

Hannes Strass

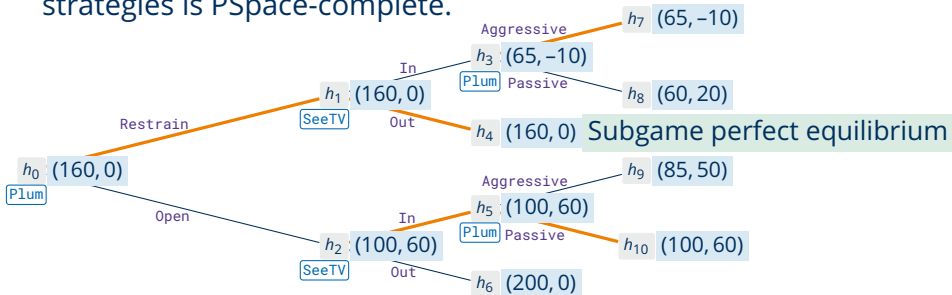
Faculty of Computer Science, Institute of Artificial Intelligence, Computational Logic Group

Playing Games: Alpha-Beta Tree Search

Lecture 4, 6th May 2024 // Algorithmic Game Theory, SS 2024

Previously ...

- **Game trees** are used to represent sequential (**extensive form**) games.
- Sequential games give rise to (different) strategic (normal form) games.
- In a game tree, a **strategy** assigns a move to each decision node.
- **Backward induction** can be used to solve sequential games.
- The **subgame perfect equilibrium** of a sequential game coincides with its backward induction solution.
- Geography is a game on graphs for which deciding existence of winning strategies is PSpace-complete.



Overview

Two-Player Zero-Sum Games

Alpha-Beta Pruning

Heuristics

Two-Player Zero-Sum Games

Zero-Sum Games

Definition

A game with players P is **zero-sum** iff for all outcomes $z \in Z$, $\sum_{i \in P} u_i(z) = 0$.

Note: Every combinatorial game is zero-sum, but not vice versa.

Examples: Penalties, Rock-Paper-Scissors, Chess, Go

In what follows, we will focus on **two-player zero-sum games**.

Observation

For a two-player zero-sum game (with $P = \{1, 2\}$), the payoffs $\mathbf{u} = (u_1, u_2)$ are fully specified by giving u_1 , as for every $z \in Z$ we have $u_2(z) = -u_1(z)$.

Two-Player Zero-Sum Sequential Games

We thus adapt our definition of sequential games with perfect information:

Definition

A **two-player zero-sum sequential game with perfect information** has:

1. The set $P = \{\text{max}, \text{min}\}$ of two (named) players.
2. A tuple $(M_{\text{max}}, M_{\text{min}})$ of sets of moves for each player; $M := M_{\text{max}} \cup M_{\text{min}}$.
3. A set H of histories, sequences $[m_1, \dots, m_k]$ of moves $m_j \in M$.
4. A subset $Z \subseteq H$ of terminal histories.
5. A player function $p: H \setminus Z \rightarrow P$ (indicating whose turn it is).
6. A utility function $u_{\text{max}}: Z \rightarrow \mathbb{R}$ for player **max**.

Starting with the empty history $[],$ in each history $h = [m_1, \dots, m_k] \in H \setminus Z,$ player $i = p(h)$ chooses a move $m \in M_i,$ leading to the history $[m_1, \dots, m_k, m].$

Histories and States

Typically, it is more useful to describe a game other than through histories:

Definition

A **state-based game model** consists of the following:

- A set S of **states** of the game, with **initial state** $S_0 \in S$, and functions:
 - **TURN**: $S \rightarrow P$ saying whose turn it is in a state.
 - **MOVES**: $S \rightarrow 2^M$ yielding the legal moves in a state.
 - **RESULT**: $S \times M \rightarrow S$ yielding the result of a move in a state (the next state).
 - **IS-TERMINAL**: $S \rightarrow \{\top, \perp\}$ indicating whether a state is terminal.
 - **UTILITY**: $S \rightarrow \mathbb{R}$ giving a terminal state's payoff for **max** (else undefined).
-
- Each history leads to exactly one state. (\square leads to S_0 .)
 - One state may be reached through different histories.

Example: A state in Chess is given by the locations of the pieces on the board.

State Spaces and Their Representation

Definition

The **state space graph** associated with a state-based game model is the edge-labelled directed graph (V, E) with $E \subseteq V \times M \times V$, where

- $V \subseteq S$ is the \subseteq -least set such that $S_0 \in V$, and:
 - if $s \in V$ and $m \in \text{MOVES}(s)$, then $\text{RESULT}(s, m) \in V$.
- $(s_1, m, s_2) \in E$ iff $\text{RESULT}(s_1, m) = s_2$.
- The state space contains all states that are reachable from the initial state by sequences of legal moves.
- The state space can be huge: for Chess, there are at least 10^{40} positions (states).
- We thus typically only search parts of the state space (game tree).

Representing Games for Search

We will assume that the game tree is not explicitly given, but implicitly specified by a state-based game model that is parsimoniously represented (e.g. using a game description language like Stanford University's GDL).

Assumption: Game Representation

A state-based game model can be represented such that:

- The set S of states is described as an efficiently decidable formal language.
- The functions **TURN**, **MOVES**, **RESULT**, **IS-TERMINAL**, and **UTILITY** can all be computed efficiently.
- The full description of the game model has a practical size.

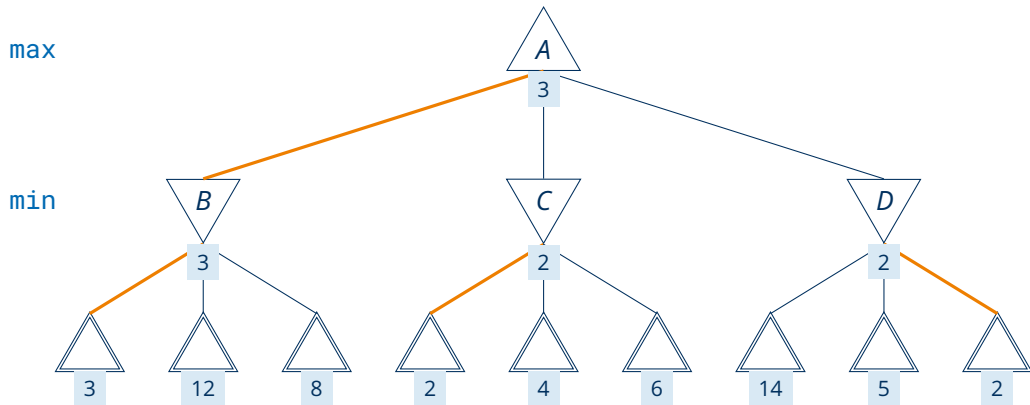
This assumption is especially relevant for games like Chess and Go, whose state-based models can be formalised (logically or through executable code), but whose game trees are too large to be explicitly represented.

Search in Game Trees

Recall: For combinatorial games, we used **backward induction** to solve them.

- For (general) zero-sum games, we also have to distinguish different utilities for the same player: Winning with 9 is better than winning with 1.
- This leads to a slightly more general algorithm: **minimax** search.
- Player **max** **maximises** their payoff u_{\max} (also called the **value** of the game).
- Player **min** maximises their payoff $u_{\min} = -u_{\max}$, thus **minimises** u_{\max} .
- Each player knows that the other player maximises/minimises and takes this into account accordingly.

Minimax Tree Search: Example



Minimax Value of a Game

Definition

For a (state-based model of a) game, the **minimax value** of a state $s \in S$ is

$$\text{minimax}(s) := \begin{cases} \text{UTILITY}(s) & \text{if IS-TERMINAL}(s), \\ \max_{m \in \text{MOVES}(s)} \text{minimax}(\text{RESULT}(s, m)) & \text{if TURN}(s) = \text{max}, \\ \min_{m \in \text{MOVES}(s)} \text{minimax}(\text{RESULT}(s, m)) & \text{if TURN}(s) = \text{min}. \end{cases}$$

The **minimax value of the game** is $\text{minimax}(S_0)$ for S_0 the initial state.

- The **minimax decision** at each node is the move leading to the maximal (resp. minimal) payoff in the next node.
- This definition of the optimal game value yields optimal responses of each player given that the respective other player also plays optimally.

Minimax Tree Search: Algorithm

```
function minimax-search(s: state) { // allows to start search in an arbitrary state s  
  if TURN(s) = max then { (v, m) := max-value(s) } else { (v, m) := min-value(s) }  
  return m } // return best move in s
```

```
function max-value(s: state) { // base case: terminal state  
  if IS-TERMINAL(s) then return (UTILITY(s), null) // initialise current maximum  
  (v*, m*) := (-∞, null) // try all moves  
  foreach m ∈ MOVES(s) do { // simulate move  
    (v', m') := min-value(RESULT(s, m))  
    if v' > v* then (v*, m*) := (v', m') // update current maximum  
  } // return maximum  
  return (v*, m*) }
```

```
function min-value(s: state) {  
  if IS-TERMINAL(s) then return (UTILITY(s), null)  
  (v*, m*) := (+∞, null)  
  foreach m ∈ MOVES(s) do {  
    (v', m') := max-value(RESULT(s, m))  
    if v' < v* then (v*, m*) := (v', m')  
  }  
  return (v*, m*) }
```

Minimax Tree Search: Complexity

Proposition

For a branching factor of b (maximal number of moves) and a depth of d (maximal length of histories), minimax search visits $O(b^d)$ terminal nodes.

↪ Minimax tree search is **impractical** for complex games.

Example

Chess has a branching factor of about 35 and average game length of about 80 ply (moves of a single player), so running minimax search to the leaves would need to expand $35^{80} \approx 10^{123}$ nodes.

There are at least two possible ways of reducing b^d :

- Reducing b : Do we really have to try out all possible moves?
↪ **alpha-beta pruning**
- Reducing d : Do we really have to play the game until the end?
↪ heuristic evaluation of states

Alpha-Beta Pruning

Alpha-Beta Tree Search: Algorithm

```
function alpha-beta-search( $s$ : state) { if TURN( $s$ ) = max then  
    ( $v$ ,  $m$ ) := max-value( $s$ ,  $-\infty$ ,  $+\infty$ ) else ( $v$ ,  $m$ ) := min-value( $s$ ,  $-\infty$ ,  $+\infty$ ); return  $m$  }
```

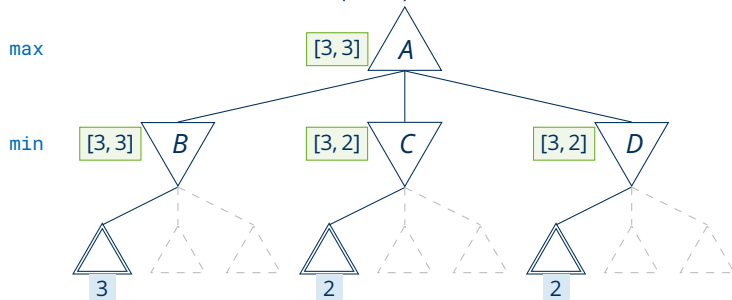
```
function max-value( $s$ : state,  $\alpha$ :  $\mathbb{R}_{\pm\infty}$ ,  $\beta$ :  $\mathbb{R}_{\pm\infty}$ ) {  
    if IS-TERMINAL( $s$ ) then return (UTILITY( $s$ ), null) // base case: terminal state  
    ( $v^*$ ,  $m^*$ ) := ( $-\infty$ , null) // initialise current maximum  
    foreach  $m \in$  MOVES( $s$ ) do { // try all moves  
        ( $v'$ ,  $m'$ ) := min-value(RESULT( $s$ ,  $m$ ),  $\alpha$ ,  $\beta$ ) // simulate move  
        if  $v' > v^*$  then { ( $v^*$ ,  $m^*$ ) := ( $v'$ ,  $m$ );  $\alpha$  :=  $\max(\alpha, v^*)$  } // update maximum and  $\alpha$   
        if  $v^* \geq \beta$  then return ( $v^*$ ,  $m^*$ ) } // prune irrelevant subtree  
    return ( $v^*$ ,  $m^*$ ) } // return maximum
```

```
function min-value( $s$ : state,  $\alpha$ :  $\mathbb{R}_{\pm\infty}$ ,  $\beta$ :  $\mathbb{R}_{\pm\infty}$ ) {  
    if IS-TERMINAL( $s$ ) then return (UTILITY( $s$ ), null)  
    ( $v^*$ ,  $m^*$ ) := ( $+\infty$ , null)  
    foreach  $m \in$  MOVES( $s$ ) do {  
        ( $v'$ ,  $m'$ ) := max-value(RESULT( $s$ ,  $m$ ),  $\alpha$ ,  $\beta$ )  
        if  $v' < v^*$  then { ( $v^*$ ,  $m^*$ ) := ( $v'$ ,  $m$ );  $\beta$  :=  $\min(\beta, v^*)$  }  
        if  $v^* \leq \alpha$  then return ( $v^*$ ,  $m^*$ ) }  
    return ( $v^*$ ,  $m^*$ ) }
```

Alpha-Beta Tree Search: Complexity

The order in which nodes are expanded matters!

- In the worst case, $O(b^d)$ terminal nodes will be visited, even with pruning.
- In the best case, only $O(b^{\frac{d}{2}}) = O(\sqrt{b^d})$ terminal nodes will be visited:



(Witnessing a winning strategy requires at least $b \cdot 1 \cdot \dots \cdot b \cdot 1 = b^{\frac{d}{2}}$ leaves.)

- However, finding a perfect move ordering amounts to solving the game.
- In practice, earlier evaluations (history) or expert knowledge can be used.

Heuristics

Heuristic Evaluation

Recall: There are at least two possible ways of reducing b^d :

- Reducing b : Do we really have to try out all possible moves?
↪ alpha-beta pruning
- Reducing d : Do we really have to play the game until the end?
↪ **heuristic evaluation of states**

Terminology

A **heuristic** aims at reducing the search space of a given problem, typically trading this off for at least one of optimality, completeness, or computation.

Main Idea: Treat non-terminal states as if they were terminal, estimate value.

- Replace function **IS-TERMINAL** : $S \rightarrow \{\top, \perp\}$ by **IS-CUTOFF** : $S \times \mathbb{N} \rightarrow \{\top, \perp\}$,
IS-CUTOFF(s, d) ... “cut off search below state s in search depth d ,”
- and function **UTILITY** : $S \rightarrow \mathbb{R}$ by **EVAL** : $S \rightarrow \mathbb{R}$,
EVAL(s) ... “estimate the prospective utility of state s (for player **max**).”

Restricting Depth: Heuristic Minimax Value

Heuristic Function **EVAL**: Technical Requirements

For all $s \in S$:

1. If **IS-TERMINAL**(s), then **EVAL**(s) = **UTILITY**(s), otherwise
2. $\min_{s \in S_T} \mathbf{UTILITY}(s) \leq \mathbf{EVAL}(s) \leq \max_{s \in S_T} \mathbf{UTILITY}(s)$
for $S_T := \{s \in S \mid \mathbf{IS-TERMINAL}(s)\}$.

- In practice, the heuristic function **EVAL** should be **computable efficiently**.
- **EVAL**(s) should strongly correlate with **max**'s "chances of winning" in s .

Definition

The **heuristic minimax value** of a state $s \in S$ (w.r.t. d , **IS-CUTOFF**, and **EVAL**) is

$$\mathbf{hmm}(s, d) := \begin{cases} \mathbf{EVAL}(s) & \text{if } \mathbf{IS-CUTOFF}(s, d), \\ \max_{m \in \mathbf{MOVES}(s)} \mathbf{hmm}(\mathbf{RESULT}(s, m), d + 1) & \text{if } \mathbf{TURN}(s) = \mathbf{max}, \\ \min_{m \in \mathbf{MOVES}(s)} \mathbf{hmm}(\mathbf{RESULT}(s, m), d + 1) & \text{if } \mathbf{TURN}(s) = \mathbf{min}. \end{cases}$$

Heuristic Evaluation Functions

- Typically require experience with or expert knowledge about the game.
- Often combine various features f_i of the state into one numerical value:

$$\text{EVAL}(s) = w_1 \cdot f_1(s) + \dots + w_m \cdot f_m(s)$$

- Possible features can be:
 - **Mobility**: Measure the number of things a player can do (e.g. number of moves, number of reachable states within the next n moves, ...).
 - **Goal proximity**: How “close” (similar) is the current state to a final state?
 - **Material**: Count number (or “strength”) of pieces (if applicable and variable).
- Further features may exploit game-specific properties, e.g. persistence of markings in Tic-Tac-Toe or Connect-Four.

Heuristic Evaluation Functions: Examples

Example: Chess

- Add up “material values” of the player’s remaining pieces:
paw $\hat{=}$ 1, knight/bishop $\hat{=}$ 3, rook $\hat{=}$ 5, queen $\hat{=}$ 9.
- Assess board control (centre is better than edges or corners).

Example: Tic-Tac-Toe, Goal proximity

- There are 9 possible first moves for X: 1 centre, 4 sides, 4 corners.
- We can e.g. estimate in how many winning final positions they occur:

centre:

X		
X		
X		

X	X	X

X		
	X	
		X

	X	X
X		

corner:

X		
X		
X		

X	X	X

X		
	X	
		X

side:

X		
X		
X		

X	X	X

Heuristic Alpha-Beta Tree Search: Algorithm

Algorithm:

In the pseudocode on Slide 17, replace the lines mentioning **IS-TERMINAL** by:

```
if IS-CUTOFF(s, d) then return (EVAL(s), null)
```

and keep track of the search depth d as for the heuristic minimax value.

When to cut off search?

- At a fixed depth d_{\max} .
- After a fixed time, using iterative deepening and keeping track of best moves (to also improve move ordering in subsequent iterations).

When not to cut off search?

- **Quiescence**: Apply heuristic evaluation only to quiescent positions, those not facing pending moves that would significantly affect the evaluation.
- **Horizon effect**: An ultimately unavoidable opponent move is pushed beyond the horizon by delay tactics and thus seemingly avoided.

Improvements of Alpha-Beta Tree Search

- **Move Ordering:**
 - Static: Use human (expert) knowledge about the game.
 - Dynamic: Use iterative deepening and the history heuristic (moves that were useful in previous search iterations will probably be useful in later ones).
- **Transposition Tables:**
 - The same game state can be reached by different histories.
 - Recognising game states that have been visited before avoids re-searching.
- **Variable Depth:**
 - Strong moves are worth searching more deeply, weak moves (e.g. those expanded later with good move ordering) less so.
- **Endgame Tables:**
 - Endgames can be completely solved (doing bottom-up search with reverse moves) whenever the number of positions can be handled in practice.
 - The resulting strategies can be put into lookup tables and consulted in search.

Conclusion

Minimax Tree Search can be extended to more than two (say n) players:

- The **UTILITY** function returns an n -tuple (v_1, \dots, v_n) of utilities.
- Every player i only maximises v_i when it is their turn to move.

Summary

- Game trees can be succinctly represented by **state-based game models**.
- **Minimax Tree Search** can be used to solve sequential (two-player zero-sum) games with perfect information.
- **Alpha-Beta Pruning** allows to reduce the search space without sacrificing solutions.
- **Heuristic Evaluation** of states can be used to reduce search depth.
- Further heuristics may reduce the search space (typically with sacrifices).