472 Recitation

Week 5

Sensorless Problem

Solution: a sequence of actions from search in the space B of belief states (b-states).

States: 2^N possible subsets of the set Σ of N physical states. $C_n^0 + C_n^1 + C_n^2 + \ldots + C_n^n = 2^n$

Initial state: S.

Actions: At the belief state b, $\text{ACTIONS}(b) = \bigcup_{s \in b} \underbrace{\text{ACTIONS}_P(s)}_{\text{Action on a physical state}}$

Goal Test: possibly if one of the states $s \in b$ passes the test;
necessarily if every state $s \in b$ passes the test.

Cost: Could be one of several values if the same action as different costs in different states.

- Use of compact description
- Incremental belief-state search to build up solution one physical state at a time

Partially Observable Environments

Transition Model:

- Prediction: computes the b-state from an action a.

estimate

$$\hat{b} = \text{PREDICT}(b, a)$$

- Possible percepts: computes the set of percepts that could be observed in the predicted b-state.

$$\text{POSSIBLE-PERCEPTS}(\hat{b}) = \{o \mid o = \text{PERCEPTS}(s) \text{ and } s \in \hat{b}\}$$

- Update: computes the set of states in $\hat{b}$ that could have produced the percept.

Algorithm: AND-OR search algorithm

Game

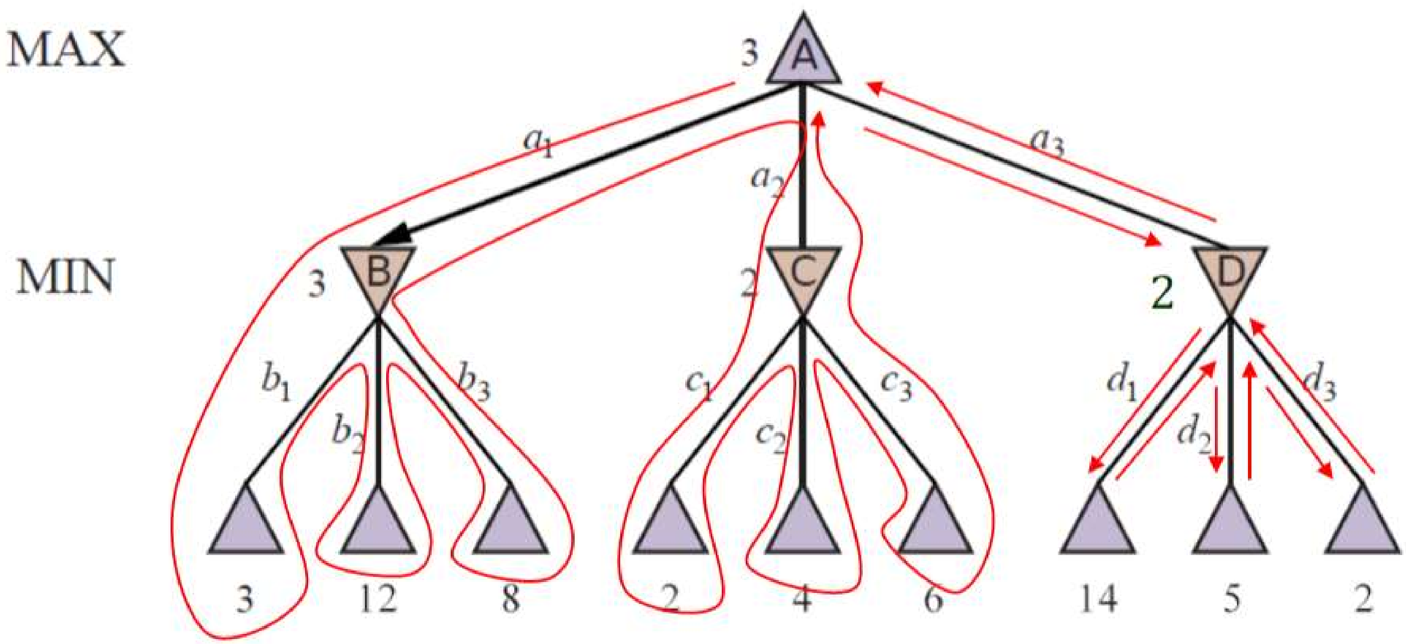
- s_0: initial state – game setup.

At a state s:

- TO-MOVE(s): the player to move in the state.
- ACTIONS(s): the set of legal moves in the state.
- RESULT(s, a): the transition model defining the next state from taking action a.
- IS-TERMINAL(s): to test if the game is over, i.e., if s is a terminal state.
- UTILITY (s, p): a utility function to return a value to the player p if the game ends in terminal state s.

Minimax

Optimal against an optimal opponent



Minimax

function MINIMAX-SEARCH(*game*, *state*) **returns** *an action*
  player ← *game*.TO-MOVE(*state*)
  value, *move* ← $\boxed{\text{MAX-VALUE}(game,\ state)}$
  return *move*

function MAX-VALUE(*game*, *state*) $\boxed{\textbf{returns}\text{ a }(utility,\ move)\text{ pair}}$
  if *game*.IS-TERMINAL(*state*) **then return** *game*.UTILITY(*state*, *player*), *null*
  $\boxed{v \leftarrow -\infty}$
  for each *a* **in** *game*.ACTIONS(*state*) **do**
    v2, *a2* ← MIN-VALUE(*game*, *game*.RESULT(*state*, *a*))
    if $\boxed{v2 > v}$ **then**
      v, *move* ← *v2*, *a*
  return *v*, *move*

function MIN-VALUE(*game*, *state*) $\boxed{\textbf{returns}\text{ a }(utility,\ move)\text{ pair}}$
  if *game*.IS-TERMINAL(*state*) **then return** *game*.UTILITY(*state*, *player*), *null*
  $\boxed{v \leftarrow +\infty}$
  for each *a* **in** *game*.ACTIONS(*state*) **do**
    v2, *a2* ← MAX-VALUE(*game*, *game*.RESULT(*state*, *a*))
    if $\boxed{v2 < v}$ **then**
      v, *move* ← *v2*, *a*
  return *v*, *move*

Minimax

Minimax Algorithm

```
function MINIMAX-DECISION(state) returns an action
  inputs: state, current state in game
  return argmax a in ACTIONS(state) MIN-VALUE(RESULT(state,a))

function MAX-VALUE(state) 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(RESULT(state,a)))
  return v

function MIN-VALUE(state) 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(RESULT(state,a)))
  return v
```

13

Criteria	Performance
Completeness	Yes (if tree is finite)
Optimality	Yes
Time Complexity	$O(b^m)$
Space Complexity	$O(bm)$

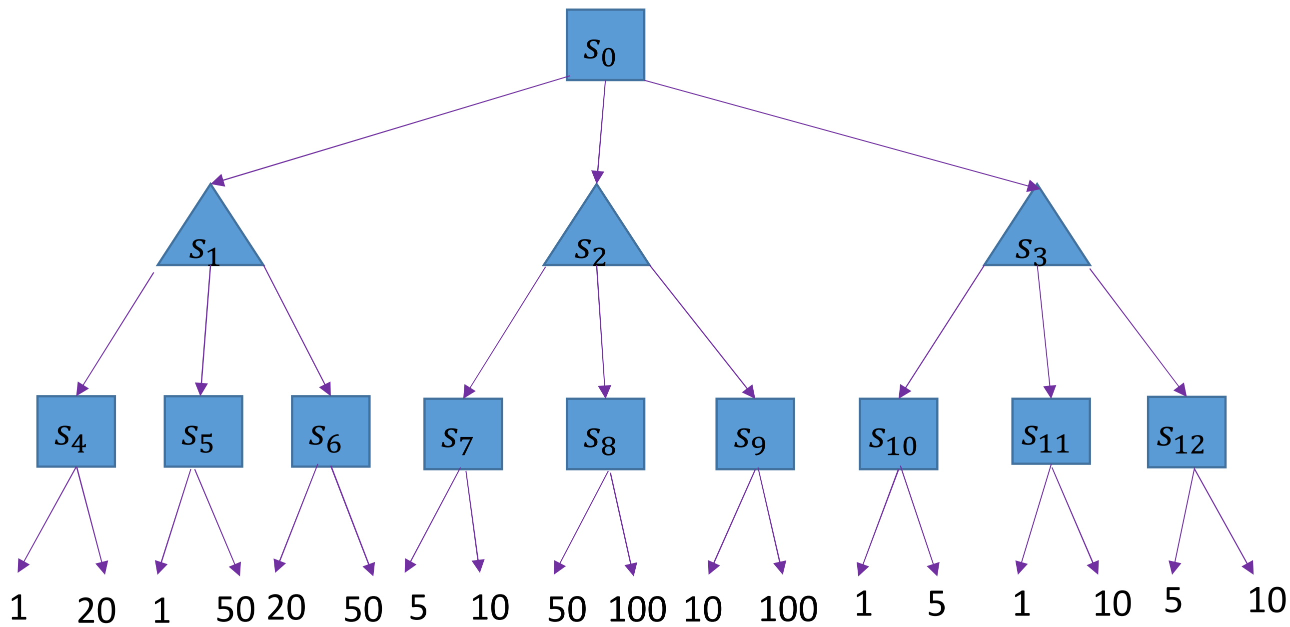
Alpha-beta pruning

The α-β algorithm

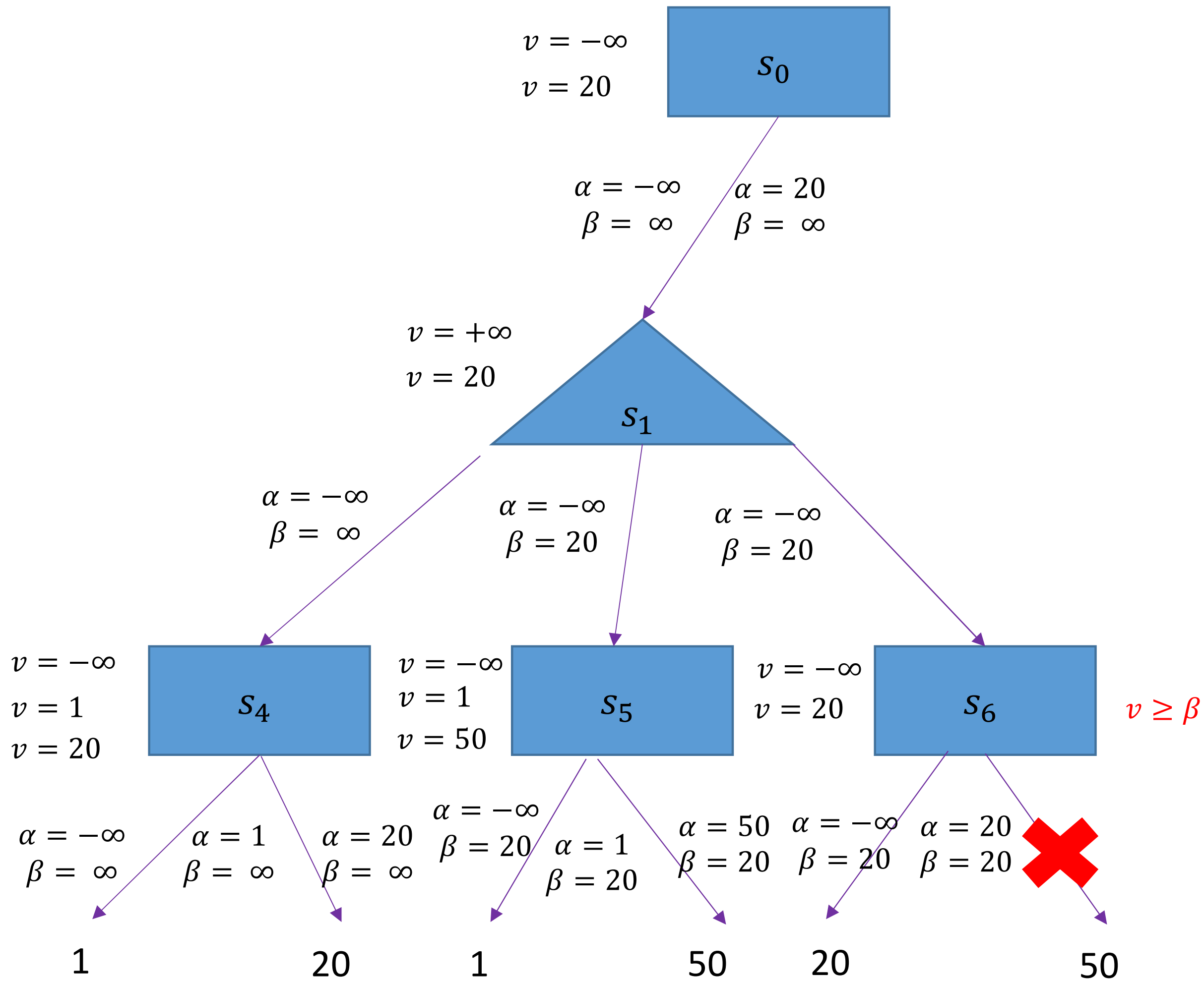
- Improve the efficiency of Minimax search

- Alpha is the value of the choice for MAX
- Beta is the value of the choice for MIN

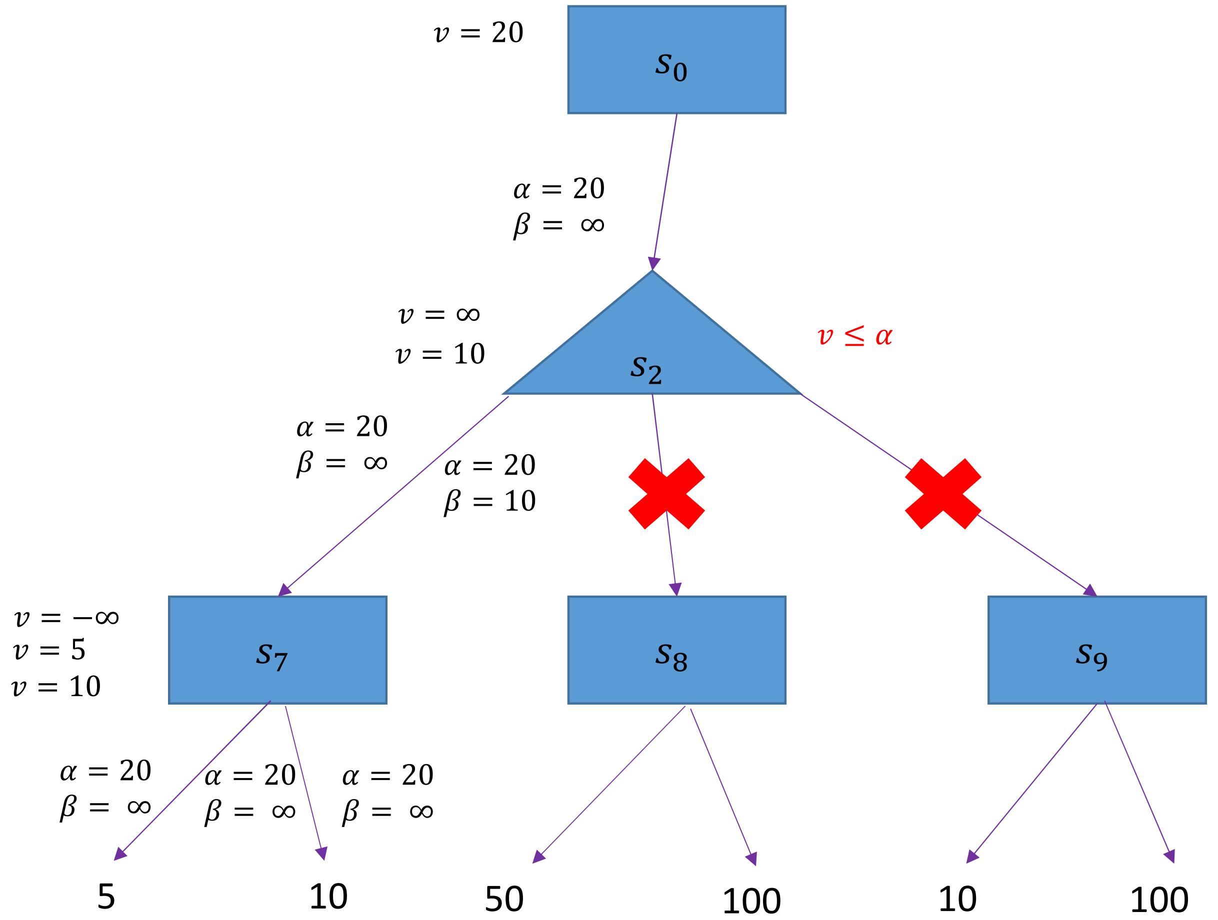
```
function ALPHA-BETA-SEARCH(state) returns an action
  inputs: state, current state in game
  v←MAX-VALUE(state, -∞ , +∞)
  return the action in ACTIONS(state) with value v

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(Result(s,a), α, β))
    if v ≥ β then return v                        Beta pruning
    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(Result(s,a), α, β))
    if v ≤ α then return v                        Alpha pruning
    β ← MIN(β ,v)
  return v
```



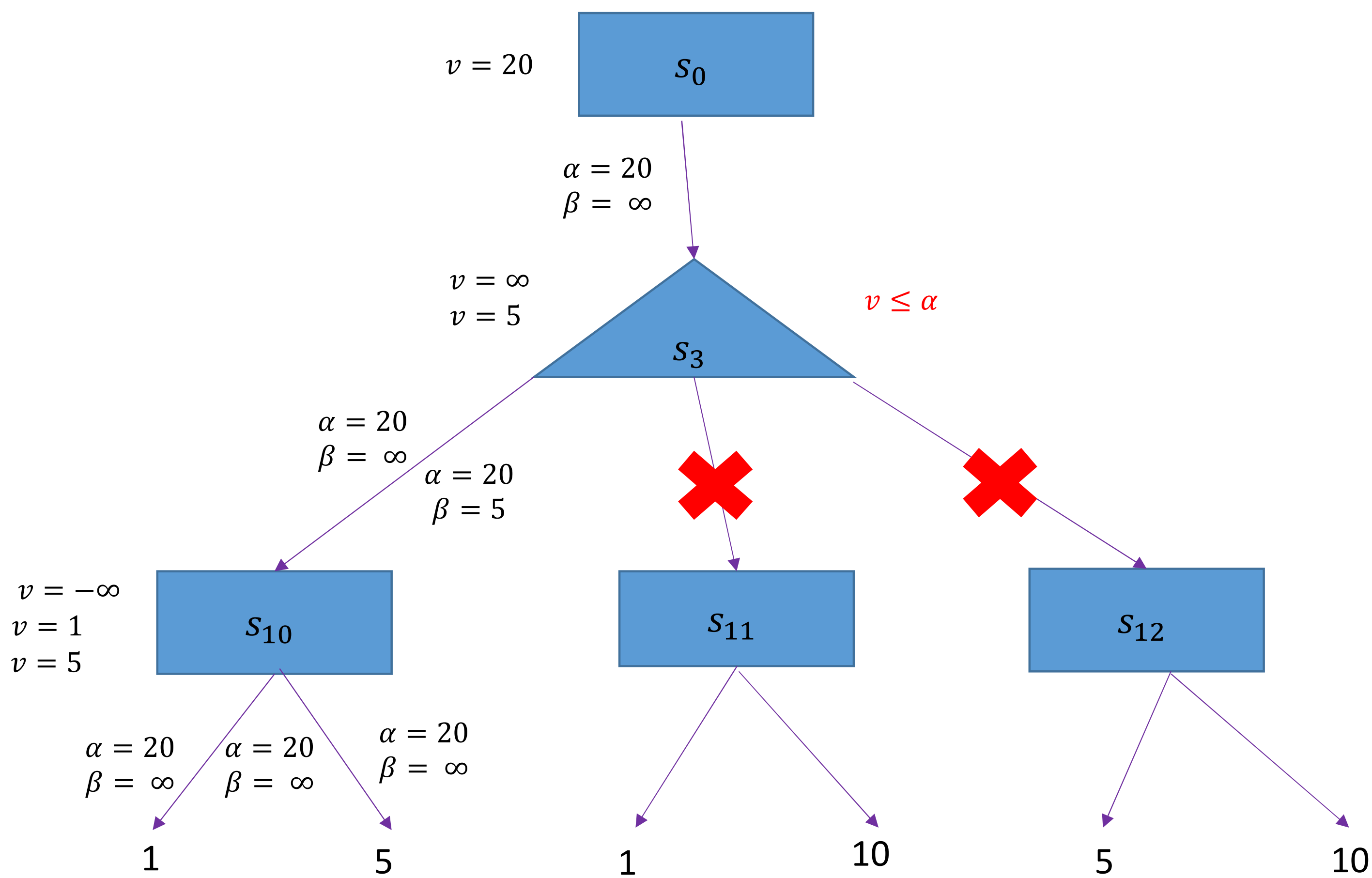
s_0
s_1
s_2
s_3
s_4
s_5
s_6
s_7
s_8
s_9
s_{10}
s_{11}
s_{12}
1
20
1
50
20
50
5
10
50
100
10
100
1
5
1
10
5
10



$v = -\infty$
$v = 20$
s_0
$\alpha = -\infty$
$\beta = \infty$
$\alpha = 20$
$\beta = \infty$
$v = +\infty$
$v = 20$
s_1
$\alpha = -\infty$
$\beta = \infty$
$\alpha = -\infty$
$\beta = 20$
$\alpha = -\infty$
$\beta = 20$
$v = -\infty$
$v = 1$
$v = 20$
s_4
$v = -\infty$
$v = 1$
$v = 50$
s_5
$v = -\infty$
$v = 20$
s_6
$v \geq \beta$
$\alpha = -\infty$
$\beta = \infty$
$\alpha = 1$
$\beta = \infty$
$\alpha = 20$
$\beta = \infty$
$\alpha = -\infty$
$\beta = 20$
$\alpha = 1$
$\beta = 20$
$\alpha = 50$
$\beta = 20$
$\alpha = -\infty$
$\beta = 20$
$\alpha = 20$
$\beta = 20$
1
20
1
50
20
50



$v = 20$
s_0
$\alpha = 20$
$\beta = \infty$
$v = \infty$
$v = 10$
s_2
$v \leq \alpha$
$\alpha = 20$
$\beta = \infty$
$\alpha = 20$
$\beta = 10$
$v = -\infty$
$v = 5$
$v = 10$
s_7
s_8
s_9
$\alpha = 20$
$\beta = \infty$
$\alpha = 20$
$\beta = \infty$
$\alpha = 20$
$\beta = \infty$
5
10
50
100
10
100



$v = 20$
s_0
$\alpha = 20$
$\beta = \infty$
$v = \infty$
$v = 5$
s_3
$v \leq \alpha$
$\alpha = 20$
$\beta = \infty$
$\alpha = 20$
$\beta = 5$
$v = -\infty$
$v = 1$
$v = 5$
s_{10}
s_{11}
s_{12}
$\alpha = 20$
$\beta = \infty$
$\alpha = 20$
$\beta = \infty$
$\alpha = 20$
$\beta = \infty$
1
5
1
10
5
10