_j, I_i)$; $E[R(A_j, I_x)] = \sum_i x_i a_{ij}$; and $E[R(A_y, I_i)] = \sum_j a_{ij} y_j$. From the trivial part of the Minimax Theorem, we have:

Theorem 3 (Yao's Principle) *For all distributions x over \mathcal{I} and y over \mathcal{A} ,*

$$\min_{A \in \mathcal{A}} E[R(A, I_x)] \leq \max_{I \in \mathcal{I}} E[R(A_y, I)]. \quad (7)$$

Observe that the right expression in Equation (7) is the worst-case running time of a randomized algorithm distributed according to y . Thus, in order to derive a lower bound on the running time of any randomized algorithm, it is enough to find some nasty distribution, with respect to which any deterministic algorithm performs poorly.

Moreover, by the other direction of the Minimax Theorem, if one produces the very worst distribution, in which the best algorithm runs longest, then this running time is equal to the worst-case expected running time of the best randomized algorithm. In this sense, Yao's method is complete: it is possible to derive from the worst distribution a lower bound which is *tight*.

Remark Yao's approach provides a compromise between the clashing views of computer-scientists and the rest of the world regarding running time: the worst-case performance of the best randomized algorithm versus deterministic inputs is equal to its running time on the worst possible randomized input.

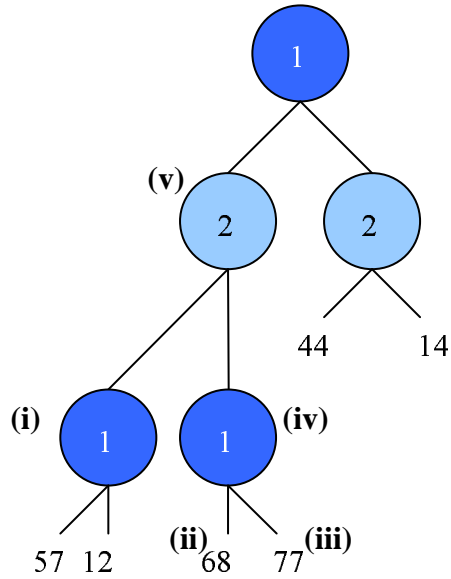


Figure 1: A zero-sum game in extensive form.

4 Evaluation of Game Trees

This section is interesting in its own right, but will ultimately serve as an example of the principles illustrated in the previous section.

Definition 4 *A zero-sum game in extensive-form consists of:*

1. *A set N of players.*
2. *A tree T , called the game tree.*
3. *A partition of the internal nodes among the players.*
4. *A utility function u assigning a real number to every leaf. This function represents the utility of player 1.*

See Figure 1 for an illustration of a game in extensive form (ignore the Roman numerals for now).

At each stage in the game, the player who owns the current node chooses to proceed to a child of this node; in the next stage, the player who owns the child makes a choice. The game ends when a leaf is reached; player 1's utility is the value of u on this leaf.

Remark A game in extensive form corresponds to a game in strategic form; a pure strategy in a game in extensive form specifies the player's action at every reachable node.

Fix a game tree and a partition of the nodes among the players. Consider the following problem: we are given the function u (the utility of the different leaves), and are asked what the value of the game is. A simple solution is the *backward induction* algorithm: we propagate nodes' values from the leaves, by taking the maximum of the children's values in each node owned by player 1, and taking the minimum in each node owned by player 2. This algorithm is rather efficient, as the running time is linear in the size of the game tree. Nevertheless, in some settings the tree is gargantuan. Is it necessary to query all the leaves?

Example 1 *Observe the game illustrated in Figure 1. Assume the algorithm first queries the children of the node marked by (i); the value of this node is 57. The algorithm then queries leaf (ii); there is no need to query leaf (iii), as the value of node (iv) is at least 68, and thus player 2 would surely go left at (v).*

A algorithm which sometimes saves us some trouble is the $\alpha - \beta$ -pruning algorithm. Here we give a somewhat informal description of the algorithm; a full specification can be found in [2, page 170]. From now on assume the game trees are binary.

```

MAX-EVAL( $T, \alpha, \beta$ )
1: if  $T$  is a leaf then
2:   return  $u(T)$ 
3: end if
4:  $v \leftarrow \max(\alpha, \text{MIN-EVAL}(\text{left}(T), \alpha, \beta))$ 
5: if  $v \geq \beta$  then
6:   return  $v$ 
7: end if
8:  $v \leftarrow \max(v, \text{MIN-EVAL}(\text{right}(T), v, \beta))$ 
9: return  $v$ 

```

α represents a lower bound for values that can affect the value of the game, and β is an upper bound. The function MIN-EVAL is defined symmetrically. The game's value is given by the execution of MAX-EVAL on the root, with $\alpha = -\infty$, $\beta = \infty$.

At this point we add the assumption that the utility function u is binary-valued (the values of the leaves are either 0 or 1).² Thus, the max operation is equivalent to the \vee operation, and min can be replaced with \wedge . We first demonstrate that in the worst case, $\alpha - \beta$ pruning doesn't prune any leaves.

Lemma 5 *Given a game tree with n leaves and an algorithm which evaluates the game, there exists an input such that the algorithm must query all the leaves.*

Proof: The proof is by induction on the number of leaves n , using an adversary argument. The base case ($n = 0$) is trivial.

²The motivation for this assumption is ease of exposition; the results can be generalized.

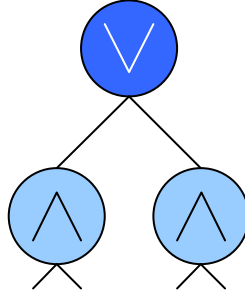


Figure 2: the tree \tilde{T} .

Assume the claim holds for any tree with $n = k - 1$ leaves. Given a tree with k leaves, the algorithm queries some leaf first. If the leaf's parent is an \vee node, the adversary answers that the value is 0; and if the leaf's parent is an \wedge node, the adversary responds with 1. Based on these answers, it is not possible to decide the value of the leaf's parent without querying the leaf's sibling. Therefore, the algorithm must still evaluate a tree with $k - 1$ leaves. By the induction assumption, all the leaves of this tree must be queried. ■

It seems possible that randomizing the algorithm might improve the expected running time. The randomized version simply switches the order of T 's children with probability $1/2$ (this coin flip is added between lines 3 and 4 of MAX-EVAL). We begin an analysis of the effectiveness of this randomized version with a modest special case: the game tree \tilde{T} illustrated in Figure 2.

Lemma 6 *Given any values for the leaves, the expected number of queries randomized $\alpha - \beta$ pruning makes on \tilde{T} is at most 3.*

Proof: We examine two cases.

Case 1: the value of the tree is 1. It follows that the value of at least one of the \wedge nodes is 1; without loss of generality the value of the left \wedge node is 1. With probability at least $1/2$ the algorithm first computes the value of the left \wedge node. If this happens, the value of the second \wedge node would not be calculated. Therefore, the expected running time is at most: $1/2 \cdot 2 + 1/2 \cdot 4 = 3$.

Case 2: the value of the tree is 0. In this setting, the value of both \wedge nodes is 0. For each one, with probability $1/2$ the algorithm would first query the value of a leaf with utility 0, and its sibling would not be queried. The expected running time is at most $2(1/2 \cdot 1 + 1/2 \cdot 2) = 3$. ■

Next week we will continue this line of inquiry. We will show that if the nodes in any path from the root to the leaves alternate between the players, then randomized $\alpha - \beta$ pruning performs well; but the performance is not as good if player 1 makes all his moves first.

References

- [1] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, 2nd edition, 2001.
- [2] S. Russel and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2nd edition, 2003.
- [3] A. C. Yao. Probabilistic computations: Towards a unified measure of complexity. In *Proceedings of the 17th Annual Symposium on Foundations of Computer Science (FOCS'77)*, pages 222–227, 1977.