IS-ZC444: Artificial Intelligence

Lecture-07: Beyond Classical Search

Dr. Kamlesh Tiwari
Assistant Professor
Department of Computer Science and Information Systems,
BITS Pilani, Pilani, Jhunjhunu-333031, Rajasthan, INDIA

September 19, 2018 (WILP @ BITS-Pilani Jul-Nov 2018)

Local Search in Continuous State

Issue: Number of next states (branching factor) becomes infinite

Local Search in Continuous State

Issue: Number of next states (branching factor) becomes infinite

Example: Induct three new airports in Romania

- Let at (x_1, y_1), (x_2, y_2) and (x_3, y_3) on the map
- Minimize sum of distances of all the cities from its nearest airport

$$f(x_1, y_1, x_2, y_2, x_3, y_3) = \sum_{i=1}^{3} \sum_{c \in C_i} ((x_i - x_c)^2 + (y_i - y_c)^2)$$

Local Search in Continuous State

Issue: Number of next states (branching factor) becomes infinite

Example: Induct three new airports in Romania

- Let at (x_1, y_1), (x_2, y_2) and (x_3, y_3) on the map
- Minimize sum of distances of all the cities from its nearest airport

$$f(x_1, y_1, x_2, y_2, x_3, y_3) = \sum_{i=1}^{3} \sum_{c \in C_i} ((x_i - x_c)^2 + (y_i - y_c)^2)$$

- If you discretize the neighborhood then there are only 12 next state (move only $\pm\delta$ in one step). One can apply hill climbing.

Local Search in Continuous State

Issue: Number of next states (branching factor) becomes infinite

Example: Induct three new airports in Romania

- Let at (x_1, y_1), (x_2, y_2) and (x_3, y_3) on the map
- Minimize sum of distances of all the cities from its nearest airport

$$f(x_1, y_1, x_2, y_2, x_3, y_3) = \sum_{i=1}^{3} \sum_{c \in C_i} ((x_i - x_c)^2 + (y_i - y_c)^2)$$

- If you discretize the neighborhood then there are only 12 next state (move only $\pm\delta$ in one step). One can apply hill climbing.
- If you attempt to use gradient $\bigtriangledown f = (\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial y_1}, \frac{\partial f}{\partial x_2}, \frac{\partial f}{\partial y_2}, \frac{\partial f}{\partial x_3}, \frac{\partial f}{\partial y_3})$ it cannot be solved as globally finding $\bigtriangledown f$ is not possible.

Local Search in Continuous State

Issue: Number of next states (branching factor) becomes infinite

Example: Induct three new airports in Romania

- Let at (x_1, y_1), (x_2, y_2) and (x_3, y_3) on the map
- Minimize sum of distances of all the cities from its nearest airport

$$f(x_1, y_1, x_2, y_2, x_3, y_3) = \sum_{i=1}^{3} \sum_{c \in C_i} ((x_i - x_c)^2 + (y_i - y_c)^2)$$

- If you discretize the neighborhood then there are only 12 next state (move only $\pm\delta$ in one step). One can apply hill climbing.
- If you attempt to use gradient $\bigtriangledown f = (\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial y_1}, \frac{\partial f}{\partial x_2}, \frac{\partial f}{\partial y_2}, \frac{\partial f}{\partial x_3}, \frac{\partial f}{\partial y_3})$ it cannot be solved as globally finding $\bigtriangledown f$ is not possible.
- Given locally correct values of $\frac{\partial f}{\partial x_1} = 2 \sum_{c \in C_1} (x_i - x_c)$ one can perform steepest-ascent using $x \leftarrow x + \alpha \bigtriangledown f$

Search with Non-Deterministic Actions

Non-Deterministic: not sure what would be the next state[1]

[1] Percepts would tell where have we reached.

Search with Non-Deterministic Actions

Non-Deterministic: not sure what would be the next state[1]

Consider erratic vacuum world

sometime 1) also cleans neighboring room 2) deposit dirt

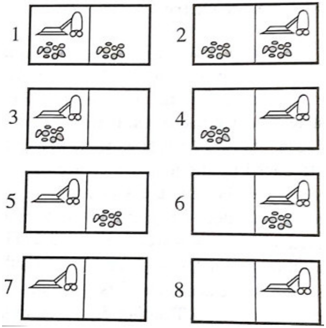
[1] Percepts would tell where have we reached.

Search with Non-Deterministic Actions

Non-Deterministic: not sure what would be the next state[1]

Consider erratic vacuum world

sometime 1) also cleans neighboring room 2) deposit dirt



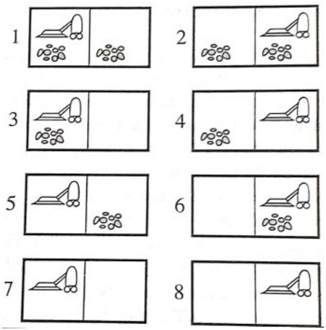
[1] Percepts would tell where have we reached.

Search with Non-Deterministic Actions

Non-Deterministic: not sure what would be the next state[1]

Consider erratic vacuum world

sometime 1) also cleans neighboring room 2) deposit dirt



- Transition would lead use to more than one state
- *suck* in 1, would lead $\{5,7\}$

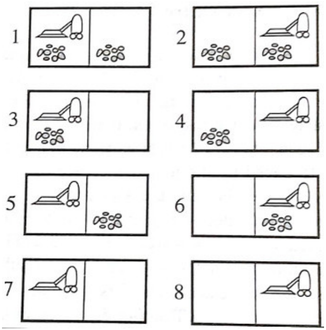
[1] Percepts would tell where have we reached.

Search with Non-Deterministic Actions

Non-Deterministic: not sure what would be the next state[1]

Consider erratic vacuum world

sometime 1) also cleans neighboring room 2) deposit dirt



- Transition would lead use to more than one state
- *suck* in 1, would lead $\{5,7\}$
- Solution would have nested if-else

[*suck*, **if** state=5 **then** [*right*,*suck* **else** []]

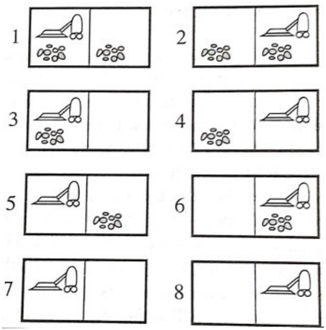
[1] Percepts would tell where have we reached.

Search with Non-Deterministic Actions

Non-Deterministic: not sure what would be the next state[1]

Consider erratic vacuum world

sometime 1) also cleans neighboring room 2) deposit dirt



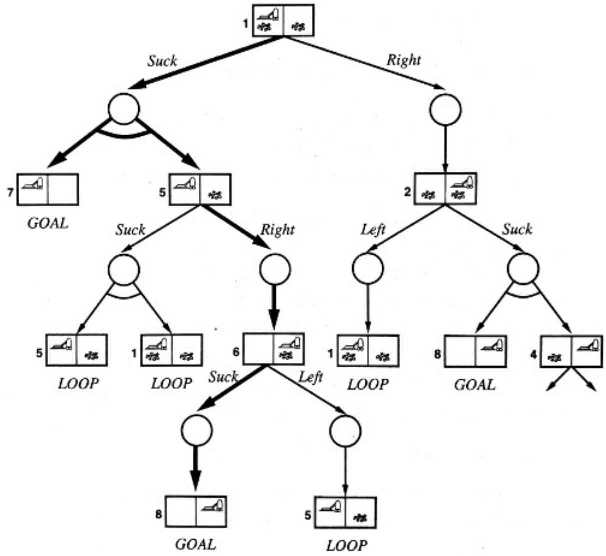
- Transition would lead use to more than one state
- *suck* in 1, would lead $\{5,7\}$
- Solution would have nested if-else

[*suck*, **if** state=5 **then** [*right,suck* **else** []]

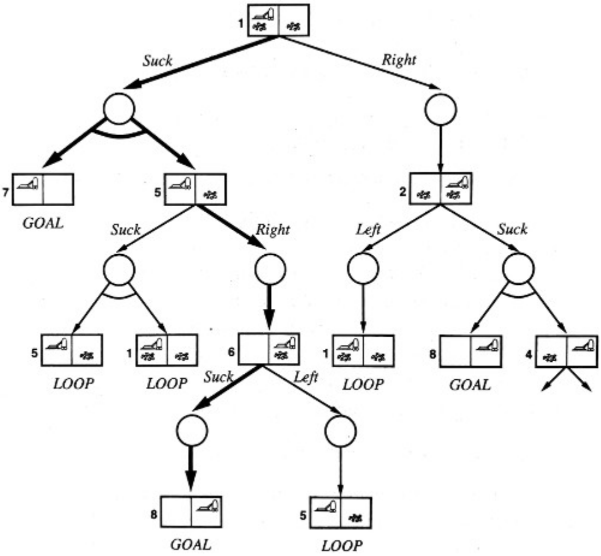
- Search tree would contain some OR nodes and some AND nodes

[1] Percepts would tell where have we reached.

AND-OR Search Tree



AND-OR Search Tree



Solution

1. has goal node at every leaf
2. takes one action at each OR node
3. includes every outcome branch at each AND node

Searching with Partial Observations

When percepts do not suffice to pin down the exact state

Searching with Partial Observations

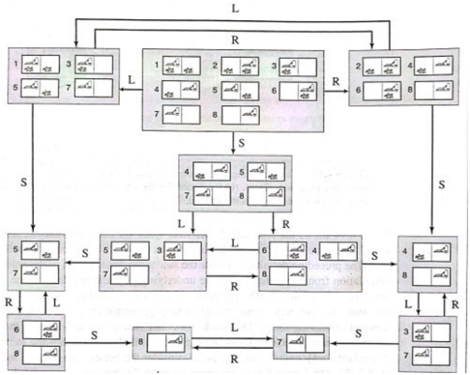
When percepts do not suffice to pin down the exact state

- **Sensor less.** consider [*right,suck,left,suck*] guarantees to reach in state 7 that is a goal state (traverses through belief states)

Searching with Partial Observations

When percepts do not suffice to pin down the exact state

- **Sensor less.** consider [*right,suck,left,suck*] guarantees to reach in state 7 that is a goal state (traverses through belief states)
- All possible belief states may not be reachable (only 12 out of 2^8)



Online Search and Unknown Environment

Agent interleaves computation and action

Online Search and Unknown Environment

Agent interleaves computation and action

Take action $\rightarrow$ observe environment $\rightarrow$ compute next action

Online Search and Unknown Environment

Agent interleaves computation and action

Take action → observe environment → compute next action

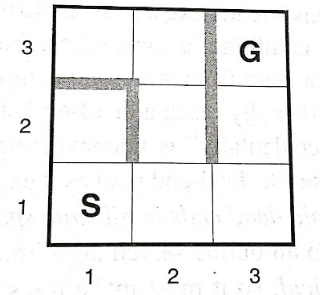
- Online Search is necessary for unknown environment

Online Search and Unknown Environment

Agent interleaves computation and action

Take action → observe environment → compute next action

- Online Search is necessary for unknown environment



1. Consider following maze problem
2. A robot need to go from S to G
3. Knows nothing about the environment

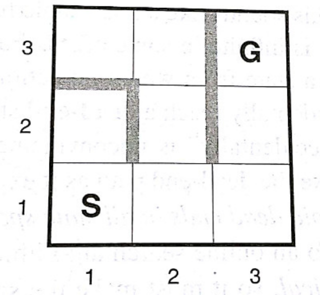
Random-walk?

Online Search and Unknown Environment

Agent interleaves computation and action

Take action → observe environment → compute next action

- Online Search is necessary for unknown environment



1. Consider following maze problem
2. A robot need to go from S to G
3. Knows nothing about the environment

Random-walk?

No algorithm can avoid dead-end in all state space

Adversarial Search (game)

Agents having conflicting goals in competitive multiagent environment

Adversarial Search (game)

Agents having conflicting goals in competitive multiagent environment

- Deterministic, fully-observable, turn-taking, two-player, zero-sum

Adversarial Search (game)

Agents having conflicting goals in competitive multiagent environment

- Deterministic, fully-observable, turn-taking, two-player, zero-sum
- Chess has roughly branching factor 35, moves 50 so tree search space is $35^{100} = 10^{154}$ however, graph has 10^{40} nodes
- Finding optimal move is infeasible but, needs an ability to decide

Adversarial Search (game)

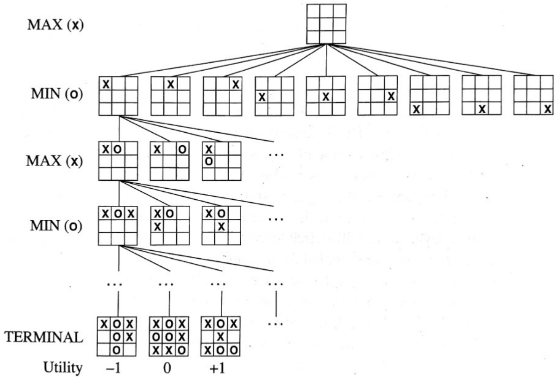
Agents having conflicting goals in competitive multiagent environment

- Deterministic, fully-observable, turn-taking, two-player, zero-sum
- Chess has roughly branching factor 35, moves 50 so tree search space is $35^{100} = 10^{154}$ however, graph has 10^{40} nodes
- Finding optimal move is infeasible but, needs an ability to decide

Game is between MAX and MIN (MAX moves first)

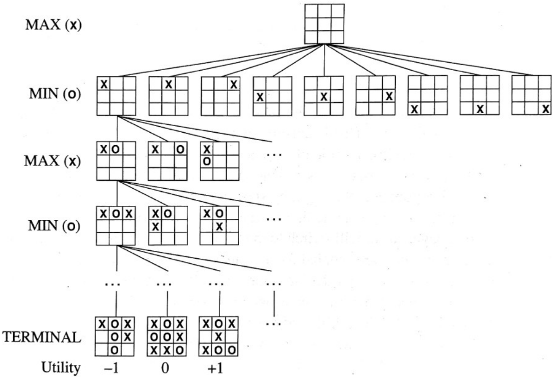
- S_0: the initial state
- PLAYER(s): defines which player has move to start
- ACTIONS(s): returns set of legal moves in a state
- RESULT(s, a):termination model defining result of a move
- TERMINAL_TEST(s): is true when game is over
- UTILITY(s, p): utility function defining reward (for chess +1,0,$1/2$)

Game Tree for tic-tac-toe



The search tree of the game has less than $9! = 362880$ nodes.

Game Tree for tic-tac-toe

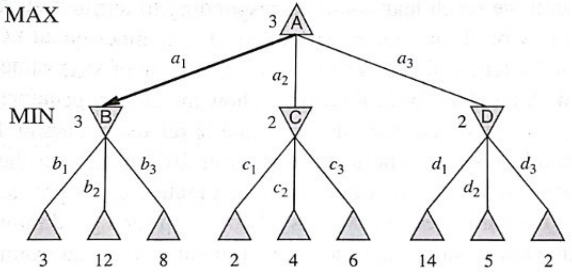


The search tree of the game has less than $9! = 362880$ nodes.

MAX must find a contingent **strategy**.

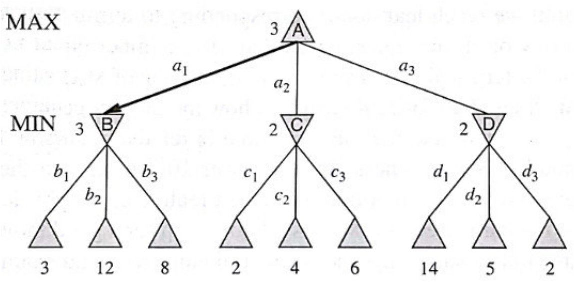
Analogous to AND-OR search (MAX plays OR and MIN plays AND)

Two half moves is one **ply**



[2] utility value for MAX of being in corresponding state (assuming then onwards both player play optimally)

Two half moves is one **ply**

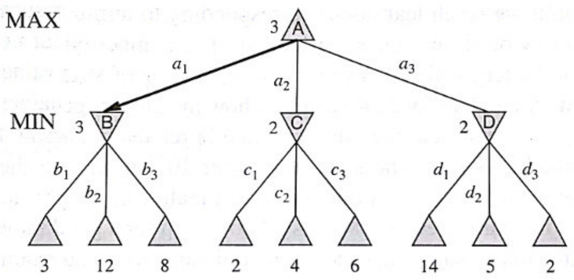


Given the game tree, optimal strategy can be determined from **minimax value** of each node.

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

[2] utility value for MAX of being in corresponding state (assuming then onwards both player play optimally)

Two half moves is one **ply**



Given the game tree, optimal strategy can be determined from **minimax value** of each node.

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

Action a_1 is the optimal choice [2]
(essentially optimizing worst-case outcome for MAX)

[2] utility value for MAX of being in corresponding state (assuming then onwards both player play optimally)

MINIMAX Algorithm

Returns the action corresponding to best move

```
function MINIMAX-DECISION(state) returns an action
  return arg max_{a ∈ ACTIONS(s)} 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(s, 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(s, a)))
  return v
```

Recursion proceeds all the way down to the leaves.

MINIMAX Algorithm

Returns the action corresponding to best move

```
function MINIMAX-DECISION(state) returns an action
   return arg max_{a ∈ ACTIONS(s)} 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(s, 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(s, a)))
   return v
```

Recursion proceeds all the way down to the leaves.Time complexity $O(b^m)$ that is impractical but provides a basis of solution.

ALPHA-BETA Pruning

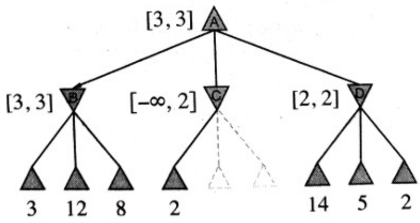
- Number of nodes to examine in minimax search is exponential in the depth of tree $O(b^m)$.
- Sometime we can make it $O(b^{m/2})$ using **alpha-beta pruning**

ALPHA-BETA Pruning

- Number of nodes to examine in minimax search is exponential in the depth of tree $O(b^m)$.
- Sometime we can make it $O(b^{m/2})$ using **alpha-beta pruning**
- When applied to standard minimax tree, it returns the same move as minimax but, prunes away branches that cannot possibly influence the decision.

ALPHA-BETA Pruning

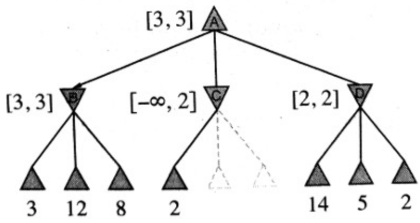
- Number of nodes to examine in minimax search is exponential in the depth of tree $O(b^m)$.
- Sometime we can make it $O(b^{m/2})$ using **alpha-beta pruning**
- When applied to standard minimax tree, it returns the same move as minimax but, prunes away branches that cannot possibly influence the decision.



Consider two unevaluated successor of node C have value x and y

ALPHA-BETA Pruning

- Number of nodes to examine in minimax search is exponential in the depth of tree $O(b^m)$.
- Sometime we can make it $O(b^{m/2})$ using **alpha-beta pruning**
- When applied to standard minimax tree, it returns the same move as minimax but, prunes away branches that cannot possibly influence the decision.

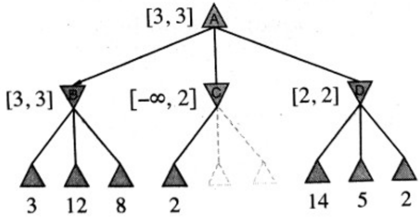


Consider two unevaluated successor of node C have value x and y

MINIMAX(root)
= max(min(3,12,8), min(2,x,y), min(14,5,2))

ALPHA-BETA Pruning

- Number of nodes to examine in minimax search is exponential in the depth of tree $O(b^m)$.
- Sometime we can make it $O(b^{m/2})$ using **alpha-beta pruning**
- When applied to standard minimax tree, it returns the same move as minimax but, prunes away branches that cannot possibly influence the decision.

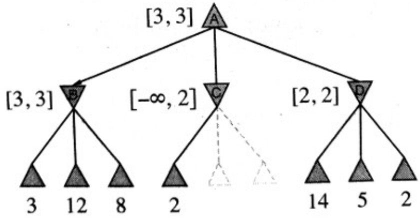


Consider two unevaluated successor of node C have value x and y

MINIMAX(root)
= max(min(3,12,8), min(2,x,y), min(14,5,2))
= max(3, min(2,x,y), 2)

ALPHA-BETA Pruning

- Number of nodes to examine in minimax search is exponential in the depth of tree $O(b^m)$.
- Sometime we can make it $O(b^{m/2})$ using **alpha-beta pruning**
- When applied to standard minimax tree, it returns the same move as minimax but, prunes away branches that cannot possibly influence the decision.

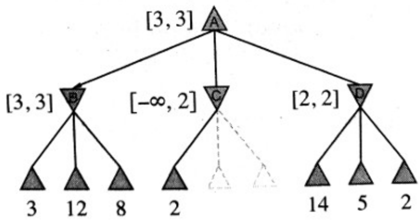


Consider two unevaluated successor of node C have value x and y

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) where z=min(2,x,y)$\leq$ 2

ALPHA-BETA Pruning

- Number of nodes to examine in minimax search is exponential in the depth of tree $O(b^m)$.
- Sometime we can make it $O(b^{m/2})$ using **alpha-beta pruning**
- When applied to standard minimax tree, it returns the same move as minimax but, prunes away branches that cannot possibly influence the decision.

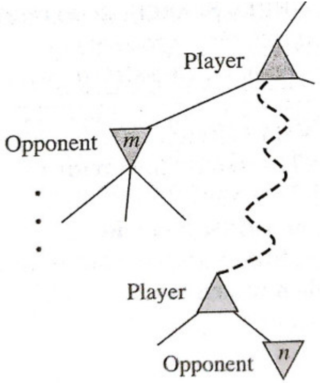


Consider two unevaluated successor of node C have value x and y

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) where $z=\min(2,x,y) \leq 2$
= 3

ALPHA-BETA Pruning

- Alpha-beta pruning can be applied to trees of any depth, and it is often possible to prune entire subtree rather than just leaves.



If *m* is better than *n* for player then we would never go to *n* in play

α	$=$	value of best choice (highest) found so far for MAX
β	$=$	value of best choice (lowest) found so far for MIN

ALPHA-BETA Pruning

```
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(RESULT(s,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(RESULT(s,a) , α, β))
    if v ≤ α then return v
    β ← MIN(β, v)
  return v
```

ALPHA-BETA Pruning

function ALPHA-BETA-SEARCH(*state*) **returns** an action
  $v \leftarrow$ MAX-VALUE(*state*, $-\infty, +\infty$)
  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*)
  $v \leftarrow -\infty$
  for each a **in** ACTIONS(*state*) **do**
    $v \leftarrow$ MAX(v, MIN-VALUE(RESULT(s,a), α, β))
    if $v \geq \beta$ **then return** v
    $\alpha \leftarrow$ MAX(α, v)
  return v

---

function MIN-VALUE(*state*, α, β) **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), α, β))
    if $v \leq \alpha$ **then return** v
    $\beta \leftarrow$ MIN(β, v)
  return v

Order matters. So, examine likely to be best successor first.

ALPHA-BETA Pruning

```
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(RESULT(s,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(RESULT(s,a) , α, β))
      if v ≤ α then return v
      β ← MIN(β, v)
   return v
```

Order matters. So, examine likely to be best successor first.

Is it possible?

ALPHA-BETA Pruning

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

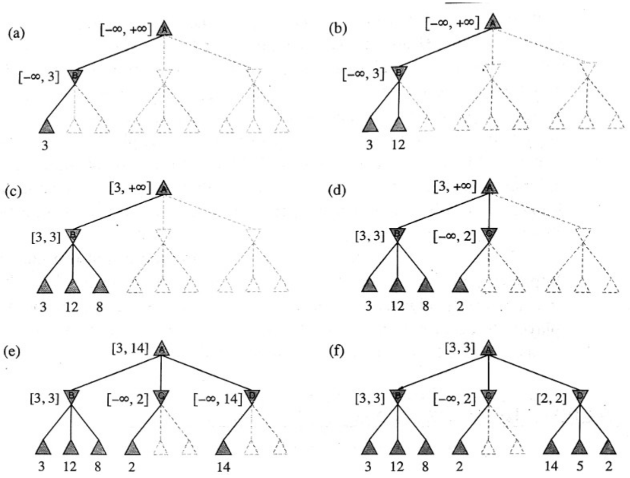
```
function MAX-VALUE(state, α, β) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← −∞
  for each a in ACTIONS(state) do
    v ← MAX(v, MIN-VALUE(RESULT(s,a), α, β))
    if v ≥ β then return v
    α ← MAX(α, v)
  return v
```

```
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(RESULT(s,a) , α, β))
    if v ≤ α then return v
    β ← MIN(β, v)
  return v
```

Order matters. So, examine likely to be best successor first.

Is it possible?
No

In-action: ALPHA-BETA Pruning



Thank You!

Thank you very much for your attention!

Queries ?

(Reference[3])

[3] 1) Book - *AIMA*, ch-04+05, Russell and Norvig.