DM842

Computer Game Programming II: AI

Lecture 10
Board Games

Christian Kudahl

Department of Mathematics & Computer Science
University of Southern Denmark

Outline

1. Board Games
    - MiniMaxing
    - Alpha-beta pruning
    - Monte Carlo Tree Search

Outline

1. Board Games
    - MiniMaxing
    - Alpha-beta pruning
    - Monte Carlo Tree Search

Combinatorial Game Theory

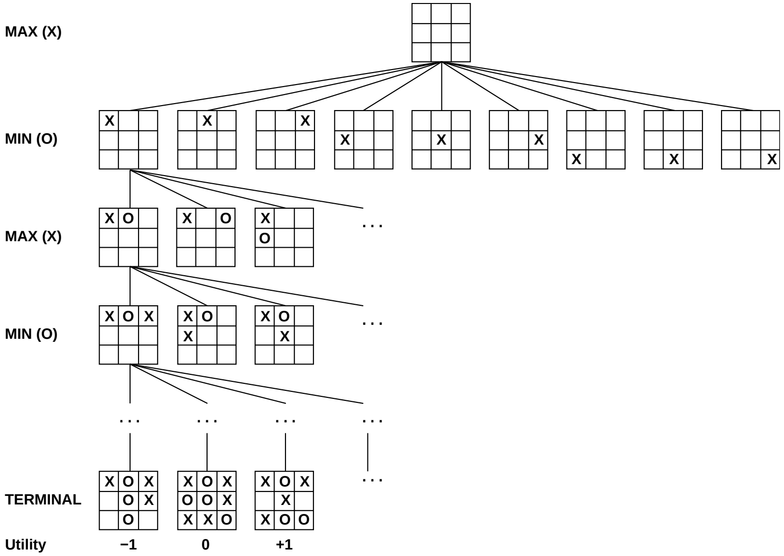
- Combinatorial game theory studies deterministic, sequential two-players games with perfect information
- Is there some move I can make, such that for all moves my opponent might make, there will then be some move I can make to win?
- Perfect play is the behavior or strategy of a player that leads to outcomes at least as good as any other strategy for that player regardless of the response by the opponent, hence even if opponent is infallible. (aka Optimal strategy)
- A solved game is a game whose outcome (win, lose, or draw) can be correctly predicted from any position, given that both players play perfectly.
- See `http://en.wikipedia.org/wiki/Solved_game` for a list of solved games.

Games vs. search problems

Search problem in a game tree (search tree overlapped on the game tree)

- initial state: root of game tree
- successor function: game rules/moves
- terminal test (is the game over?)
- utility function, gives a value for terminal nodes (eg, +1, -1, 0)

Game tree (2-player, deterministic, turns)



Solved Games

A two-player game can be "solved" on several levels:

- Ultra weak: Prove whether the first player will win, lose, or draw from the initial position, given perfect play on both sides.
  It can be non-constructive.

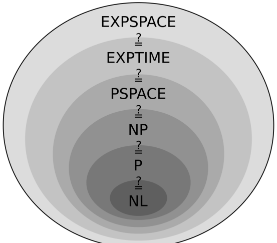
- Weak: Provide an algorithm that secures a win for one player, or a draw for either, against any possible moves by the opponent, from the beginning of the game.

- Strong: Exhaustively search a game tree to figure out what would happen if perfect play were realized. Provides optimal strategy from any position.

A minimax algorithm that exhaustively traverse the game tree would provide a strong proof.

Measures of game complexity

- State-space complexity: number of legal positions reachable from the initial state. Most often, an upper bound that includes illegal positions.
- Game tree size: total number of possible games that can be played, ie, number of leaves of the game tree.
- Computational complexity of the generalized game (eg, played on a $n \times n$ board): often PSPACE-complete (set of all decision problems that can be solved by a Turing machine using a polynomial amount of space and for which every other problem that can be solved in polynomial space can be transformed to one of these problems in polynomial time.)



Eg: Quantified Boolean formulas

$$\exists x_1, \forall x_2 \exists x_3 : (x_1 \vee x_2) \wedge (x_2 \vee x_3)$$

Outline

1. Board Games
    MiniMaxing
    Alpha-beta pruning
    Monte Carlo Tree Search

MiniMaxing

Starting from the bottom of the tree, scores are bubbled up according to the minimax rule:

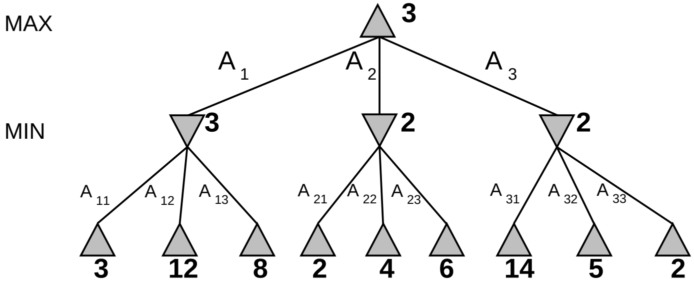
- on our moves, we are trying to maximize our score
- on opponent moves, the opponent is trying to minimize our score

(Perfect play for deterministic, perfect-information games)

Important Choice: Search whole tree (solves the game but only feasible for very simple games) or use a max search depth + heuristic.

Example

2-ply game:



Minimax algorithm

Recursive Depth First Search:

```
function MINIMAX-DECISION(state) returns an action
    return arg max_{a ∈ ACTIONS(s)} MIN-VALUE(RESULT(state, a))
```

```
function MAX-VALUE(state) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    $v \leftarrow -\infty$
    for each a in ACTIONS(state) do
        $v \leftarrow$ MAX(v, MIN-VALUE(RESULT(s, a)))
    return v
```

```
function MIN-VALUE(state) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    $v \leftarrow \infty$
    for each a in ACTIONS(state) do
        $v \leftarrow$ MIN(v, MAX-VALUE(RESULT(s, a)))
    return v
```

Properties of minimax

Time complexity: $O(b^m)$
Space complexity: $O(m)$ (depth-first exploration)

b branching factor
m search depth
But do we need to explore every path?

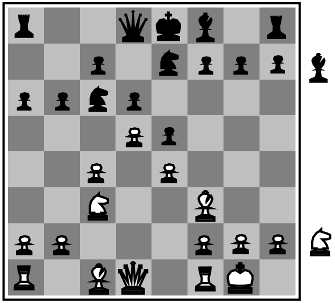
MiniMaxing with heuristic

Instead of searching to terminal position, search a fixed depth and use a heuristic (we are no longer sure that we play optimally).

static evaluation function: heuristic to score a state of the game for one player

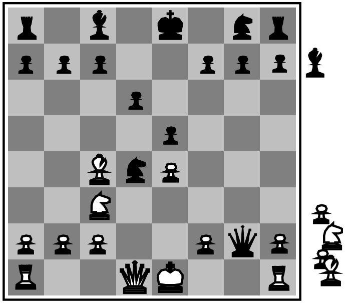
- it reflects how likely a player is to win the game from that board position
- knowledge of how to play the game (ie, strategic positions) enters here.
- the domain is the natural numbers ($-100; +100$)
- Eg. in Chess: ± 1000 for a win or loss, 10 for the value of a pawn
- there may be several scoring functions which are then combined in a single value (eg, by weighted sum, weigths can depend on the state of the game)

Evaluation functions



Black to move

White slightly better



White to move

Black winning

For chess, typically weighted sum of features

$$Eval(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$$

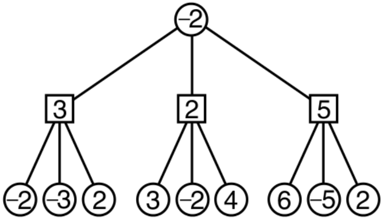
e.g., $w_1 = 9$ with
$f_1(s)$ = (number of white queens) – (number of black queens), etc.

Negamaxing

For two player and zero sum games:
If one player scores a board at -1, then the opponent should score it at $+1$

$\rightsquigarrow$ simplify the minimax algorithm.

- adopt the perspective of the player that has to move
- at each stage of bubbling up, all the scores from the previous level have their signs changed
- largest of these values is chosen at each time

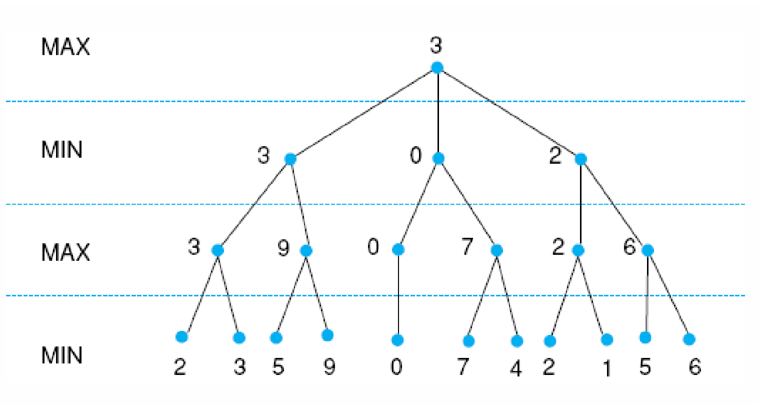


Simpler implementation but same complexity

Outline

1. Board Games
    - MiniMaxing
    - Alpha-beta pruning
    - Monte Carlo Tree Search

Example



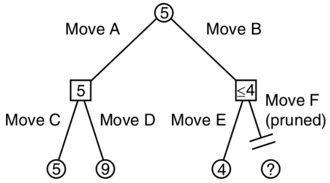
Alpha-beta pruning

Ignore sections of the tree that cannot contain the best move. Maintain two values:

- Alpha - Smallest value that the max player is assured of.
- Beta - Largest value that the min player is assured of.

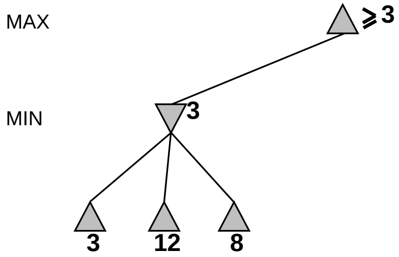
When investigating its children, what if the min player, when investigating node *v* finds a node with value smaller than alpha?

Good News? No. Since the parent of *v* is a max node, it will simply not allow the play to go to *v*. This means that we do not need to consider the subtre rooted in *v* anymore, since play will never to there.

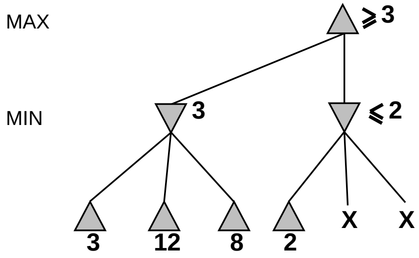


Similar reasoning holds for when the max player finds a node with value greater than beta.

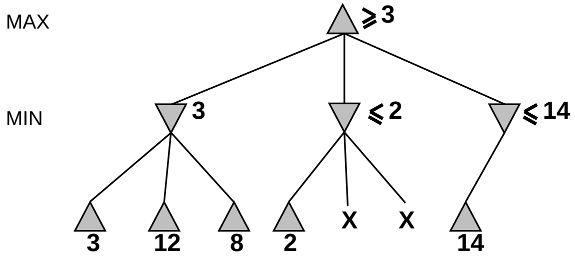
Example



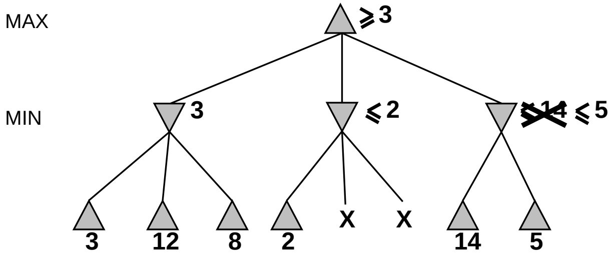
Example



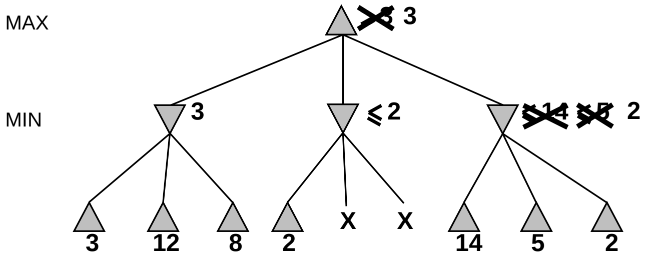
Example



Example



Example



The α–β algorithm

```
function ALPHA-BETA-SEARCH(state) returns an action
  $v \leftarrow$ MAX-VALUE(state, $-\infty, +\infty$)
  return the action in ACTIONS(state) with value $v$

function MAX-VALUE(state, $\alpha, \beta$) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  $v \leftarrow -\infty$
  for each $a$ in ACTIONS(state) do
    $v \leftarrow$ MAX($v$, MIN-VALUE(RESULT($s$,$a$), $\alpha, \beta$))
    if $v \geq \beta$ then return $v$
    $\alpha \leftarrow$ MAX($\alpha$, $v$)
  return $v$

function MIN-VALUE(state, $\alpha, \beta$) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  $v \leftarrow +\infty$
  for each $a$ in ACTIONS(state) do
    $v \leftarrow$ MIN($v$, MAX-VALUE(RESULT($s$,$a$) , $\alpha, \beta$))
    if $v \leq \alpha$ then return $v$
    $\beta \leftarrow$ MIN($\beta$, $v$)
  return $v$
```

Properties of α–β

- (α, β) search window: we (max player) will never choose to make moves that score less than alpha, and our opponent will never let us make moves scoring more than beta.
- Pruning *does not* affect final result
- Good move ordering improves effectiveness of pruning (shrinks window) consider first most promising moves: Use heuristics
- With "perfect ordering," time complexity = $O(b^{m/2})$
- Can be used in Negamax (in each step, swap alpha and beta and invert their signs)

Deterministic games in practice

- Checkers: Chinook ended 40-year-reign of human world champion Marion Tinsley in 1994. Weakly solved in 2007, the game is a draw with perfect play.
- Kalaha (6,6) solved at IMADA in 2011
- Chess: Deep Blue defeated human world champion Gary Kasparov in a six-game match in 1997. Deep Blue searches 200 million positions per second, uses very sophisticated evaluation, and undisclosed methods for extending some lines of search up to 40 ply.
- Othello: human champions refuse to compete against computers, who are too good.
- Go (2014): human champions refuse to compete against computers, who are too bad. In go, $b > 300$, so most programs use pattern knowledge bases to suggest plausible moves.

Deterministic games in practice

- Go (2014): human champions refuse to compete against computers, who are too bad. In go, $b > 300$, so most programs use pattern knowledge bases to suggest plausible moves. Not true anymore! In May 2017 AlphaGo beat the no. 1 human player Ke Jie.
  AlphaGo uses a variety of techniques including Monte Carlo Tree Search and Neural Networks.

Outline

1. Board Games
    - MiniMaxing
    - Alpha-beta pruning
    - Monte Carlo Tree Search

Monte Carlo Tree Search

Good heuristics can be very hard to come up with. Especially if not much is known about the game.

Idea: From a given position, use random playouts to evaluate it. If a player wins 73/100 playouts from a position, then that position is likely to be good for the player.

In reality, we do something slightly more sophisticated. We would like to bias our random search towards the most promising parts of the tree.

Tool: Multi-Armed Bandit Problem

100 different Multi-Armed Bandits. Which is best?

Stategy UCB1: Pick best machine according to upper bound of confidence interval given by

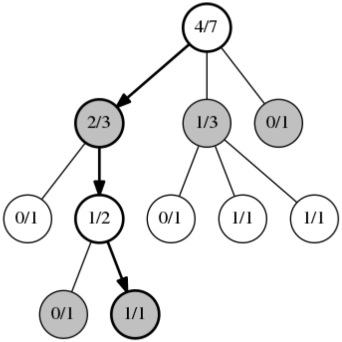
$$\bar{x}_i \pm \sqrt{\frac{2 \ln n}{n_i}}$$

$\bar{x}_i$ is mean payout on machine i
n_i is number of plays on machine i
n is total number of plays

(balances playing all of the machines to gather information with concentrating your plays on the observed best machine)

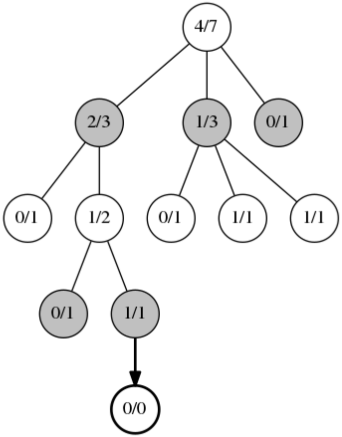
1. Selection

Game tree with recorded number of wins from each node.



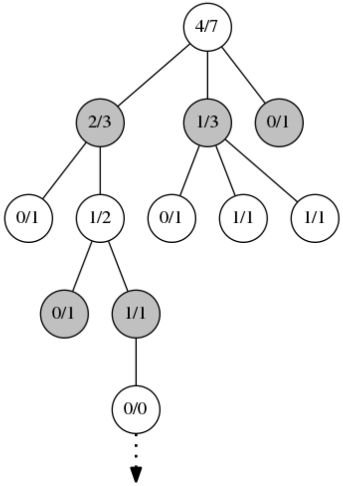
Start from a given position. Use UCB1 to select a child until you reach a node which has a child with a 0/0 record.

2. Expansion



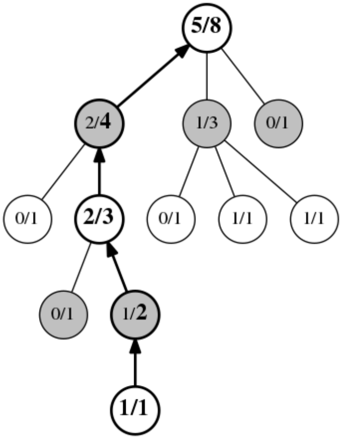
Chose a random 0/0 child of the previously selected node.

3. Simulation



Simulate a random game from the selected node.

4. Back-Propagation



Update the winner in all nodes on the path.

Moving

After running step 1-4 for the allowed amount of time, we pick the best move according UCB1.
We can keep using the tree we have already built (both in this play and future plays).

Summary: AI4GP

1. Movement
2. Pathfinding
3. Decision making
4. Tactical and strategic AI
5. Board game AI