

Lecture 4: Pure Maximin Strategies and Perfect Information Games

Lecturer: Peter Bro Miltersen

Scribe: Sarah Zakarias

1 Deciding existence of pure maximin strategies

Below is the game we considered last lecture.

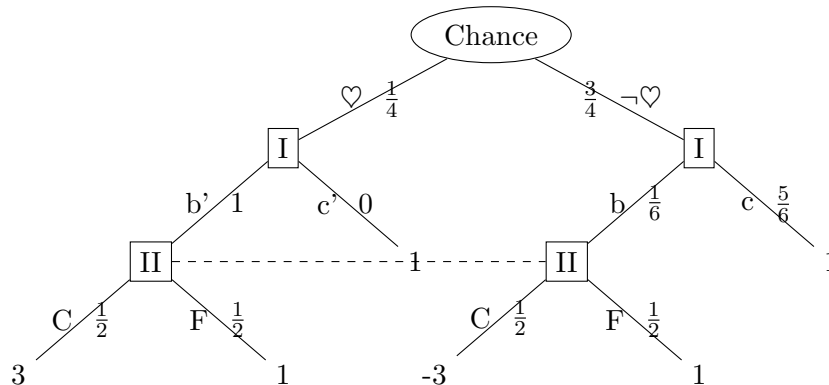


Figure 1: Basic endgame in poker

For this game we found the unique maximin/minimax strategies for player1/player2. These are shown as probabilities on the edges. In general we would like to find pure maximin/minimax strategies if they exist. As an example we now look at a game which is a slight modification of the above. We deal Player 1 a card from a deck of 24 cards, six hearts, six diamonds, six clubs and six spades, including the four aces.

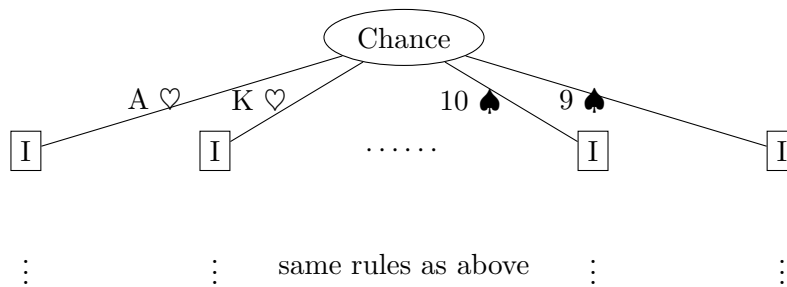


Figure 2: Modified poker game

This game has a pure maximin behavior strategy for player 1, namely:

- If ♠ bet
- If ¬♠
 - If ace bet

– If \neg ace check

In this way we let the card decide the “randomness” we needed before where player 1 in the last case should bet with probability $\frac{1}{6}$ and check with probability $\frac{5}{6}$. In general we have the following:

Computational Problem 1 *Given a two-player zero-sum extensive form game with perfect recall, does it have a pure maximin behavior strategy¹?*

Proposition 1 *Computational problem 1 is NP-hard, even if chance nodes are restricted to uniform distribution.*

Proof The proof is done by a reduction from Exact Binpacking. Recall the definition of this problem:

Exact Binpacking: Given a set of positive integers: $A = \{a_1, \dots, a_n\}$ and a positive integer m , we want to answer the following: Can $\{1, 2, \dots, n\}$ be partitioned in m parts $I_0, I_1, I_2, \dots, I_m$ such that $\sum_{i \in I_j} a_i$ is the same for all j ?

We know that this problem is NP-hard (in fact strongly NP-hard) so doing a reduction from this yields the result. In the reduction we use a gadget, namely the game which in strategic form is the matrix game where the matrix is just the $m \times m$ identity matrix negated. E.g., if $m = 3$:

$$\mathbf{M} = \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{bmatrix}$$

M describes a game where player 1 commits to a number $\in \{0, 1, 2, 3, \dots, m-1\}$ and player 2 guesses which. If he guesses correct player 1 gives him a cookie. This game has value $-\frac{1}{m}$ and the maximin/minimax strategy is the uniform distribution on m rows/columns².

The reduction goes as follows: Given an instance of Exact Binpacking, $A = \{a_1, \dots, a_n\}$ and m , we construct the following game, $G(A)$ (illustrated for $m = 3$):

Lemma 2 *$G(A)$ has a pure maximin strategy $\Leftrightarrow A$ is a yes instance.*

Proof “ \Leftarrow ” Assuming A is a yes instance, i.e. it can indeed be divided into m equally large parts, we want to convince ourselves that $G(A)$ has a pure maximin strategy. This is the same as with the modified poker game, we get the randomness needed from the random item distributed.

“ \Rightarrow ” Now we assume that $G(A)$ has a pure maximin strategy and we have to construct a partition of A . We do this by putting items in bins matching player 1’s choice in the strategy.

¹We note here that in this context it does not make a big difference if we are talking about a plan or a strategy. The only difference is whether we specify behavior in every information set or not. In a plan we don’t specify behavior in nodes that are not reachable with the given choices.

²Note that this game is *not* symmetric. M is symmetric and for a game to be symmetric, its matrix must be skewsymmetric.

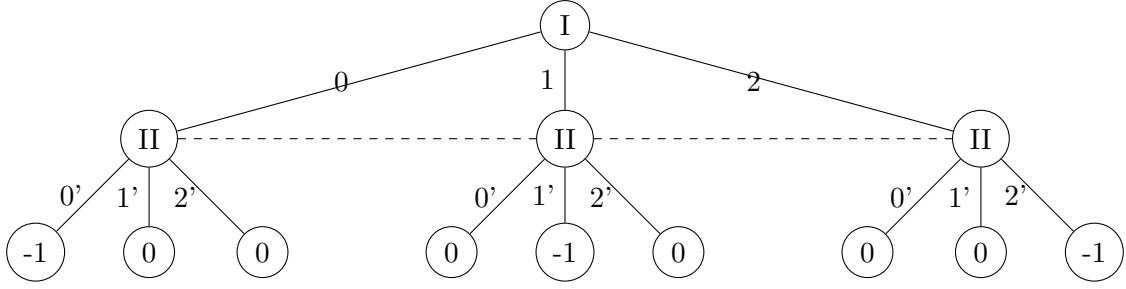


Figure 3: Game tree for M

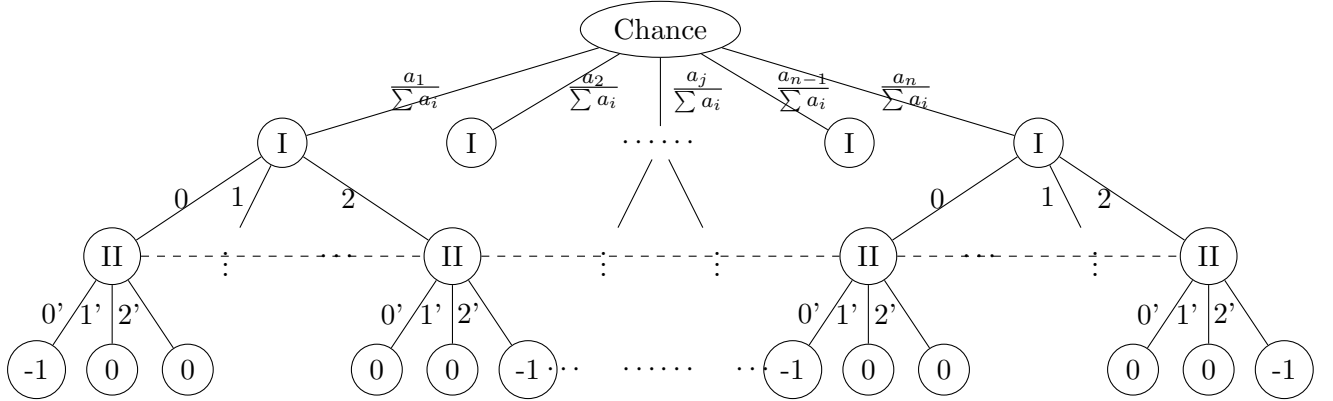


Figure 4: Game tree for $G(A)$

For example if in node I where the edge from the parent has probability $\frac{a_j}{\sum_{i=1}^n a_i}$ the choice is 0 we put item a_j into bin I_0 and so on. The claim is, that this is a correct partition. Assume for contradiction that it is not. Then $\exists j, \text{wlog } j = 0$ such that $\sum_{i \in I_0} a_i > \frac{\sum_{i=1}^n a_i}{m}$. But then the pure strategy of player 2 can always choose 0 which makes player 1's payoff $< -\frac{1}{m}$. But this means that it is not a pure maximin strategy, and we have the contradiction! \square

With the proof of Lemma 2 we have completed the reduction and thereby the proof of the first part of Proposition 1. \square

To prove the "even ..." part we first need to introduce the following concept:

Definition 3 (Strongly NP-hard) *A problem is strongly NP-hard if it is NP-hard when numbers in the instances are represented using unary notation rather than binary or decimal. (For example, $\text{unary}(8) = 11111111$)*

Proof (Proposition 1: "even ...") We are again given an instance of Exact Binpacking, but now with unary representation of numbers: $A = \{a_1, \dots, a_n\} = \{\text{unary}(a_1), \dots, \text{unary}(a_n)\}$ and m . The problem is still NP-hard, as Exact Binpacking is strongly NP-hard. In the reduction, if for example $a_1 = 7$, instead of having one edge with probability $\frac{a_1}{\sum_{i=1}^n a_i}$ in the game tree, we have 7 edges, each with a subtree like the subtree the original node had. In this way we have uniform distribution on the edges from the chance node. The proof of correctness is the same as before. \square

Fact 4 (Exercise) For two-player zero-sum games of perfect recall without chance nodes, existence of pure maximin strategies can be determined in linear time by a tree traversal if the value of the game is known.

2 Perfect Information Games

Definition 5 (Perfect information games) A perfect information game is an extensive form game where all information sets are singletons ("A plain game tree").

An example of such a game is seen in Figure 5. Note that for such games we can cut off any subtree thereby yielding a subgame. The reason we can do this, is because we will never cut in such a way that we split two information sets. In general if we cut off a subtree without splitting information sets we get a subgame.

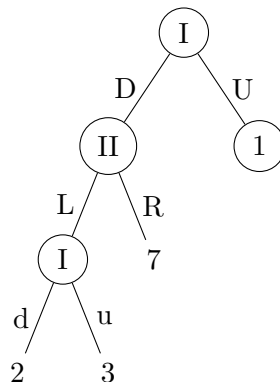


Figure 5: A perfect information game

Proposition 6 Perfect information games have pure maximin/minimax strategies

Proof (Backward induction) We do induction in the height of the tree. We have three cases, which are shown in Figures 6, 7, 8.

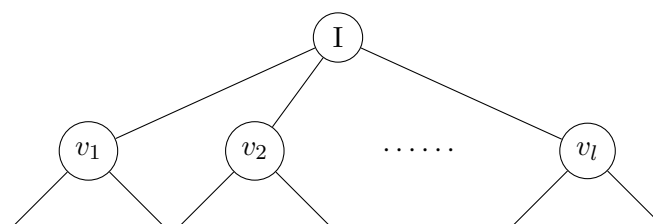


Figure 6: Case 1

In the first case the root node belongs to player 1. Each of the l subgames have pr. induction a pure maximin strategy and a value. Player 1's strategy will simply be to choose the subgame with the largest value, so we get $v = \max(v_1, \dots, v_l)$. The second case is very similar, here instead the value will be the minimum: $v = \min(v_1, \dots, v_l)$. And in the last case we have a chance node at the root. Here the value will be the weighted value of the subgames: $v =$

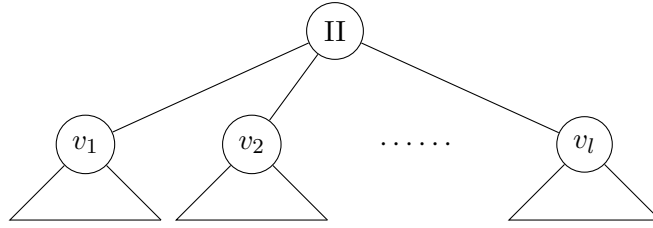


Figure 7: Case 2

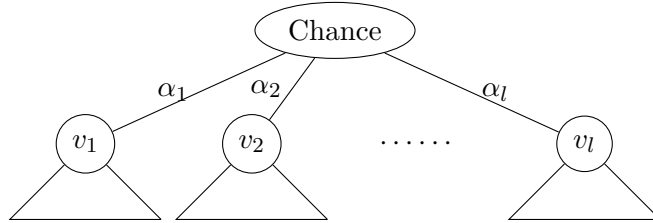


Figure 8: Case 3

$\sum_{i=1}^n v_i \alpha_i$. In this case we have to specify behavior in all subgames because we can end up in all of these (where $\alpha_i \neq 0$).

□

We can apply the technique from the proof on the example of Figure 5. This is done in Figure 9. We see that the value of the game and all of the subgames in this case is 3. This computation can be done by a tree traversal in linear time.

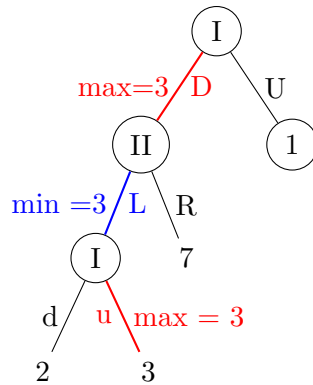


Figure 9: Computing value for game and subgames

Corollary 7 *Given a two-player, zero-sum perfect information game, value and pure maximin strategy can be computed in linear time.*

3 Sublinear time evaluation of game trees

Proposition 8 *Sublinear time for computing value and pure maximin strategy of a two-player zero-sum perfect information game is not possible in general.*

Proof Consider the extensive form game consisting of a single position belonging to Player 1 and with N leaves, each with a payoff of 0. Suppose an algorithm terminates outputting “value 0” without having examined all the leaves. Then, it is consistent with the part of the input that the algorithm has read that the value of the game is, say, 1, so the algorithm is not correct. \square

Definition 9 (Perfect Game Tree) A perfect game tree is a special case of perfect information games where we furthermore have:

- In each position there are exactly two options/ each node has exactly two children
- There is perfect alternation between the two players
- The tree is perfectly balanced
- Payoff's are 1 or -1

Note that a maximin plan of a perfect game tree with N leaves consists of only approximately \sqrt{N} distinguished actions. It is conceivable that we could compute one in sublinear time. In fact, we can.

Theorem 10 (Saks and Wigderson, 1986) There is a randomized algorithm that computes the value and a maximin plan for a perfect game tree and runs in expected time $O(N^{\log(\frac{1}{4} + \frac{\sqrt{33}}{4})})$, i.e. $O(N^{0.8})$ where N is the size of the tree.

Note that a maximin plan of a perfect game tree with N leaves consists of only approximately \sqrt{N} distinguished actions.

Theorem 11 (Saks and Wigderson, 1986) No randomized algorithm (which always gives the right answer) has a better expected running time than the one in Theorem 10.

Proof (Theorem 11) We won't cover the proof, but simply report that the proof uses the minimax theorem! \square

Proof (Theorem 10) We only do the proof for computing the value of the game. We did not have time to do the whole proof, but we did the setup and in the next lecture we will finish the proof.

Recall that when we compute the value in the tree traversal it corresponds to taking the maximum when the node belongs to player I and the minimum for player II, see Figure 10.

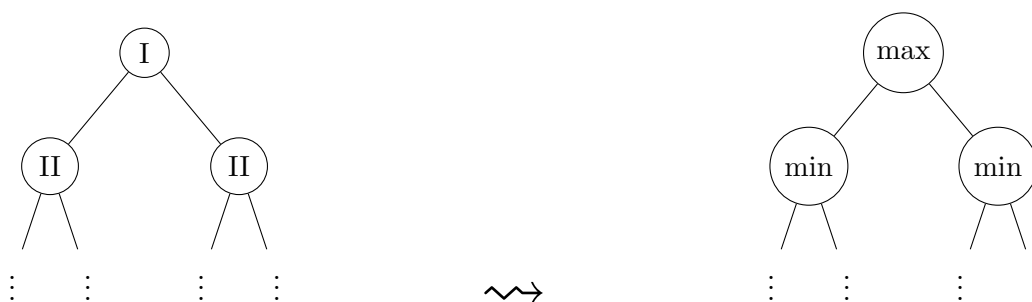


Figure 10: Converting a game tree to a formula

What we want is an algorithm that is able to evaluate a balanced alternating Max/Min formula with inputs from $\{-1, 1\}$. To simplify this we note that $\min(a, b) = -\max(-a, -b)$. We now get a tree that looks like the following:

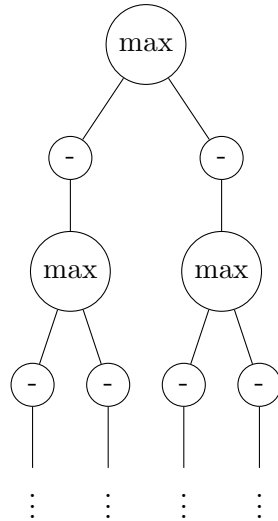


Figure 11: Converting min to max in tree

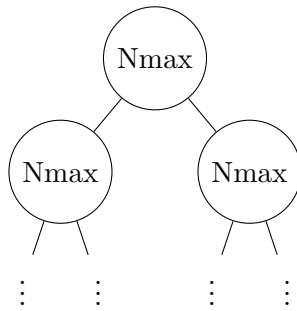


Figure 12: Nmax formula tree

If we now define $Nmax = \max(-a,-b)$ the above tree can be redrawn as:

We now have a Negamax expression, perfectly balanced tree. If we evaluate Nmax an all possible inputs we get: $Nmax(1,1) = -1$, $Nmax(-1,1) = 1$, $(1,-1) = 1, (-1,-1) = 1$ and observe if one of the inputs are -1, then Nmax is 1. We can use this information to optimize an algorithm evaluating the expression.

The time bound will be proven in the next lecture. □

Algorithm 1 *Eval*(NMax(f_1, f_2))

```
1: Flip a coin and choose  $i \in \{1, 2\}$  uniformly at random.
2:  $v = \text{Eval}(f_i)$ 
3: if (  $v = -1$  ) then
4:   return 1
5: else
6:    $v' = \text{Eval}(f_{3-i})$ 
7:   if ( $v' = -1$ ) then
8:     return 1
9:   else
10:    return -1
11:  end if
12: end if
```
