Adversarial Search

CE417: Introduction to Artificial Intelligence
Sharif University of Technology
Spring 2016

Soleymani

“Artificial Intelligence: A Modern Approach”, 3rd Edition, Chapter 5

Outline

- Game as a search problem
- Minimax algorithm
- α-β Pruning: ignoring a portion of the search tree
- Time limit problem
  - Cut off & Evaluation function

Games as search problems

- Games
  - **Adversarial search** problems (goals are in conflict)
    - Competitive multi-agent environments

- Games in AI are a specialized kind of games (in the game theory)

Primary assumptions

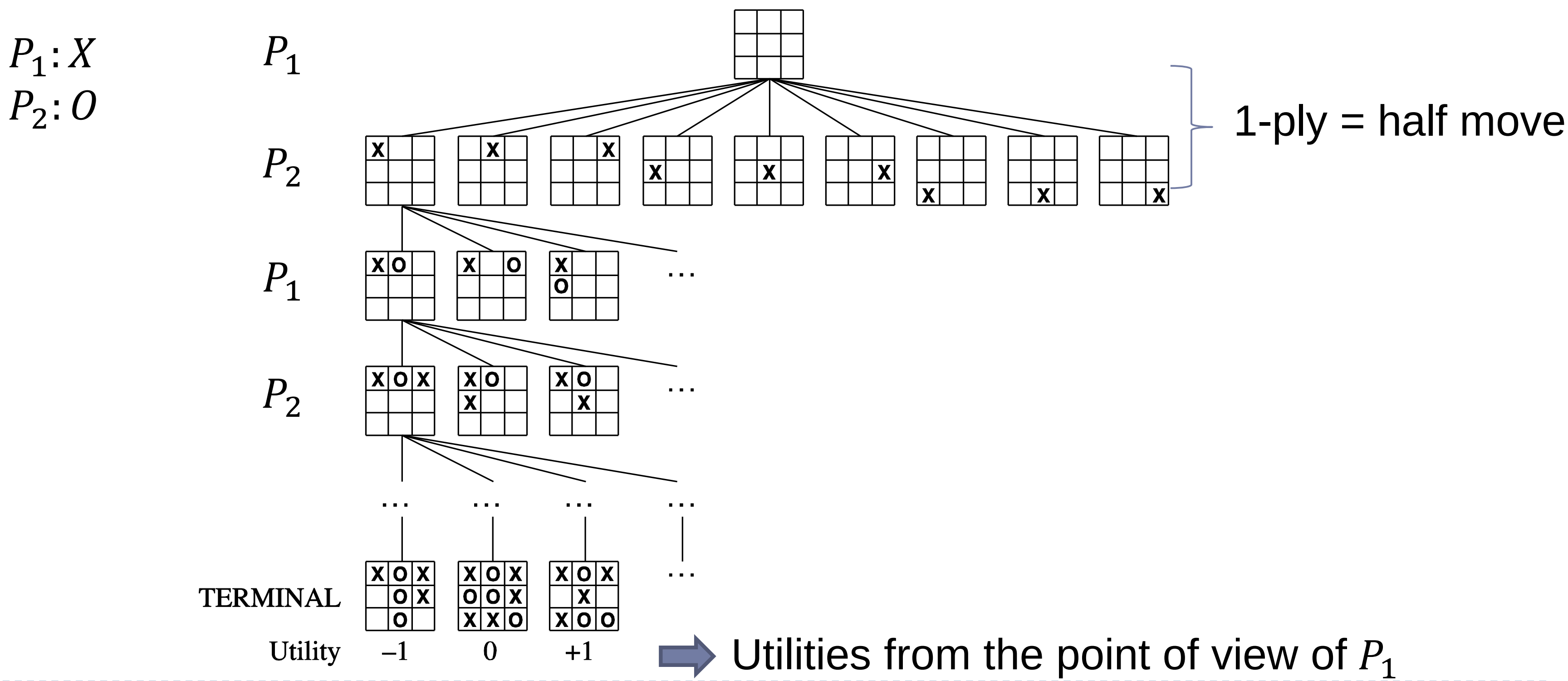
- Common games in AI:
  - Two-player
  - Turn taking
    - agents act alternately
  - Zero-sum
    - agents' goals are in conflict: sum of utility values at the end of the game is zero or constant
  - Deterministic
  - Perfect information
    - fully observable

Game as a kind of search problem

- Initial state S_0, set of states (each state contains also the turn), $ACTIONS(s)$, $RESULTS(s, a)$ like standard search
- $PLAYERS(s)$: Defines which player takes turn in a state
- $TERMINAL\_TEST(s)$: Shows where game has ended
- $UTILITY(s, p)$: utility or payoff function $U: S \times P \rightarrow \mathbb{R}$ (how good is the terminal state s for player p)
  - Zero-sum (constant-sum) game: the total payoff to all players is zero (or constant) for every terminal state
  - We have utilities at end of game instead of sum of action costs

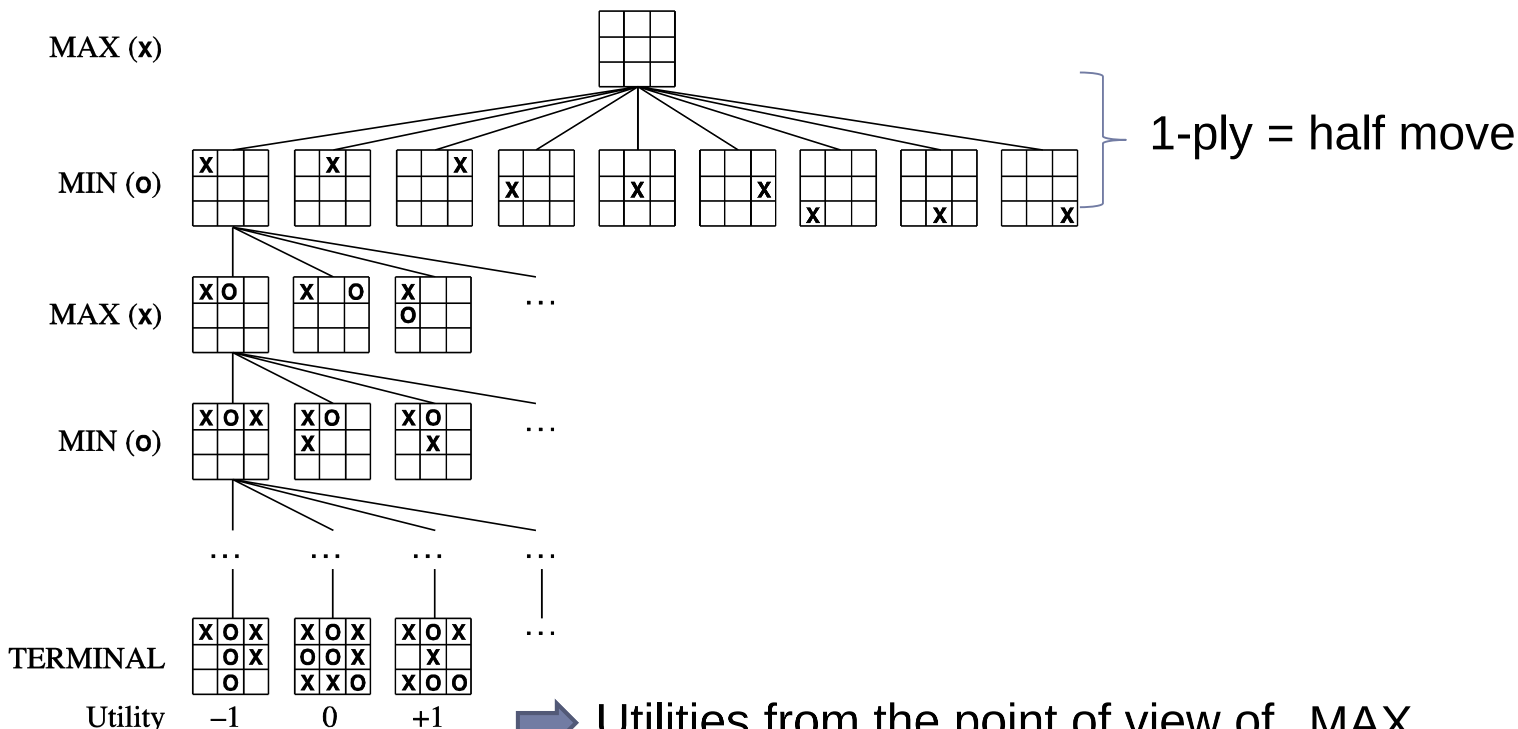
Game tree (tic-tac-toe)

- Two players: P_1 and P_2 (P_1 is now searching to find a good move)
  - Zero-sum games: P_1 gets $U(t)$, P_2 gets $C - U(t)$ for terminal node t



Game tree (tic-tac-toe)

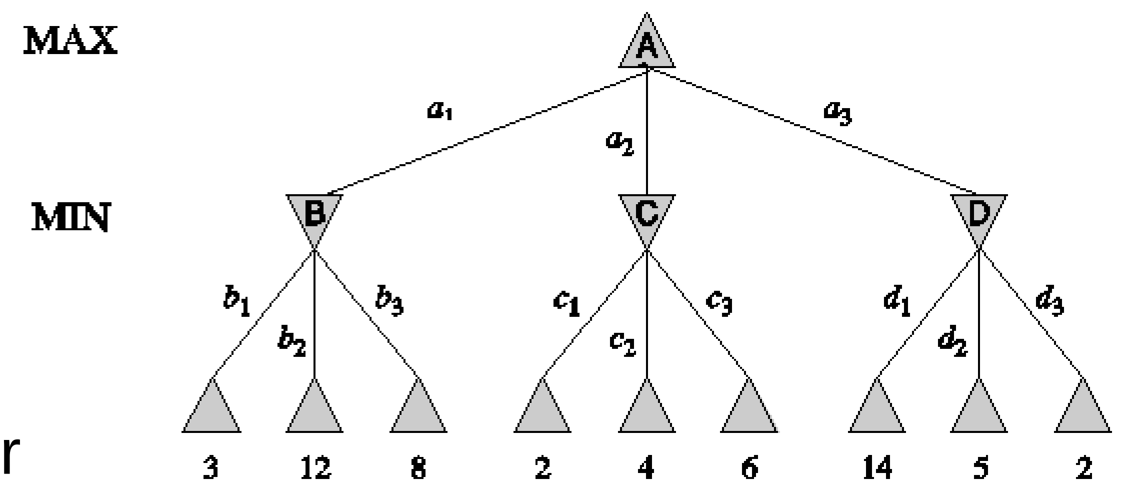
- Two players: P_1 and P_2 (P_1 is now searching to find a good move)
  - Zero-sum games: P_1 gets $U(t)$, P_2 gets $C - U(t)$ for terminal node t



Utilities from the point of view of MAX

Optimal play

- Opponent is assumed optimal
- Minimax function is used to find the utility of each state.
  - MAX/MIN wants to maximize/minimize the terminal payoff



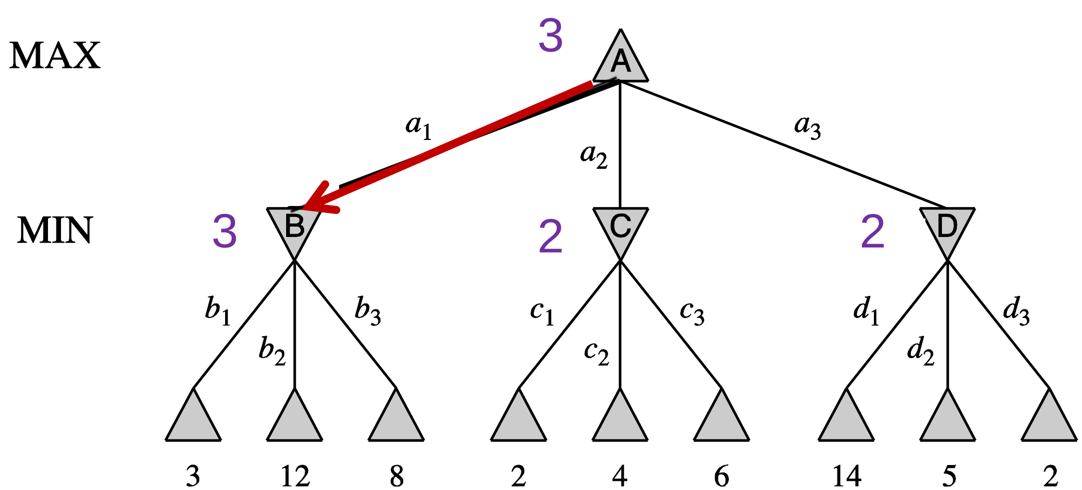
MAX gets $U(t)$ for terminal node t

Minimax

$$MINIMAX(s) = \begin{cases} UTILITY(s, MAX) & if\ TERMINAL\_TEST(s) \\ max_{a \in ACTIONS(s)} MINIMAX(RESULT(s,a)) & PLAYER(s) = MAX \\ min_{a \in ACTIONS(s)} MINIMAX(RESULT(s,a)) & PLAYER(s) = MIN \end{cases}$$

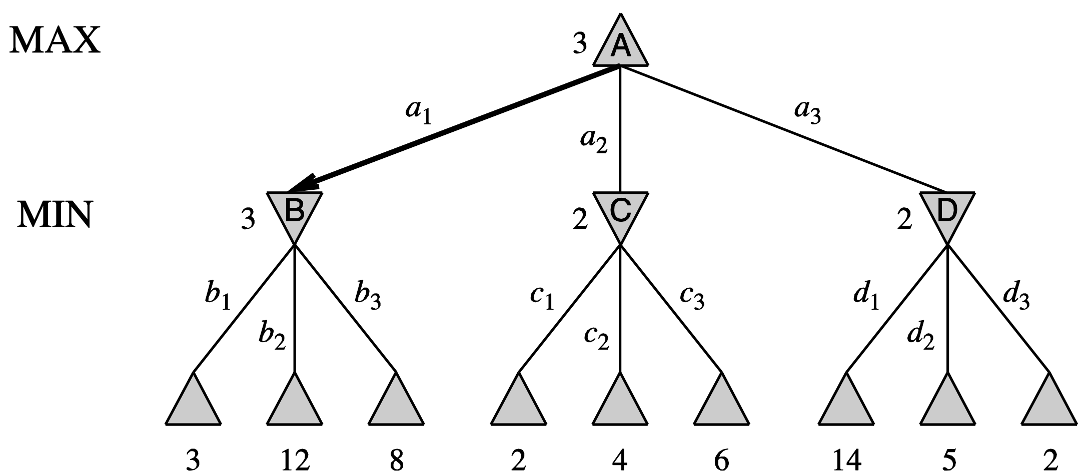
Utility of being in state s

- $MINIMAX(s)$ shows the best achievable outcome of being in state s (assumption: optimal opponent)



Minimax (Cont.)

- Optimal strategy: move to the state with highest minimax value
  - Best achievable payoff against best play
    - Maximizes the worst-case outcome for MAX
  - It works for zero-sum games



Minimax algorithm

Depth first search

function $MINIMAX\_DECISION(state)$ **returns** $an\ action$
  return $\underset{a \in ACTIONS(state)}{\text{argmax}}\ MIN\_VALUE(RESULT(state, a))$

---

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**
    $v \leftarrow MAX(v, MIN\_VALUE(RESULTS(state, a)))$
  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**
    $v \leftarrow MIN(v, MAX\_VALUE(RESULTS(state, a)))$
  return v

Properties of minimax

- Complete? Yes (when tree is finite)

- Optimal? Yes (against an optimal opponent)

- Time complexity: $O(b^m)$

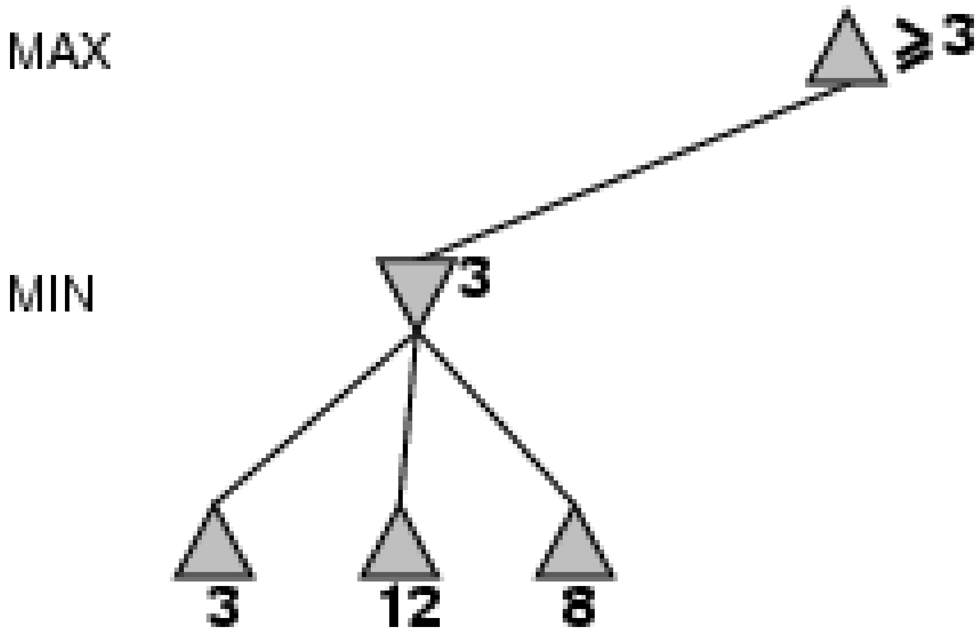
- Space complexity: $O(bm)$ (depth-first exploration)

- For chess, $b \approx 35, m > 50$ for reasonable games
  - Finding exact solution is completely infeasible

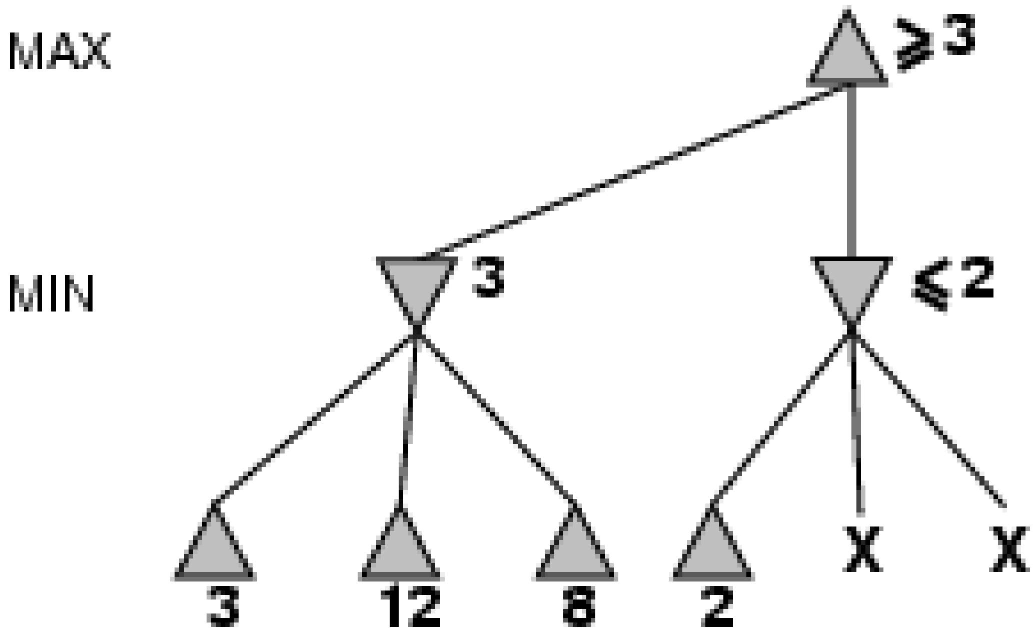
Pruning

- Correct minimax decision without looking at every node in the game tree
  - α-β pruning
    - Branch & bound algorithm
    - Prunes away branches that cannot influence the final decision

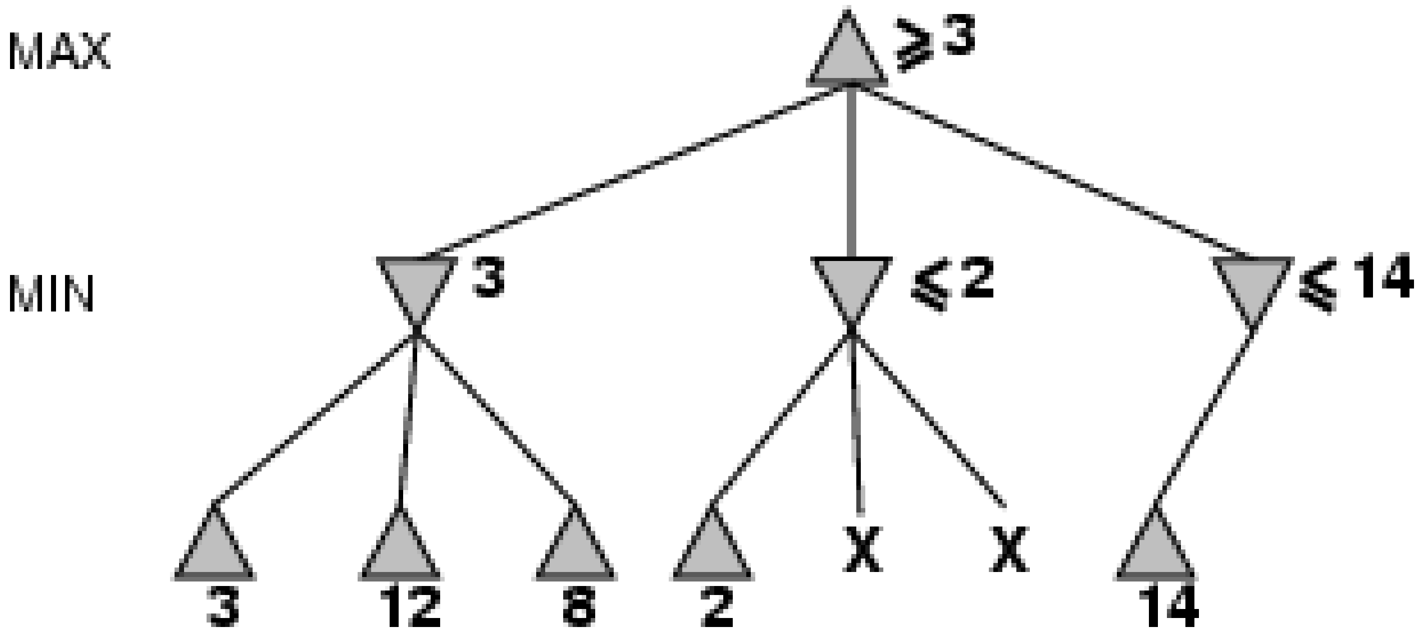
α-β pruning example



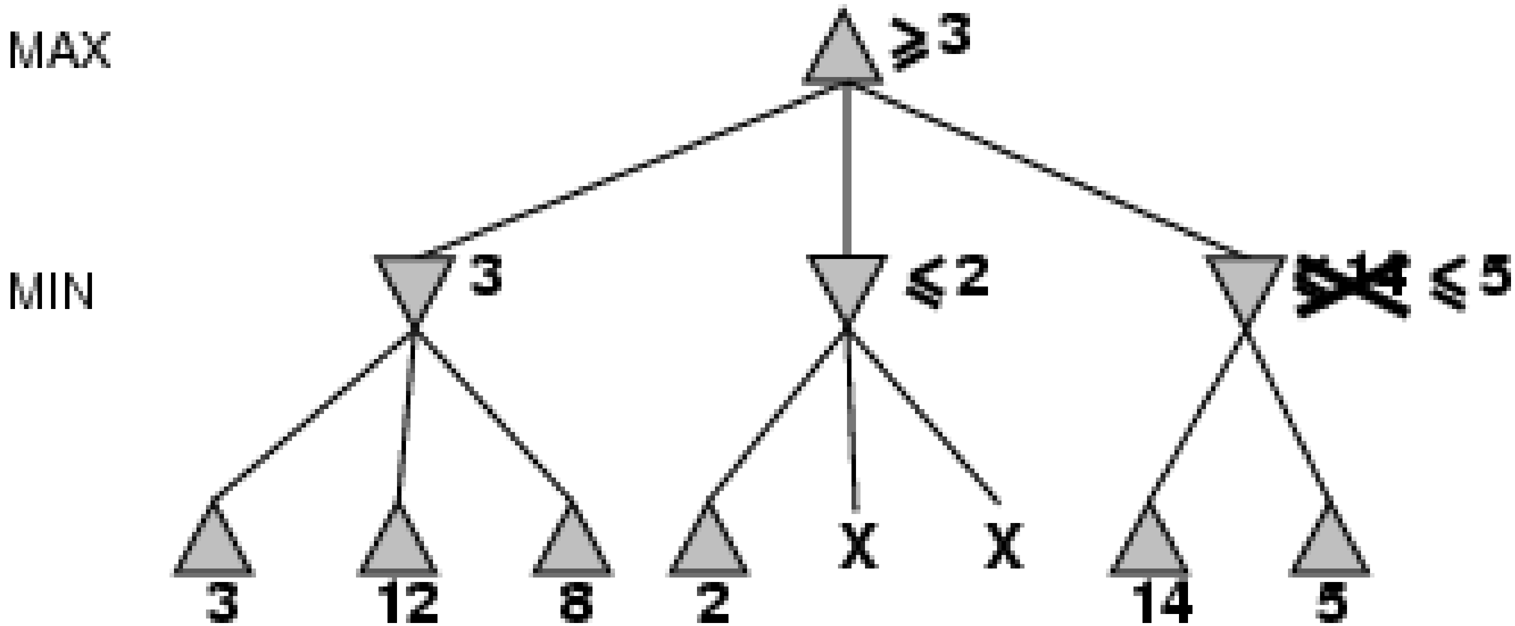
α-β pruning example



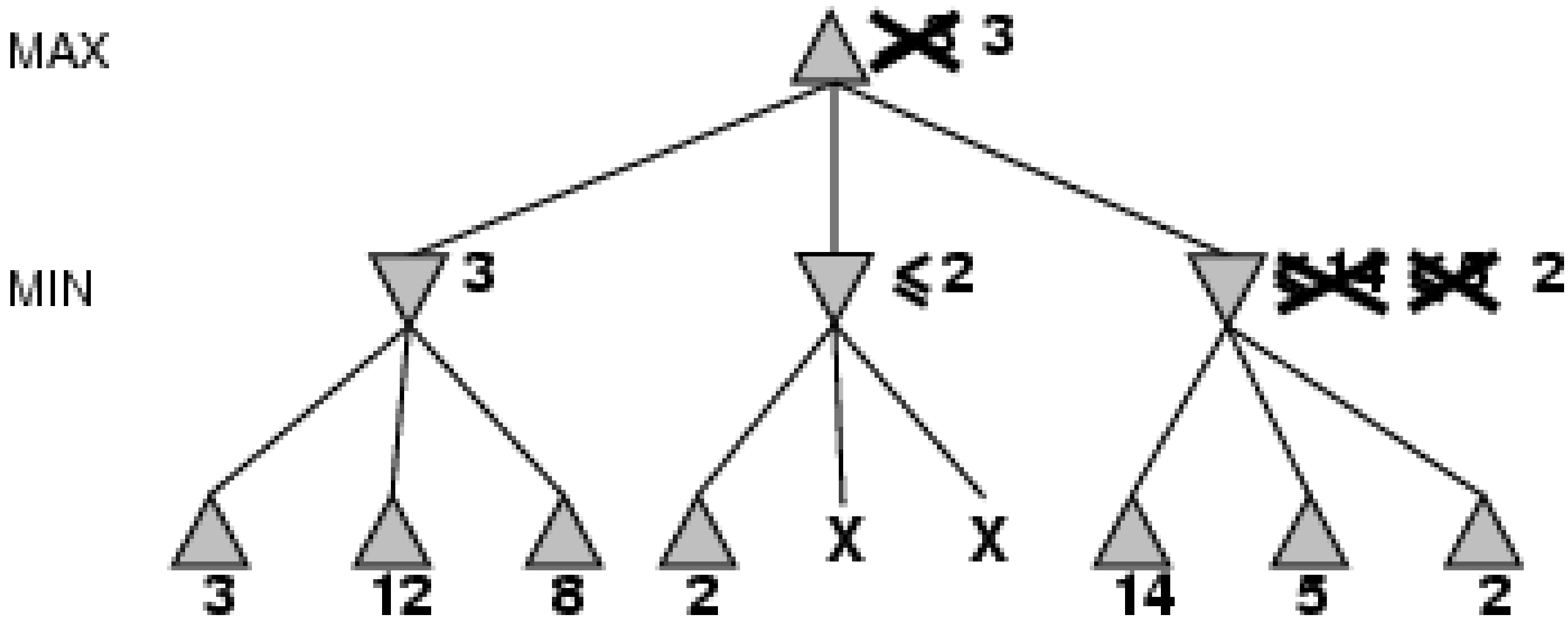
α-β pruning example



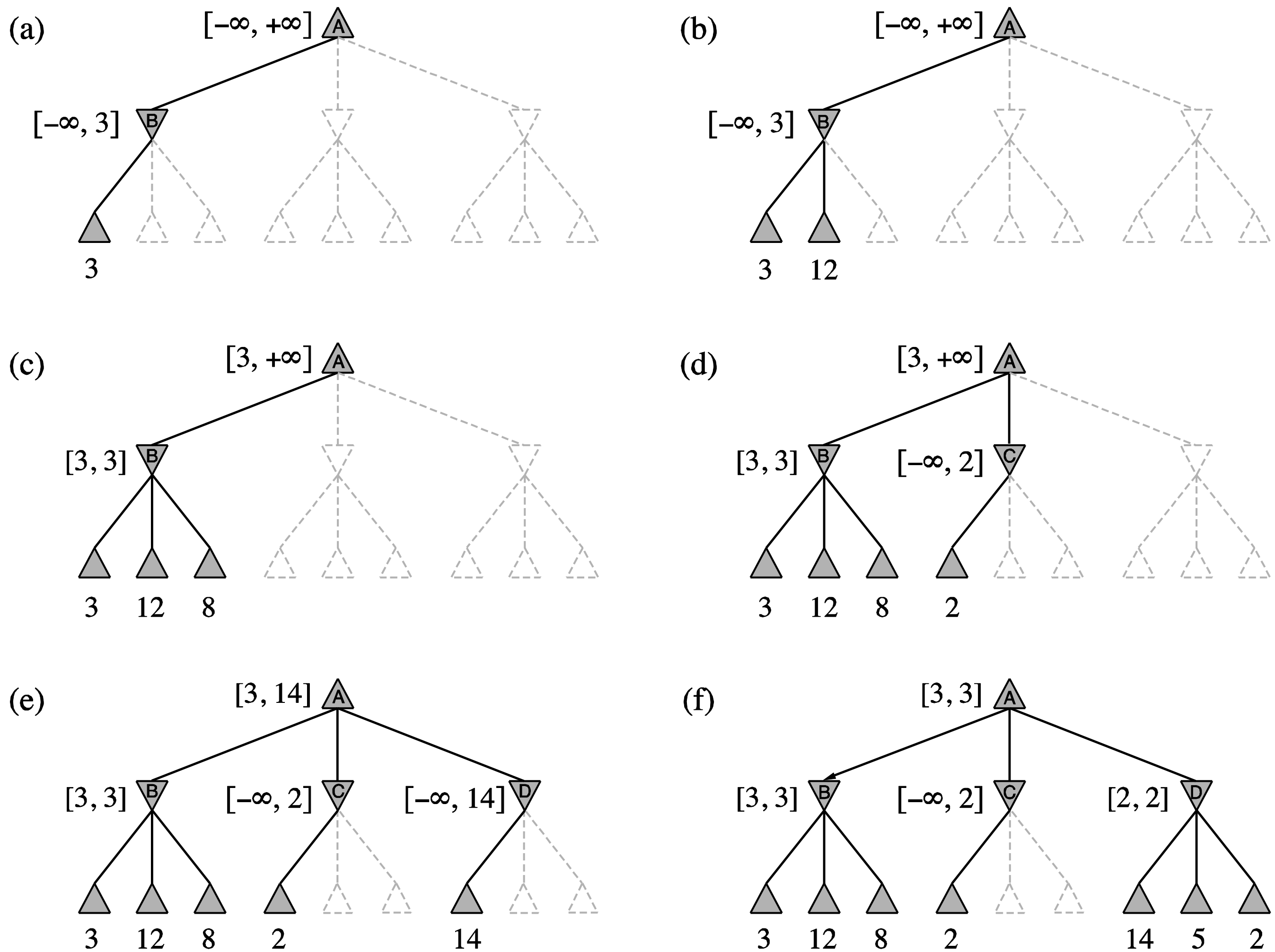
α-β pruning example



α-β pruning example



α-β progress

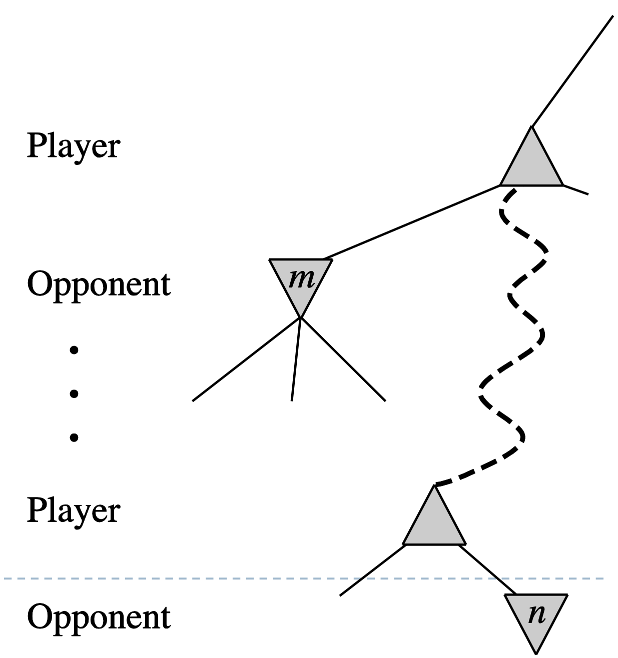


(a)
[−∞, +∞]
A
[−∞, 3]
B
3
(b)
[−∞, +∞]
A
[−∞, 3]
B
3
12
(c)
[3, +∞]
A
[3, 3]
B
3
12
8
(d)
[3, +∞]
A
[3, 3]
B
[−∞, 2]
C
3
12
8
2
(e)
[3, 14]
A
[3, 3]
B
[−∞, 2]
C
[−∞, 14]
D
3
12
8
2
14
(f)
[3, 3]
A
[3, 3]
B
[−∞, 2]
C
[2, 2]
D
3
12
8
2
14
5
2

α-β pruning

- Assuming depth-first generation of tree
  - We prune node n when player has a better choice m at (parent or) any ancestor of n

- Two types of pruning (cuts):
  - pruning of max nodes (α-cuts)
  - pruning of min nodes (β-cuts)



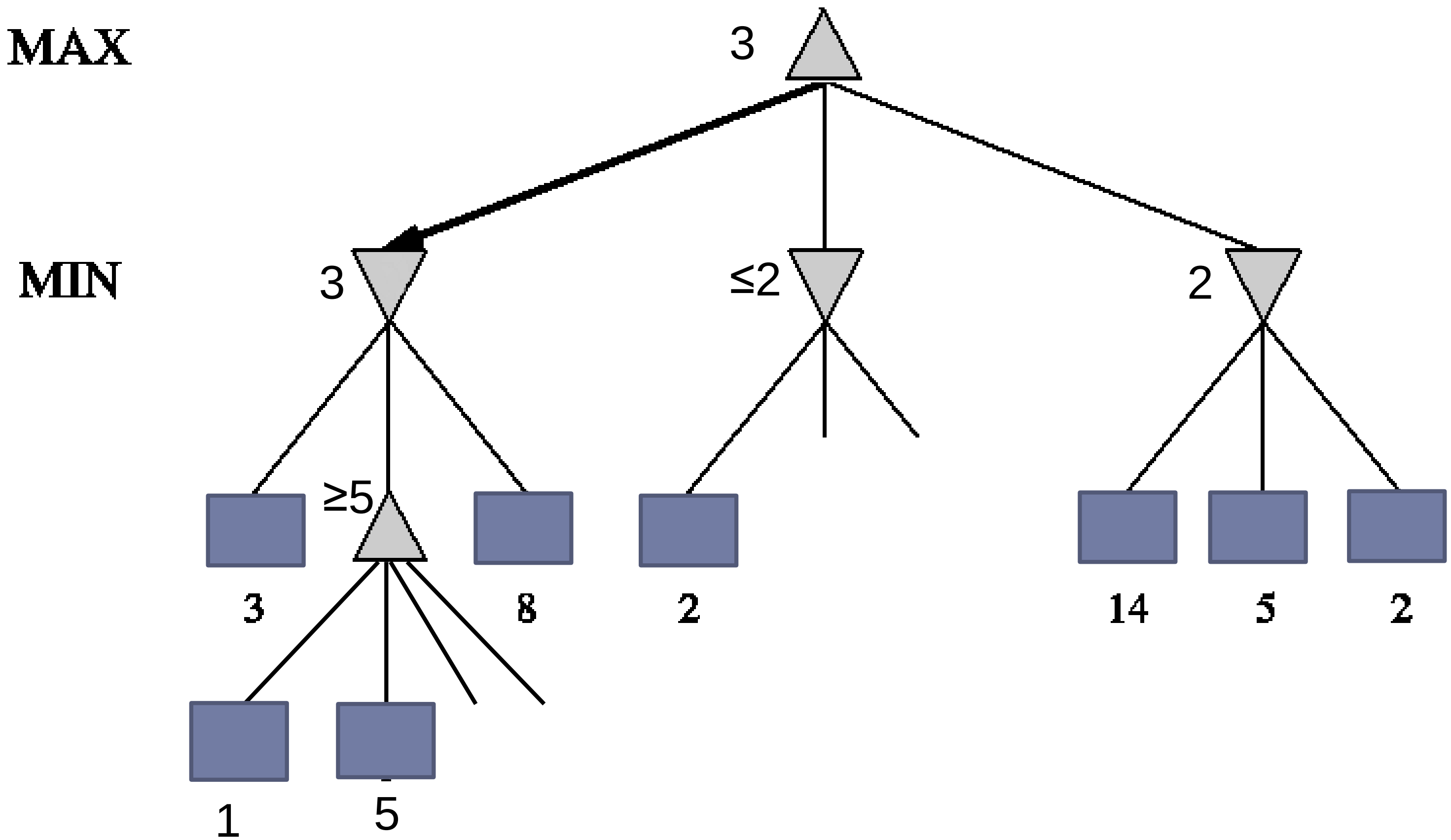
Why is it called α-β?

- α: Value of the best (<u>highest</u>) choice found so far at any choice point along the path for MAX
- β: Value of the best (<u>lowest</u>) choice found so far at any choice point along the path for MIN

- Updating α and β during the search process

- For a MAX node once the value of this node is known to be more than the current β ($v \geq \beta$), its remaining branches are pruned.
- For a MIN node once the value of this node is known to be less than the current α ($v \leq \alpha$), its remaining branches are pruned.

α-β pruning (an other example)



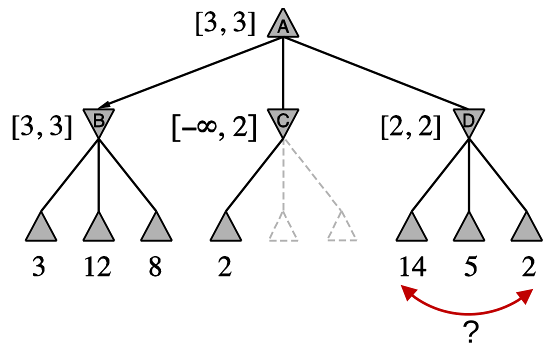
```
function ALPHA_BETA_SEARCH(state) returns an action
    v ← MAX_VALUE(state, −∞, +∞)
    return the action in ACTIONS(state) with value v
```

```
function MAX_VALUE(state, α, β) returns a utility value
    if TERMINAL_TEST(state) then return UTILITY(state)
    v ← −∞
    for each a in ACTIONS(state) do
        v ← MAX(v, MIN_VALUE(RESULTS(state, a), α, β))
        if v ≥ β then return v
        α ← MAX(α, v)
    return v
```

```
function MIN_VALUE(state, α, β) returns a utility value
    if TERMINAL_TEST(state) then return UTILITY(state)
    v ← +∞
    for each a in ACTIONS(state) do
        v ← MIN(v, MAX_VALUE(RESULTS(state, a), α, β))
        if v ≤ α then return v
        β ← MIN(β, v)
    return v
```

Order of moves

- Good move ordering improves effectiveness of pruning



- Best order: time complexity is $O(b^{m/2})$
- Random order: time complexity is about $O(b^{3m/4})$ for moderate b
  - α-β pruning just improves the search time only partly

Computational time limit (example)

- 100 secs is allowed for each move (game rule)
- 10^4 nodes/sec (processor speed)
- We can explore just 10^6 nodes for each move
  - $b^m = 10^6$, $b=35 \Longrightarrow m=4$

  (4-ply look-ahead is a hopeless chess player!)

Computational time limit: Solution

- We must make a decision even when finding the optimal move is infeasible.

- Cut off the search and apply a heuristic evaluation function
  - cutoff test: turns non-terminal nodes into terminal leaves
    - Cut off test instead of terminal test (e.g., depth limit)
  - evaluation function: estimated desirability of a state
    - Heuristic function evaluation instead of utility function

- This approach does not guarantee optimality.

Heuristic minimax

$$H_{MINIMAX(s,d)} =$$

$$\begin{cases} EVAL(s, MAX) & if\ CUTOFF\_TEST(s,d) \\ max_{a \in ACTIONS(s)} H\_MINIMAX(RESULT(s,a), d+1) & PLAYER(s) = \text{MAX} \\ min_{a \in ACTIONS(s)} H\_MINIMAX(RESULT(s,a), d+1) & PLAYER(s) = \text{MIN} \end{cases}$$

Evaluation functions

- For terminal states, it should order them in the same way as the true utility function.
- For non-terminal states, it should be strongly correlated with the actual chances of winning.
- It must not need high computational cost.

Evaluation functions based on features

- Example: features for evaluation of the chess states
  - Number of each kind of piece: number of white pawns, black pawns, white queens, black queens, etc
  - King safety
  - Good pawn structure

Evaluation functions

- Weighted sum of features
  - Assumption: contribution of each feature is independent of the value of the other features

$$EVAL(s) = w_1 \times f_1(s) + w_2 \times f_2(s) + \cdots + w_n \times f_n(s)$$

- Weights can be assigned based on the human experience or machine learning methods.
  - Example: Chess
    - Features: number of white pawns (f_1), number of white bishops (f_2), number of white rooks (f_3), number of black pawns (f_4), …
    - Weights: $w_1 = 1, w_2 = 3, w_3 = 5, w_4 = -1, \dots$

Cutting off search: simple depth limit

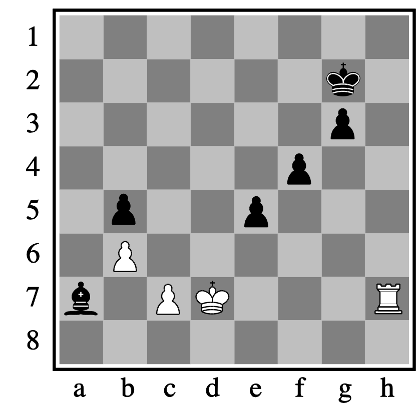
- Simple: depth limit d_0

$$CUTOFF\_TEST(s,d) = \begin{cases} true & if\ d > d_0\ or\ TERMINAL\_TEST(s) = TRUE \\ false & otherwise \end{cases}$$

Cutting off search: simple depth limit

- Problem1: non-quiescent positions
  - Few more plies make big difference in evaluation value

- Problem 2: horizon effect
  - Delaying tactics against opponent's move that causes serious unavoidable damage (because of pushing the damage beyond the horizon that the player can see)



More sophisticated cutting off

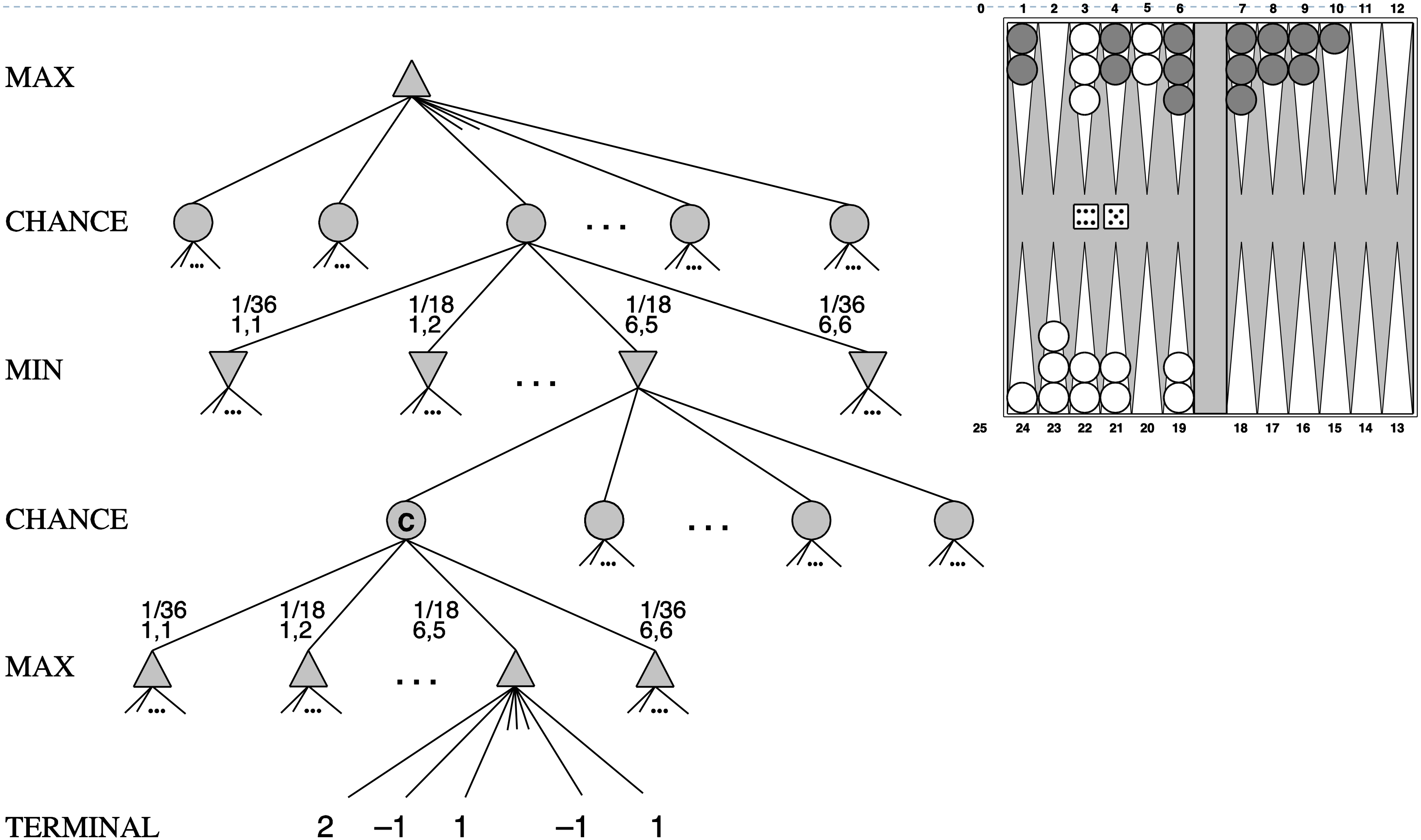
- Cutoff only on quiescent positions
  - Quiescent search: expanding non-quiescent positions until reaching quiescent ones

- Horizon effect
  - Singular extension: a move that is clearly better than all other moves in a given position.
    - Once reaching the depth limit, check to see if the singular extension is a legal move.
    - It makes the tree deeper but it does not add many nodes to the tree due to few possible singular extensions.

Speed up the search process

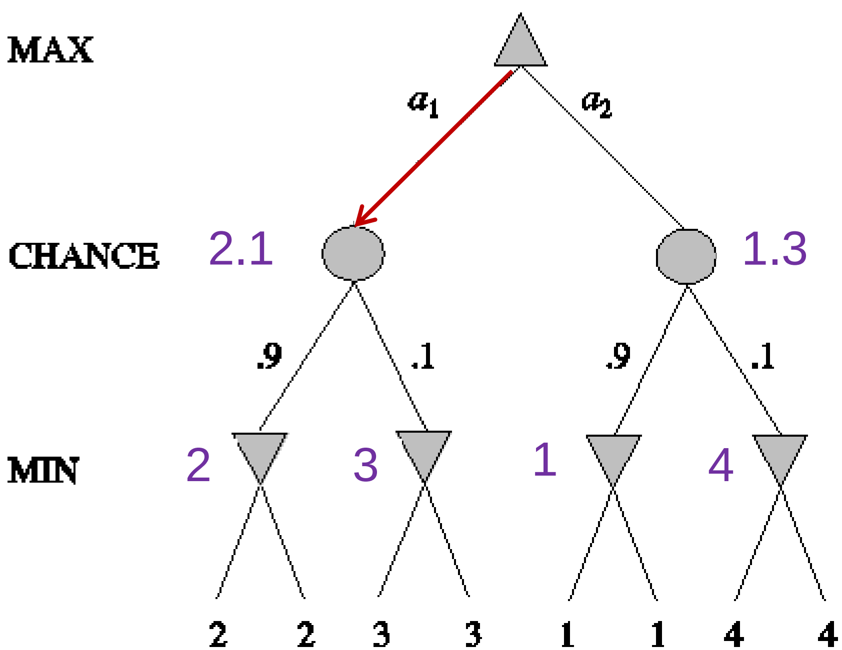
- Table lookup rather than search for some states
  - E.g.,for the opening and ending of games (where there are few choices)
- Example: Chess
  - For each opening, the best advice of human experts (from books describing good plays) can be copied into tables.
  - For endgame, computer analysis is usually used (solving endgames by computer).

Stochastic games: Backgammon



Stochastic games

- Expected utility: Chance nodes take average (expectation) over all possible outcomes.
  - It is consistent with the definition of rational agents trying to maximize expected utility.



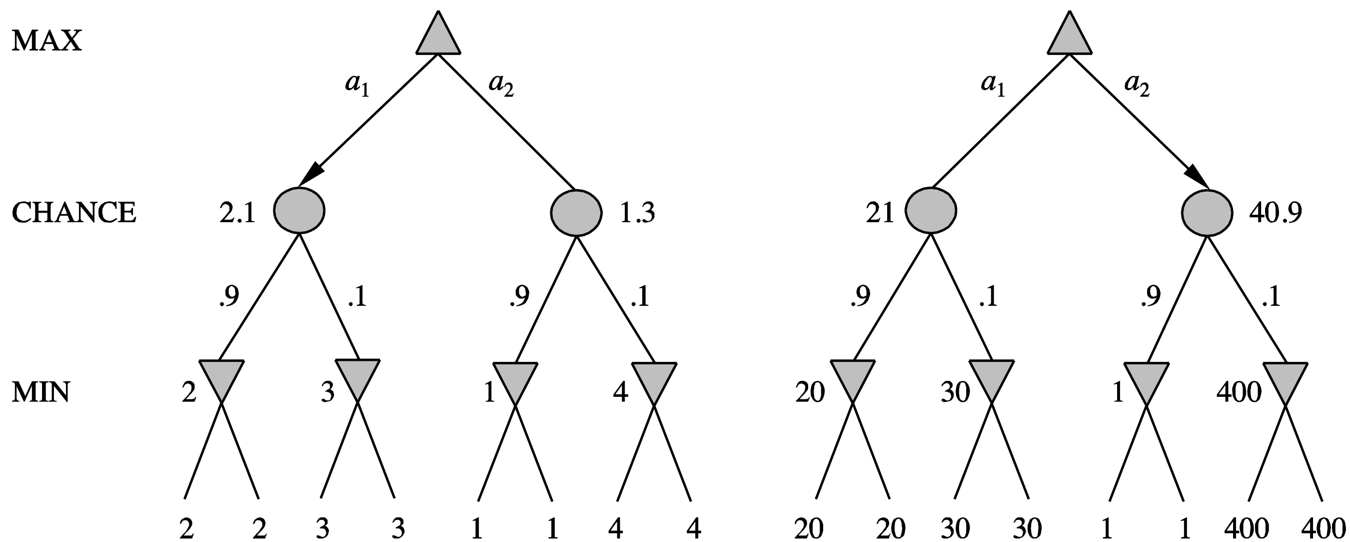
average of the values weighted by their probabilities

Stochastic games

$$EXPECT\_MINIMAX(s) = \begin{cases} UTILITY(s, MAX) & if\ TERMINAL\_TEST(s) \\ max_{a \in ACTIONS(s)} EXPECT\_MINIMAX(RESULT(s, a)) & PLAYER(s) = \text{MAX} \\ min_{a \in ACTIONS(s)} EXPECT\_MINIMAX(RESULT(s, a)) & PLAYER(s) = \text{MIN} \\ \sum_r P(r)\, EXPECT\_MINIMAX(RESULT(s, r)) & PLAYER(s) = \text{CHANCE} \end{cases}$$

Evaluation functions for stochastic games

- An order preserving transformation on leaf values is not a sufficient condition.



- Evaluation function must be a positive linear transformation of the expected utility of a position.

Properties of search space for stochastic games

- $O(b^m n^m)$
  - Backgammon: $b \approx 20$ (can be up to 4000 for double dice rolls), $n = 21$ (no. of different possible dice rolls)
    - 3-plies is manageable ($\approx 10^8$ nodes)

- Probability of reaching a given node decreases enormously by increasing the depth (multiplying probabilities)
  - Forming detailed plans of actions may be pointless
    - Limiting depth is not such damaging particularly when the probability values (for each non-deterministic situation) are close to each other
  - But pruning is not straightforward.

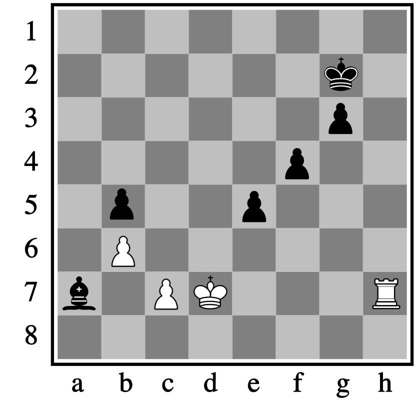
Search algorithms for stochastic games

- Advanced alpha-beta pruning
  - Pruning MIN and MAX nodes as alpha-beta
  - Pruning chance nodes (by putting bounds on the utility values and so placing an upper bound on the value of a chance node)

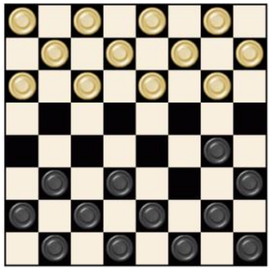
- Monte Carlo simulation to evaluate a position
  - Starting from the corresponding position, the algorithm plays thousands of games against itself using random dice rolls.
    - Win percentage as the approximated value of the position (Backgammon)

State-of-the-art game programs

- Chess ($b \approx 35$)
  - In 1997, Deep Blue defeated Kasparov.
    - ran on a parallel computer doing alpha-beta search.
    - reaches depth 14 plies routinely.
      - techniques to extend the effective search depth
  - Hydra: Reaches depth 18 plies using more heuristics.

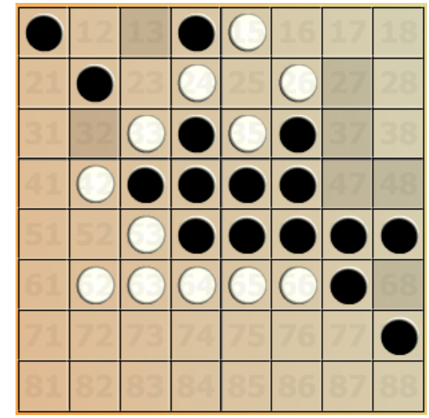


- Checkers ($b < 10$)
  - Chinook (ran on a regular PC and uses alpha-beta search) ended 40-year-reign of human world champion Tinsley in 1994.
  - Since 2007, Chinook has been able to play perfectly by using alpha-beta search combined with a database of 39 trillion endgame positions.

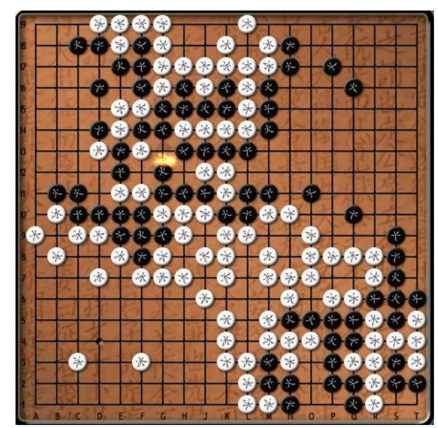


State-of-the-art game programs (Cont.)

- Othello (b is usually between 5 to 15)
  - Logistello defeated the human world champion by six games to none in 1997.
  - Human champions are no match for computers at Othello.

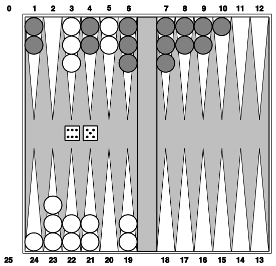


- Go ($b > 300$)
  - Human champions refuse to compete against computers (current programs are still at advanced amateur level).
  - MOGO avoided alpha-beta search and used Monte Carlo rollouts.

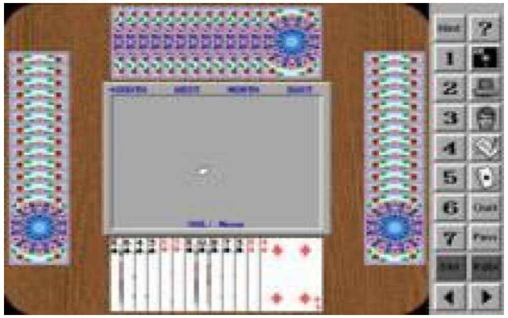


State-of-the-art game programs (Cont.)

- Backgammon (stochastic)
  - TD-Gammon (1992) was competitive with top human players.
    - Depth 2 or 3 search along with a good evaluation function developed by learning methods



- Bridge (partially observable, multiplayer)
  - In 1998, GIB was 12th in a filed of 35 in the par contest at human world championship.
  - In 2005, Jack defeated three out of seven top champions pairs. Overall, it lost by a small margin.



- Scrabble (partially observable & stochastic)
  - In 2006, Quackle defeated the former world champion 3-2.