1 Introduction

In this lecture, the focus is on the Game theory principle of decision trees. In class we focused on the following subjects:

- Game tree evaluation
- n-round 2 player game
- Randomized strategies
- Minimax Principle
- Strategy bounds

2 Game tree

A game tree is a tree in which various decisions can lead to reaching a leaf node ending in a win or lose situation. Within our class, we describe this game tree as a binary tree in which the leaf nodes are labeled either a 1 or 0. The internal nodes which are odd number distance from the root are the MAX nodes and the even number of distance are the MIN nodes.

A 1 represents a "win" and a 0 represents a "lose" and these leaf node labels are computed by some oracle. Evaluating the game tree will be determining a strategy for which the MAX returns the larger value of its children while the MIN nodes return the smaller value of its children.

3 n-round 2 player game

To create a more well-defined game, we can use a game tree between 2 players alternating turns in the decision tree.

The tree will be a balanced binary tree with a 1 or 0 stored as the label of the leaf nodes. These values are created by an oracle as a function of the path take to reach the leaf node.

In this game, starting at the root, the player will alternate making 1-bit choices to go down the left or right path with a second player. The game ends when the leaf node is reached, and the player only wins if the leaf node reached is a 1.

The goal is to produce a strategy which can be used as a witness to verify which player is guaranteed to win.

3.1 Redefining Internal Nodes

Instead of only labeling the leaf nodes of the game with values, we can label all nodes as a function of the winner of that level. If the 1-bit choice of the player deciding to go left or right on the $n^{th}$ level will lead to a winning path on the remaining subtree, label the root of the subtree a 1, otherwise a 0.

In this method, the MAX nodes and MIN nodes are both trying to achieve a value of 1.

This can be calculated by recursively taking the logical NAND of the children of a node in the decision tree.
3.2 Deterministic Strategy

When the 1st player has a deterministic method for determining a winning strategy from root, the result requires $O(2^n)$ queries. This can be done using the above method taking the NAND of the labels of each nodes children recursively.

This requires $O(2^n)$ queries because the leaf nodes are labeled with an unknown oracle, and the only way to determine the label for the root node is to recursively get the NAND of the children and with a n-height binary tree, there are $2^n$ leaf nodes.

Verifying the witness however, only requires $O(2^{n/2})$ queries. This is because the root player can provide their strategy, and in the strategy, each decision cuts out half the remaining search space. They cut half the remaining search space every other step in the decision tree. This results in only considering $2^{n/2}$ possible leaf nodes.

3.3 Randomized Strategy

Because the deterministic strategy requires $O(2^n)$ queries, we can use randomness to reduce the average number of queries needed assuming a random leaf node labeling.

The technique works using the following:

1. Starting at root, recursively pick a random child
2. If child node evaluates to 0 (using the NAND of children labels), recursively evaluate the other child.
3. Return the root’s label.

To calculate the average number of queries needed to determine, let:

$$w_n = \mathbb{E} \left[ \text{queries for tree of height } n \text{ where root wins} \right]$$

$$l_n = \mathbb{E} \left[ \text{queries for tree of height } n \text{ where root loses} \right]$$

Then, as a node is labeled as the NAND of its children’s labels,

$$l_n = 2w_{n-1}$$

When there is a strategy for a winning subtree, at least one of the children must evaluate to 0 and there is a 50% chance this child is selected first. This makes:

$$w_n \leq l_{n-1} + \frac{1}{2}w_{n-1}$$

Applying recursively makes:

$$l_n \leq 2l_{n-1} + w_{n-2}$$

$$w_n \leq w_{n-1} + 1/2l_{n-2} + 1/4w_{n-2}$$

Then

$$Something_n \leq 3Something_{n-2}$$

This results in an expected $3^n$ queries using this strategy.

4 Minimax Principle

In a game like rock-paper scissors, we can construct matrices which show how strategies can succeed.

Rock-Paper-Scissors is considered a zero-sum game. This implies that no cooperation exists, and when playing between two players, the sum of the scores must be constant.

We can construct a payoff matrix $M$ for these zero-sum games played between two people. A strategy where Player 1 selects row $i$, and Player 2 selects column $j$, produces a payoff to player 1 of $M_{i,j}$
In Rock-Paper-Scissors let $M = \begin{bmatrix} R & P & S \\ R & 0 & -1 & 1 \\ P & 1 & 0 & -1 \\ S & -1 & 1 & 0 \end{bmatrix}$

In the table above, a 0 means a tie, -1 means Player 2 won, and 1 means Player 2 won. In rock paper scissors, it is trivial to show that any deterministic strategy will fail, because the other player will simply select the hand that beats the strategy.

The fix for playing rock paper scissors is to use randomized algorithm which uses a probability distribution vector $p$.

Then the expected probability of winning is $p^T M q$ where $p$ is the probability distribution for player 1 choosing Rock, Paper, or Scissors and $q$ is the probability distribution for player 2 choosing Rock, Paper, or Scissors.

Then for player 1 to maximize their chances of winning, they need to find the best strategy $p$ for the any strategy $q$ when $q$ is chosen to minimize player 1 winning. This can be written as:

$$= \sup_p \inf_q p^T M q$$

This is thought of as the player 1 choosing the best strategy when player 2 is choosing the worst strategy for player 1.

## 5 Maximin Theorem

Because in game theory, when 2 players are playing against each other in zero-sum games, and the game decision can be written as a matrix $M$, Von Nuemann’s maximin theorem states:

$$\sup_p \inf_q p^T M q = \inf_q \sup_p p^T M q$$

The best payoff player 1 can guarantee to receive is the same as the smallest payoff player 2 can guarantee player 1 receives.

We can then prove optimal lower bounds by finding distributions such that any deterministic algorithm has bad average case.

## 6 Back to NAND Game tree

We can apply some of these ideas to make proofs about the lower bounds in randomized algorithms for game trees. We can show that there is a distribution of labels in the leaves of the game tree such that any deterministic algorithm has a lower bounded running time.

We can let $p = \frac{\sqrt{5} - 1}{2}$ and all leaf nodes be set to 1 with probability $p$ and 0 with probability $1-p$. Then using the probability if a win at distance $n$ from the root is:

$$w(n) = w(n - 1) + pw(n)$$

which results in:

$$w(n) = (1 + p)^n = 1.618^n \leq \sqrt{3}^n$$

This provides a lower bound on the number of queries needed to query a decision tree using a randomized strategy. It shows that our strategy described in the previous sections is also bounded.