Games

Introduction to Artificial Intelligence

G. Lakemeyer

Winter Term 2018/19

© G. Lakemeyer

Games

Games like chess have been studied in AI since the 50-ies.

They are a special variant of search problems.

The states are usually accessible.

The actions are the possible moves of a player.

Complication: More than 1 actor.

From one player's point of view, his or her actions have uncertain outcomes, since the reaction of the opponent is usually not foreseeable.

In this sense games belong to the class of contingency problems (uncertain knowledge of action effects).

Note: The opponent usually chooses actions to do maximal damage.

© G. Lakemeyer

What Makes Games Interesting

Problem: Almost all interesting games are unsolvable in practice.

Chess:

Each player has about 50 moves with about 35 actions per move,
i.e. there are about 35^{100} nodes in the search tree
(with "only" 10^{40} legal chess positions).

Good game programs have the following features:

a) Early pruning of useless sub-trees.
b) good evaluation function of states without doing a complete search.

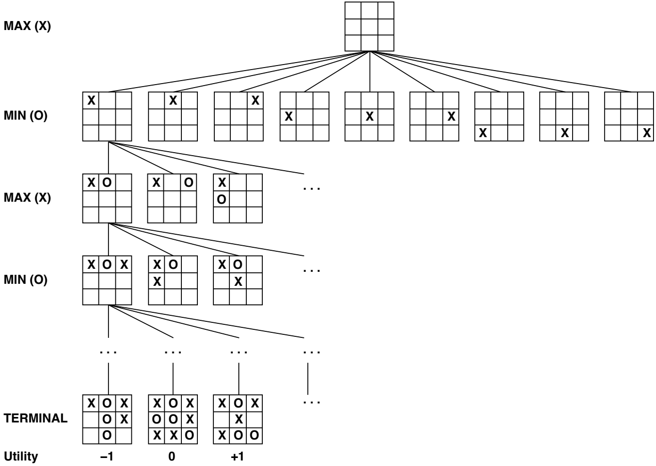
© G. Lakemeyer

2-Person Games – Some Terminology

Players:	MAX and MIN with MAX doing the first move.
Initial state:	e.g. initial board position, assignment of MAX and MIN.
Operators:	legal moves.
Terminal test:	when a game ends.
Terminal state	game over.
Utility function (payoff):	gives a numeric value to the outcome of a game; often simply $+1$ (win), -1 (loss), 0 (draw). Backgammon has a range between $+192$ and -192.
Strategy:	in contrast to regular search, MAX needs to find a path which leads to a winning terminal state for *every* possible reaction of MIN.

© G. Lakemeyer

An Example: Tic-Tac-Toe



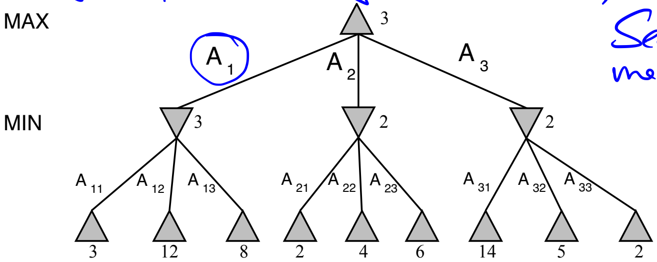
Each level of the search tree (also called game tree) is labelled with the player whose turn it is (MAX and MIN levels). When the complete game tree can be generated, the optimal strategy for MAX can be computed.

© G. Lakemeyer

Minimax

John McCarthy (50-ies)

Search method DFS



1. Generate a complete game tree.
2. Apply utility function to each terminal state.
3. Starting from the terminal states, calculate the values for the parent node as follows:
   - If the parent node is at a MIN level, then assign it the minimum of the values of the children.
   - If the parent node is at a MAX level, then assign it the maximum of the values of the children.
   - Then at the root MAX chooses the action which leads to the child with maximum utility (Minimax decision).

Note: Minimax assumes that MIN plays perfectly rational, i.e. always chooses the optimal move.

© G. Lakemeyer

Minimax-Algorithm (DFS)

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 Cut-off-test ~~TERMINAL-TEST~~[*game*](*state*) **then**
return Eval ~~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*)

© G. Lakemeyer

Evaluation Functions

When the search space is large, the game tree can only be generated up to a certain depth.

Minimax works then as well. Simply replace TERMINAL-TEST by CUT-OFF-TEST and the utility function UTILITY by the evaluation function EVAL.

The trick is then to correctly evaluate the goodness of the leaves.

Simple criteria for chess:

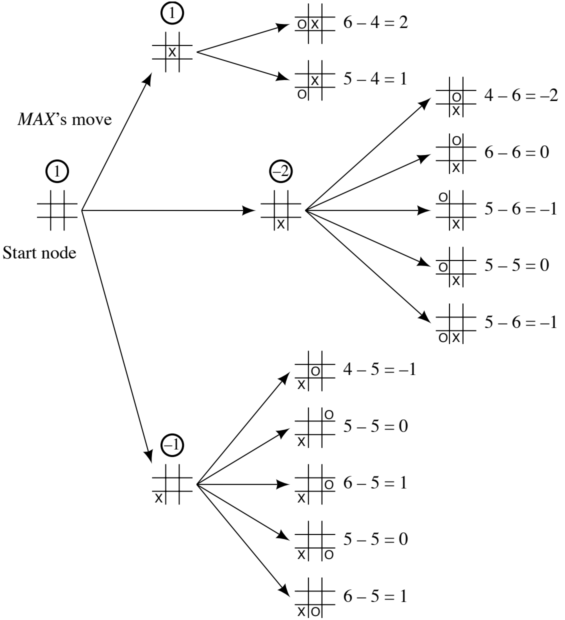
- Material value: pawn = 1, knight = 3, bishop = 3, rook = 5, queen = 9.
- Other features: safety of the king, good pawn structure.
- Rule of thumb: 3-point advantage = certain victory.

© G. Lakemeyer

Example: Tic-Tac-Toe (1)

(symmetric states not shown)

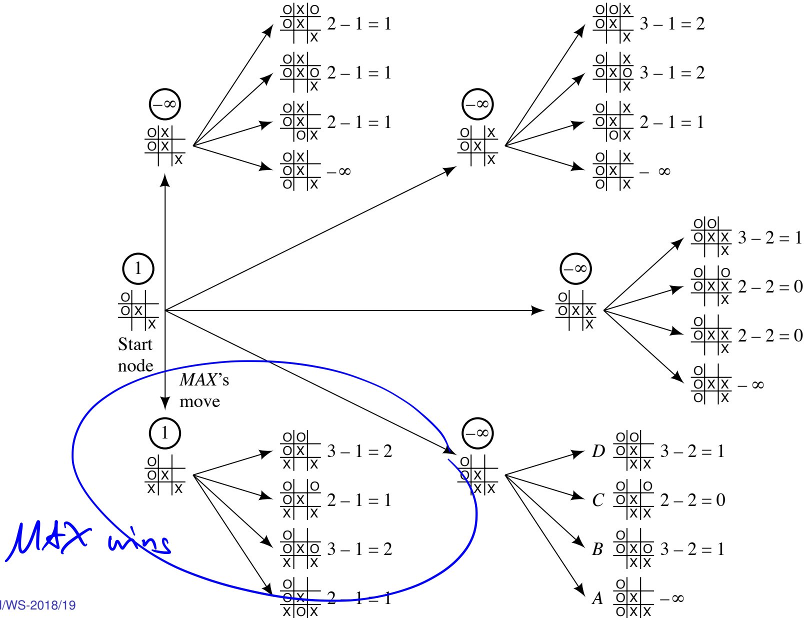
First stage, search depth 2



© G. Lakemeyer

Example: Tic-Tac-Toe (2)

Final stage, search depth 2



© G. Lakemeyer

Form of Evaluation Functions

a good eval. fct should reflect the probability of winning in a state randomly chosen from all states with the same value

The choice of an evaluation function is critical.

It should be easy to compute and accurately reflect the chance of winning.

Chance of winning for a given material value means the probability to win averaged over all positions with the same material value.

Usually evaluation functions are weighted linear functions:

$$w_1 f_1 + w_2 f_2 + \ldots + w_n f_n$$

(E.g. MAX = White: $w_1 = 3$, f_1 = number of white knights on the board.)

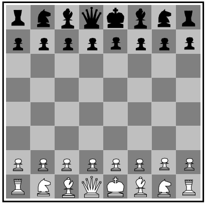
Assumption: The criteria are independent of each other. (Simplification)

© G. Lakemeyer

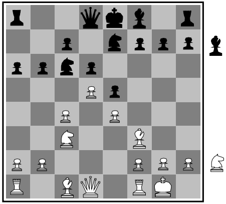
When to Cut Off Search?

Cut-off only in quiescent states, i.e. those which do not lead to dramatic subsequent changes, since the evaluation function is otherwise misleading.

Example:

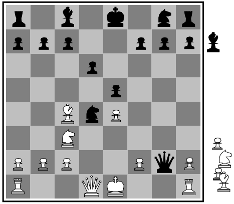


(a) White to move
Fairly even

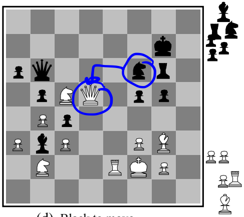


(b) Black to move
White slightly better

material — advant. for black is +5



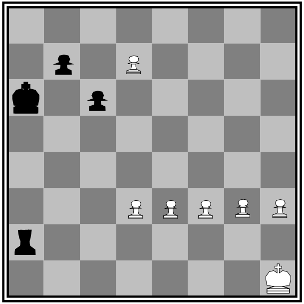
(c) White to move
Black winning



(d) Black to move
White about to lose

© G. Lakemeyer

Horizon Problem

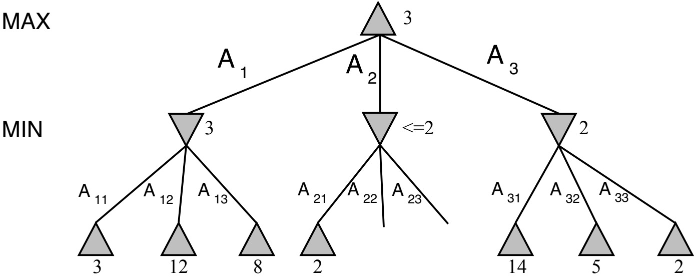


Black to move

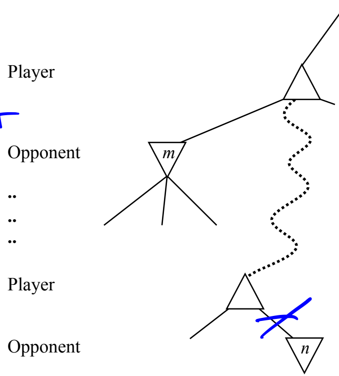
- Black has a slight advantage in material value;
- black loses the game eventually (pawn turns into a queen eventually);
- fixed depth search may not detect this since the “disaster” can be pushed over the horizon.

© G. Lakemeyer

Alpha Beta



Motto:
if you know that the situation is bad, then don't try to find out how bad it really is.



if $n \leq m$

© G. Lakemeyer

Alpha-Beta Search Algorithm

```
function MAX-VALUE(state, game, α, β) returns the minimax value of state
   inputs: state, current state in game
           game, game description
           α, the best score for MAX along the path to state
           β, the best score for MIN along the path to state

   if CUTOFF-TEST(state) then return EVAL(state)
   for each s in SUCCESSORS(state) do
       α ← MAX(α, MIN-VALUE(s, game, α, β))
       if α ≥ β then return β
   end
   return α
```

```
function MIN-VALUE(state, game, α, β) returns the minimax value of state

   if CUTOFF-TEST(state) then return EVAL(state)
   for each s in SUCCESSORS(state) do
       β ← MIN(β, MAX-VALUE(s, game, α, β))
       if β ≤ α then return α
   end
   return β
```

Initialization: $\alpha = -\infty$, $\beta = +\infty$.
Max-Value is applied to nodes at MAX levels, *Min-Value* at MIN levels.

© G. Lakemeyer

Alpha-Beta Search Algorithm Version 2

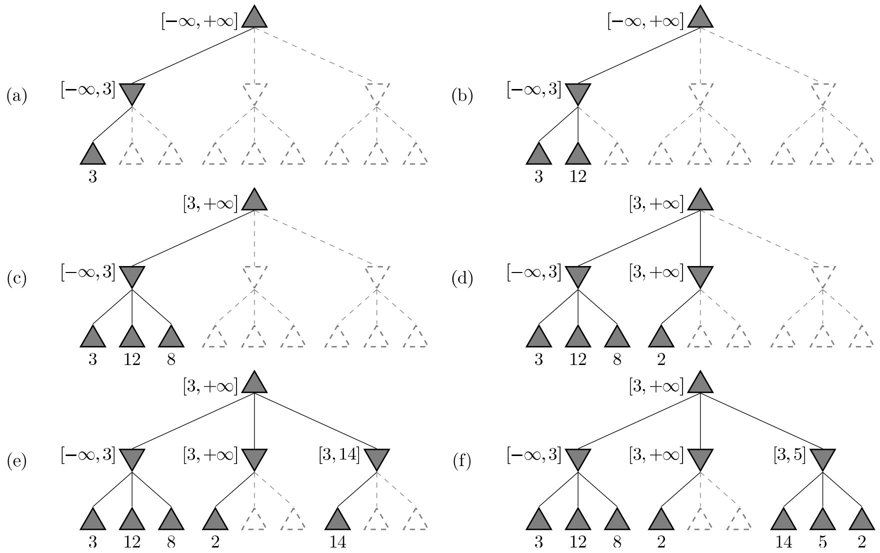
```
function ALPHA-BETA-SEARCH(state) returns an action
  inputs: state, current state in game

  v ← MAX-VALUE(state, −∞, +∞)
  return the action in SUCCESSORS(state) with value v
```

```
function MAX-VALUE(state, α, β) returns a utility value
  inputs: state, current state in game
          α, the value of the best alternative for MAX along the path to state
          β, the value of the best alternative for MIN along the path to state

  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← −∞
  for a, s in SUCCESSORS(state) do
    v ← MAX(v, MIN-VALUE(s, α, β))
    if v ≥ β then return v
    α ← MAX(α, v)
  return v
```

```
function MIN-VALUE(state, α, β) returns a utility value
  inputs: state, current state in game
          α, the value of the best alternative for MAX along the path to state
          β, the value of the best alternative for MIN along the path to state

  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← +∞
  for a, s in SUCCESSORS(state) do
    v ← MIN(v, MAX-VALUE(s, α, β))
    if v ≤ α then return v
    β ← MIN(β, v)
  return v
```

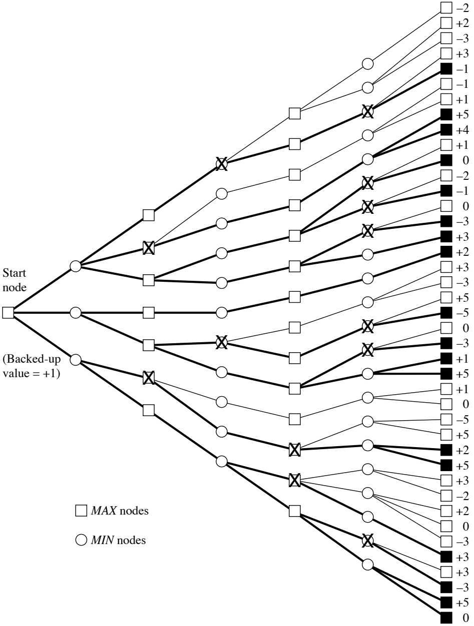
© G. Lakemeyer

Alpha-Beta Example



© G. Lakemeyer

Alpha-Beta Example



© G. Lakemeyer

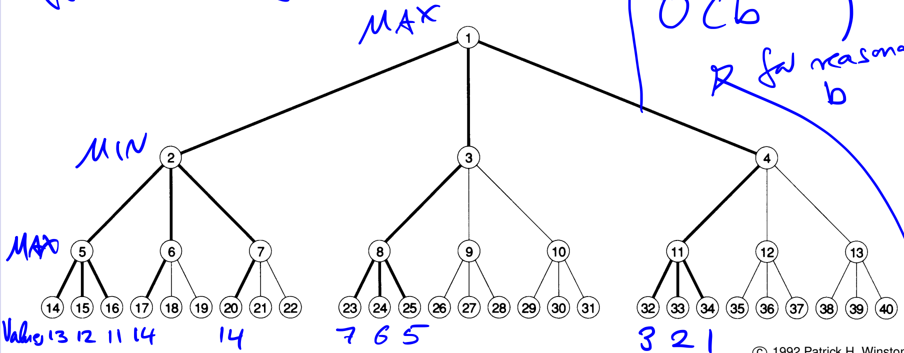
© 1998 Morgan Kaufman Publishers

Alpha-Beta: the Ideal Case

number of nodes expanded $O(b^{d/2})$

eff. branching factor $b^* = \sqrt{b}$

$O(b^{3/4\,d})$ for reasonable b



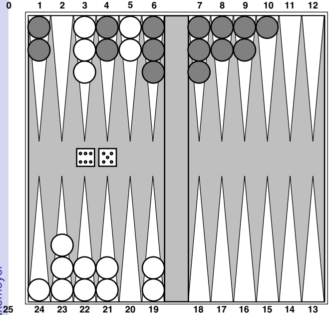
© 1992 Patrick H. Winston

for randomly ordered leaf nodes: $O((b/\log b)^d)$ [Knuth+Moore 75]

only for $b > 1000$

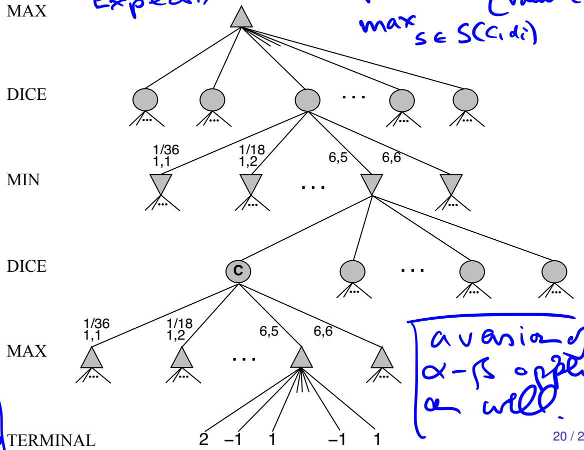
© G. Lakemeyer

Games with an Element of Chance



Let $p(d_i)$ be the prob. that d_i is the result of rolling dice.
Let $S(C, d_i)$ be the set of all states reachable by MAX after rolling d_i

$$\text{ExpectiMAX} = \sum_i p(d_i) * \left(\max_{s \in S(C,d_i)} \text{Value}(s)\right)$$



moves for white:

5 → 10, 5 → 11

5 → 10, 10 → 16

5 → 11, 11 → 16

19 → 24, 5 → 11

complexity

$O(b^d \cdot n^d)$

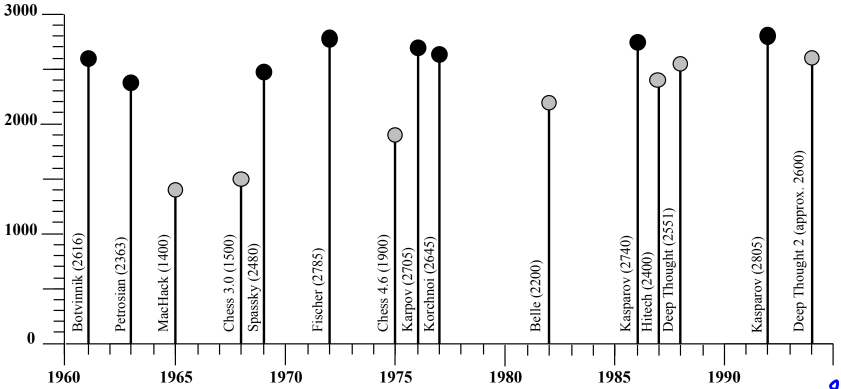
n # distinct rolls

a variant of α-β applies as well.

© G. Lakemeyer

Chess Programs

M. Carlsen
2882



2014

2018 Computer Chess World Champ.
"Komodo" (Dailey, Loffler, Kaufmann)
Elo 3404

© G. Lakemeyer