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. Although of the aforementioned games are *deterministic*, while Backgammon is a *stochastic*, two-player, alternating-move, zero-sum games.

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

- P is a set of n players
- 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 require that $\sum_{i=1}^{n} v_i(x) = 0$, 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:

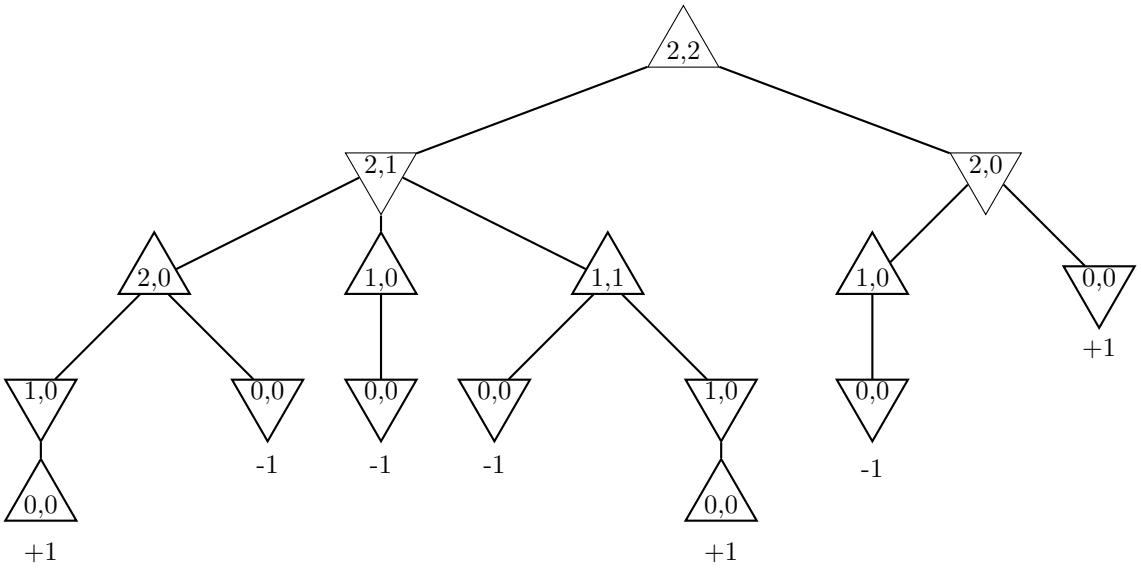

Figure 1: Game tree for 2,2-Nim. MAX nodes are upright triangles; MIN nodes are upside-down triangles. The minimax value of this game is -1: there is a winning strategy for MIN.

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

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 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 [2] 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** [1], 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.

The definition of the minimax value of a game tree suggests a breadth-first-search style computation. In practice, however, the minimax algorithm traverses nodes in depth-first-search (DFS) order to ensure that space is managed efficiently.

We present recursive versions of the MINIMAX and $\alpha\beta$-PRUNING adversarial search algorithms. These algorithms search smaller and smaller game trees. Hence, we introduce the following notation: given (game) tree Γ, we let Γ_z denote the sub(game) tree of Γ rooted at node z.

3.1 Recursive MiniMax

This recursive minimax algorithm traverses nodes in depth-first-search order, by virtue of the fact that the algorithm is recursive. The initial call initializes $\alpha = -1$ (lower bound) and $\beta = +1$ (upper bound). The algorithm updates α values at MAX nodes and β values at MIN nodes. In the base case, the terminal nodes' values are returned. In the inductive step, two cases arise. If the root node x is a MIN (MAX) node, then as its successor's are evaluated, their values are "backed up"—"minned" ("maxed") with the value of x—until x's value is computed, at which point this updated value, β (α), is returned.

References

[1] Laszló Kalmár. Zur theorie der abstrakten spiele. *Acta Scientiarum Mathematicarum (Szeged)*, 4(1–2):65–85, 1928–29.

[2] Ernst Zermelo. Über eine anwendung der mengenlehre auf die theorie des schachspiels. In *Proceedings of the fifth international congress of mathematicians*, volume 2, pages 501–504. Cambridge University Press, 1913.

MiniMax(Γ_x, α, β)

Inputs: game tree Γ_x rooted at x
lower bound α
upper bound β

Output: minimax value

1. if $x \in T$
   (a) if $l(x) =$ Min, `return` $v_{\text{MIN}}(x)$
   (b) if $l(x) =$ Max, `return` $v_{\text{MAX}}(x)$
2. if $l(x) =$ Min
   (a) for all $y \in \delta(x)$
      i. $\beta = +\infty$
      ii. $\beta = \min\{\beta, \text{MiniMax}(\Gamma_y, \alpha, \beta)\}$
   (b) `return` β
3. if $l(x) =$ Max
   (a) for all $y \in \delta(x)$
      i. $\alpha = -\infty$
      ii. $\alpha = \max\{\alpha, \text{MiniMax}(\Gamma_y, \alpha, \beta)\}$
   (b) `return` α

Table 1: Recursive MiniMax.