Foundations of AI

5. Board Games

Search Strategies for Games, Games with Chance, State of the Art
Wolfram Burgard and Luc De Raedt

Why Board Games?
Board games are one of the oldest branches of AI (Shannon und Turing 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.

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" ca. \( 10^{40} \) legal chess positions).
- **Go**: On average 200 possible actions with ca. 300 moves \( \rightarrow 200^{300} \) nodes.

**Good game programs have the properties that they**
- delete irrelevant branches of the game tree,
- use good evaluation functions for in-between states, and
- look ahead as many moves as possible.
Terminology of Two-Person Board Games

- Players are MAX and MIN, where MAX begins.
- Initial position (e.g., board arrangement)
- Operators (= legal moves)
- Termination test, determines when the game is over. Terminal state = game over.
- 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.

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.
Minimax Algorithm

Recursively calculates the best move from the initial state.

```plaintext
function MINIMAX-DECISION(game) returns an operator
for each op in OPERATIONS(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 MIN is to move in state then
  return the highest MINIMAX-VALUE of SUCCESSOR(state)
else
  return the lowest MINIMAX-VALUE of SUCCESSOR(state)
```

Note: Minimax only works when the game tree is not too deep. Otherwise, the minimax value must be approximated.

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.

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:

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

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).

When Should we Prune the Tree?

- Fixed-depth search (so the goal limit is not overstepped)
- Better: iterative deepening search (with cut-off at the goal limit)
- ...but only evaluate "peaceful" positions that won't cause large fluctuations in the evaluation function in the following moves.

→ i.e., continue searching and follow a sequence of moves through to the end.
Horizon Problem

- Black has a slight material advantage
- ...but must eventually lose (pawn becomes a queen)
- A fixed-depth search cannot 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).

Alpha-Beta Pruning

We do not need to consider all nodes.

Alpha-Beta Pruning in Tic-Tac-Toe (1)

Recall: Minimax algorithm with depth-first search
α = best worth for MAX on a path
β = best worth for MIN on a path

Tic-Tac-Toe Example:
Possible evaluation $e(p)$ of a game state: Number of still attainable complete goals for MAX minus number of still attainable complete goals for MIN.

```
  x
  o
```

$e(p) = 6 - 4 = 2$

MAX can still win in two columns, two rows, and two diagonals (=6). MIN can only win in the top and bottom rows and in the two outer columns (=4).

Calculation of the Lower Bound

First move by MAX:

```
  x
  o
```

By exploiting the symmetry of the board, MIN can react:

```
  x  x  x  x  x  x
  o  o  o  o  o  o
```

For MAX nodes, there is a lower bound, since its worth in every case is only $\geq -1$, $\rightarrow \alpha = -1$.

α = the value of the best (i.e., highest-value) choice we have found so far at any choice point along the path for MAX.
Calculation of the Upper Bound

<table>
<thead>
<tr>
<th></th>
<th>MAX</th>
<th>MIN (-1)</th>
<th>MAX (-1)</th>
</tr>
</thead>
</table>

- For the MIN nodes, there is currently a best possible of -1. Since on the MIN step, the minimax algorithm always chooses the minimum of the successor nodes, every successive evaluation through the expansion of branches can only decrease.

- The result is an upper bound of $\beta = -1$

- $\beta$ = 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 apply:

$\alpha$ 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 Pruning: General

If $m > n$ we will never reach node $n$ in the game.

Alpha-Beta Search Algorithm

Initial call with MAX-VALUE(initial-state, $-\infty$, $+\infty$)
**Alpha-Beta Pruning Example**

![Alpha-Beta Pruning Example Diagram]

**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 expenditure is reduced to $O(b^{d/2})$.
- In the average case (randomly distributed moves), the search expenditure is reduced to $O((b/\log b)^d)$.
- For $b < 100$, we attain $O(b^{3d/4})$.
- **Practical case:** A simple ordering heuristic brings the performance close to the best case.
- We can search twice as deep in the same amount of time.

$\Rightarrow$ In chess, we can thus reach a depth of 6-7 moves.

**Games that Include an Element of Chance**

![Games that Include an Element of Chance Diagram]

White has just rolled 6-5 and has 4 legal moves.

**Game Tree for Backgammon**

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

Utility function for chance nodes $C$ over $\text{MAX}$:

- $d_i$: possible dice rolls
- $P(d_i)$: probability of obtaining that roll
- $S(C,d_i)$: attainable positions from $C$ with roll $d_i$
- $\text{utility}(s)$: Evaluation of $s$

$$\text{expectimax}(C) = \sum_i P(d_i) \max_{s \in S(C,d_i)} \text{utility}(s)$$

$\text{expectimin}$ likewise

---

State of the Art

**Checkers, draughts (by international rules):** A program called CHINOOK is the official world champion in man-computer competition (acknowledges by ACF and EDA) and the highest-rated player:
- CHINOOK: 2712
- R on King: 2632
- A sa long: 2631
- D on Lafferty: 2625

**Backgammon:** The BKG program defeated the official world champion in 1980. A newer program is among the top 3 players.

**Othello:** Very good, even on normal computers. Programs are not allowed at tournaments.

**Go:** The best programs play a little better than beginners (branching factor > 300). There is a $2 \text{ Mi. US }$ prize for the first program to defeat a world master.

---

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 $O(bxn)^d$, where $n$ is the number of possible dice outcomes.

  \[ \rightarrow \text{In Backgammon } (n=21, b=20, \text{can be } 4000) \text{ the maximum } d \text{ can be is } 2. \]

---

Chess (1)

Chess as “Drosophila” of AI research.

- A limited number of rules produces an unlimited number of courses of play. In a game of 40 moves, there are $1.5 \times 10^{126}$ possible courses of play.
- Victory comes through logic, intuition, creativity, and previous knowledge.
- Only special chess intelligence, no “general knowledge”

  **Playing Strength**
  - G. Kasparow: 2828
  - V. Anand: 2758
  - A. Karpow: 2710
  - Deep Blue: 2680

At the moment, there are only ca. 64 super-master players with a rating of over 2600 ELO points.
**Chess (2)**

In 1997, world chess master G. Kasparow was beaten by a computer in a match of 6 games.

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
- Duel between the “machine-like human Kasparow vs. the human machine Deep Blue.”

**Chess (3)**

<table>
<thead>
<tr>
<th>Game</th>
<th>1996</th>
<th>1997</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>DB won</td>
<td>DB won</td>
</tr>
<tr>
<td>2</td>
<td>Draw</td>
<td>DB won (K. gave up)</td>
</tr>
<tr>
<td>3</td>
<td>Draw</td>
<td>Draw</td>
</tr>
<tr>
<td>4</td>
<td>DB lost</td>
<td>Draw</td>
</tr>
<tr>
<td>5</td>
<td>DB lost</td>
<td>Draw</td>
</tr>
<tr>
<td>6</td>
<td>DB lost</td>
<td>DB won (K. gave up)</td>
</tr>
</tbody>
</table>

**Chess (4)**

- The branching factor in chess is ca. 35.
- In tournament chess, each move is limited to 150 seconds.
- “Normal” chess computers expand/evaluate ca. 1000 positions per second.
  - $\rightarrow$ 150 000 searchable positions: 3 half-moves.
- Deep Blue 1996: Horizon of 3 moves / 15 half-moves.
- Deep Blue 1997: Horizon of 7 moves (50-100 billion positions in 3 minutes).

**Chess (5)**

![Graph showing the development of chess-playing computers from 1960 to 2000.](image)
Chess (6)

Kasparow: There were moments when I had the feeling that these boxes are possibly closer to intelligence that 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 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 game termination 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 terminal 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.