Adversarial Search

1 History of Games in AI

For a brief history of game playing in AI, we refer you to this AAAI 2020 panel, moderated by Professor Greenwald. Panelists include AI luminaries Murray Campbell (Chief architect of Deep Blue), Mike Bowling (Leader of the Computer Poker Research Group at Alberta), and David Silver (creator of AlphaGo), as well as Chess Grandmaster Garry Kasparov.

2 Two-Player Zero-Sum Games

A game is a very general model of agent interactions. One class of games that has been a very common object of study among AI practitioners is two-player, alternating-move, zero-sum games, such as Tic-Tac-Toe, Chess, and Poker. Among these examples, Tic-Tac-Toe are Chess are perfect-information games, while Poker—a card game—is a game of imperfect information, because some of the players' cards are hidden. Moreover, while Tic-Tac-Toe and Chess are *deterministic*, as is Poker once the cards are dealt, in Backgammon, there is *stochasticity* throughout, because every move depends on the roll of the dice.

Alternating-move games of perfect information can be represented as tuples $\Gamma = \langle P, X, S, T, \mathcal{T}, l, \boldsymbol{v} \rangle$, where

- P is a set of n players[1]
- X is a finite set of states
- $S \subseteq X$ is a nonempty set of start states
- $T \subseteq X$ is a nonempty set of terminal states
- $\mathcal{T} : X \rightrightarrows X$ is a state transition function
  $\mathcal{T}(x)$ is the set of successor states of x
- $l : X \to P$ labels state x with the player who moves at x
- $\boldsymbol{v} : T \to [-1, 1]^n$ maps terminal states into real-valued vectors
  $v_i(x) \in [-1, 1]$ is the payoff to player i at state x

Zero-sum games are so-called because $\sum_{i=1}^{n} v_i(x) = 0$, for all $x \in T$. In two-player, zero-sum games the two players are viewed as adversaries; one player is the maximizer (MAX); the other is the minimizer (MIN), i.e., $v_{\mathsf{MAX}}(x) = -v_{\mathsf{MIN}}(x)$. In such games, it suffices to use one value v to represent the value of a state: if $v(x) > 0$ ($v(x) < 0$), then MAX (MIN) is the winner; if $v(x) = 0$, then the game is a draw.

As an example, consider the game of m, p-NIM. Initially, there are p piles of m matches. To move, a player removes any number of matches from exactly one pile. The losing (or winning) player is s/he who removes the final match. The game tree representation of 2,2-NIM (for two players) is depicted in Figure 1, where it is modeled as follows:

[1] In this formalism, stochasticity can be modeled as a **chance** player, whose moves depend on the outcome of randomness.

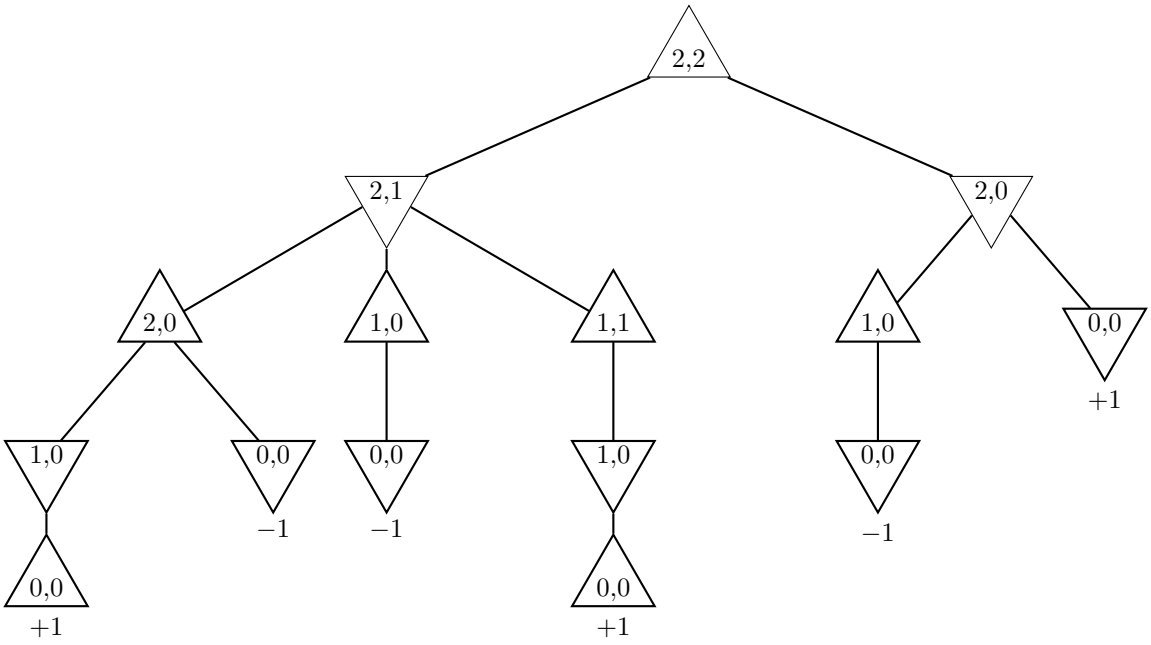

Figure 1: Game tree for 2,2-Nim. MAX nodes are upright triangles; MIN nodes are upside-down triangles. The minimax value of this game is -1: there is a winning strategy for MIN.

- $P = \{\textsc{Max}, \textsc{Min}\}$
- $X = Y \times P$: e.g., $((1,1), \textsc{Max}) \in X$
  $Y = \{(2,2),(2,1),(2,0),(1,1),(1,0),(0,0)\}$
  canonical representation where $a \geq b$ for all pairs (a,b)
- $S = \{((2,2), \textsc{Max}), ((2,2), \textsc{Min})\}$
- $T = \{((0,0), \textsc{Max}), ((0,0), \textsc{Min})\}$
- $\mathcal{T}((2,2), \textsc{Max}) = \{((2,1), \textsc{Min}), ((2,0), \textsc{Min})\}$,
  $\mathcal{T}((2,1), \textsc{Min}) = \{((1,1), \textsc{Max}), ((1,0), \textsc{Max}), ((2,0), \textsc{Max})\}$, ...
- $l((2,2), \textsc{Max}) = \textsc{Max}$,
  $l((2,1), \textsc{Min}) = \textsc{Min}$,
  $l((2,0), \textsc{Min}) = \textsc{Min}$, ...
- $\boldsymbol{v}((0,0), \textsc{Max}) = (+1,-1)$,
  $\boldsymbol{v}((0,0), \textsc{Min}) = (-1,+1)$

3 Minimax Search

To solve a game is to predict its outcome. That is, to predict how the game will be played. To predict how a game will be played, it is necessary to make some assumptions about the players'—or agents'—behavior. The most common assumption is to assume all agents behave rationally. **Rationality** in the context of games (including non-zero sum games) means acting so as to maximize **utility**, where utility is a measure

MinimaxValue
Inputs game tree Γ_x rooted at x
Output minimax value

1. if $x \in T$, `return` v
2. if $l(x) =$ Max, `return` MaxValue(Γ_x)
3. if $l(x) =$ Min, `return` MinValue(Γ_x)

MaxValue
Inputs game tree Γ_x rooted at x
Output minimax value

1. $v = -\infty$
2. for all $y \in \mathcal{T}(x)$
   (a) $v = \max\{v, \text{MinimaxValue}(\Gamma_y)\}$
3. `return` v

MinValue
Inputs game tree Γ_x rooted at x
Output minimax value

1. $v = +\infty$
2. for all $y \in \mathcal{T}(x)$
   (a) $v = \min\{v, \text{MinimaxValue}(\Gamma_y)\}$
3. `return` v

Table 1: Algorithm for computing the minimax value of a game.

of an agent's satisfaction with an outcome. In zero-sum games, specifically, a rational (Max) agent acts to *maximize* the value of the game, while a rational (Min) agent acts to *minimize* the value of the game.

Perhaps one of the longest-standing open questions in game theory is whether white (or black) has a winning strategy in Chess. That is, can either agent *force* a win, meaning no matter how black (resp. white) plays, can white (resp. black) guarantee a win? We cannot (yet) answer this question for Chess, but Zermelo's theorem [**?**] tells us that one of the following three statements must be true: either white has a winning strategy, or black has a winning strategy, or optimal play for both results in a draw.

The proof of Zermelo's theorem is by **backward induction** [**?**], and applies to all *finite* two-player, alternating-moves, zero-sum games of perfect information. A game's value at each leaf node is dictated by the rules of the game. It is therefore conceptually simple to solve games of depth 1: it suffices to look at all the successors and choose one that is utility maximizing: i.e., one that yields a value of $+1$ when it is Max's turn to move, and -1 when it is Min's turn. But now that we know the value of a game of depth 1, we can apply the same procedure to compute the value of a game of depth 2. And so on.

In this way, a (finite) game's value can be computed by backing up values from the leaf nodes to the root. Hence, the name "backward" induction. The ensuing value is called the **minimax value** of the game. If the minimax value of a game is $+1$, then there exists a winning strategy for Max; if the minimax value is -1, then there exists a winning strategy for Min; if the value of the root node is 0, then neither player has a winning strategy, and optimal play results in a draw.

Figure 2:

The definition of the minimax value of a game suggests a breadth-first-search style computation, as each interior node's value is the maximum or the minimum of its childrens' values. In practice, however, the minimax algorithm usually traverses nodes in depth-first-search (DFS) order to ensure that space is managed efficiently. We present a recursive version of minimax, which naturally traverses nodes in DFS order.

Like any recursive algorithm, the minimax algorithm computes the values of smaller and smaller problem instances before backing up any values. Hence, we use the following notation in our pseudocode (see Table 1): given game Γ, we let Γ_z denote the subgame Γ rooted at node z.

In the base case, i.e., at the terminal nodes, values are determined by the game. In the inductive step, two cases arise: The value at a MAX node y is initialized to $-\infty$, and then minimax is called on Γ_y. Similarly, the value at a MIN node z is initialized to $+\infty$, and then minimax is called on Γ_y. When these recursive calls return, the values returned are "maxed" or "minned" as appropriate with the current value of x.

The pseudocode presented in Table 1 returns the minimax value of the game, but it does not return a minimax strategy. Formally, a strategy in a game, often called a **policy**, is a function from states to actions. It is straightforward to generalize our pseudocode to return an action that produces the minimax value of a node, rather than just the minimax value itself. An **optimal** policy comprises such actions.

4 Depth-Limited Search

Although a minimax solution always exists, it is intractable to compute one in most interesting games.[2] Instead, in most implementations of adversarial search, search proceeds to some limited depth or for some limited time, at which point expansion of the search tree is truncated, and an evaluation function is employed to estimate the minimax value. Thus, as in informed search, the name of the game (no pun intended!) is to encode domain knowledge into a heuristic to guide the search.

An **evaluation function** for a game is a function from game states to values. For example, to evaluate a state in Tic-Tac-Toe from the point of view of, say, X, you might ask how many opportunities there are for X to complete a line from that state. From the initial state (an empty board), X can move to a state with an X in a corner, along a side, or in the middle (see Figure 4). A corner state presents the opportunity to complete three lines, while a side state offers only two opportunities, and the middle state offers four.

A more general approach to designing evaluation functions is to enumerate the important features of a game state, and to then score those features individually, before combining them to produce a heuristic value. For example, in Chess, the features might be how many of each type of piece are on the board: Let q_w denote the no. white queens; r_w, the no. white rooks; k_w, the no. white knights; s_w, the no. white bishops; and p_w, the no. white pawns; and likewise, for Black's pieces. We can then linearly combine these features, weighting each by a score that corresponds to its material value. In this way, we can evaluate the quality of a state x for each of White and Black as follows:

$$w(x) = 9q_w + 5r_w + 3k_w + 3s_w + 1p_w$$

$$b(x) = 9q_b + 5r_b + 3k_b + 3s_b + 1p_b$$

One simple evaluation function is then $e(x) = w(x) - b(x)$.[3] A positive value indicates an advantage for

[2]Indeed, games like TIC-TAC-TOE for which computing the minimax solution is tractable could be said to be uninteresting for precisely that reason!

[3]If desired, these values can be normalized to lie in the range $[-1, +1]$, by dividing $w(x) - b(x)$ by $w(x) + b(x)$, but they need not be.

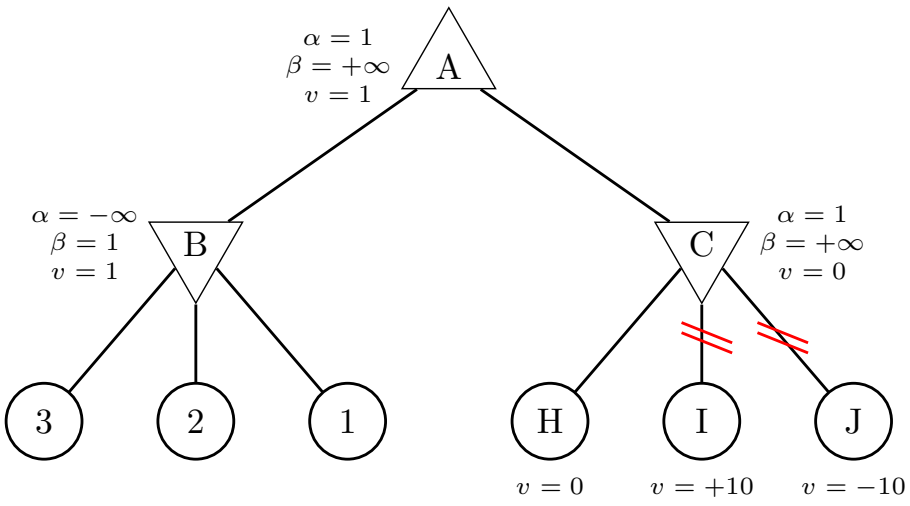

Figure 3: An example of $\alpha\beta$pruning. The subtrees rooted at I and J can pruned because their values do not impact the value of node A. Indeed, the pruning test passes: $v = 0 \leq 1 = \alpha$.

White; a negative value, an advantage for Black; and zero, no advantage either way.

Exercise What are some limitations of this heuristic evaluation function?

Evaluation functions return heuristic estimates, which are not perfect. As a result, choosing a depth or time at which to truncate search in game trees is a delicate matter. If $e(x)$ is changing rapidly, then single moves dramatically affect the (apparent) value of x. One popular heuristic is to allow search to proceed until *quiescence*. Another difficulty is that search algorithms cannot recognize drastic changes in the values of nodes if delay tactics push drastic moves beyond the horizon; this is called the *horizon effect*. One proposed solution to this problem is to engage in a secondary search beyond the seemingly best node. This technique is called *singular extension*. If it is determined that this path degrades, then a secondary search is performed on the second-best node; but it is impractical to conduct a secondary search on all nodes.

5 $\alpha\beta$-Pruning

The $\alpha\beta$-pruning algorithm is an alternative to minimax, which also computes the minimax value of a game, but it scales better, because—as the name suggests—it prunes subtrees once it determines that their values cannot alter the minimax value computed so far. Pseudocode for a recursive version of $\alpha\beta$-pruning appears in Table 2. This pseudocode varies only very slightly from the minimax algorithm itself. Thus like minimax, it is straightforward to implement a depth-limited version of $\alpha\beta$-pruning.

The main idea of $\alpha\beta$ pruning can be understood by example. Consider the game tree in Figure 5. Max is the first to move in this game. Were she to move left, her value would be 1, the minimum of Min's three moves at node B, which yield 3, 2, and 1, respectively. Instead, by taking the right branch, and letting Min evaluate his leftmost branch (node H), Max learns that her value at node C is bounded above by 0. Why? Because it is Min's move at node C, and although C's value may ultimately be less than 0, Min will ensure that its value is never *more* than 0. Therefore, since Max can guarantee a value of 1 by moving left, and cannot earn any more than 0 by moving right, any branches below node C can be pruned—their values have no bearing on the minimax value of the game.

The $\alpha\beta$-pruning algorithm is nearly identical to minimax. The only difference is that two additional book-keeping parameters, unsurprisingly called α and β, are maintained throughout the computation, and applied for pruning purposes. The former, α, represents the best known value for MAX along an alternative path in the game tree; analogously, the latter, β, represents the best known value for MIN along an alternative path in the game tree. In this way, if ever during normal minimax operations, it is determined that $v \geq \beta$ or $v \leq \alpha$, all remaining subtrees of the tree currently being evaluated can be pruned, because it has been determined that there is a higher (resp. lower) value for MAX (resp. MIN) along an alternative path.

In our example, the α value at the root is updated to $+1$ after the value of the left branch (node B) is determined. Thus, $\alpha = +1$ when $\alpha\beta$pruning is called recursively on the subtree rooted at C. Now, when 0 is returned as the value at node H, and C's value is updated accordingly, all subtrees beneath C can be pruned, because the current value at C, namely 0, is less than α, namely $+1$.

The initial call to $\alpha\beta$-pruning initializes α and β to their lower and upper bounds, respectively, namely, -1 and $+1$, as no values anywhere in the game tree have as yet been determined. When game values are estimated by a heuristic evaluation function that ranges between $-\infty$ and $+\infty$, as is forthcoming in Section 4, α and β should be initialized to these values instead.

In practice, $\alpha\beta$pruning may prune many nodes, or it may prune very few nodes. Its effectiveness depends on the order in which nodes are evaluated. At its best, it can search to a depth twice that of minimax. For example, in the game of Chess, $\alpha\beta$ pruning might enable search to a depth of 40 rather than 20, which sounds fantastic since the length of an average Chess game is around 40; however the branching factor in Chess is 35, and 35^{20} is still about 7.6×10^{30}!

One sensible way to order nodes during $\alpha\beta$pruning is according to the heuristic evaluation function e. Another trick is to use iterative deepening on top of depth-limited $\alpha\beta$pruning. In this way, search can continue until the allotted time is elapsed, but a solution is guaranteed at any time.

References

[1] Donald E. Knuth. Representing numbers using only one 4. *Mathematics Magazine*, 37(5):308–310, 1964.

$\alpha\beta$PRUNING

Inputs: game tree Γ_x rooted at x
α: the best known value for MAX along an alternative path
β: the best known value for MIN along an alternative path
Output: minimax value

1. if $x \in T$, `return` v
2. if $l(x) =$ MAX, `return` MAXVALUE$(\Gamma_x, \alpha, \beta)$
3. if $l(x) =$ MIN, `return` MINVALUE$(\Gamma_x, \alpha, \beta)$

MAXVALUE

Inputs: game tree Γ_x rooted at x
Output: minimax value

1. $v = -\infty$
2. for all $y \in \mathcal{T}(x)$
   (a) $v = \max\{v, \alpha\beta\text{PRUNING}(\Gamma_y, \alpha, \beta)\}$
   (b) if $v \geq \beta$, return v
   (c) $\alpha = \max\{\alpha, v\}$
3. `return` v

MINVALUE

Inputs: game tree Γ_x rooted at x
Output: minimax value

1. $v = +\infty$
2. for all $y \in \mathcal{T}(x)$
   (a) $v = \min\{v, \alpha\beta\text{PRUNING}(\Gamma_y, \alpha, \beta)\}$
   (b) if $v \leq \alpha$, return v
   (c) $\beta = \min\{\beta, v\}$
3. `return` v

Table 2: $\alpha\beta$pruning algorithm for computing the minimax value of a game. The meanings of these parameters are: α (resp. β) is the best known value for MAX (resp. MIN) along an alternative path through the tree.