Course 440 : Introduction To Artificial Intelligence
Lecture 5

Solving Problems by Searching: Adversarial Search

Abdeslam Boularias

Friday, October 7, 2016

RUTGERS
THE STATE UNIVERSITY
OF NEW JERSEY

Outline

We examine the problems that arise when we make decisions in a world where other agents are also acting, possibly against us.

1. The minimax algorithm
2. Alpha-beta pruning
3. Imperfect fast decisions
4. Stochastic games
5. Partially observable games

Games

- **Multiagent environments** are environments where more than one agent is acting, simultaneously or at different times.
- **Contingency plans** are necessary to account for the unpredictability of other agents.
- Each agent has its own personal **utility function**.
- The corresponding **decision-making** problem is called a **game**.
- A game is **competitive** if the utilities of different agents are maximized in different states.
- In **zero-sum games**, the sum of the utilities of all agents is **constant**.
- Zero-sum games are purely competitive.

Games

- The abstract nature of games, such as chess, makes them appealing to study in AI.
- The state of a game is easy to represent and agents typically have a small number of actions to choose from.

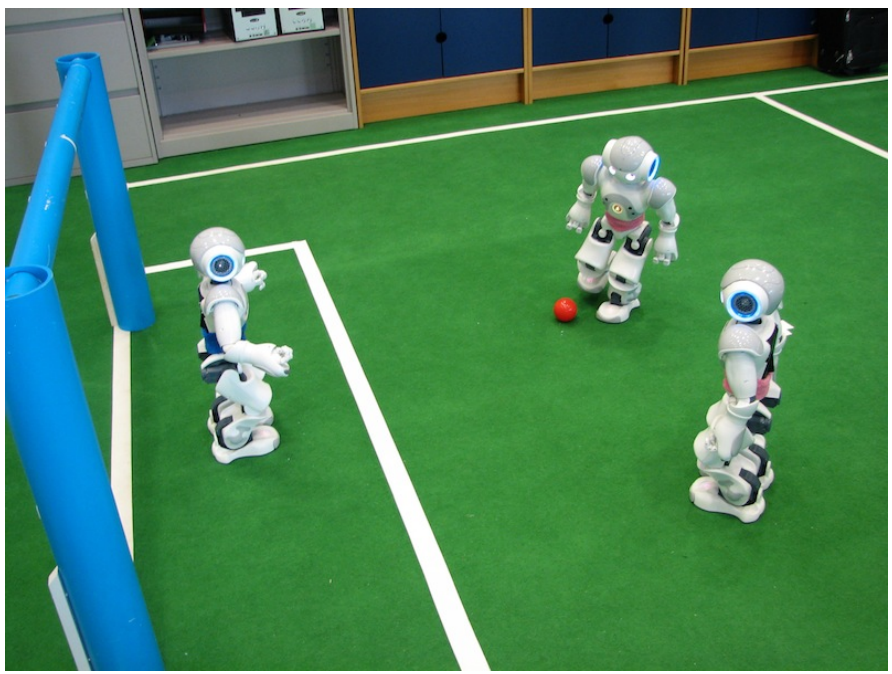
(a)

(b)

Left : Computer chess pioneers Herbert Simon and Allen Newell (1958). Right : John McCarthy and the Kotok-McCarthy program on an IBM 7090 (1967)

Games

- The abstract nature of games, such as chess, makes them appealing to study in AI.
- The state of a game is easy to represent and agents typically have a small number of actions to choose from.
- Physical games, such as soccer, are more difficult to study due to their continuous state and action spaces.



Robot soccer
(from ri.cmu.edu)

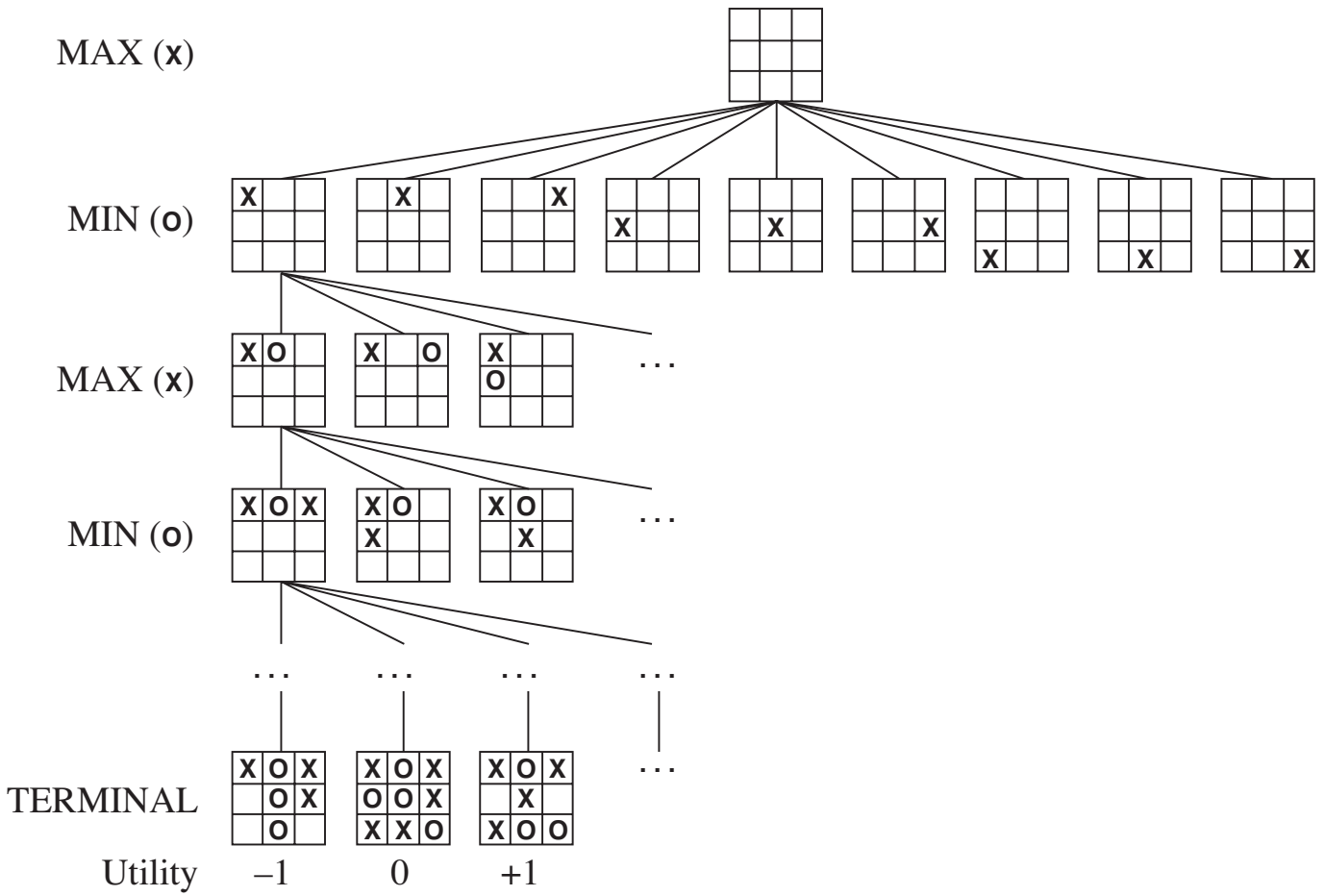
A game is described by :

- $\mathbf{S_0}$: the initial state (how is the game set up at the start ?).
- **PLAYER(s)** : Indicates which player has the move in state s (whose turn is it ?).
- **ACTIONS(s)** : Set of legal actions in state s.
- **RESULT(s, a)** : returns the next state after we play action a in state s.
- **TERMINAL-TEST(s)** : Indicates if s is a terminal state.
- **UTILITY(s, p)** : (also called objective or payoff function) defines a numerical value for a game that ends in terminal state s for player p

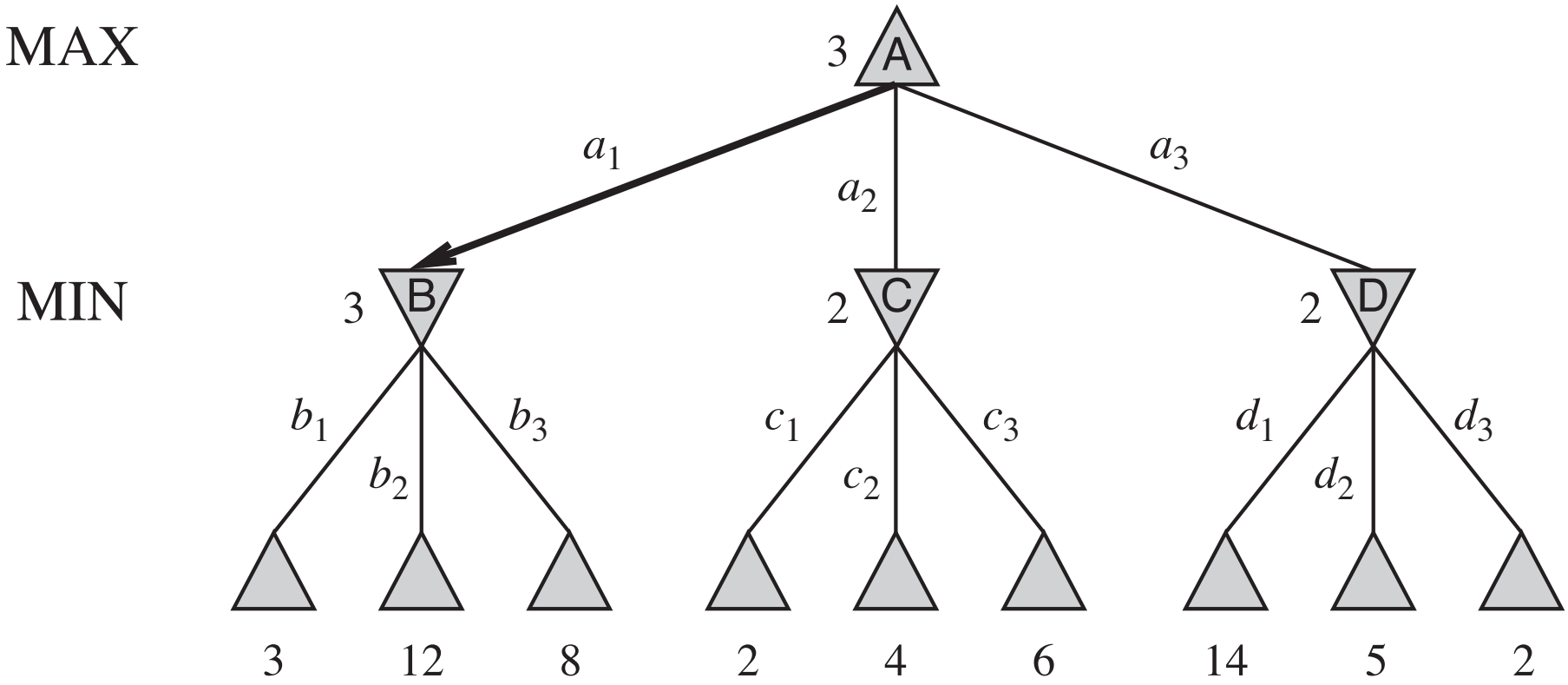
Example : tic-tac-toe

We suppose there are two players in a zero-sum game

- We call our player **MAX**, she tries to maximize our utility.
- We call our opponent **MIN**, she tries to minimize our utility (i.e. maximize her utility).



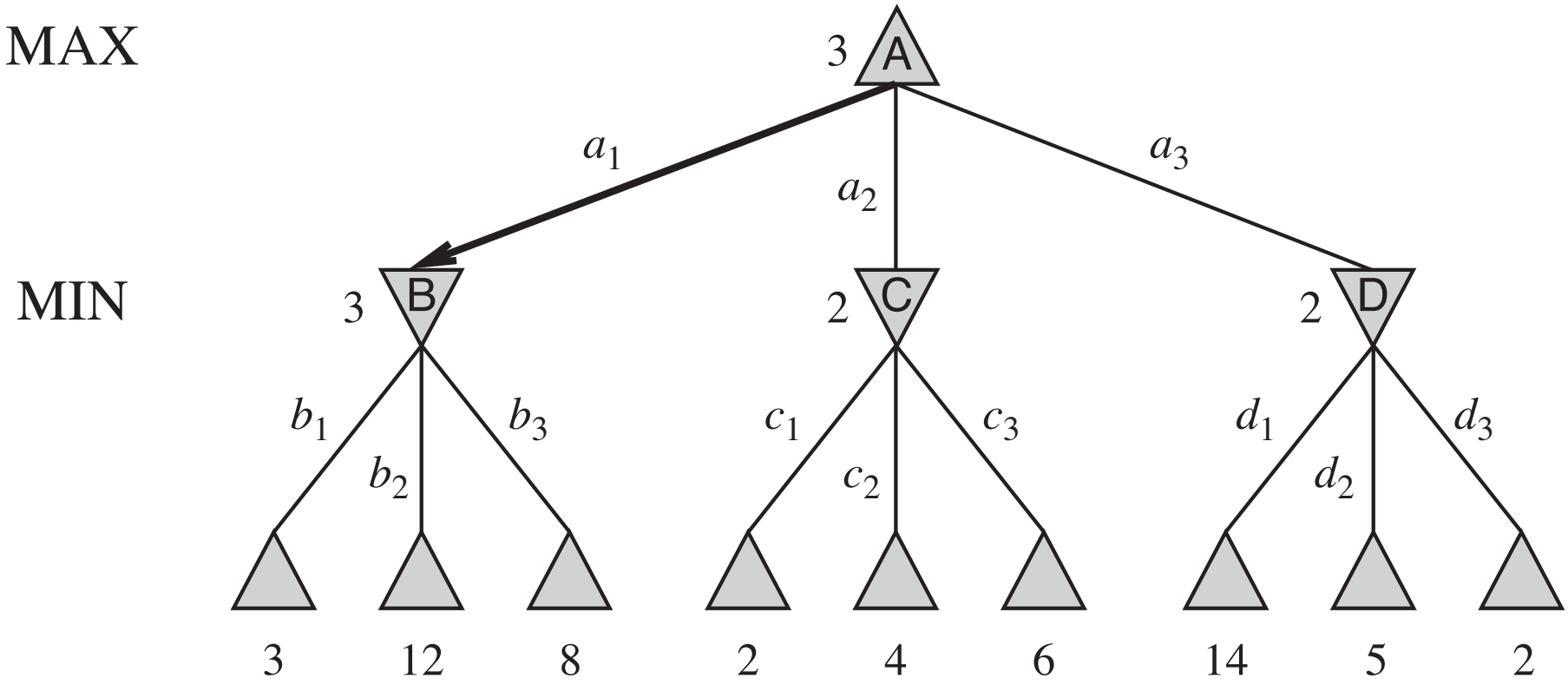
Optimal games



- $\triangle$ Indicates states where MAX should play.
- ∇ Indicates states where MIN should play.

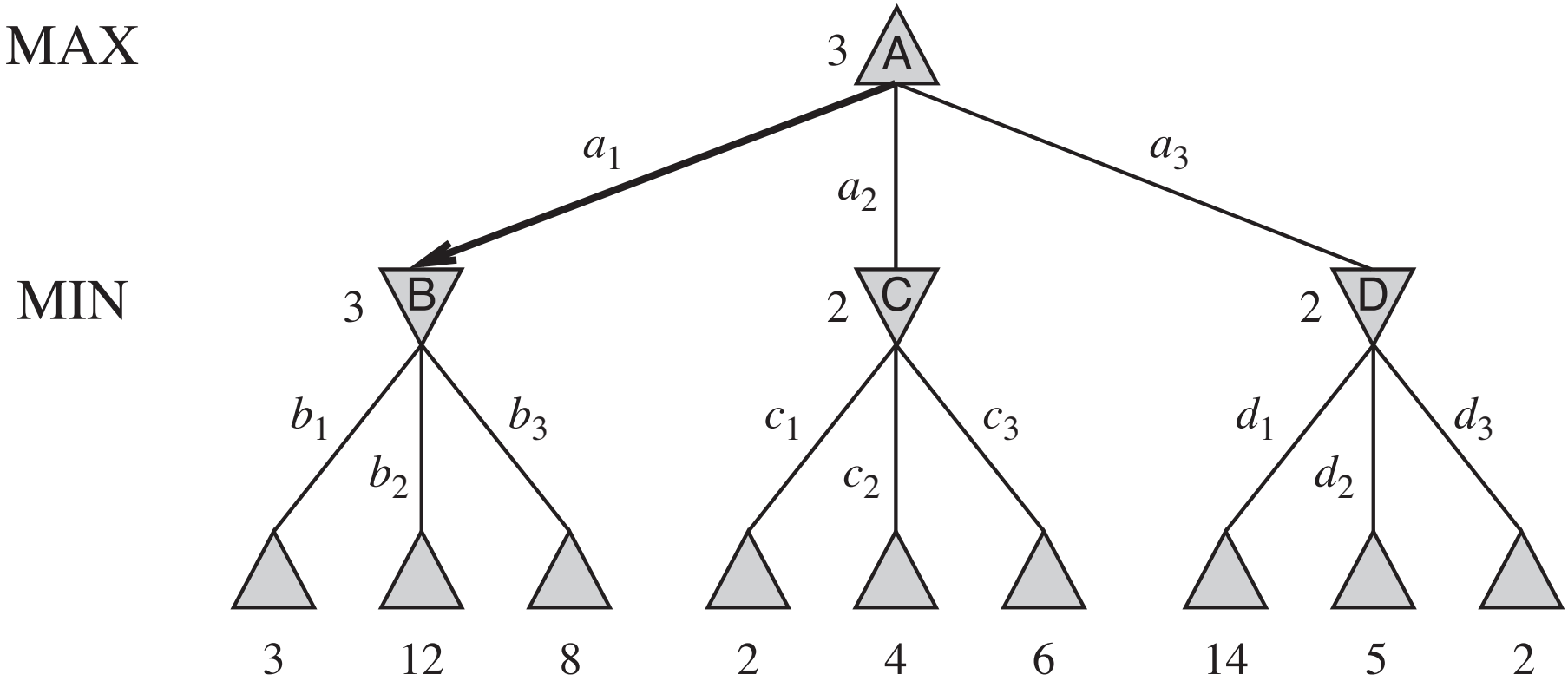
Which action among $\{a_1, a_2, a_3\}$ should MAX play ?

Minimax strategy



- We don't take any risk, we assume that MIN will play optimally.
- We look for the best action for the worst possible scenario.
- What if our opponent is not optimal ? Can we learn the opponent's behaviour ?
- What if our player is trying to fool us by behaving in a certain way ?

Minimax strategy



$$\text{MINIMAX}(s) = \begin{cases} \text{UTILITY}(s) & \text{if TERMINAL-TEST}(s) \\ \max_{a \in Actions(s)} \text{MINIMAX}(\text{RESULT}(s,a)) & \text{if PLAYER}(s) = MAX, \\ \min_{a \in Actions(s)} \text{MINIMAX}(\text{RESULT}(s,a)) & \text{if PLAYER}(s) = MIN. \end{cases}$$

Minimax algorithm

function MINIMAX-DECISION($state$) **returns** *an action*
  return $\arg\max_{a \in \text{ACTIONS}(s)}$ MIN-VALUE(RESULT($state, a$))

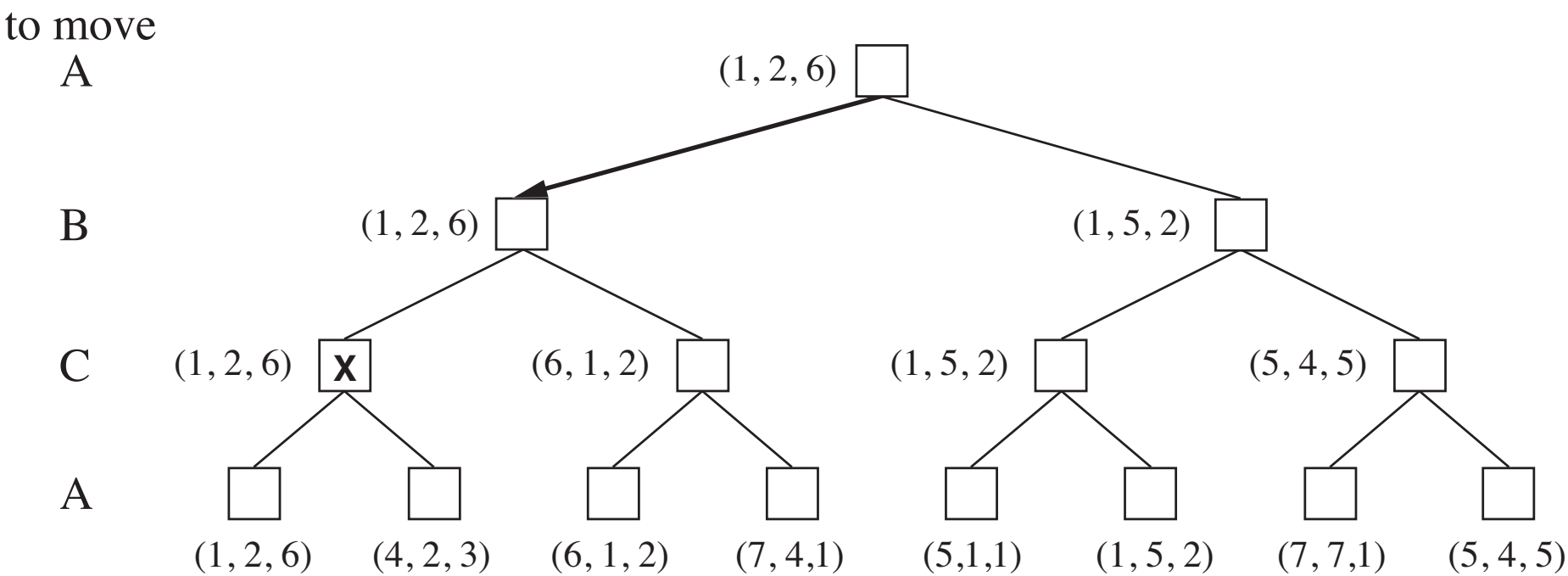
---

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

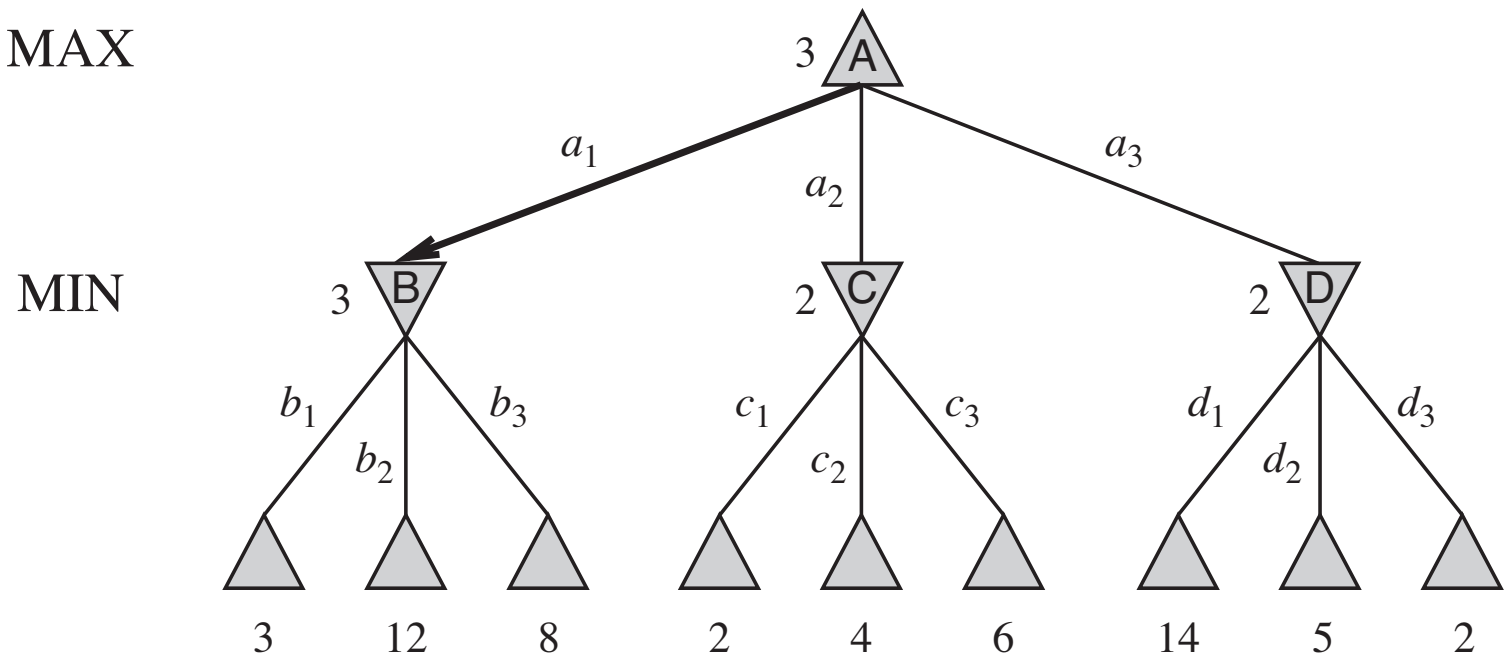
Minimax strategy in multiplayer games



- We have three players A, B, and C.
- The utilities are represented by a 3-dimensional vector (v_A, v_B, v_C).
- We apply the same principle : assume that every player is optimal.
- If the game is not zero-sum, implicit collaborations may occur.

Alpha-Beta pruning

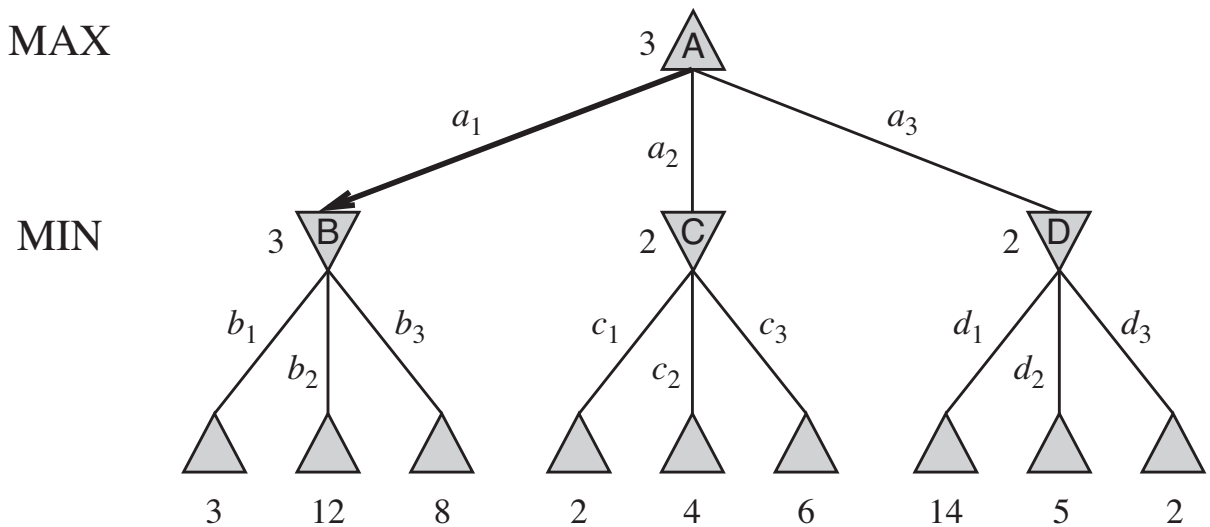
- Time is a major issue in game search trees. Searching the complete tree takes $O(b^m)$ operations, where b is the branching factor and m is the depth of the tree (the horizon).
- Do we really need to parse the whole tree to find a minimax strategy ?



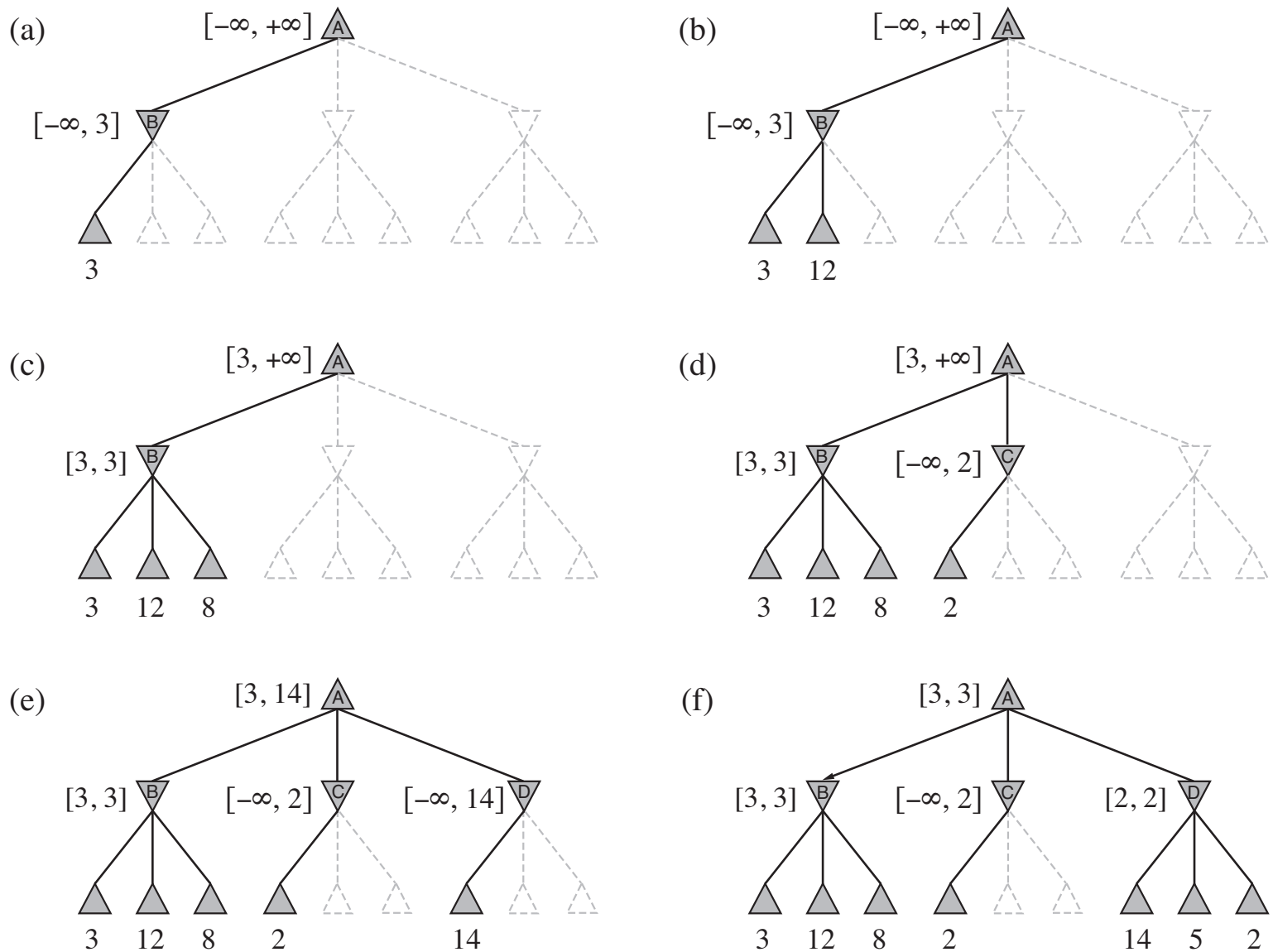
Example

- Assume that the search tree has been parsed except for actions c_2 and c_3.
- Let us denote the utilities of c_2 and c_3 by x and y respectively.

$$\begin{aligned}
\text{MINIMAX(root)} &= \max(\min(3,12,8), \min(2,x,y), \min(14,5,2)) \\
&= \max(3, \min(2,x,y), 2) \\
&= \max(3,z,2) \text{ where } z = \min(2,x,y) \leq 2 \\
&= 3.
\end{aligned}$$



Alpha-Beta pruning



Imperfect fast decisions

- The minimax algorithm generates the entire search tree.
- The alpha-beta algorithm allows us to prune large parts of the search tree, but its complexity is still exponential in the branching factor (number of actions).
- This is still non-practical because moves should be made very quickly.

Cutting-off the search

- A cutoff test is used to decide when to stop looking further.
- A heuristic **evaluation function** is used to estimate the utility where the search is cut off.

$$\text{H-MINIMAX}(s,d) = \begin{cases} \text{EVAL}(s) & \text{if CUTOFF-TEST}(s,d) = true, \\ \max_{a \in Actions(s)} \text{H-MINIMAX}(\text{RESULT}(s,a), d+1) & \text{if PLAYER}(s) = MAX, \\ \min_{a \in Actions(s)} \text{H-MINIMAX}(\text{RESULT}(s,a), d+1) & \text{if PLAYER}(s) = MIN. \end{cases}$$

Evaluation functions

- Human chess players have ways of judging the value of a position without imagining all the moves ahead until a check-mate.
- A good evaluation function should order the actions correctly according to their true utilities.
- Evaluation functions should be computed very quickly.

Example : evaluation functions in chess

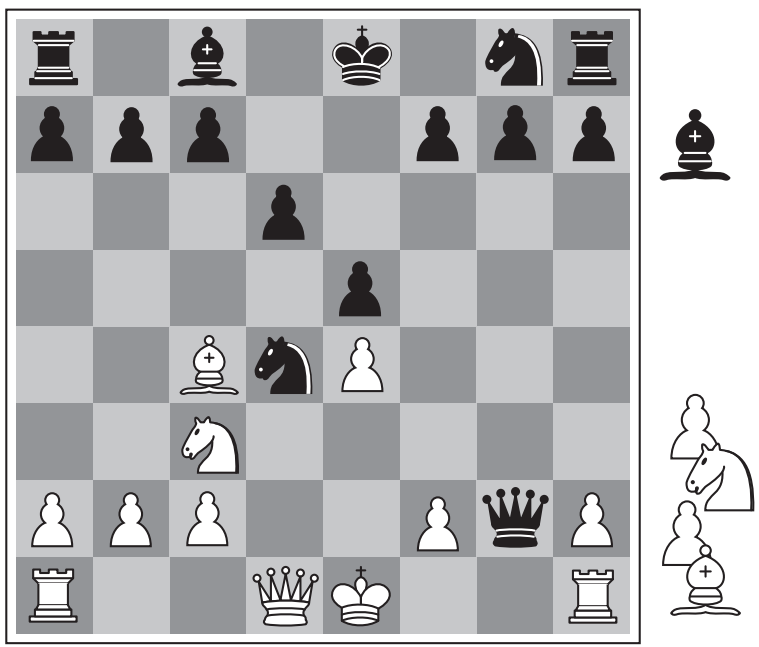
- Evaluation functions use **features** of given positions in the game.
- Example : number of pawns in the given position. If we know from experience that 72% of positions "two pawns vs one pawn" lead to a win (utility $+1$) ; 20% to a loss (0), and 8% to a draw (1/2), then the expected value of these positions is 0.76.
- Other functions such as the advantage in each piece, "good pawn structure", and "king safety" can be used as features f_i.
- The evaluation function can be given using a weighted linear model :

$$\text{EVAL}(s) = \sum_{i=1}^{n} w_i f_i(s),$$

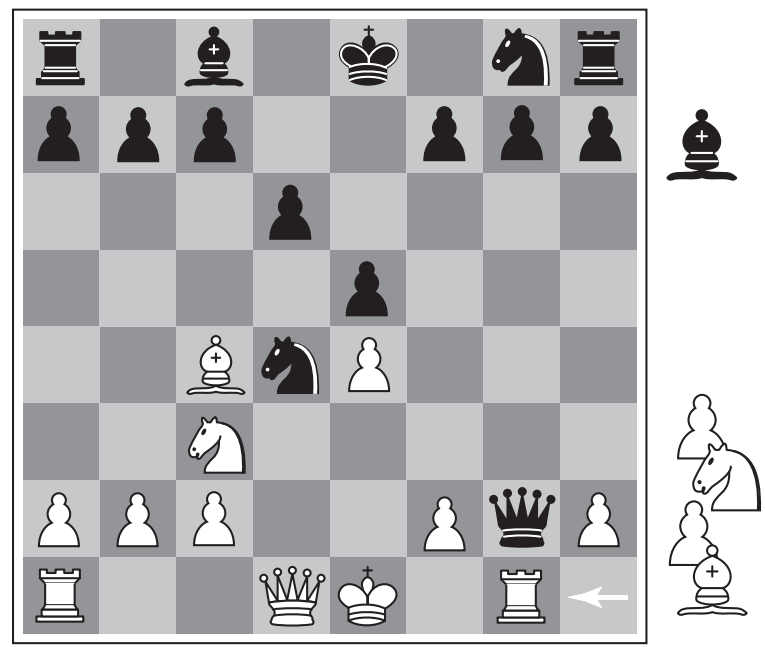
where w_i is the importance of feature f_i.

Example : evaluation functions in chess

- Linear models assume that the features are independent, which is not always true (bishops are more efficient at endgame).
- Values of some features do not increase linearly (two knights are way more useful than one knight).



(a) White to move

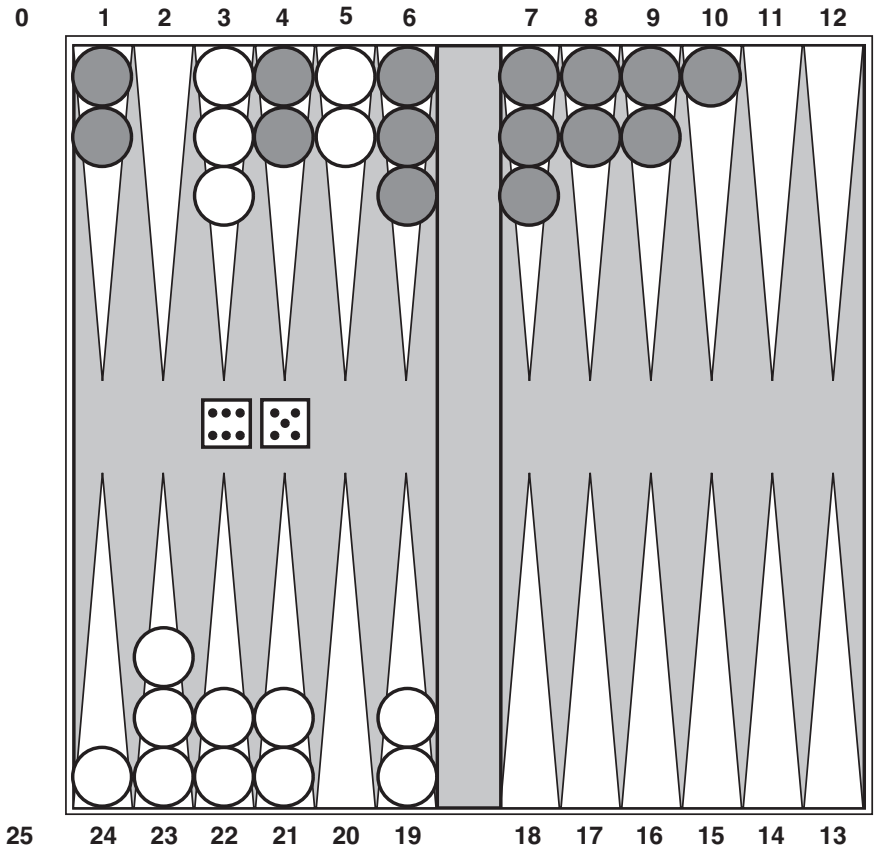


(b) White to move

In the two positions, the two players have the same number of pieces. The position on the right is much worse than the one on the left for Black. What if the search cutoff happens in the left ?

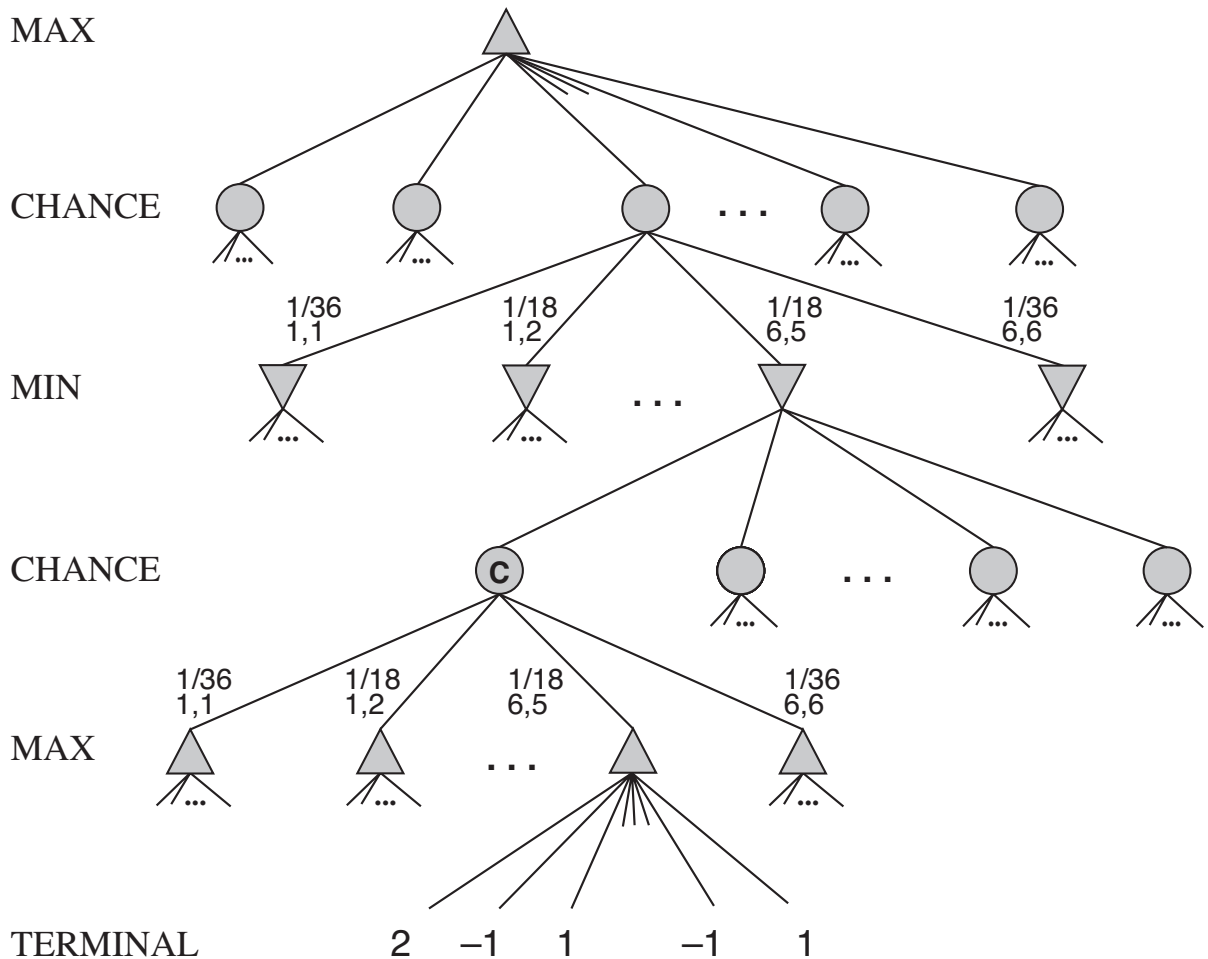
Stochastic games

- In some games, the state of the game changes randomly depending on the selected actions
- Backgammon is a typical game that combines luck and skill.



Stochastic games

We can use the same minimax strategy, but we need to take the randomness into account by computing the expected utilities.



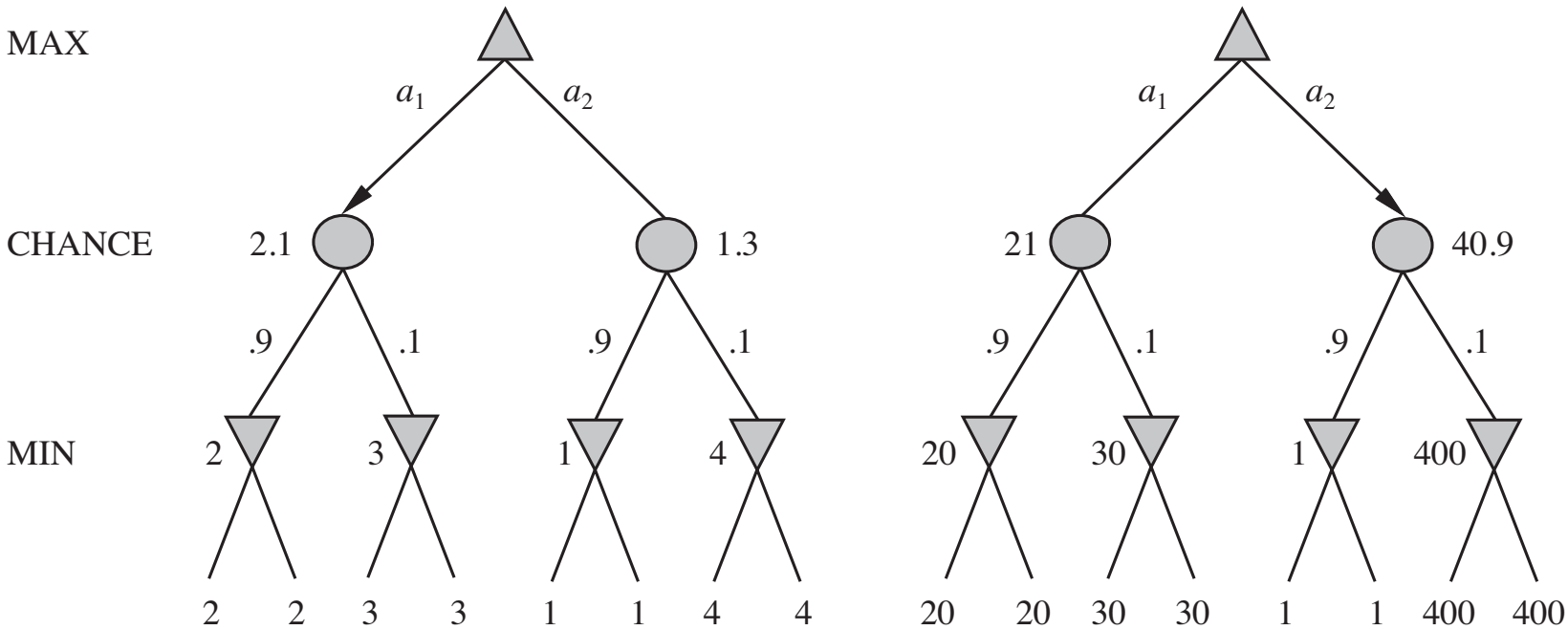
Stochastic games

We can use the same minimax strategy, but we need to take the randomness into account by computing the expected utilities.

$$\text{EXPECMINIMAX}(s) = \begin{cases} \text{UTILITY}(s) & \text{if TERMINAL-TEST}(s) = true, \\ \max_{a \in Actions(s)} \text{EXPECMINIMAX}(\text{RESULT}(s, a)) & \text{if PLAYER}(s) = MAX, \\ \min_{a \in Actions(s)} \text{EXPECMINIMAX}(\text{RESULT}(s, a)) & \text{if PLAYER}(s) = MIN \\ \sum_r P(r)\text{EXPECMINIMAX}(\text{RESULT}(s, r)) & \text{if PLAYER}(s) = CHANCE, \end{cases}$$

where r is a chance event (e.g, dice roll).

Trouble with evaluation functions in stochastic games



Partially observable

- In some games, the state of the game is not fully known.
- Cards and Kriegspiel are examples of such games.
- The state of the game can be tracked by remembering past actions and observations.