Data Structures and Algorithms (INFO-F413) Assignment 2: Zero-Sum Games and Lower Bounds

Jean Cardinal (`jean.cardinal@ulb.be`)

November 16, 2023

Background

Definitions

An $n \times m$ payoff matrix M specifies a two-person zero-sum game. Let the two players be R (for rows) and C (for columns). The value M_{ij} is the payoff due to R by C when R chooses strategy i and C chooses strategy j.

For such games, von Neumann's Minimax Theorem, proved in 1928, states that

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

where p is a discrete probability distribution on $\{1, 2, \ldots, n\}$, q is a discrete probability distribution on $\{1, 2, \ldots, m\}$, and e_i is a unit vector with a 1 at position i and 0 elsewhere. The distributions p and q correspond to *randomized*, or *mixed strategies*. For a given distribution p, the row player chooses

Figure 1: John von Neumann, father of Game Theory

strategy i with probability p_i. Similarly, the column player chooses strategy j with probability q_j.

The value considered in the theorem is the expected payoff at equilibrium, that is, when both R and C choose optimal vectors q and p. These vectors form a *Nash equilibrium* for the game.

Equilibria and Linear programming

The value in the Minimax Theorem is the optimum value of a pair of *linear programs*. Consider for example the following payoff matrix:

$$\begin{pmatrix} 3 & -1 \\ -2 & 1 \end{pmatrix}.$$

For a mixed strategy p_1, p_2 of the row player (where $0 \leq p_1, p_2 \leq 1$ and $p_1 + p_2 = 1$), the expected payoff is $\min\{3p_1 - 2p_2, -p_1 + p_2\}$. So the row player's best strategy is the pair p_1, p_2 that is a solution of

$$\max_{p_1,p_2} \min\{3p_1 - 2p_2, -p_1 + p_2\}$$

This can be expressed by the following optimization problem:

$$\begin{array}{rcl} & \max z & \\ \text{such that} & & \\ z & \leq & 3p_1 - 2p_2 \\ z & \leq & -p_1 + p_2 \\ p_1 + p_2 & = & 1 \\ p_1 & \geq & 0 \\ p_2 & \geq & 0 \end{array}$$

This is an optimization problem involving a linear function (here z) and linear constraints, hence a *linear programming* problem, for which there exists polynomial-time algorithms. One can write a similar *dual* problem for computing the right-hand term in the Minimax Theorem, that is, the best strategy for the column player. The optimal values for those two problems must be equal.

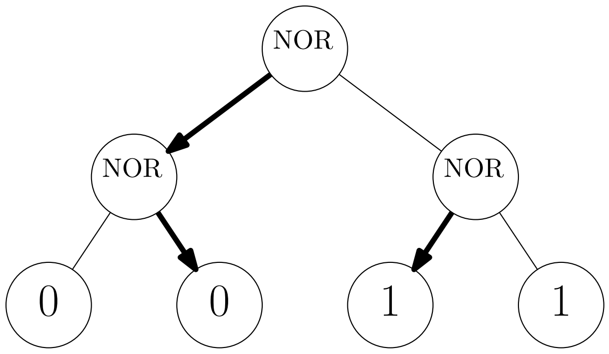

Figure 2: An instance of the Game tree evaluation. There $k = 2$, the set of possible inputs is $\mathcal{I} = \{0, 1\}^4$, the input is 0011, and the algorithm $A \in \mathcal{A}$ is the order of evaluation given by the thick arrows. The algorithm only probes the first two bits of the input.

Application to Game tree evaluation

In the interpretation of Yao's Minimax principle, the two players are the input $\mathcal{I}$ (rows), and a set $\mathcal{A}$ of deterministic algorithms (columns), and the payoff is the running time of the algorithm on the input (as measured, for instance, by the number of probes to the input or the number of elementary operations).

We propose to consider zero-sum games derived from the Game tree evaluation problem, in a binary tree of height $2k$ with NOR nodes. Here the set $\mathcal{I}$ of inputs is the set of binary strings in $\{0, 1\}^n$, with $n = 4^k$. We will restrict (this can be shown to be without loss of generality) to the set $\mathcal{A}$ of deterministic *recursive pruning* algorithms. Such an algorithm can be encoded by associating a single bit with internal NOR node, indicating which of the two subtree is evaluated first. The measure of complexity is the number of bits of the input that are probed by the algorithm.

For instance, the case $k = 2$ is illustrated on Figure 2. The thick arrows in the tree point to the node that is explored first. The only case when an internal NOR node must return 1 is when both subtrees return 0. Hence collecting a 1 in the first subtree allows the algorithm to avoid exploring the other subtree. In the example, the algorithm uses two probes: the right subtree of the root need not be explored.

Your Work

1. Write a program that, given a payoff matrix, finds the Nash equilibrium of the corresponding zero-sum game together with its expected payoff, by solving the corresponding linear program. For this purpose, feel free to use libraries for linear programming, such as `scipy` (and the function `scipy.optimize.linprog`) in Python, or the `GLPK` in C.

2. Write a program to generate instances of payoff matrices derived from the Game tree evaluation problem as explained above, for some small values of k.

3. Use the first program on the instances generated by the second to find the equilibria. Comment on the best strategies p (for the input player) and q (for the algorithm player). Can you explain the results?

Requirements:

1. A report, typeset in LaTeX, describing your methods and results for the three points above.

2. In an appendix or a separate file, the source codes of the programs performing the above tasks, in your favorite programming language.

Further Readings

- *Game Theory*, Thomas S. Ferguson, Mathematics Department, UCLA. Second Edition, 2014.
- *Understanding and using linear programming*, J. Matoušek and B. Gärtner, Springer, 2007.

Deadline

Monday December 4, 2023.