Artificial Intelligence - Methods and Applications

Fall 2011

Lecture 2: Adverserial Search

Henrik Björklund
Computing Science
Umeå University

Games

- There are many kinds of games
  - What makes backgammon different from chess?
  - What makes Stone, Paper, Scissors different from backgammon?
  - What makes poker different from backgammon?
- Besides games that are played for fun, there are "theoretical games"
  - Philosophy
  - Economics
  - Biology
  - Computer Science (Satisfiability, Model Checking, etc.)
- Why interpret model checking as a game?

"Chess-Like" Games

- Two players
- Turn-based
- Perfect information
- Deterministic
- Zero sum
- Finite

Games can be hard

- Even "ideal" games like chess can be very hard to solve.
  - Chess has an average branching factor of about 35
  - A game can last for more than 100 ply
  - This gives rise to a game tree with 35^100 leaves
- Checkers has been completely solved (2007)
  - Alpha-Beta search
  - Endgame databases
- There are chess engines that play at (or even above) elite level.
  - This does *not* mean that chess has been completely solved.
- The best current go engines play at an intermediate level.

Game trees

- A deterministic turn-based finite game with perfect information can be described by a game tree.
- The root is the initial position.
- The children of the root are the possible positions after one ply.
- The children of the children of the root are the possible positions after two ply.
- Et cetera.
- The leaves represent possible endings. They contain information on the value (utility) of the ending to each of the players.
- In a two-player zero-sum game, what one player gains, the other one loses.

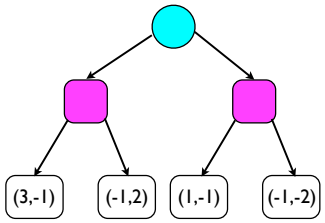
Game of matches

- Rules
  - Initially, there are ***n*** matches.
  - The players take turns picking 1, 2, or 3 matches.
  - The one that takes the last match loses.
- The branching factor is 3.
- For small ***n***, the tree has a reasonable size, but it grows exponentially with ***n***.
- Is there a cleverer way to compute a strategy?

Tic-Tac-Toe

- In tic-tac-toe, the initial branching factor is 9.
- It then decreases by one per ply.
- In total, the game tree has less than (why less?) 9! nodes, with 9! = 362.880.
- This size is trivial to handle on a modern computer.
- What do we want to compute?
  - The outcome, when both players play optimally.
  - A perfect strategy for the winner.

Think about the opponent!



In a game tree, we cannot just search for the best outcome for the first player.

The MiniMax Algorithm

The MiniMax algorithm searches a game tree for the best outcome the first player can achieve, ***assuming that the other player does what is best for her.***

In a zero-sum game, we can look only at the utility for the first player. The first player then tries to ***maximize*** the outcome, while the second player tries to ***minimize*** it.

Thus, we search the game tree for the

- **maximum** over the possible first moves of the
- **minimum** over the possible second moves of the
- **maximum** over the possible third moves of the
- **...**

MiniMax

- **MiniMax**(search tree node ***s***)
  - **if**(***s*** is a leaf)
    - **return** utility(***s***)
  - **if**(***s*** is a Max node)
    - **return** max(MiniMax(***t***)) : ***t*** is a child of ***s***
  - **if**(***s*** is a Min node)
    - **return** min(MiniMax(***t***)) : ***t*** is a child of ***s***

Alpha-Beta

- **Alpha-Beta**(search tree node *s*)
  - **return MaxValue**(*s*, -infty, +infty)

- **MaxValue**(search tree node *s*, *alpha*, *beta*)
  - **if**(*s* is a leaf)
    - **return** utility(*s*)
  - *value* = -infty
  - **for each** child *s'* of *s*
    - *value* = **Max**(value, **MinValue**(*s'*,*alpha*,*beta*))
    - **if**(*value* >= *beta*) **return** *value*
    - *alpha* = **Max**(*alpha*, *value*)
  - **return** *value*

MinValue

- **MinValue**(search tree node *s*, *alpha*, *beta*)
  - **if**(*s* is a leaf)
    - **return** utility(*s*)
  - *value* = +infty
  - **for each** child *s'* of *s*
    - *value* = **Min**(value, **MaxValue**(*s'*,*alpha*,*beta*))
    - **if**(*value* <= alpha) **return** *value*
    - *beta* = **Min**(*beta*, *value*)
  - **return** *value*

Alpha-Beta vs. Minimax

- MiniMax must, in the worst case, visit all tree nodes. With branching factor ***b*** and depth ***d*** this amounts to ***b^d***.
- How many nodes Alpha-Beta visits depends on the order in which moves are examined.
- If the best move is always examined first, Alpha-Beta visits ***b^(d/2)*** nodes. (This is unrealistic though.)
- We can use domain-specific knowledge to achieve reasonable move orderings.
- For chess ***b^(3d/4)*** is achievable.

Chess Engines

- Alpha-Beta search
- Move ordering
  - Killer moves
  - Iterative deepening
- Transposition tables
- Advanced (but easy to compute) heuristic functions
- Quiescence search
- Opening databases
- Endgame databases
- Efficient representation of position (move generation is a bottleneck)