Foundations of AI

6. Adversarial Search

Search Strategies for Games, Games with Chance, State of the Art

Wolfram Burgard & Bernhard Nebel
Contents

• Game Theory
• Board Games
• Minimax Search
• Alpha-Beta Search
• Games with an Element of Chance
• State of the Art
Games & Game Theory

• When there is *more than one agent*, the future is not anymore easily predictable for the agent
• In *competitive* environments (when there are conflicting goals), *adversarial search* becomes necessary
• Mathematical *game theory* gives the theoretical framework (even for non-competitive environments)
• In AI, we usually consider only a special type of games, namely, board games, which can be characterized in game theory terminology as
  – *Extensive, deterministic, two-player, zero-sum* games with *perfect information*
Why Board Games?

- Board games are one of the oldest branches of AI (Shannon, Turing, Wiener, and Shanon 1950).
- Board games present a very abstract and pure form of competition between two opponents and clearly require a form on “intelligence”.
- The states of a game are easy to represent.
- The possible actions of the players are well defined.
- Realization of the game as a search problem
- The world states are fully accessible
- It is nonetheless a contingency problem, because the characteristics of the opponent are not known in advance.

Note: Nowadays, we also consider sport games
Problems

• Board games are not only difficult because they are **contingency problems**, but also because the search trees can become astronomically large.

• **Examples:**
  
  – **Chess**: On average 35 possible actions from every position, 100 possible moves \( \rightarrow 35^{100} \) nodes in the search tree (with “only” approx. \( 10^{40} \) legal chess positions).
  
  – **Go**: On average 200 possible actions with approx. 300 moves \( \rightarrow 200^{300} \) nodes.
What are Our Goals?

• **Good game programs** try to
  
  – look ahead as many moves as possible
  – delete *irrelevant branches* of the game tree
  – use good *evaluation functions* in order to estimate how good a position is
Terminology of Two-Person Board Games

- **Players** are MAX and MIN, where MAX begins.
- **Initial position**, e.g. board arrangement
- **Operators** are the legal moves
- **Termination test** determines when the game is over. Terminal state = game over.
- **Utility function** computes the value of a terminal state, e.g., -1, 0, or 1.
- **Strategy**. In contrast to regular searches, where a path from beginning to end is simply a solution, MAX must come up with a strategy to reach a terminal state regardless of what MIN does → correct reactions to all of MIN’s moves.
Every step of the search tree, also called game tree, is given the player’s name whose turn it is (MAX- and MIN-steps).

When it is possible, as it is here, to produce the full game tree, the \textit{minimax algorithm} computes an \textit{optimal strategy for MAX}. 
Minimax

1. Generate the complete game tree using depth-first search.
2. Apply the utility function to each terminal state.
3. Beginning with the terminal states, determine the utility of the predecessor nodes as follows:
   - Node is a MIN-node
     Value is the minimum of the successor nodes
   - Node is a MAX-node
     Value is the maximum of the successor nodes
   - From the initial state (root of the game tree), MAX chooses the move that leads to the highest value (minimax decision).

Note: Minimax assumes that MIN plays perfectly. Every weakness (i.e. every mistake MIN makes) can only improve the result for MAX.

Note: Human strategy may be different trying to exploit the weakness of the opponent.
Minimax Example

MAX

A₁

A₁₁ A₁₂ A₁₃
3 12 8

A₂

A₂₁ A₂₂ A₂₃
2 4 6

A₃

A₃₁ A₃₂ A₃₃
14 5 2

MIN
Minimax Algorithm

Recursively calculates the best move from the initial state.

function MINIMAX-DECISION(game) returns an operator

    for each op in OPERATORS[game] do
        VALUE[op] ← MINIMAX-VALUE(APPLY(op, game), game)
    end

    return the op with the highest VALUE[op]

function MINIMAX-VALUE(state, game) returns a utility value

    if TERMINAL-TEST(game)(state) then
        return UTILITY(game)(state)
    else if MAX is to move in state then
        return the highest MINIMAX-VALUE of SUCCESSORS(state)
    else
        return the lowest MINIMAX-VALUE of SUCCESSORS(state)
Evaluation Function

When the search space is too large, the game tree can be created to a certain depth only. The art is to correctly evaluate the playing position of the leaves, which are not terminal states.

Example of simple evaluation criteria in chess:

- Material worth: pawn=1, knight =3, rook=5, queen=9.
- Other: king safety, good pawn structure
- Rule of thumb: 3-point advantage = certain victory

The choice of evaluation function is decisive!

The value assigned to a state of play should reflect the chances of winning, i.e. the chance of winning with a 1-point advantage should be less than with a 3-point advantage.
Evaluation Function - General

The preferred evaluation functions are weighted, linear functions (easy to compute):

\[ w_1 f_1 + w_2 f_2 + \ldots + w_n f_n \]

where the \( w \)'s are the weights, and the \( f \)'s are the features. [e.g. \( w_1 = 3 \), \( f_1 = \) number of our own knights on the board]

**Assumption**: The criteria are independent.

The weights can be learned. The criteria, however, must be given (no one knows how they can be learned).
Cutting Off Search

- Fixed-depth search (so the goal limit is not overstepped)
- Better: iterative deepening search (with cut-off at time limit)
- ...but only evaluate quiescent positions that won’t cause large fluctuations in the evaluation function in the following moves.
- ... if bad situations can be pushed behind the horizon, try to search in order to find out
Two Similar Positions

- Very similar positions, but in (b) black will lose
- Search for quiescent position
Horizon Problem

- Black has a slight material advantage
- …but will eventually lose (pawn becomes a queen)
- A fixed-depth search (<14) will not detect this because it thinks it can avoid it (on the other side of the horizon - because black is concentrating on the check with the rook, to which white must react).
Pruning Branches

• Often, it becomes clear early on that a branch cannot lead to better results than the one we have explored already. 

➤ Prune away such branches that cannot improve our results!

• What are the conditions under which we are allowed to do that?
Pruning Irrelevant Branches
Pruning Branches: General Idea

If $m > n$ we will never reach node $n$ in the game. Once we have enough information (an upper bound) about the node $n$, we can prune.
Alpha-Beta Pruning: The Method

- $\alpha = \text{the value of the best (i.e., highest value) choice we have found so far at any choice point along the path for MAX}$
  - In the example: $m$
- $\beta = \text{the value of the best (i.e., lowest value) choice we have found so far at any choice point along the path for MIN}$
When Can We Prune?

The following applies:

\(\alpha\) values of MAX nodes can never decrease

\(\beta\) values of MIN nodes can never increase

(1) Prune below the MIN node whose \(\beta\)-bound is less than or equal to the \(\alpha\)-bound of its MAX-predecessor node.

(2) Prune below the MAX node whose \(\alpha\)-bound is greater than or equal to the \(\beta\)-bound of its MIN-predecessor node.

→ Delivers results that are just as good as with complete minimax searches to the same depth (because only irrelevant nodes are eliminated).
Alpha-Beta Search Algorithm

function $\text{MAX-VALUE}(state, game, \alpha, \beta)$ returns the minimax value of $state$

inputs: $state$, current state in game
$game$, game description
$\alpha$, the best score for MAX along the path to $state$
$\beta$, the best score for MIN along the path to $state$

if $\text{CUTOFF-TEST}(state)$ then return $\text{EVAL}(state)$
for each $s$ in $\text{SUCCESSORS}(state)$ do
    $\alpha \leftarrow \text{MAX}(\alpha, \text{MIN-VALUE}(s, game, \alpha, \beta))$
    if $\alpha \geq \beta$ then return $\beta$
end
return $\alpha$

function $\text{MIN-VALUE}(state, game, \alpha, \beta)$ returns the minimax value of $state$

if $\text{CUTOFF-TEST}(state)$ then return $\text{EVAL}(state)$
for each $s$ in $\text{SUCCESSORS}(state)$ do
    $\beta \leftarrow \text{MIN}(\beta, \text{MAX-VALUE}(s, game, \alpha, \beta))$
    if $\beta \leq \alpha$ then return $\alpha$
end
return $\beta$

Initial call with $\text{MAX-VALUE}(\text{initial-state}, game, -\infty, +\infty)$
Alpha-Beta Trace

\[ \alpha = -\infty, \quad \beta = +\infty \]

3 12 8 2 4 6 14 5 2
Efficiency Gain

• The alpha-beta search cuts the largest amount off the tree when we examine the best move first.
• In the best case (always the best move first), the search cost is reduced to $O(b^{d/2})$.
• In the average case (randomly distributed moves), the search cost is reduced to $O((b/\log b)^d)$
• For $b < 100$, we get $O(b^{3d/4})$.
• Practical case: A simple ordering heuristic brings the performance close to the best case.
• i.e. we can search twice as deep in the same amount of time
Transposition Tables

• As in search trees, also in game trees there is the problem of \textit{repeated states}
• In chess, e.g. the game tree may have $35^{100}$ nodes, but there are only $10^{40}$ different board positions
• Similar to \textit{closed list} in search, maintain a \textit{transposition table}
  
  \begin{itemize}
  \item \textbf{Got its name from the fact that the same state is reached by a transposition of moves.}
  \end{itemize}
Games that Include an Element of Chance

White has just rolled 6-5 and has 4 legal moves.
Game Tree for Games with an Element of Chance

In addition to MIN- and MAX nodes, we need chance nodes (for rolling the dice).
Calculation of the Expected Value

• *Expectiminimax* instead of *Minimax*:

\[
\text{Expectiminimax}(n) =
\begin{align*}
\text{Utility}(n) & \quad \text{if } n \text{ is a terminal state} \\
\max_{s \in \text{Successors}(n)} \text{Expectiminimax}(s) & \quad \text{if } n \text{ is a MAX node} \\
\min_{s \in \text{Successors}(n)} \text{Expectiminimax}(s) & \quad \text{if } n \text{ is a MIN node} \\
\sum_{s \in \text{Successors}(n)} P(s) \cdot \text{Expectiminimax}(s) & \quad \text{if } n \text{ is a chance node}
\end{align*}
\]
Problems

- **Order-preserving transformations on evaluation values change the best move:**

  ![Game tree diagram](image)

- **Search costs increase:** Instead of $O(b^d)$, we get $O(bnx)^d$, where $n$ is the number of possible dice outcomes.
  - In Backgammon ($n = 21$, $b = 20$ but can be 4000) the maximum $d$ is 3.
  - Variation of alpha-beta search can be used.
Card Games

• Recently card games such as bridge and poker have been addressed as well
• One approach: simulate play with open cards and then average over all possible plays (or make a Monte Carlo simulation)
  – Averaging over clairvoyancy
• Although “incorrect”, seems to give reasonable results
State of the Art (1)

**Checkers, draughts** (by international rules): A program called *CHINOOK* is the official world champion in man-computer competition (acknowledged by ACF and EDA) and the highest-rated

**Backgammon**: The *BKG* program defeated the official world champion in 1980. A newer program called *TD-Gammon* (that used a reinforcement learning to learn the evaluation function) is among the top 3 players.

**Othello**: Very good, even on normal computers. Programs are not allowed at tournaments. *Logistello* defeated the world champion in 1997 the human world champion.
State of the Art (2)

**Bridge:** The Bridge Baron program won the 1997 computer bridge championship. GIB (using the averaging over clairvoyancy) won in 2000. In general, they are not a match for humans, though

**Tic-Tac-Toe, Go-Moku (five in a row), Nine-Men’s Morris** are all solved by exhaustive analysis.

**Go:** The best programs play a little better than beginners (10 kyu) (branching factor on average 200).

→ There is a $2 Mi. US-$ prize for the first program to defeat a world master.
Chess (1)

- Chess as “Drosophila” of AI research.
- A limited number of rules produces an virtually unlimited number of courses of play. In a game of 40 moves, there are $1.5 \times 10^{128}$ possible courses of play.
- Victory comes through logic, intuition, creativity, and previous experience.
- In 1997, the world chess master G. Kasparow was beaten by Deep Blue in a match of 6 games.
- Recently, Kasparow played a draw against Deep Junior
Chess (2)

- **Deep Blue** (IBM Thomas J. Watson Research Center)
- **Special hardware** (32 processors with 8 chips, 2 Mi. calculations per second)
- Heuristic search
- Case-based reasoning and learning techniques
- 1996 Knowledge based on 600,000 chess games
- 1997 Knowledge based on 2 million chess games
- Training through grand masters
Kasparow: There were moments when I had the feeling that these boxes are possibly closer to intelligence than we are ready to admit.

From a certain point on it seems, in chess at least, that great quantity translates into quality.

I see rather a great chance for fine creativity and brute force computational capacity to complement each other in a new form of information acquisition. The human and electronic brain together would produce a new quality of intelligence – an intelligence worthy of this name.
The Reasons for Success…

• Alpha-Beta-Search
• … with dynamic decision/making for uncertain positions
• Good (but usually simple) evaluation functions
• Large databases of opening moves.
• Very large end-game databases (for checkers, all 8-piece situations)
• And very fast and parallel processors!
Summary

• A **game** can be defined by the **initial state**, the **operators** (legal moves), a **termination test** and a **utility function** (outcome of the game).
• In two-player games, the **minimax algorithm** can determine the best move by enumerating the entire game tree.
• The **alpha-beta algorithm** produces the same result but is more efficient because it prunes away irrelevant branches.
• Usually, it is not feasible to construct the complete game tree, so the utility of some states must be determined by an **evaluation function**.
• **Games of chance** can be handled by an extension of the alpha-beta algorithm.