Game Theory I

1 Game Tree Evaluation

1.1 AND/OR Trees

Consider a tree with nodes labeled min or max on alternating levels. Each leaf has a real value, and each internal node has value equal to the min or max of its children.

The intuition is that two players take turns and play perfectly; one wants to maximize the outcome, and the other wants to minimize it. The leaf values state who won the game (or by how much). We assume this is a zero-sum game: one player's gain is the other's loss.

More generally, consider game trees of height $2k$, and let all leaves have values either 0 or 1. Then, min levels correspond to an OR operation, and max levels correspond to an AND operation. This is called an AND/OR tree. We will also assume that the tree is binary and complete (it contains 2^{2k} leaves). Our goal is to evaluate the tree while querying only few leaves.

Observe that every deterministic algorithm can be forced to query all leaves. Because the algorithm is deterministic, the adversary can predict what queries the algorithm makes next – we can treat the adversary as responding to the algorithm's queries online. The proof is then by induction on the height of subtrees, and we can show that the adversary can force the algorithm to query all leaves in a subtree to learn its value.

Concretely, the adversary just makes the first revealed subtree of an OR a 0, and the first revealed subtree of an AND a 1. Then, the deterministic algorithm has to query every leaf.

1.2 Snir's Algorithm

If the algorithm is instead randomized, this will restrict what the adversary can do, because the adversary must commit to values of all leaves – it cannot act in an online manner.

Key observation: if an OR node evaluates to 1 or an AND node evaluates to 0, there is at least one subtree which by itself determines the value of the node. If the algorithm evaluates that subtree first, it does not need to evaluate the other subtree. A deterministic algorithm cannot achieve this, but a randomized algorithm can flip a coin and evaluate the "helpful" subtree first with probability at least 1/2 (more if both subtrees are helpful). Note that this does not work is an OR node evaluates to 0 or an AND node evaluates to 1 – in that case, both subtrees must be evaluated.

Algorithm 1 Snir's Algorithm

1: Let T be the tree to be evaluated with subtrees T_1, T_2.
2: Choose uniformly randomly either T_1 or T_2.
3: Evaluate it recursively.
4: If this does not determine the value of T, evaluate the other tree recursively.

Snir's Algorithm is a Las Vegas algorithm, as the runtime is never more than $\mathcal{O}(2^{2k})$.

1.3 Runtime Analysis

Let X_k be the expected number of queries of Snir's Algorithm on trees of height $2k$.
Claim 1. $X_k \leq 3^k$.

Proof. The key idea is that if the root AND node evaluates to 1, then both OR child nodes evaluate to 1, so for each OR node there is at least one good choice of a subtree to evaluate first. It is chosen first with probability 1/2. If the root AND node evaluates to 0, then one OR child node evaluates to 0. It is chosen first with probability $\geq 1/2$.

Formally, let $E^{0/1}_{OR/AND}$ be the expected number of queries to evaluate these OR or AND nodes when the outcome is 0 or 1.

$$\begin{aligned}
E^0_{OR} &= \mathbb{E}[X_k + X_k] && \text{must evaluate both subtrees} \\
&\leq 2 \cdot 3^k \\
E^1_{OR} &\leq X_k + \frac{1}{2} X_k && \text{evaluate one tree, then possibly another} \\
&= \frac{3}{2} \cdot 3^k \\
E^0_{AND} &= \max(E^0_{OR}, E^0_{OR} + \frac{1}{2} E^1_{OR}) && \text{max of both 0, or one 0 and one 1} \\
&= E^0_{OR} + \frac{1}{2} E^1_{OR} \\
&\leq 2 \cdot 3^k + \frac{3}{4} 3^k \\
&\leq 3^{k+1} \\
E^1_{AND} &= 2E^1_{OR} \\
&\leq 3^{k+1} && \text{must evaluate both subtrees}
\end{aligned}$$

□

Thus,

$$\begin{aligned}
\mathbb{E}[X_{k+1}] &\leq \max(E^0_{AND}, E^1_{AND}) \\
&\leq 3^{k+1}.
\end{aligned}$$

Because $k = \log_4 n$, the number of evaluations is

$$\begin{aligned}
\mathcal{O}(3^{\log_4 n}) &= \mathcal{O}(n^{\log_4 3}) \\
&= \mathcal{O}(n^{0.793}).
\end{aligned}$$

So a sublinear number of leaves must be evaluated. This tells us that for every variable assignment, there is a "witness" of size $\mathcal{O}(n^{0.793})$ for the value of the tree.

1.4 Followup Questions

What can we say about the optimality of Snir's Algorithm? Is there a more clever and more efficient randomized algorithm?

One lower bound is information theoretic – in general, there is no witness of size $o(\sqrt{n})$, so each algorithm must query $\Omega(\sqrt{n})$ leaves just to prove the value. [Aside: there always is a witness of size $\mathcal{O}(\sqrt{n})$].

Can we prove a stronger lower bound on what an algorithm can achieve? More generally, given how unpredictable randomized algorithms can be, how do we prove lower bounds for them? This will be answered by Yao's Minimax Theorem, which guarantees that you can prove lower bounds on randomized algorithms by proving lower bounds on deterministic algorithms against random inputs. We derive this with a detour into some game theory.

2 Game Theory

One analogy is that a game is played between an algorithm designer and an input designer. The algorithm designer chooses an algorithm $\mathscr{A}$ and the input designer chooses an input $\mathscr{I}$. Together, they result in a "cost" $T(\mathscr{A}, \mathscr{I})$ – think runtime, memory, approximation guarantee, etc. The algorithm designer wants to minimize $T(\mathscr{A}, \mathscr{I})$, while the input designer wants to maximize it. Thus, this is a zero-sum game.

An important consideration is turn order. As in rock-paper-scissors and other zero-sum games, the first player is at a disadvantage in general. To improve chances, the first player should randomize. In the context of the algorithm-input game, this means that if the algorithm player goes first, they should choose a distribution over algorithms (a randomized algorithm); if the input player goes first, they should choose a distribution over inputs (a randomized input).

The general setup for zero-sum 2-player games is that we are given a (visible) payoff matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$ from which the row player chooses a row r and the column player picks a column c; the outcome is $a_{r,c}$. The row player wants to maximize the outcome, while the column player wants to minimize it. We assume both play perfectly.

If the row player goes first, he chooses r anticipating a response c that is best for the column player. The outcome is $\max_r \min_c a_{r,c}$. If the column player goes first, he chooses c anticipating a response r that is best for the row player. The outcome is $\min_c \max_r a_{r,c}$.

Claim 2. The first player has no advantage. That is, $\max_r \min_c a_{r,c} \leq \min_c \max_r a_{r,c}$.

Proof. Let c_o minimize $\max_r a_{r,c}$. Then

$$\max_r \min_c a_{r,c} \leq \max_r a_{r,c_o} = \min_c \max_r a_{r,c}.$$

□

If allowed, players typically prefer to randomize. A randomized (also called mixed) strategy is a distribution over rows or columns. Write $\boldsymbol{p} \in \mathbb{R}^m$ for a distribution over the m rows, and $\boldsymbol{q} \in \mathbb{R}^n$ for a distribution over the n columns. So, $\sum_{i=1}^m p_i = 1$ with $p_i \geq 0$, and $\sum_{j=1}^n q_i = 1$ with $q_i \geq 0$.

The expected value of the game when the row player plays $\boldsymbol{p}$, the column player plays $\boldsymbol{q}$, and the two players randomize independently is the weighted sum:

$$\begin{aligned}\sum_{r,c} a_{r,c} \cdot \Pr[(r,c) \text{ chosen}] &= \sum_{r,c} a_{r,c} p_r q_c \\ &= \boldsymbol{p}^\intercal \mathbf{A} \boldsymbol{q}.\end{aligned}$$

If the row player commits to a (randomized) strategy $\boldsymbol{p}$ first, and the column player responds optimally with $\boldsymbol{q}$, the outcome is

$$\max_{\boldsymbol{p}} \min_{\boldsymbol{q}} \boldsymbol{p}^\intercal \mathbf{A} \boldsymbol{q}.$$

If the column player commits first, the outcome is

$$\min_{\boldsymbol{q}} \max_{\boldsymbol{p}} \boldsymbol{p}^\intercal \mathbf{A} \boldsymbol{q}.$$

Theorem 3. *(von Neumann Minimax Theorem):*

$$\max_{\boldsymbol{p}} \min_{\boldsymbol{q}} \boldsymbol{p}^\intercal \mathbf{A} \boldsymbol{q} = \min_{\boldsymbol{q}} \max_{\boldsymbol{p}} \boldsymbol{p}^\intercal \mathbf{A} \boldsymbol{q}.$$

Proof. LP duality. □

The implication is that if the players can randomize, the order of play does not matter. An observation is that if the other player has committed to a (randomized) strategy, there is always a deterministic best response. Let $\boldsymbol{e_i}$ be the vector with 1 in component i and 0 in all others.

Theorem 4. *(Loomis):*

$$\max_{\boldsymbol{p}} \min_{i} \boldsymbol{p}^\intercal \mathbf{A} \boldsymbol{e_i} = \min_{\boldsymbol{q}} \max_{j} \boldsymbol{e_j}^\intercal \mathbf{A} \boldsymbol{q}.$$

That is, the second player can always choose an optimal deterministic strategy.

Proof. Pick the row/column with best expectation under the other player's distribution. □

Corollary 5. *If $\hat{\boldsymbol{p}}, \hat{\boldsymbol{q}}$ are any randomized strategies (not necessarily optimal), then*

$$\begin{aligned} \min_{i} \hat{\boldsymbol{p}}^\intercal \mathbf{A} \boldsymbol{e_i} &\leq \max_{\boldsymbol{p}} \min_{i} \boldsymbol{p}^\intercal \mathbf{A} \boldsymbol{e_i} \\ &= \min_{\boldsymbol{q}} \max_{j} \boldsymbol{e_j} \mathbf{A} \boldsymbol{q} \\ &\leq \max_{j} \boldsymbol{e_j}^\intercal \mathbf{A} \hat{\boldsymbol{q}}. \end{aligned}$$

The idea is to apply this to algorithm design (as rows) and input design (as columns). A randomized algorithm is just a distribution over deterministic algorithms. Crucially, we need that the number of algorithms and number of inputs are both finite. This applies, for instance, if we look at a fixed input size n, and the algorithm's runtime is deterministically bounded (*e.g.,* as a function of n). This could be problematic for some Las Vegas algorithms whose runtime may grow unboundedly with small probability (*e.g.,* coupon collector).