# Artificial Intelligence

For HEDSPI Project

**Lecturer 6 - Advanced search methods**

Lecturers :
Le Thanh Huong
Tran Duc Khanh
Dept of Information Systems
Faculty of Information Technology - HUT



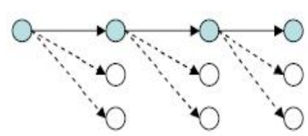
## Outline

- Local beam search
- Game and search
- Alpha-beta pruning

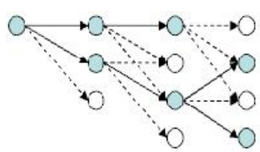


## Local beam search

- Like greedy search, but keep K states at all times:
  - Initially: *k* random states
  - Next: determine all successors of *k* states
  - If any of successors is goal → finished
  - Else select *k* best from successors and repeat.



Greedy Search



Beam Search



## Local beam search

- Major difference with random-restart search
  - Information is shared among k search threads: If one state generated good successor, but others did not → "come here, the grass is greener!"
- Can suffer from lack of diversity.
  - Stochastic variant: choose k successors at proportionally to state success.
- The best choice in MANY practical settings

## Games and search

- Why study games?
- Why is search a good idea?

- Majors assumptions about games:
  - Only an agent's actions change the world
  - World is deterministic and accessible

5

## Why study games?

May 1997
Deep Blue - Garry Kasparov
3.5 - 2.5

machines are better than humans in:
othello
humans are better than machines in:
go
here: perfect information zero-sum games

6

## Why study games?

- Games are a form of *multi-agent environment*
  - What do other agents do and how do they affect our success?
  - Cooperative vs. competitive multi-agent environments.
  - Competitive multi-agent environments give rise to adversarial search a.k.a. *games*

- Why study games?
  - Fun; historically entertaining
  - Interesting subject of study because they are hard
  - Easy to represent and agents restricted to small number of actions

7

## Relation of Games to Search

- Search – no adversary
  - Solution is (heuristic) method for finding goal
  - Heuristics and CSP techniques can find *optimal* solution
  - Evaluation function: estimate of cost from start to goal through given node
  - Examples: path planning, scheduling activities
- Games – adversary
  - Solution is strategy (strategy specifies move for every possible opponent reply).
  - Time limits force an *approximate* solution
  - Evaluation function: evaluate "goodness" of game position
  - Examples: chess, checkers, Othello, backgammon
- Ignoring computational complexity, games are a perfect application for a complete search.
- Of course, ignoring complexity is a bad idea, so games are a good place to study resource bounded searches.

8

## Types of Games

| | deterministic | chance |
|---|---|---|
| perfect information | chess, checkers, go, othello | backgammon monopoly |
| imperfect information | battleships, blind tictactoe | bridge, poker, scrabble nuclear war |

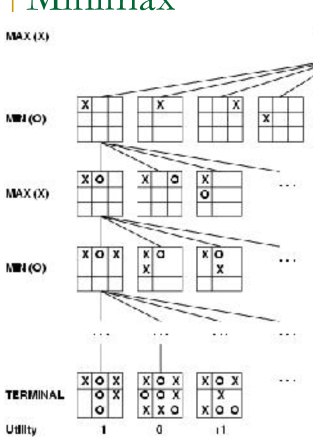


## Minimax

- Two players: MAX and MIN
- MAX moves first and they take turns until the game is over. Winner gets award, looser gets penalty.
- Games as search:
  - **Initial state:** e.g. board configuration of chess
  - **Successor function:** list of (move,state) pairs specifying legal moves.
  - **Terminal test:** Is the game finished?
  - **Utility function:** Gives numerical value of terminal states.
  - E.g. win (+1), loose (-1) and draw (0) in tic-tac-toe
- MAX uses search tree to determine next move.
- Perfect play for deterministic games



## Minimax




- From among the moves available to you, take the best one
- The best one is determined by a search using the MiniMax strategy



## Optimal strategies

- MAX maximizes a function: find a move corresponding to max value
- MIN minimizes the same function: find a move corresponding to min value

At each step:

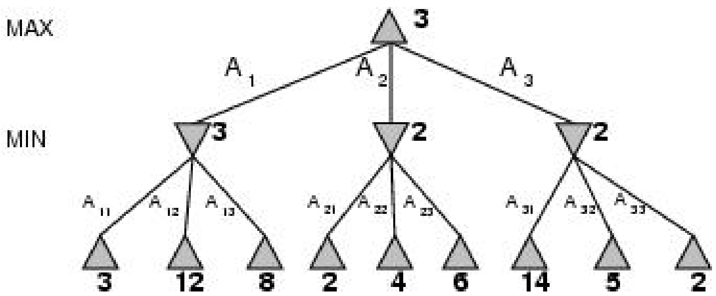
- If a state/node corresponds to a MAX move, the function value will be the maximum value of its childs
- If a state/node corresponds to a MIN move, the function value will be the minimum value of its childs

Given a game tree, the optimal strategy can be determined by using the minimax value of each node:

MINIMAX-VALUE($n$)=

| | |
|---|---|
| UTILITY($n$) | If $n$ is a terminal |
| $\max_{s \in successors(n)}$ MINIMAX-VALUE($s$) | If $n$ is a max node |
| $\min_{s \in successors(n)}$ MINIMAX-VALUE($s$) | If $n$ is a min node |

## Minimax






## Minimax algorithm

```
function MINIMAX-DECISION(state) returns an action
   v ← MAX-VALUE(state)
   return the action in SUCCESSORS(state) with value v

function MAX-VALUE(state) returns a utility value
   if TERMINAL-TEST(state) then return UTILITY(state)
   v ← −∞
   for a, s in SUCCESSORS(state) do
      v ← MAX(v, MIN-VALUE(s))
   return v

function MIN-VALUE(state) returns a utility value
   if TERMINAL-TEST(state) then return UTILITY(state)
   v ← ∞
   for a, s in SUCCESSORS(state) do
      v ← MIN(v, MAX-VALUE(s))
   return v
```



## Properties of minimax

- Complete? Yes (if tree is finite)
- Optimal? Yes (against an optimal opponent)
- Time complexity? $O(b^m)$
- Space complexity? O(bm) (depth-first exploration)

- For chess, b ≈ 35, m ≈100 for "reasonable" games
  → exact solution completely infeasible



## Problem of minimax search

- Number of games states is exponential to the number of moves.
  - Solution: Do not examine every node

⇒ Alpha-beta pruning:
- Remove branches that do not influence final decision
- Revisit example …

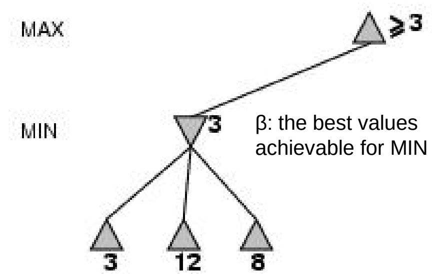
## α-β pruning

- Alpha values: the best values achievable for MAX, hence the max value so far
- Beta values: the best values achievable for MIN, hence the min value so far
- At MIN level: compare result V of node to alpha value. If V>alpha, pass value to parent node and BREAK
- At MAX level: compare result V of node to beta value. If V<beta, pass value to parent node and BREAK



## α-β pruning

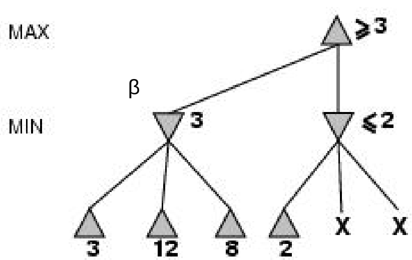
α: the best values achievable for MAX





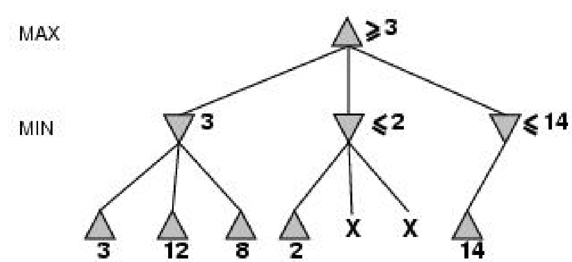

## α-β pruning example

Compare result V of node to β. If V< β, pass value to parent node and BREAK

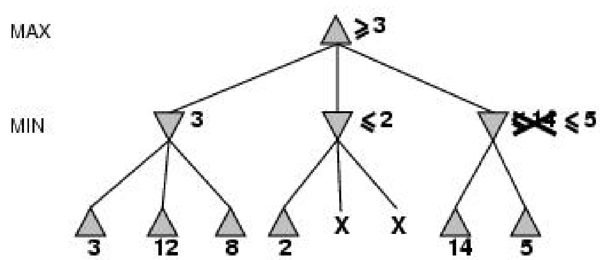





## α-β pruning example

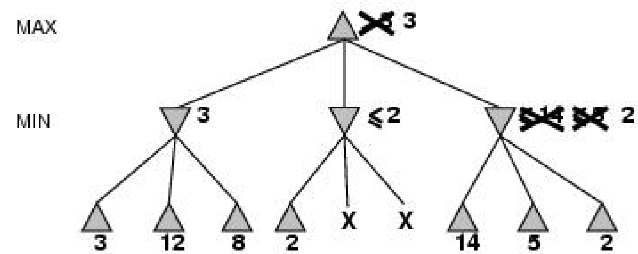


## α-β pruning example






## α-β pruning example





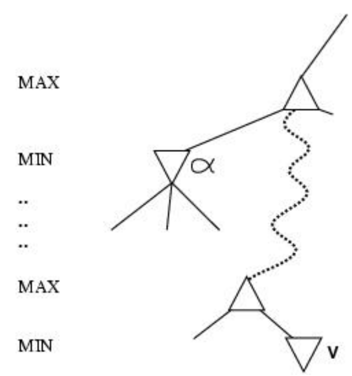

## Properties of α-β

- Pruning does not affect final result
- Entire sub-trees can be pruned.
- Good move ordering improves effectiveness of pruning. With "perfect ordering"
  - time complexity = O($b^{m/2}$)
    - → doubles depth of search
  - Branching factor of sqrt(b) !!
  - Alpha-beta pruning can look twice as far as minimax in the same amount of time
- Repeated states are again possible.
  - Store them in memory = transposition table
- A simple example of the value of reasoning about which computations are relevant (a form of metareasoning)



## Why is it called α-β?

- α is the value of the best (i.e., highest-value) choice found so far at any choice point along the path for *max*
- If *v* is worse than α, *max* will avoid it
  - → prune that branch
- Define β similarly for *min*



## The α-β algorithm

```
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
```



## The α-β algorithm

```
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
```



## Imperfect, real-time decisions

- Minimax and alpha-beta pruning require too much leafnode evaluations.
- May be impractical within a reasonable amount of time.
- Suppose we have 100 secs, explore $10^4$ nodes/sec → $10^6$ nodes per move
- Standard approach (SHANNON, 1950):
  - Cut off search earlier (replace TERMINAL-TEST by CUTOFF-TEST)
  - Apply heuristic evaluation function EVAL (replacing utility function of alpha-beta)



## Cut-off search

- Change:

  **if** TERMINAL-TEST(*state*) **then return** UTILITY(*state*)

  into:

  **if** CUTOFF-TEST(*state,depth*) **then return** EVAL(*state*)

- Introduces a fixed-depth limit *depth*
  - Is selected so that the amount of time will not exceed what the rules of the game allow.
- When cut-off occurs, the evaluation is performed.

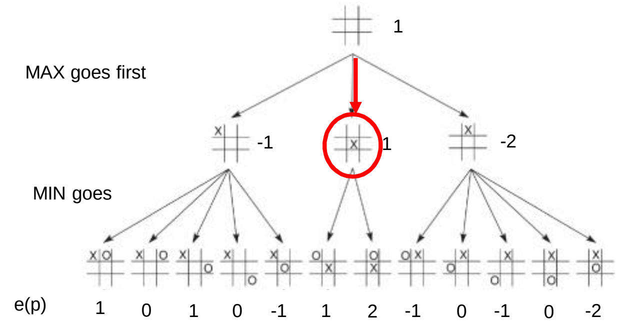
# Heuristic evaluation (EVAL)

- Idea: produce an estimate of the expected utility of the game from a given position.
- Requirements:
  - EVAL should order terminal-nodes in the same way as UTILITY.
  - Computation may not take too long.
  - For non-terminal states the EVAL should be strongly correlated with the actual chance of winning.
- Example:
  Expected value e(p) for each state p:
  E(p) = (# open rows, columns, diagonals for MAX)
  - (# open rows, columns, diagonals for MIN)
- MAX moves all lines that don't have o; MIN moves all lines that don't have x



Expected value e(p) for each state p:
E(p) = (# open rows, columns, diagonals for MAX)
- (# open rows, columns, diagonals for MIN)
MAX moves all lines that don't have o; MIN moves all lines that don't have x

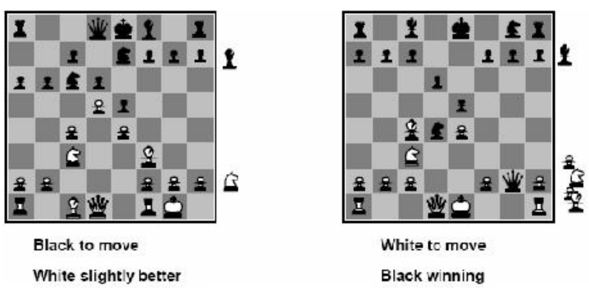
## on the symmetry of the states




→ A kind of depth-first search



# Evaluation function example




- For chess, typically linear 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.



# Chess complexity

- PC can search 200 millions nodes/3min.
- Branching factor: ~35
  - $35^5$ ~ 50 millions
  - if use minimax, could look ahead ***5 plies***, defeated by average player, planning 6-8 plies.
- Does it work in practice?
  - 4-ply ≈ human novice → hopeless chess player
  - 8-ply ≈ typical PC, human master
  - 12-ply ≈ Deep Blue, Kasparov
- To reach grandmaster level, needs a better *extensively tuned evaluation* and a *large database of optimal opening and ending* of the game

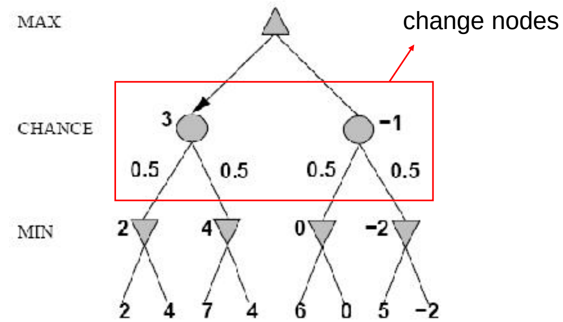
## Deterministic games in practice

- Checkers: Chinook ended 40-year-reign of human world champion Marion Tinsley in 1994. Used a precomputed endgame database defining perfect play for all positions involving 8 or fewer pieces on the board, a total of 444 billion positions.
- Chess: Deep Blue defeated human world champion Garry 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: 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.

33

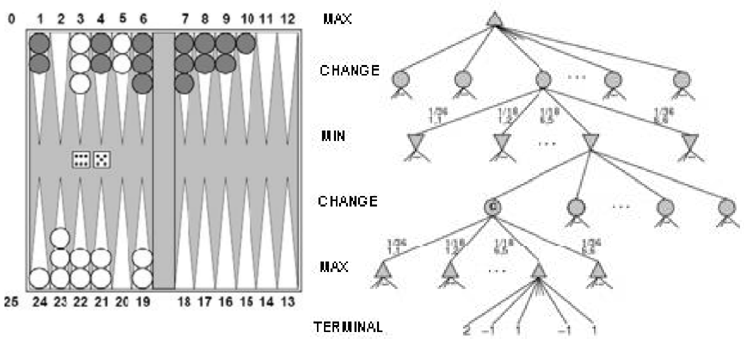
## Nondeterministic games

- Chance introduces by dice, card-shuffling, coin-flipping...
- Example with coin-flipping:




34

## Backgammon




Possible moves: (5-10,5-11), (5-11,19-24),(5-10,10-16) and (5-11,11-16)

35

## Expected minimax value

```
...
if state is a MAX node then
    return the highest EXPECTIMINIMAX-VALUE of SUCCESSORS(state)
if state is a MIN node then
    return the lowest EXPECTIMINIMAX-VALUE of SUCCESSORS(state)
if state is a chance node then
    return average of EXPECTIMINIMAX-VALUE of SUCCESSORS(state)
```

**EXPECTED-MINIMAX-VALUE(*n*)=**

| | |
|---|---|
| UTILITY(*n*) | If *n* is a terminal |
| max$_{\text{sčsuccessors}(n)}$ EXPECTEDMINIMAX(*s*) | If *n* is a max node |
| min$_{\text{sčsuccessors}(n)}$ EXPECTEDMINIMAX(*s*) | If *n* is a max node |
| $\Sigma_{\text{sčsuccessors}(n)}$ *P(s)* .EXPECTEDMINIMAX(*s*) | If *n* is a chance node |

*P(s) is probability of s occurence*

36

# Games of imperfect information

- E.g., card games, where opponent's initial cards are unknown
- Typically we can calculate a probability for each possible deal
- Seems just like having one big dice roll at the beginning of the game
- Idea: compute the minimax value of each action in each deal, then choose the action with highest expected value over all deals
- Special case: if an action is optimal for all deals, it's optimal.
- GIB, current best bridge program, approximates this idea by
  - generating 100 deals consistent with bidding information
  - picking the action that wins most tricks on average