DATA130008 Introduction to Artificial Intelligence

复旦大学大数据学院
School of Data Science, Fudan University
袁建 罗瑞璞

Lab 2

October 21th 2020

Outline

- Gomoku
  - Final project
- Alpha-Beta Pruning
  - Submit in class via OJ
- Constraint Satisfaction Problems
  - Take home as an assignment (Project 2)

Outline

- Gomoku
  - Final project
- Alpha-Beta Pruning
  - Submit in class via OJ
- Constraint Satisfaction Problems
  - Take home as an assignment (Project 2)

- Gomoku Rule
- Solve Gomoku
  - Board Representation
  - Monte Carlo Tree Search
  - Proof-Number Search
  - Threat-Space Search
  - Genetic Algorithm

- Gomoku Rule
- Solve Gomoku
  - Board Representation
  - Monte Carlo Tree Search
  - Proof-Number Search
  - Threat-Space Search
  - Genetic Algorithm

Gomoku Rule

- Goal: Five in a row, 15 $\times$ 15
  - Proved: Black first leads to win (1899)
- Gomoku without forbidden shape:
  - Free: 5 or more than 5
  - Standard: only 5
  - Swap 2 Rule
- Renju: forbid some shape for Black

- Focus on: Free Gomoku

- Gomoku Rule
- Solve Gomoku
  - Board Representation
  - Monte Carlo Tree Search
  - Proof-Number Search
  - Threat-Space Search
  - Genetic Algorithm

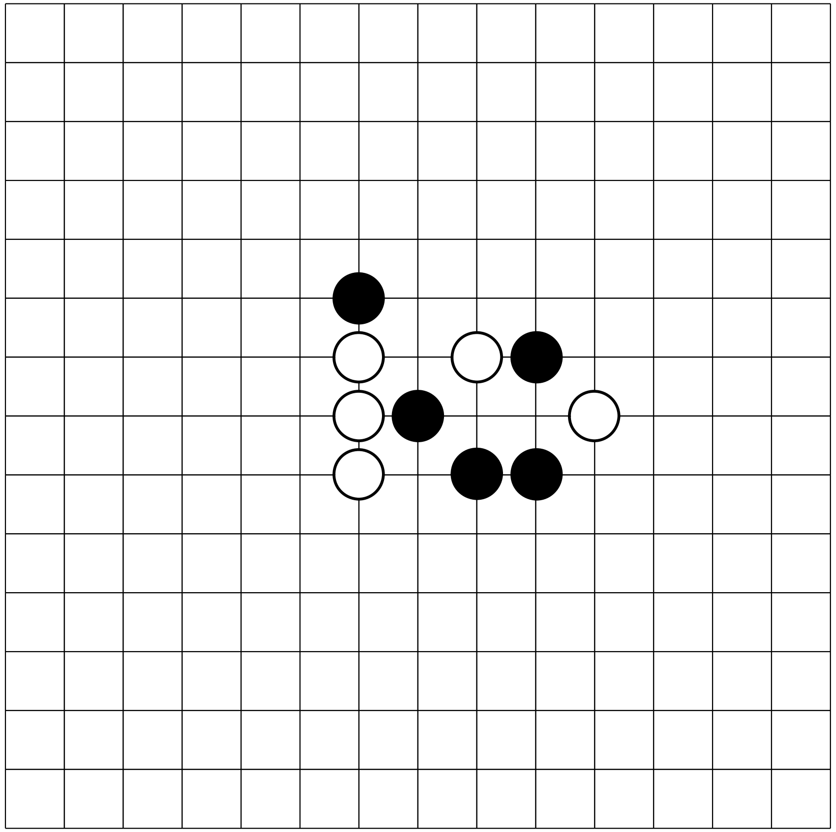
Solve Gomoku

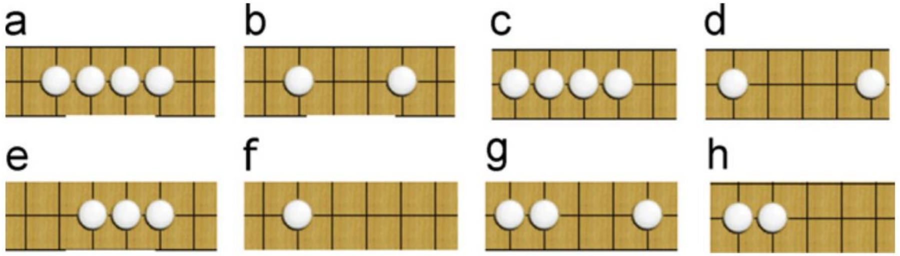
- Input
  - Current board state
- Goal
  - Search for next step

- Gomoku Rule
- Solve Gomoku
  - Board Representation
  - Monte Carlo Tree Search
  - Proof-Number Search
  - Threat-Space Search
  - Genetic Algorithm

Solve Gomoku: Board Representation

- Board representation
  - Atomic
    - (x, y, ●/○/ ·)
  - Structured: Features
    - Patterns
    - Turns
    - Offensive/Defensive



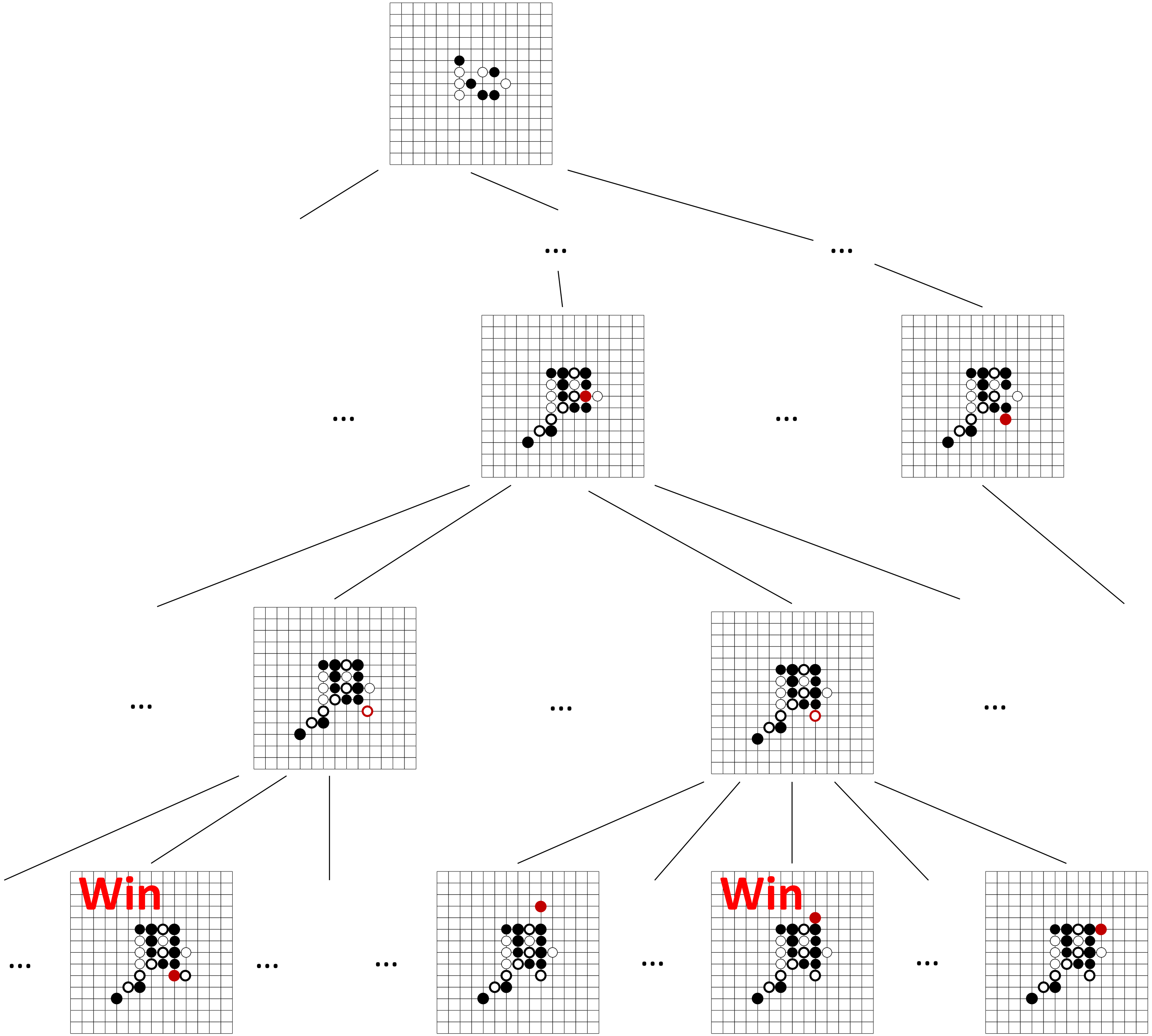


Dongbin Zhao, Zhen Zhang, and Yujie Dai.
Self-teaching adaptive dynamic programming for Gomoku.
Neurocomputing, 78(1):23 - 29, 2012.

Zhentao Tang, Dongbin Zhao, Kun Shao, and Le Lv.
ADP with MCTS algorithm for Gomoku.
2016 IEEE Symp. Ser. Comput. Intell. SSCI 2016,

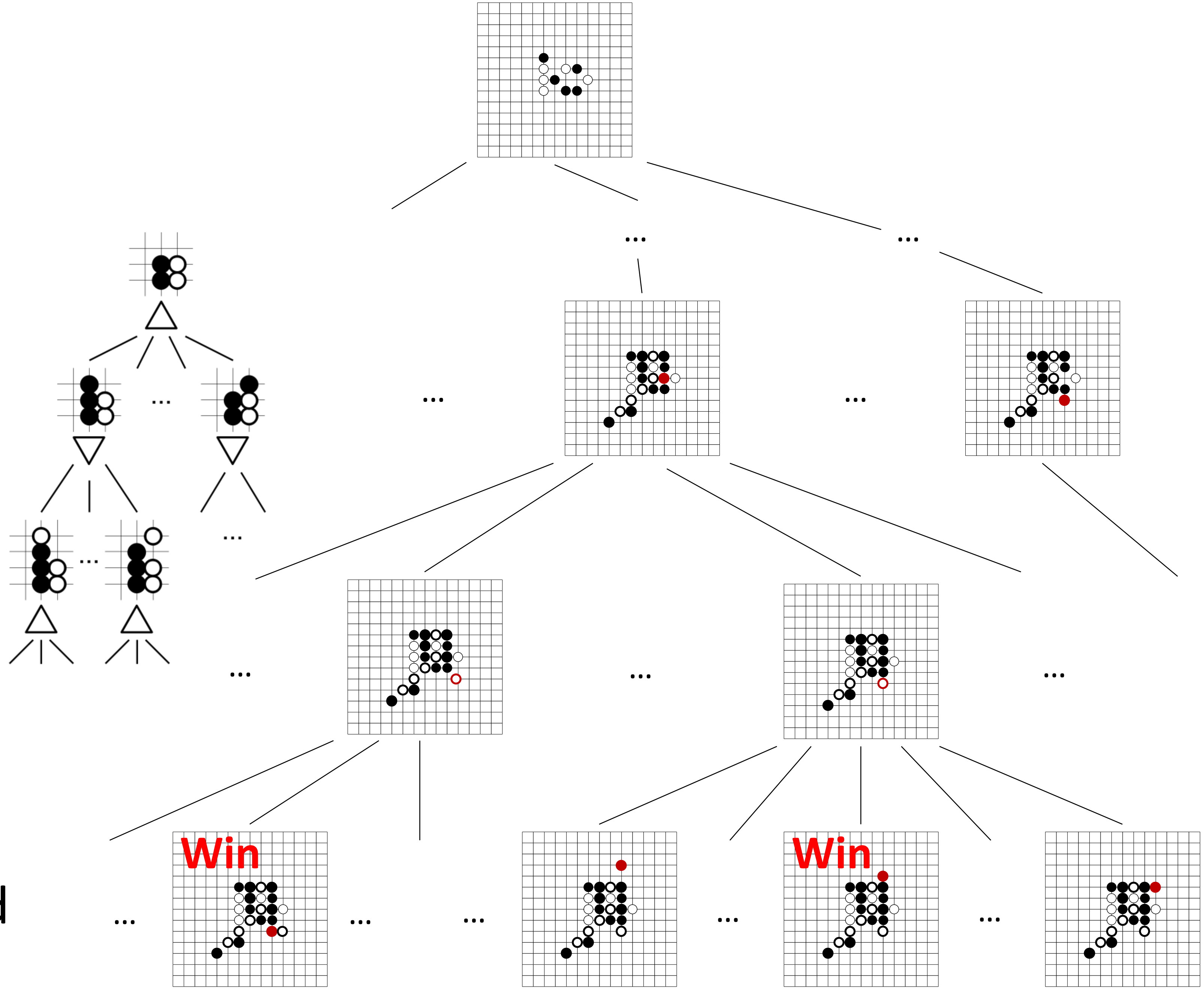
Solve Gomoku

- Input
  - Current board state
- Goal
  - Search for next step
    - Single agent
    - Adversarial agent
- Construct a search tree
  - Node: Gomoku board



Solve Gomoku

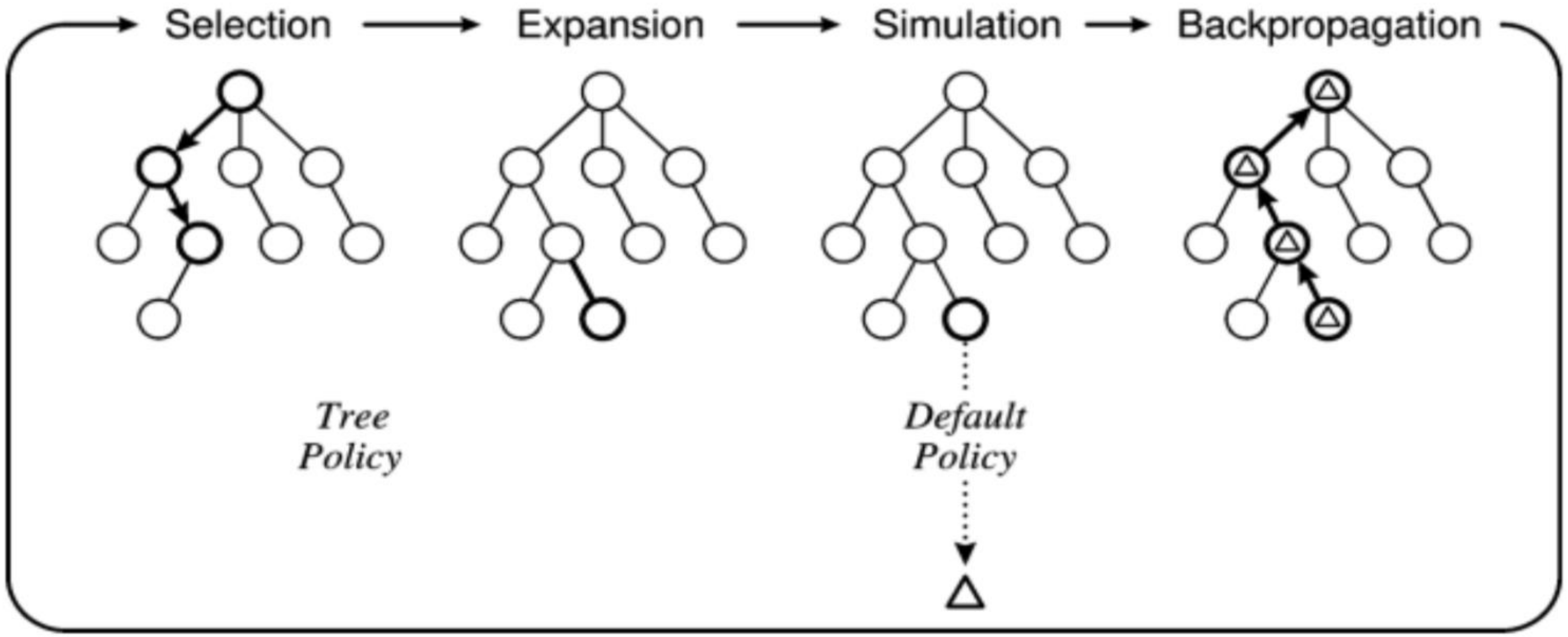
- Input
  - Current board state
- Goal
  - Search for next step
    - Single agent
    - Adversarial agent
- Construct a search tree
  - Node: Gomoku board



- Gomoku Rule
- Solve Gomoku
  - Board Representation
  - Monte Carlo Tree Search
  - Proof-Number Search
  - Threat-Space Search
  - Genetic Algorithm

Solve Gomoku: Monte Carlo Tree Search

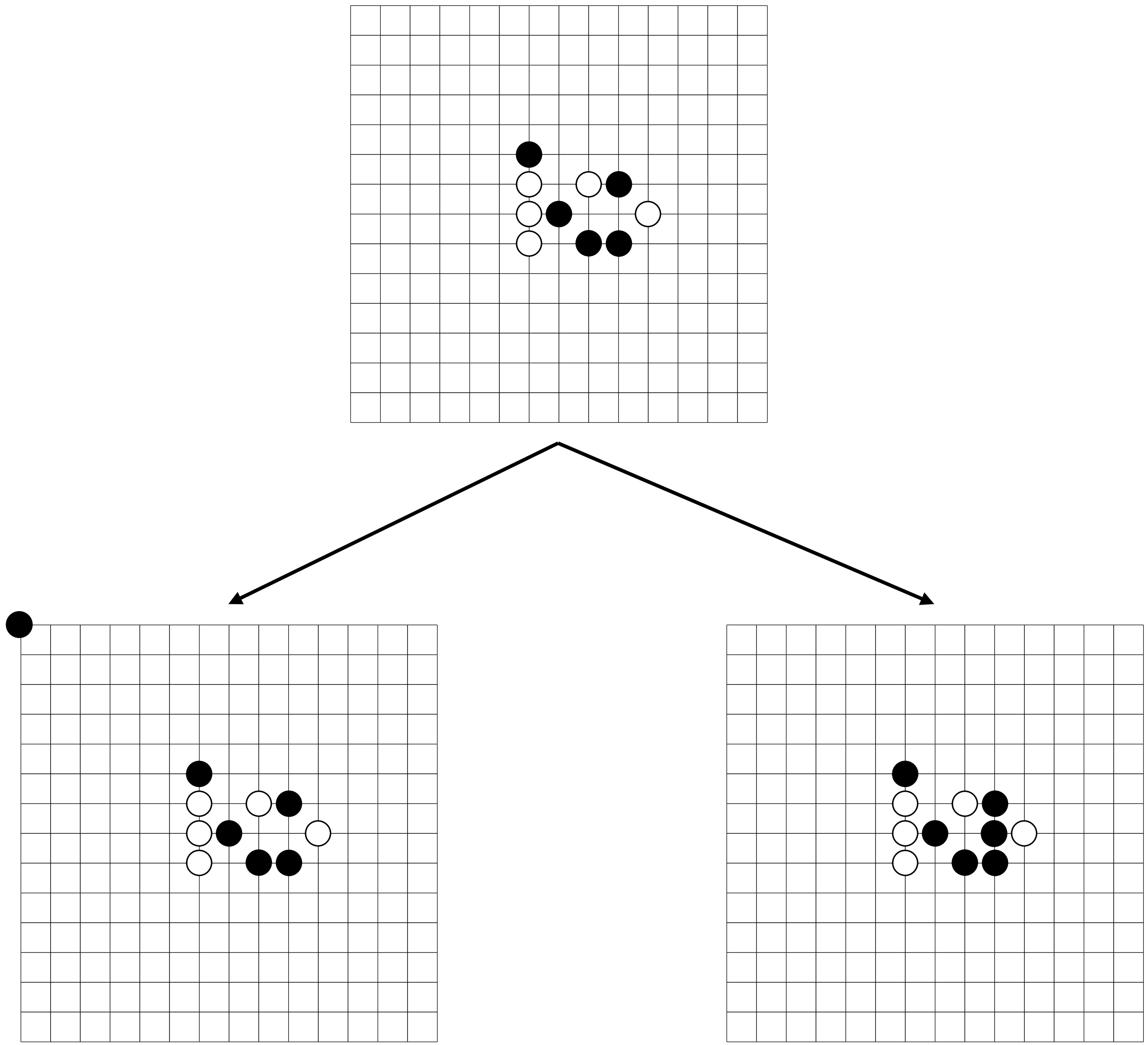
- Monte Carlo Tree Search
  - Simulation, Expectation
  - Steps
    - Selection
    - Expansion
    - Simulation
    - Backpropagation



Zhentao Tang, Dongbin Zhao, Kun Shao, and Le Lv.
ADP with MCTS algorithm for Gomoku.
2016 IEEE Symp. Ser. Comput. Intell. SSCI 2016, (61273136), 2017.

Solve Gomoku: Monte Carlo Tree Search

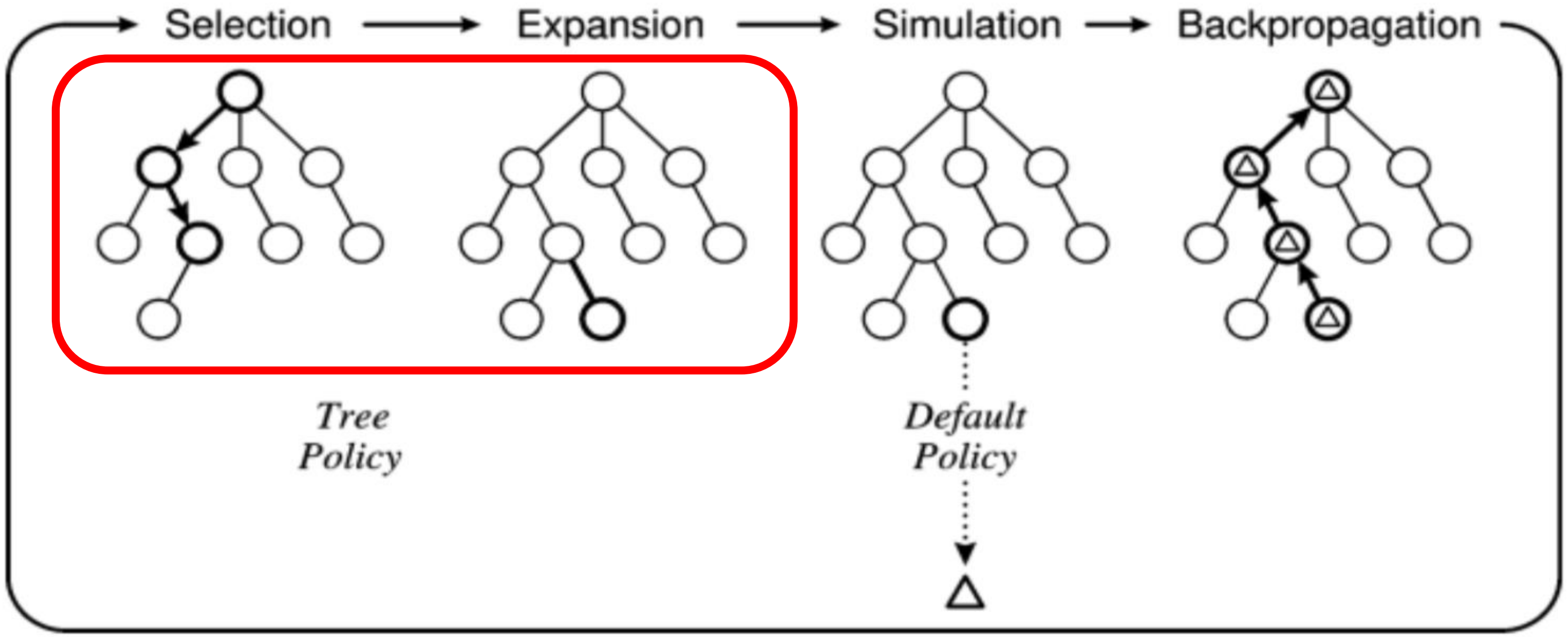
- Monte Carlo Tree Search
  - Simulation, Expectation
  - Steps
    - Selection
    - Expansion
    - Simulation
    - Backpropagation



Zhentao Tang, Dongbin Zhao, Kun Shao, and Le Lv.
ADP with MCTS algorithm for Gomoku.
2016 IEEE Symp. Ser. Comput. Intell. SSCI 2016, (61273136), 2017.

Solve Gomoku: Monte Carlo Tree Search

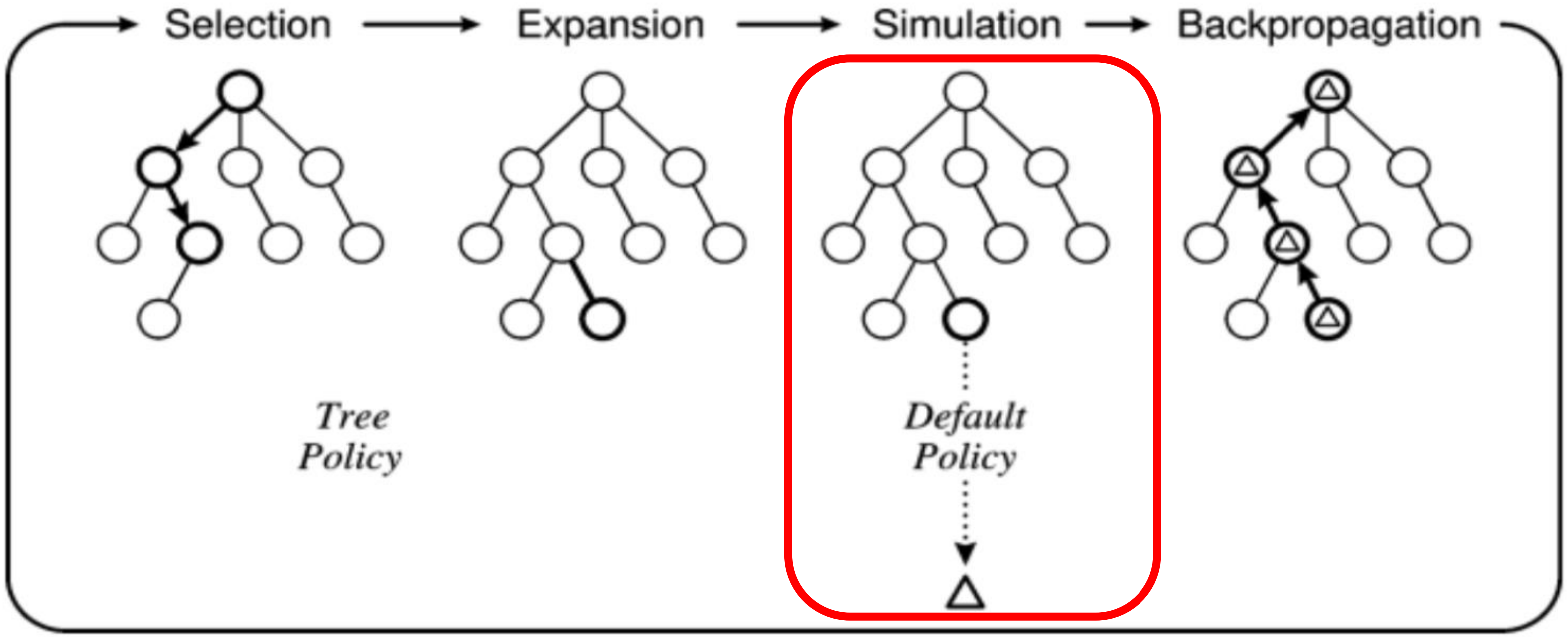
- Monte Carlo Tree Search
  - Simulation, Expectation
  - Steps
    - Selection
    - Expansion
    - Simulation
    - Backpropagation



Zhentao Tang, Dongbin Zhao, Kun Shao, and Le Lv.
ADP with MCTS algorithm for Gomoku.
2016 IEEE Symp. Ser. Comput. Intell. SSCI 2016, (61273136), 2017.

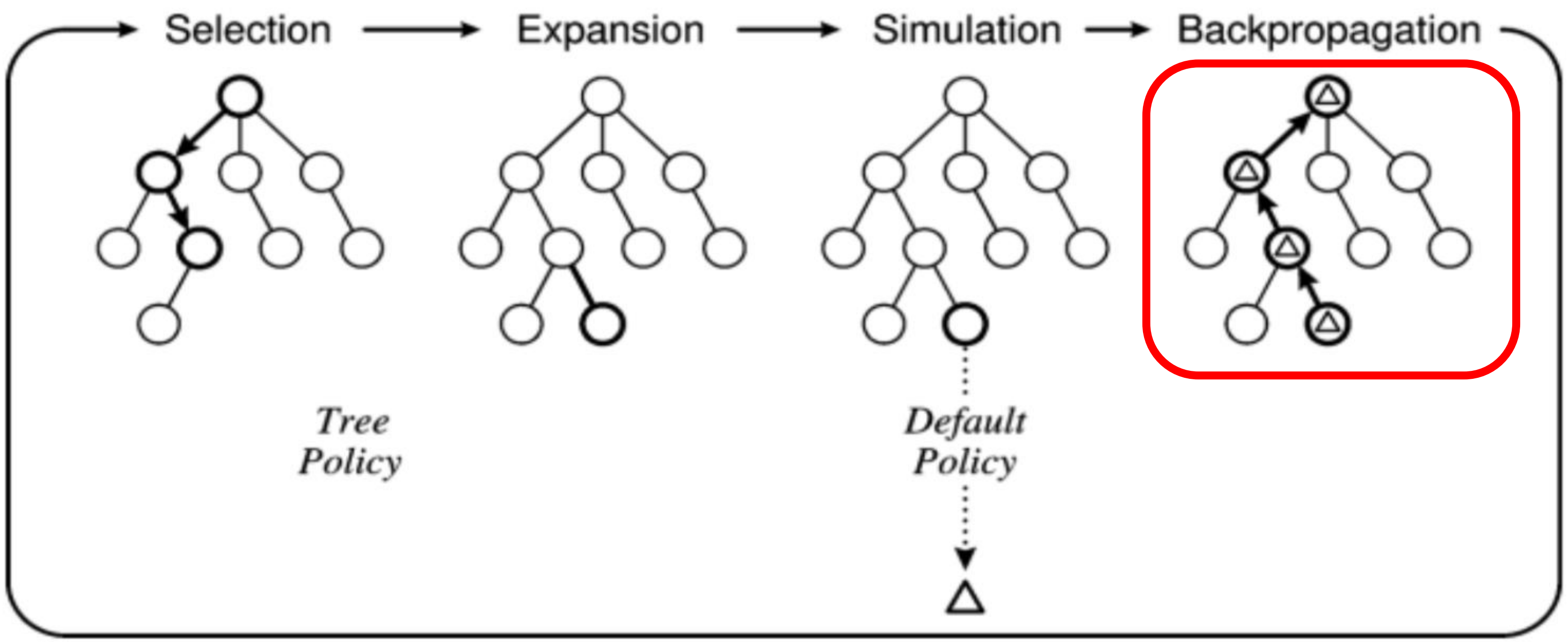
Solve Gomoku: Monte Carlo Tree Search

- Monte Carlo Tree Search
  - Simulation, Expectation
  - Steps
    - Selection
    - Expansion
    - Simulation
    - Backpropagation



Zhentao Tang, Dongbin Zhao, Kun Shao, and Le Lv.
ADP with MCTS algorithm for Gomoku.
2016 IEEE Symp. Ser. Comput. Intell. SSCI 2016, (61273136), 2017.

- Monte Carlo Tree Search
  - Simulation, Expectation
  - Steps
    - Selection
    - Expansion
    - Simulation
    - Backpropagation



Zhentao Tang, Dongbin Zhao, Kun Shao, and Le Lv.
ADP with MCTS algorithm for Gomoku.
2016 IEEE Symp. Ser. Comput. Intell. SSCI 2016, (61273136), 2017.

Solve Gomoku: Monte Carlo Tree Search

- **Input** original state s0
- **Output** action a corresponding to the highest value of MCTS

add Heuristic Knowledge;
obtain possible action moves M from state s_0;
for each move m in moves M **do**
  reward $r_{total} \leftarrow 0$;
  while simulation times < assigned times **do**
    reward $r \leftarrow$ Simulation($s(m)$);
    $r_{total} \leftarrow r_{total} + r$;
    simulation times add one;
  end while
  add (m, r_{total}) into *data*;
  end for each
return action Best(*data*)

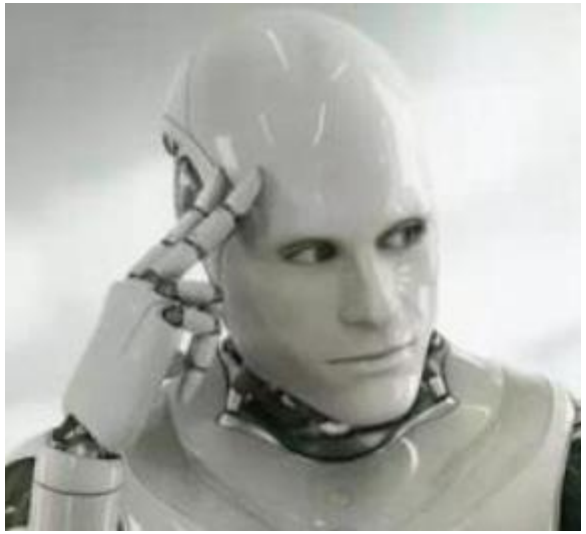
Simulation(state s_t)
  if (s_t is win **and** s_t is terminal) **then return** 1.0;
    else return 0.0;
  end if
  if (s_t satisfied with Heuristic Knowledge)
    then obtain forced action a_f;
      new state $s_{t+1} \leftarrow f(s_t, a_f)$;
    else choose random action $a_r \in$ untried actions;
      new state $s_{t+1} \leftarrow f(s_t, a_r)$;
  end if
  return Simulation(s_{t+1})

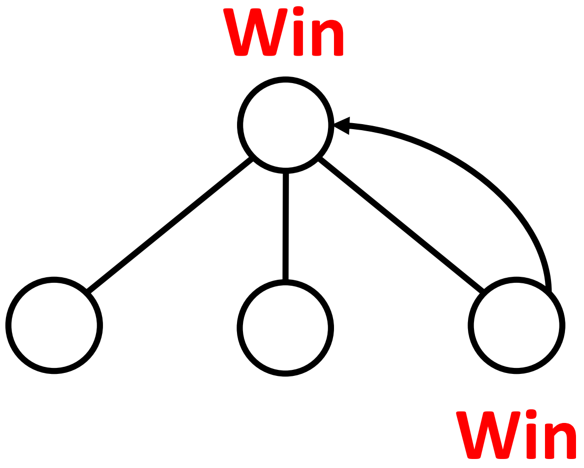
Best(*data*)
  return action a //the maximum r_{total} of m from data

Zhentao Tang, Dongbin Zhao, Kun Shao, and Le Lv.
ADP with MCTS algorithm for Gomoku.
2016 IEEE Symp. Ser. Comput. Intell. SSCI 2016,

- Gomoku Rule
- Solve Gomoku
  - Board Representation
  - Monte Carlo Tree Search
  - Proof-Number Search
  - Threat-Space Search
  - Genetic Algorithm

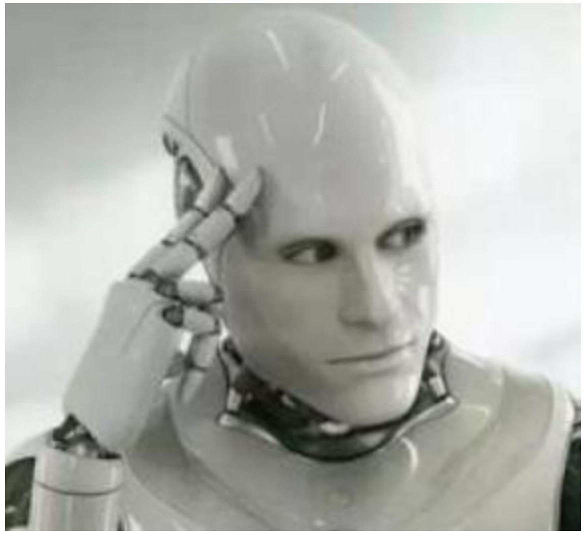
- When will the Black win?

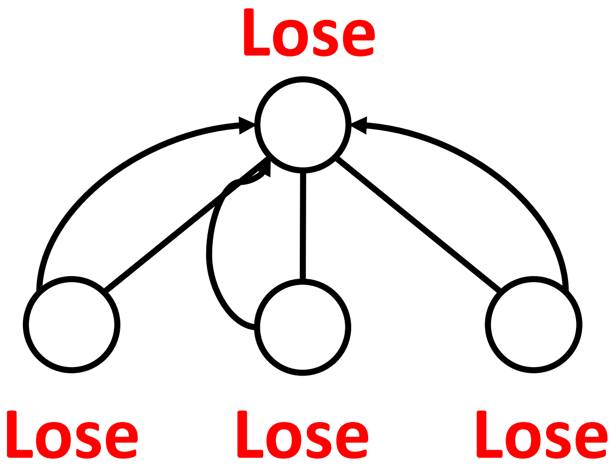




Louis Victor Allis.
Searching for Solutions in Games and Artificial Intelligence.
1994.

- When will the Black lose?





Louis Victor Allis.
Searching for Solutions in Games and Artificial Intelligence.
1994.

Solve Gomoku: Proof-Number Search

- Board situation: Win, Lose, Unknown
- 2 Nodes:
  - Black Turn (OR)
    - Win if there is an action (White take) leading to Black win
    - Lose if all actions leading to Black lose
  - White (AND)
    - Win if all actions leading to Black win
    - Lose if there is an action leading to Black lose

Louis Victor Allis.
Searching for Solutions in Games and Artificial Intelligence.
1994.

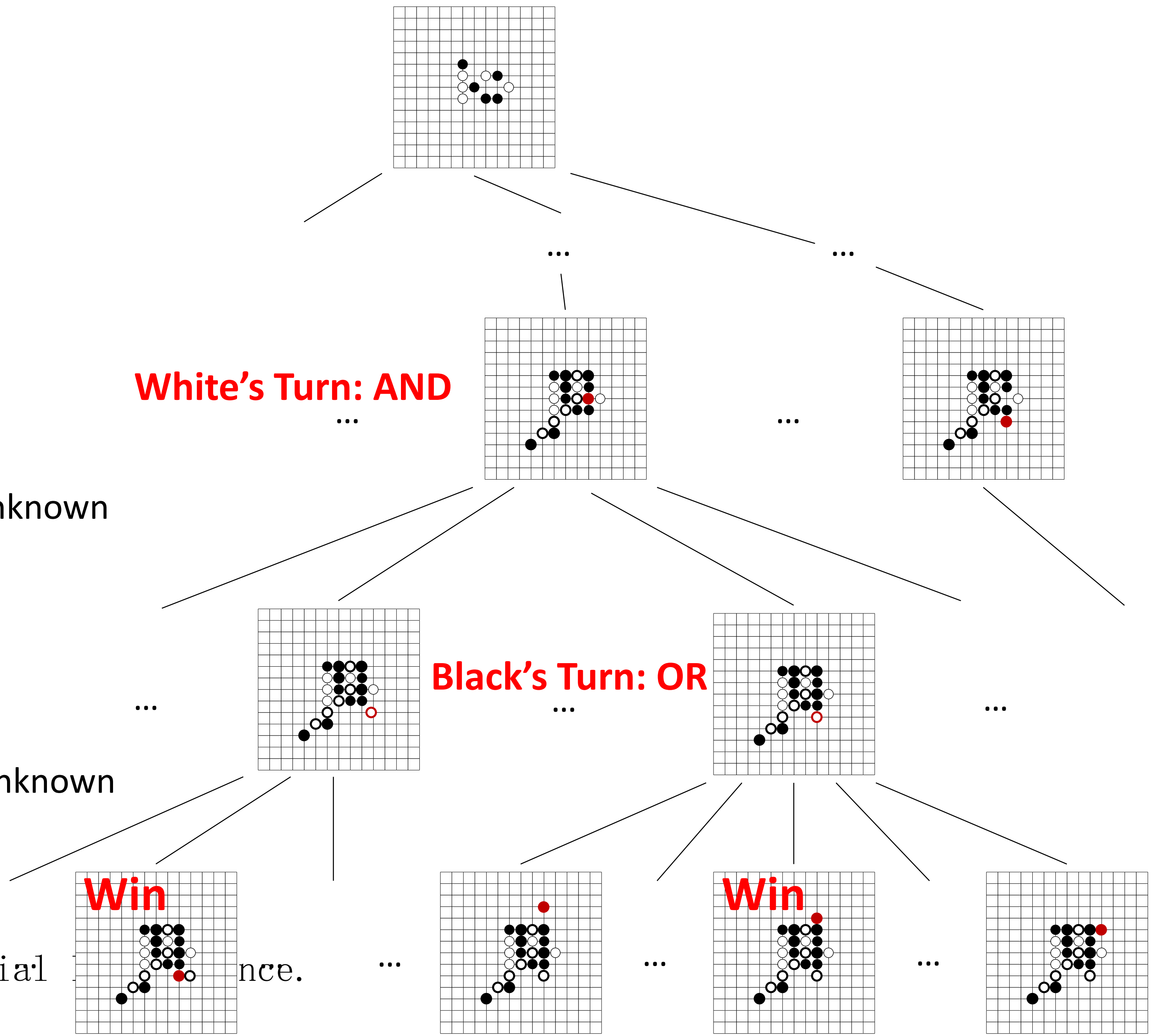
Solve Gomoku: Proof-Number Search

- Board situation: Win, Lose, Unknown
- 2 Nodes: **Win or Lose: BLACK's View**
  - Black Turn (OR)
    - Win if there is an action (White take) leading to Black win
    - Lose if all actions leading to Black lose
  - White (AND)
    - Win if all actions leading to Black win
    - Lose if there is an action leading to Black lose

Louis Victor Allis.
Searching for Solutions in Games and Artificial Intelligence.
1994.

Solve Gomoku: Proof-Number Search

- AND/OR Tree
  - 3 Values: true, false, unknown
  - 2 Nodes: AND, OR
    - Black: OR
      - Win if one child is win
      - Unknown if no win and has unknown
      - Lose if all children are lose
    - White: AND
      - Lose if one child is lose
      - Unknown if no lose and has unknown
      - Win if all children are win

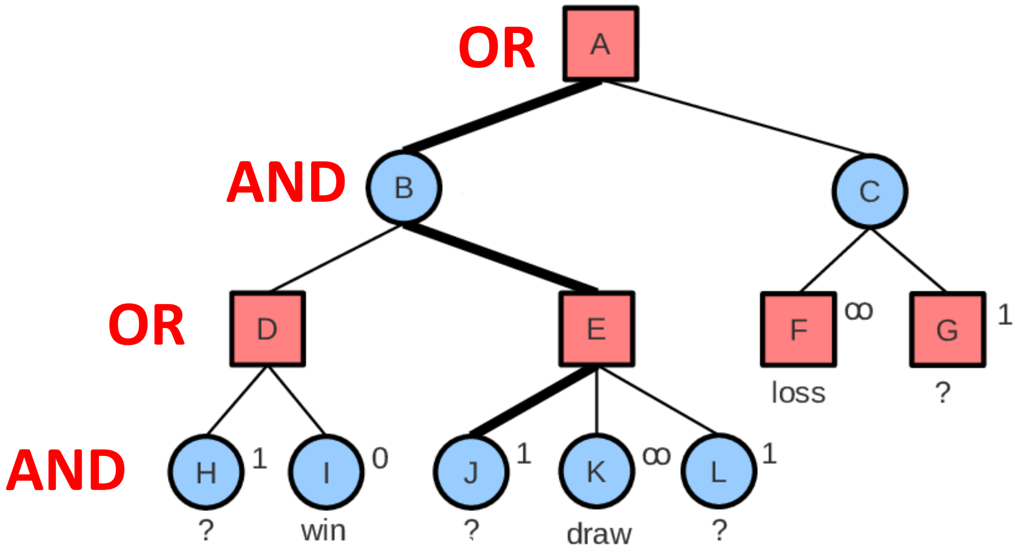


Louis Victor Allis.
Searching for Solutions in Games and Artificial Intelligence.
1994.

Solve Gomoku: Proof-Number Search

- AND/OR Tree
  - Proof set
    - Proof set: a set of frontier nodes S is a proof set if proving all nodes within S proves T
    - The proof number of T is defined as the cardinality of the smallest proof set of T
  - Disproof set
  - State of leaf nodes
    - Win: 0, ∞
    - Lose: ∞, 0
    - Unknown: 1, 1

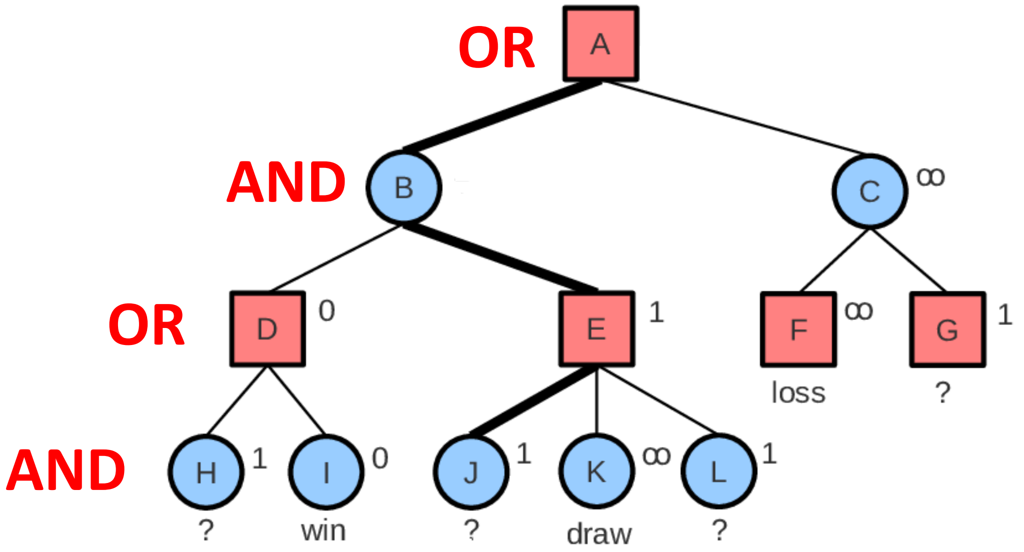
AND: Circle; OR: Square



Louis Victor Allis.
Searching for Solutions in Games and Artificial Intelligence.
1994.

- AND/OR Tree
  - Proof set
    - Proof set: a set of frontier nodes S is a proof set if proving all nodes within S proves T
    - The proof number of T is defined as the cardinality of the smallest proof set of T
  - Disproof set
  - State of leaf nodes
    - Win: 0, ∞
    - Lose: ∞, 0
    - Unknown: 1, 1

AND: Circle; OR: Square

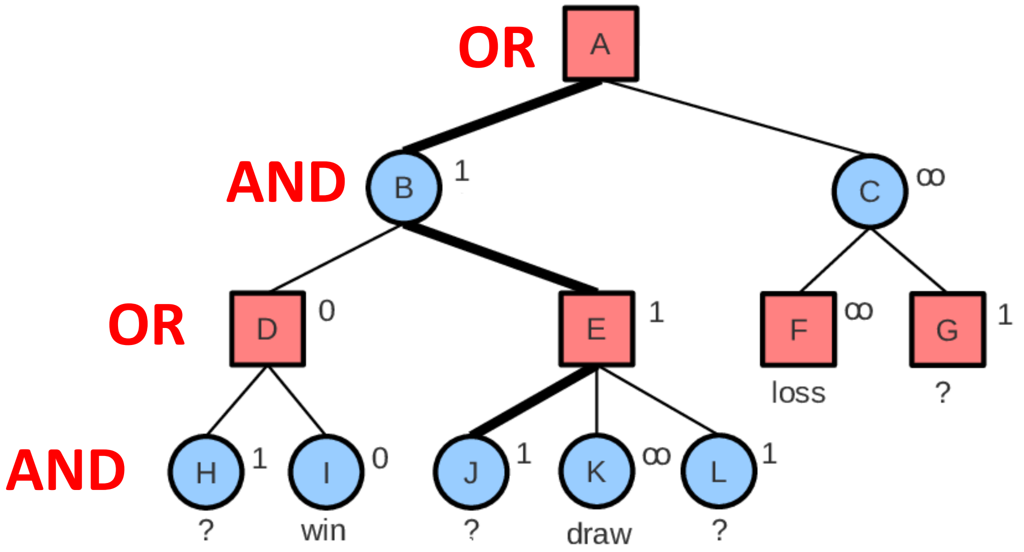


Louis Victor Allis.
Searching for Solutions in Games and Artificial Intelligence.
1994.

Solve Gomoku: Proof-Number Search

- AND/OR Tree
  - Proof set
    - Proof set: a set of frontier nodes S is a proof set if proving all nodes within S proves T
    - The proof number of T is defined as the cardinality of the smallest proof set of T
  - Disproof set
  - State of leaf nodes
    - Win: 0, ∞
    - Lose: ∞, 0
    - Unknown: 1, 1

AND: Circle; OR: Square

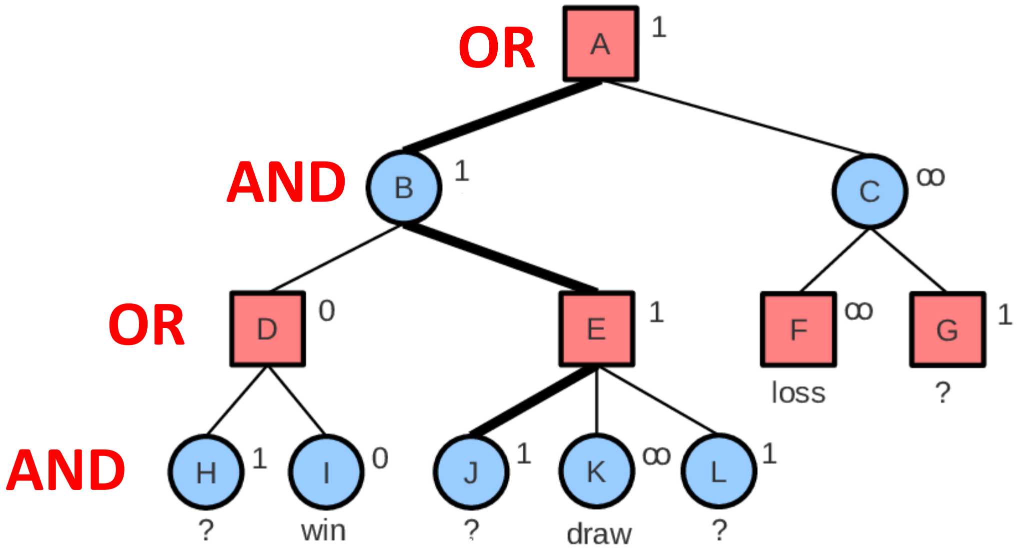


Louis Victor Allis.
Searching for Solutions in Games and Artificial Intelligence.
1994.

Solve Gomoku: Proof-Number Search

- AND/OR Tree
  - Proof set
    - Proof set: a set of frontier nodes S is a proof set if proving all nodes within S proves T
    - The proof number of T is defined as the cardinality of the smallest proof set of T
  - Disproof set
  - State of leaf nodes
    - Win: 0, ∞
    - Lose: ∞, 0
    - Unknown: 1, 1

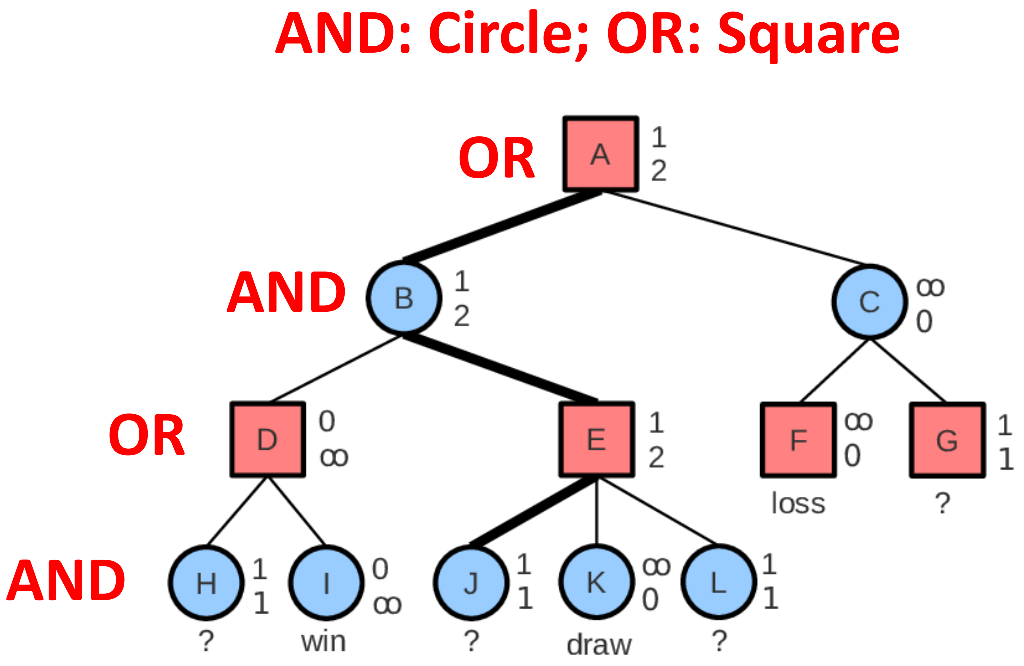
AND: Circle; OR: Square



Louis Victor Allis.
Searching for Solutions in Games and Artificial Intelligence.
1994.

Solve Gomoku: Proof-Number Search

- AND/OR Tree
  - Set proof number
    - AND: sum OR: min
  - Set disproof number
    - AND: min OR: sum

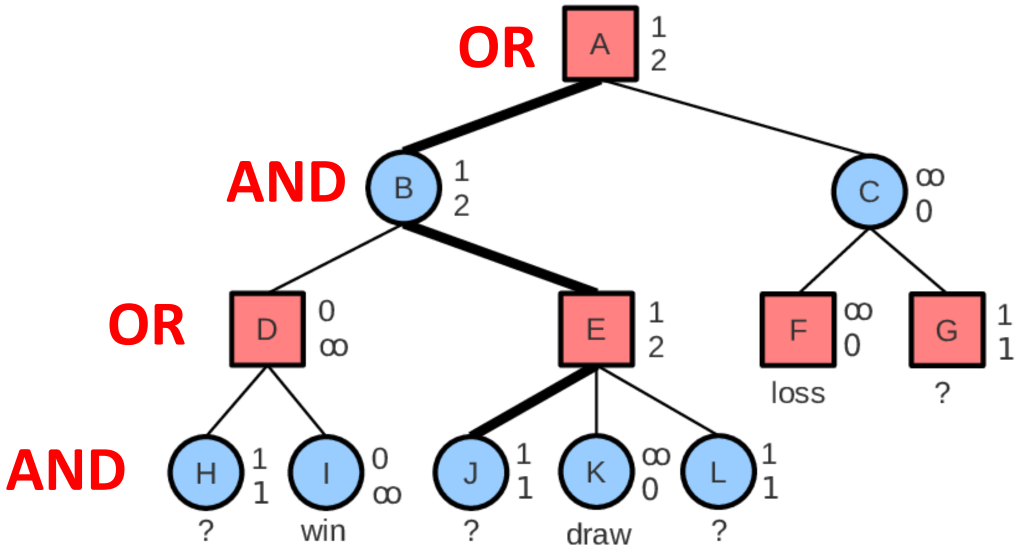


Louis Victor Allis.
Searching for Solutions in Games and Artificial Intelligence.
1994.

Solve Gomoku: Proof-Number Search

- AND/OR Tree
  - Most-proving node
    - Proved: each pair consisting of a smallest proof set and a smallest disproof set has a non-empty intersection.
    - i.e. There must exist at least one most-proving node.

AND: Circle; OR: Square

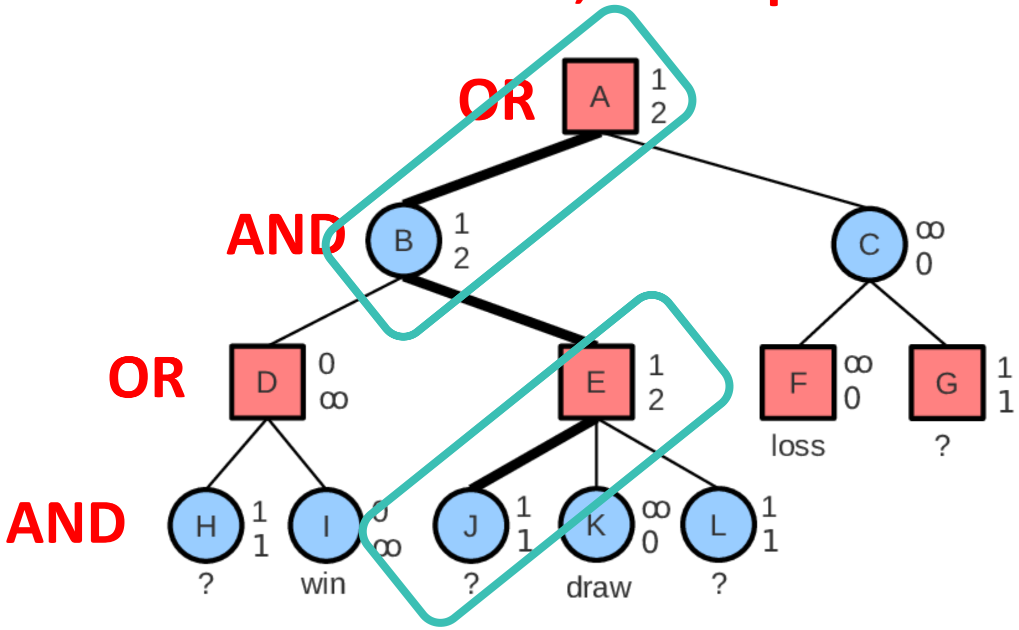


Louis Victor Allis.
Searching for Solutions in Games and Artificial Intelligence.
1994.

Solve Gomoku: Proof-Number Search

- AND/OR Tree
  - Most-proving node
    - Proved: each pair consisting of a smallest proof set and a smallest disproof set has a non-empty intersection.
    - i.e. There must exist at least one most-proving node.

AND: Circle; OR: Square

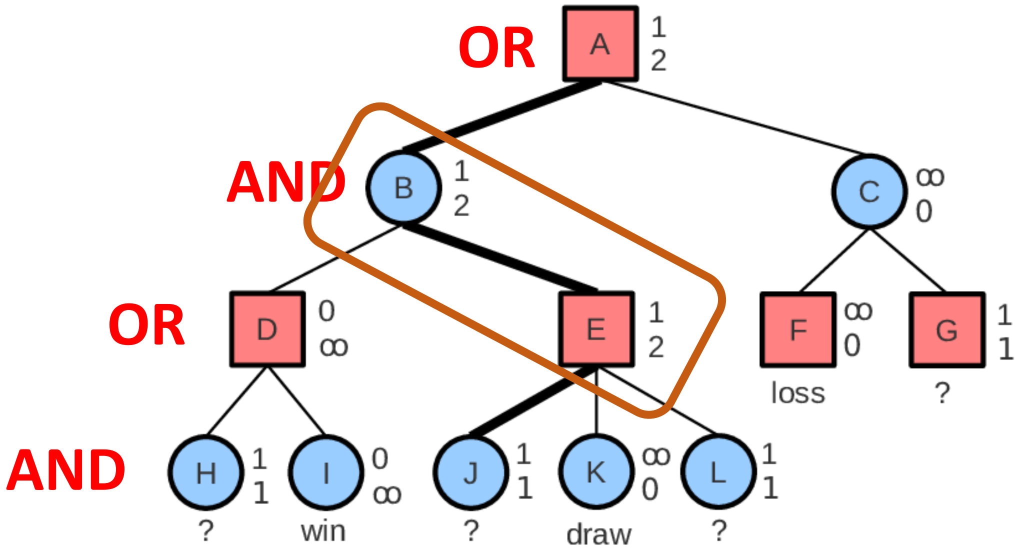


Louis Victor Allis.
Searching for Solutions in Games and Artificial Intelligence.
1994.

Solve Gomoku: Proof-Number Search

- AND/OR Tree
  - Most-proving node
    - Proved: each pair consisting of a smallest proof set and a smallest disproof set has a non-empty intersection.
    - i.e. There must exist at least one most-proving node.

AND: Circle; OR: Square



Louis Victor Allis.
Searching for Solutions in Games and Artificial Intelligence.
1994.

- Algorithm

```
procedure ProofNumberSearch(root);
    Evaluate(root);
    SetProofAndDisproofNumbers(root);
    while root.proof ≠ 0 and root.disproof ≠ 0 and
            ResourcesAvailable() do
        mostProvingNode := SelectMostProving(root);
        DevelopNode(mostProvingNode);
        UpdateAncestors(mostProvingNode)
    od ;
    if root.proof = 0 then root.value := true
    elseif root.disproof = 0 then root.value := false
    else root.value := unknown
    fi
end
```

```
function SelectMostProving(node);
    while node.expanded do
        case node.type of
            or :
                i := 1;
                while node.children[i].proof ≠ node.proof do
                    i := i+1
                od
            and :
                i := 1;
                while node.children[i].disproof ≠ node.disproof do
                    i := i+1
                od
        esac ;
        node := node.children[i]
    od ;
    return node
end
```

Louis Victor Allis.
Searching for Solutions in Games and Artificial Intelligence.
1994.

Solve Gomoku: Proof-Number Search

- Algorithm

```
procedure SetProofAndDisproofNumbers(node);
    if node.expanded then
        case node.type of
            and :
                node.proof := Σ_{N∈Children(node)} N.proof;
                node.disproof := Min_{N∈Children(node)} N.disproof
            or :
                node.proof := Min_{N∈Children(node)} N.proof;
                node.disproof := Σ_{N∈Children(node)} N.disproof
        esac
    elseif node.evaluated then
        case node.value of
            false : node.proof := ∞; node.disproof := 0
            true : node.proof := 0; node.disproof := ∞
            unknown : node.proof := 1; node.disproof := 1
        esac
    else node.proof := 1; node.disproof := 1
    fi
end
```

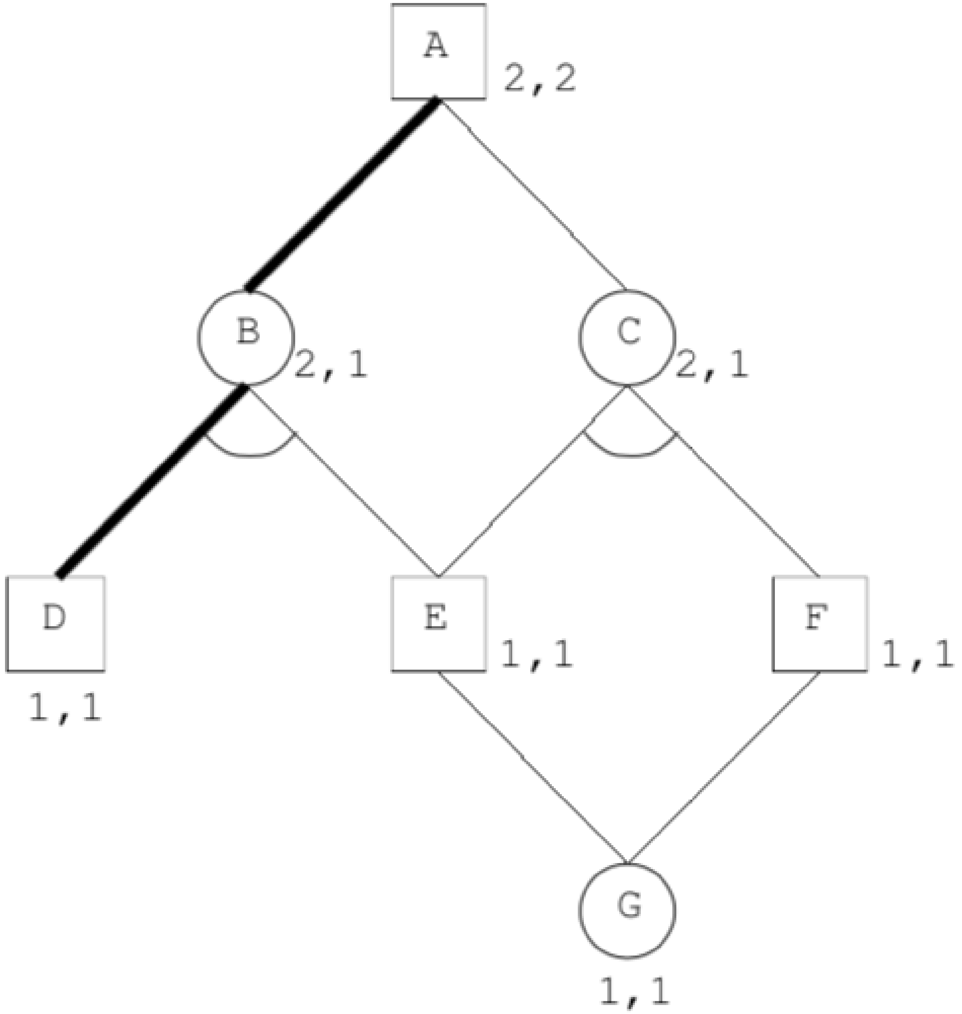
```
procedure DevelopNode(node);
    GenerateAllChildren(node);
    for i := 1 to node.numberOfChildren do
        Evaluate(node.children[i]);
        SetProofAndDisproofNumbers(node.children[i])
    od
end
```

```
procedure UpdateAncestors(node);
    while node ≠ nil do
        SetProofAndDisproofNumbers(node);
        node := node.parent
    od
end
```

Louis Victor Allis.
Searching for Solutions in Games and Artificial Intelligence.
1994.

Solve Gomoku: Proof-Number Search

- Transposition
  - Hash table
  - Directed Acyclic Graphs



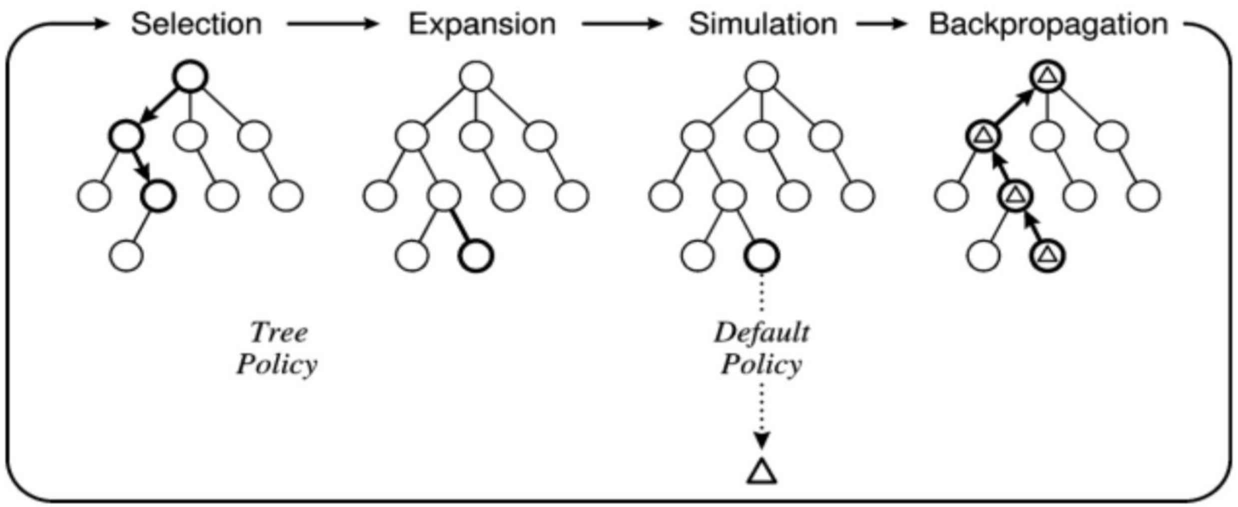
Louis Victor Allis.
Searching for Solutions in Games and Artificial Intelligence.
1994.

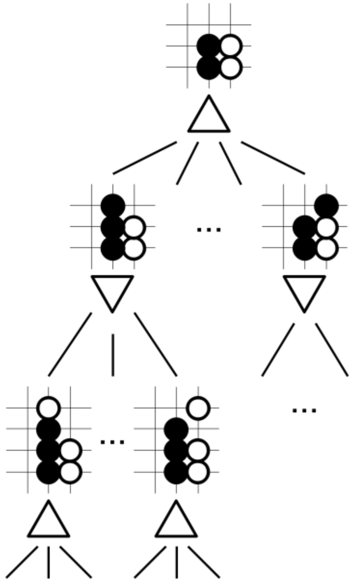
Solve Gomoku: Monte Carlo Tree Search

- Monte Carlo Tree Search
  - Simulation, Expectation
  - Steps
    - Selection
    - Expansion
    - Simulation
    - Backpropagation

Proof-Number Search

Threat-Space Search, Genetic Algorithm





Zhentao Tang, Dongbin Zhao, Kun Shao, and Le Lv.
ADP with MCTS algorithm for Gomoku.
2016 IEEE Symp. Ser. Comput. Intell. SSCI 2016,

Gomoku

- Gomoku Rule
- Solve Gomoku
  - Board Representation
  - Monte Carlo Tree Search
  - Proof-Number Search
  - Threat-Space Search
  - Genetic Algorithm

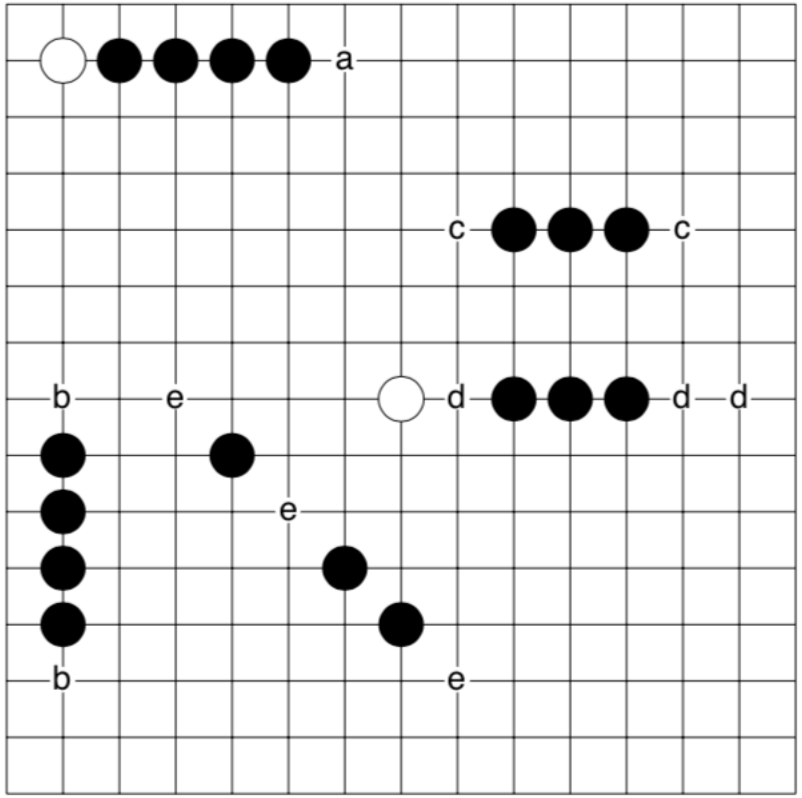
Solve Gomoku: Threat-Space Search

- Goal: Five in a row

Four in a row
Three in a row
…

Threat

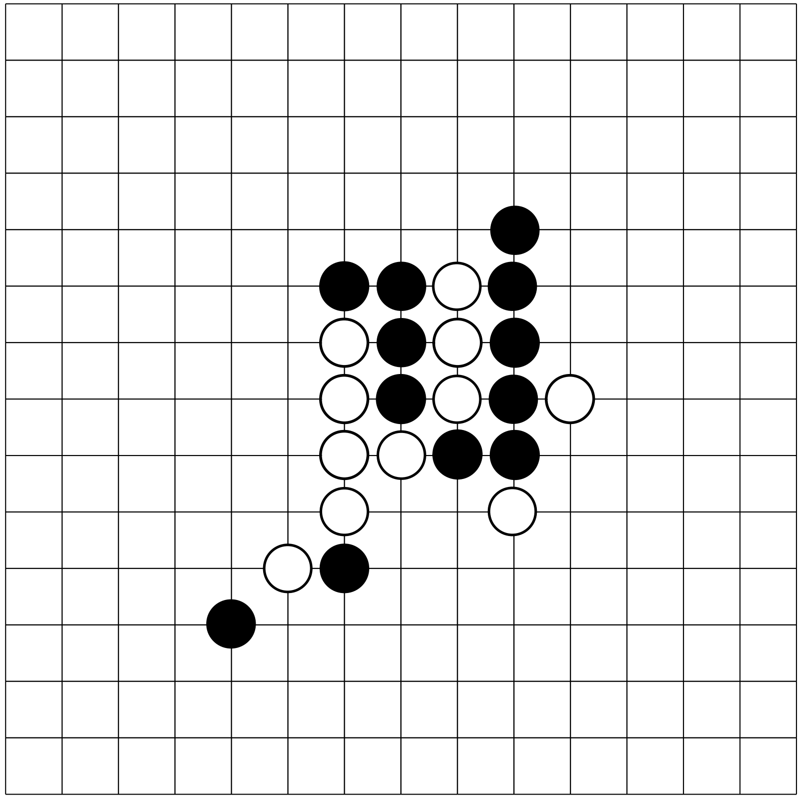
Threat Sequence



Louis Victor Allis and Hj Van Den Herik.
Go-moku and threat-space search.
. Ist. Psu. Edu/, 1993.

Solve Gomoku: Threat-Space Search

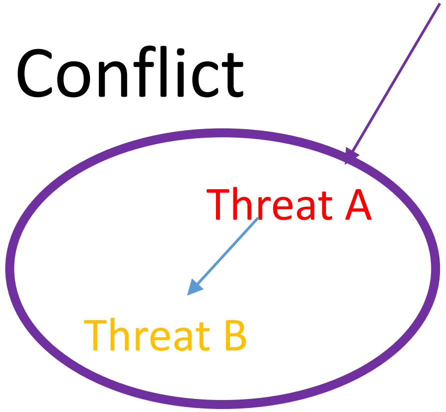
- Threat Sequence
- Winning Threat Sequence

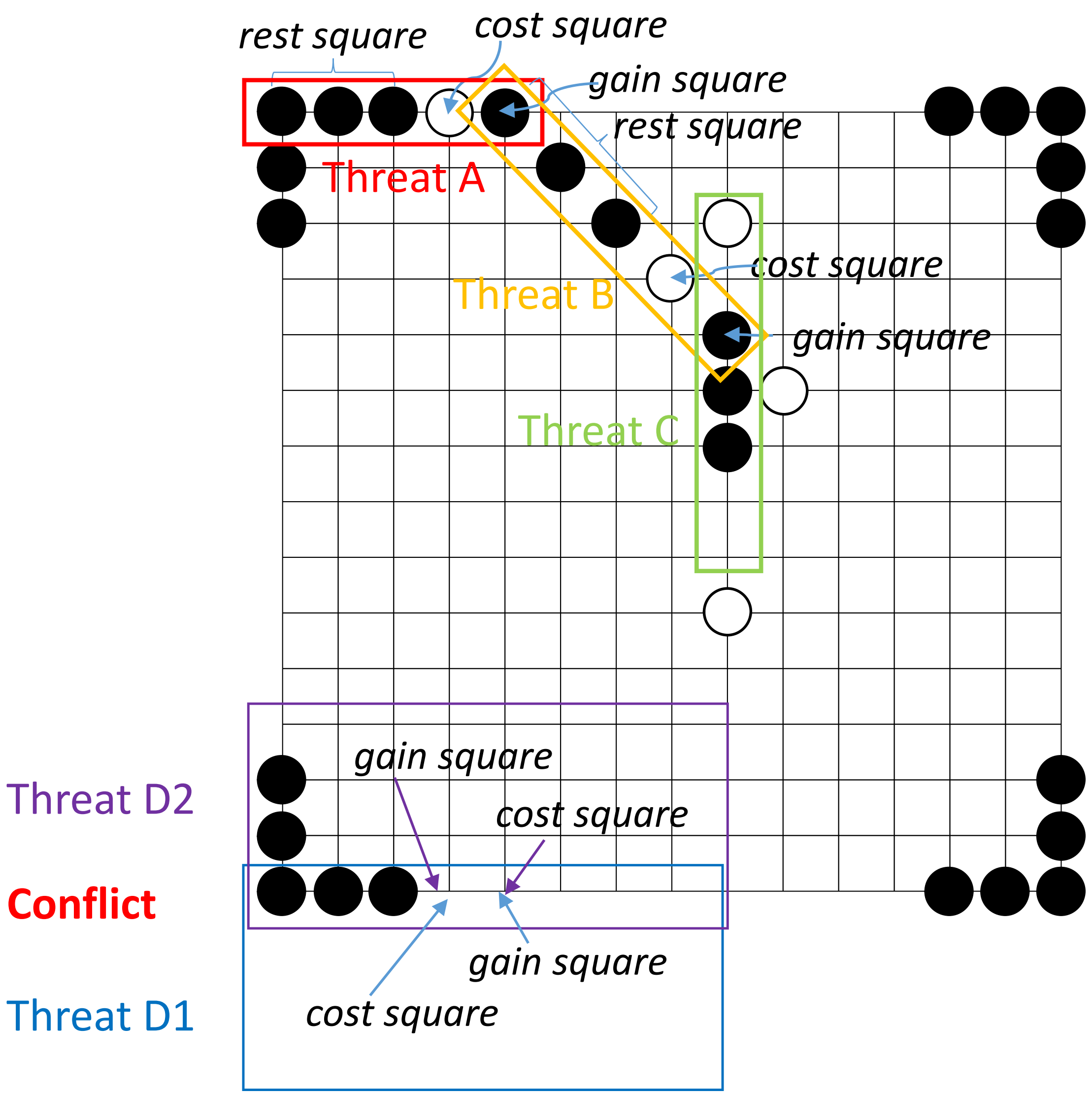


Louis Victor Allis and Hj Van Den Herik.
Go-moku and threat-space search.
. Ist. Psu. Edu/, 1993.

Solve Gomoku: Threat-Space Search

- Gain square
- Cost square
- Rest square
- Dependent
  - Dependency tree
- Conflict



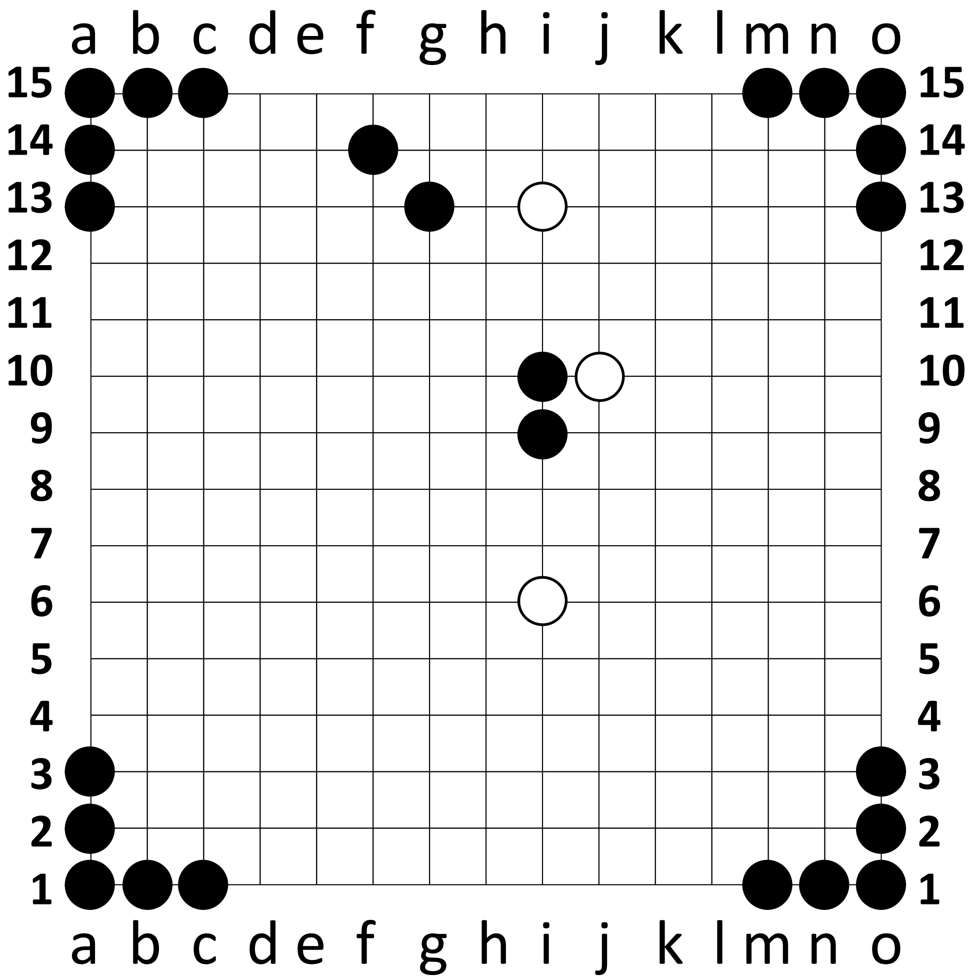


Louis Victor Allis and Hj Van Den Herik.
Go-moku and threat-space search.
. Ist. Psu. Edu/, 1993.

Solve Gomoku: Threat-Space Search

- Threat Search tree:
  - Threat A being independent of threat B is not allowed to occur in the search tree of threat B.
  - Only threats for the attacker are included.

Depth	Type of threat	Gain square	Cost squares
1	Four	*l15*	*k15*
1	Four	*k15*	*l15*
1	Four	*e15*	*d15*
2	Four	*i11*	*h12*
3	Straight Four	*i8*	*i7*
2	Four	*h12*	*i11*
1	Four	*d15*	*e15*
1	Four	*o12*	*o11*
1	Four	*o11*	*o12*
1	Four	*a12*	*a11*
1	Four	*a11*	*a12*
1	Three	*i11*	*i7,i8,i12*
2	Four	*h12*	*e15*
2	Four	*e15*	*h12*
3	Five	*d15*	
1	Three	*i8*	*i7,i11,i12*
1	Four	*o5*	*o4*
1	Four	*o4*	*o5*
1	Four	*l1*	*k1*
1	Four	*k1*	*l1*
1	Four	*e1*	*d1*
1	Four	*d1*	*e1*
1	Four	*a5*	*a4*
1	Four	*a4*	*a5*



Louis Victor Allis and Hj Van Den Herik.
Go-moku and threat-space search.
. Ist. Psu. Edu/, 1993.

Solve Gomoku: Threat-Space Search

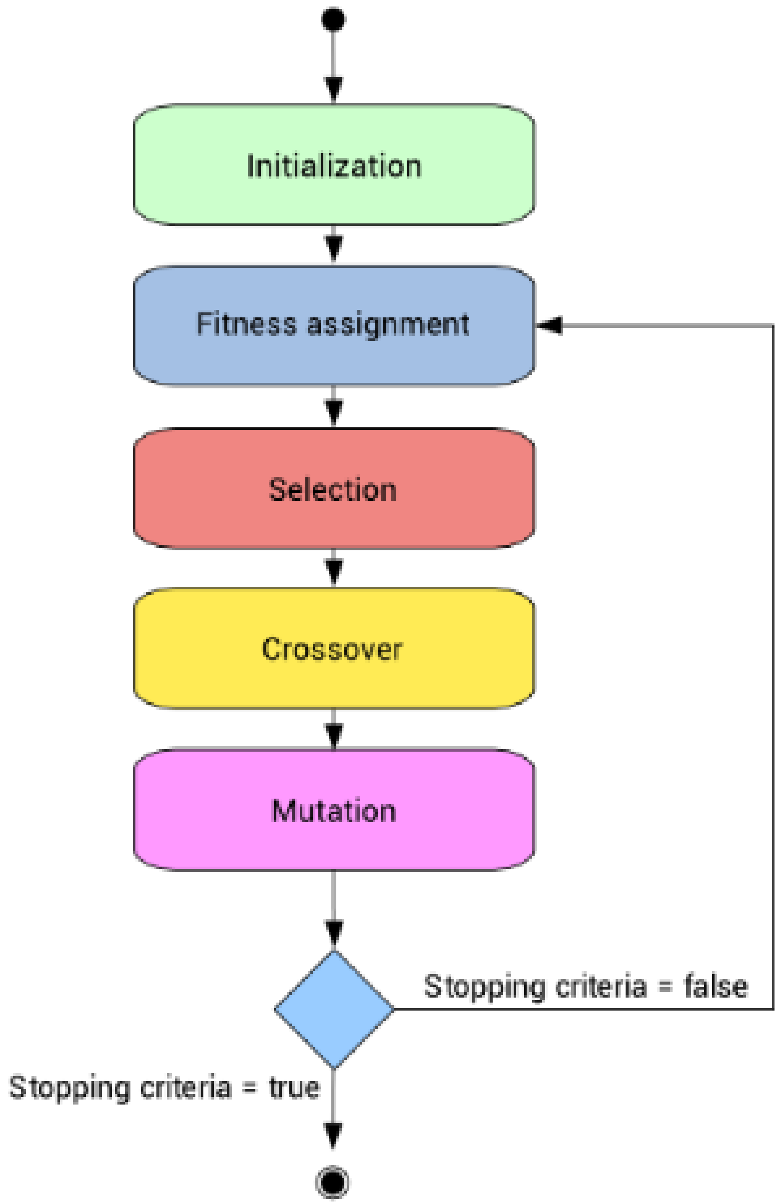
- Victoria
  - Threat-space search
  - Proof-number search
- Threat-Space Search
  - a module capable of quickly determining whether a winning threat sequence exists
  - used as a first evaluation function
    - Win for the attacker
    - No win: proof-number search
  - a heuristic evaluation procedure

Louis Victor Allis and Hj Van Den Herik.
Go-moku and threat-space search.
. Ist. Psu. Edu/, 1993.

- Gomoku Rule
- Solve Gomoku
  - Board Representation
  - Monte Carlo Tree Search
  - Proof-Number Search
  - Threat-Space Search
  - Genetic Algorithm

Solve Gomoku: Genetic Algorithm

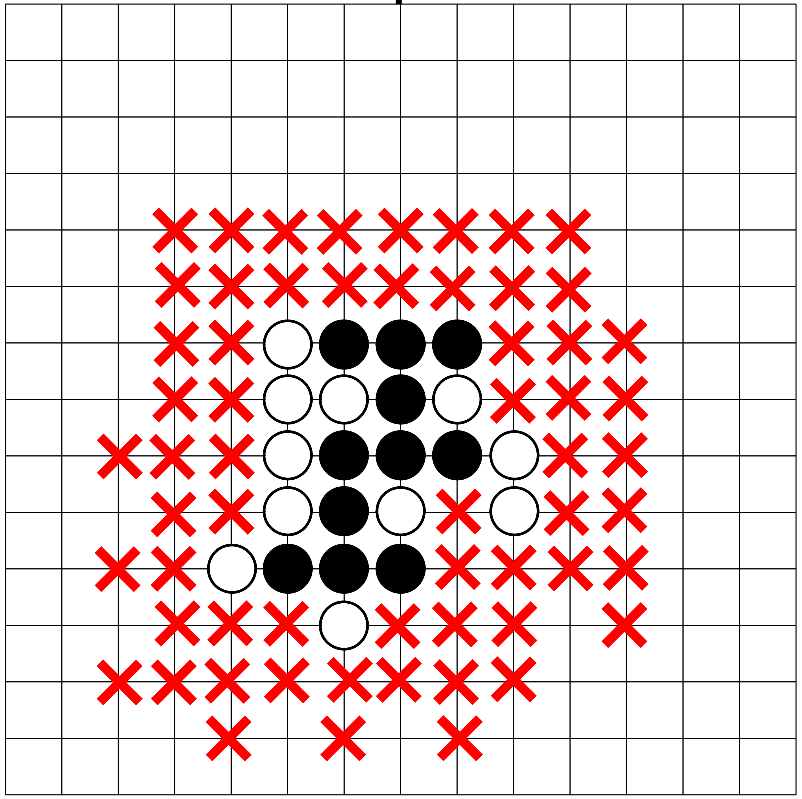
- Initialization: Coding Scheme
- Fitness assignment
- Selection
- Crossover
- Mutation

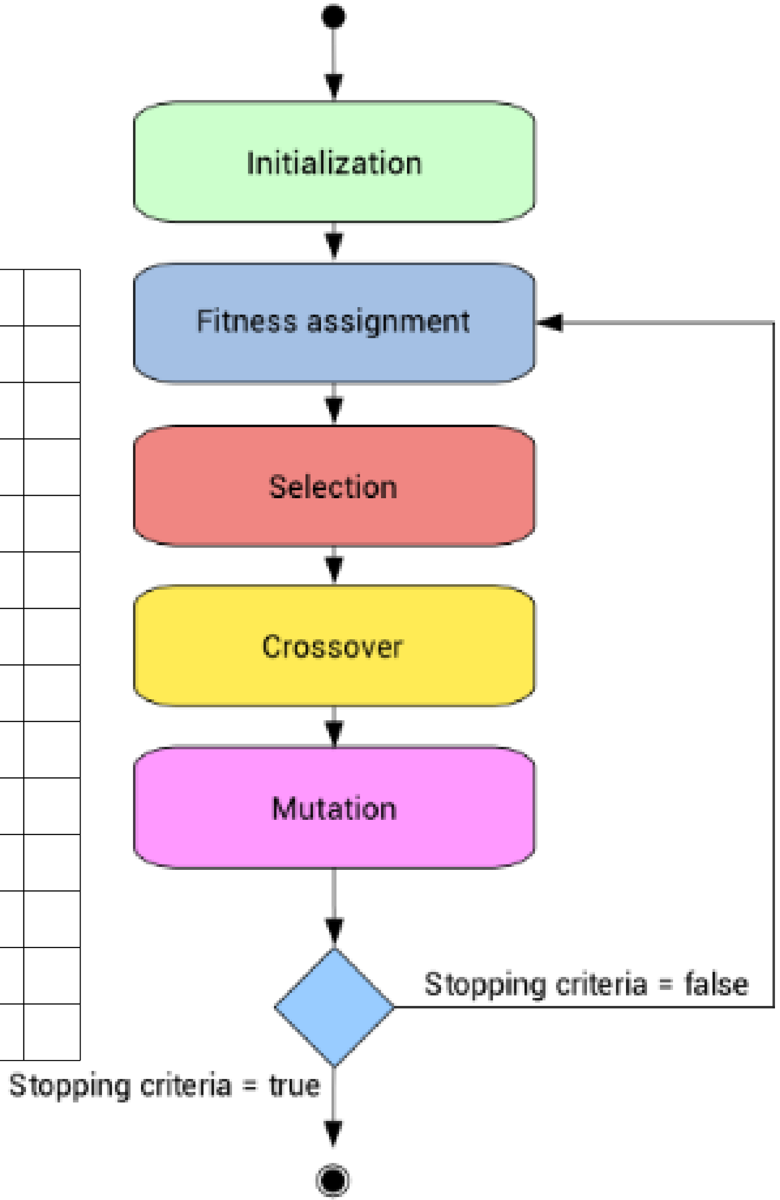


Solve Gomoku: Genetic Algorithm

- Initialization: Coding Scheme
  - Coordinates of consecutive K steps
  - N sequences
  - Representation:
    - $A_2A_1A_7A_3A_8A_5A_6$
    - $A_5A_3A_1A_4A_9A_2A_8$
    - $A_1A_6A_8A_2A_5A_3A_4$
    - $A_2A_9A_3A_4A_7A_8A_6$
    - $A_1A_4A_7A_6A_5A_8A_2$

Both you and the enemy





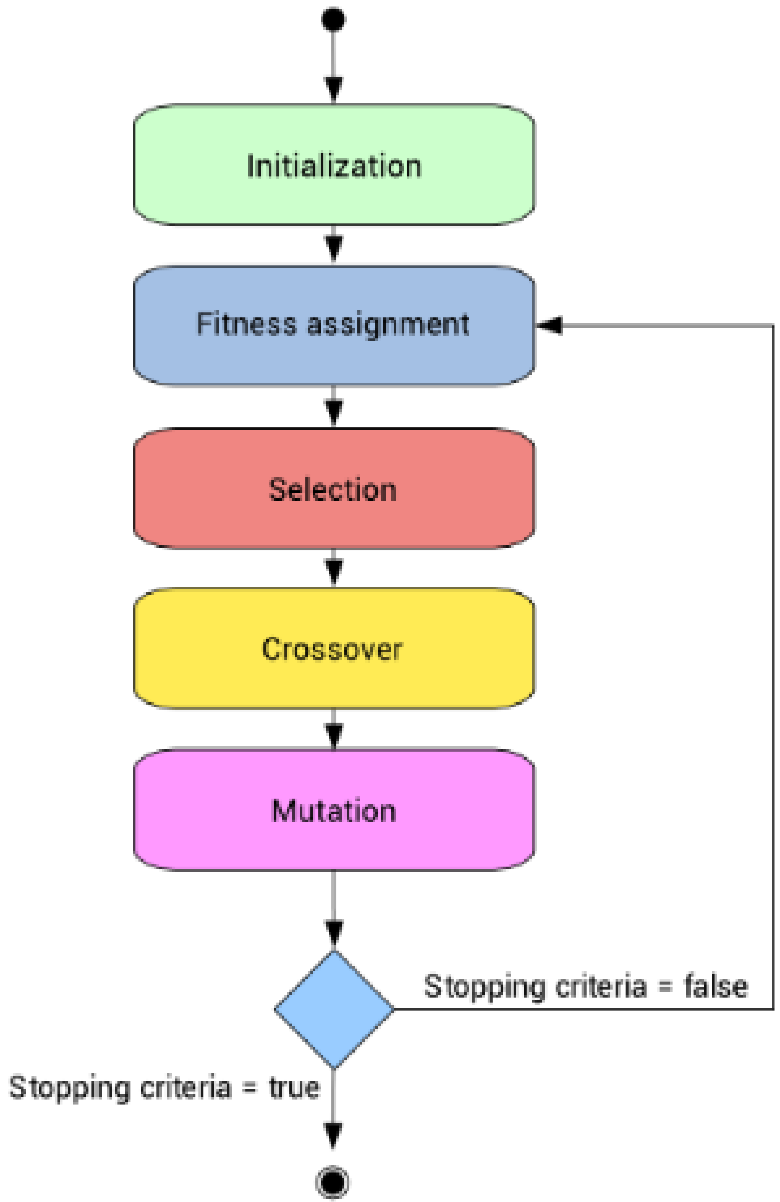
Solve Gomoku: Genetic Algorithm

- Initialization: Coding Scheme
- Fitness assignment
  - Example：

    f(s) = 4800 * (number of four structures in neighborhood)

    + 97 * (number of three structures in neighborhood)

    + 17 * (number of two structures in neighborhood)
- Selection

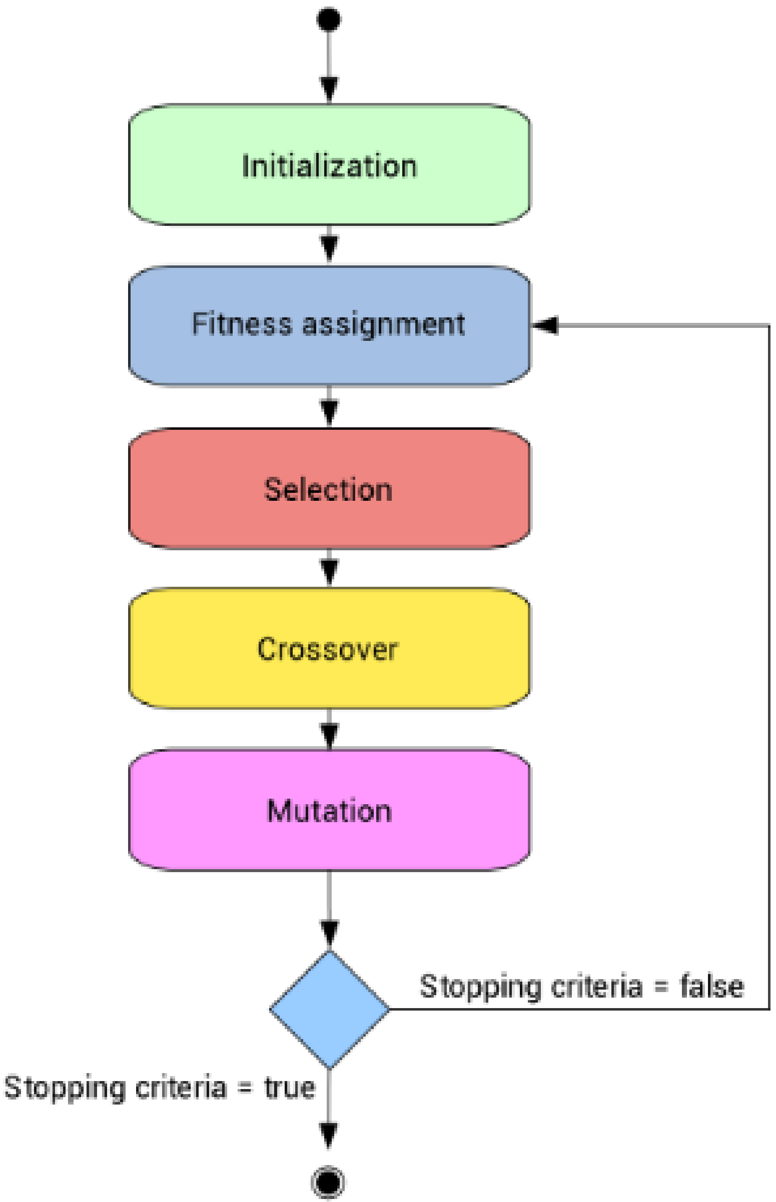


Solve Gomoku: Genetic Algorithm

- Initialization: Coding Scheme
- Fitness assignment
- Selection
- Crossover
  - Parents: $A_2A_1A_7A_3A_8A_5A_6$
    $A_5A_3A_1A_4A_9A_2A_8$
  - Children: $A_2A_1A_7A_3A_9A_2A_8$
    $A_5A_3A_1A_4A_8A_5A_6$
- Mutation

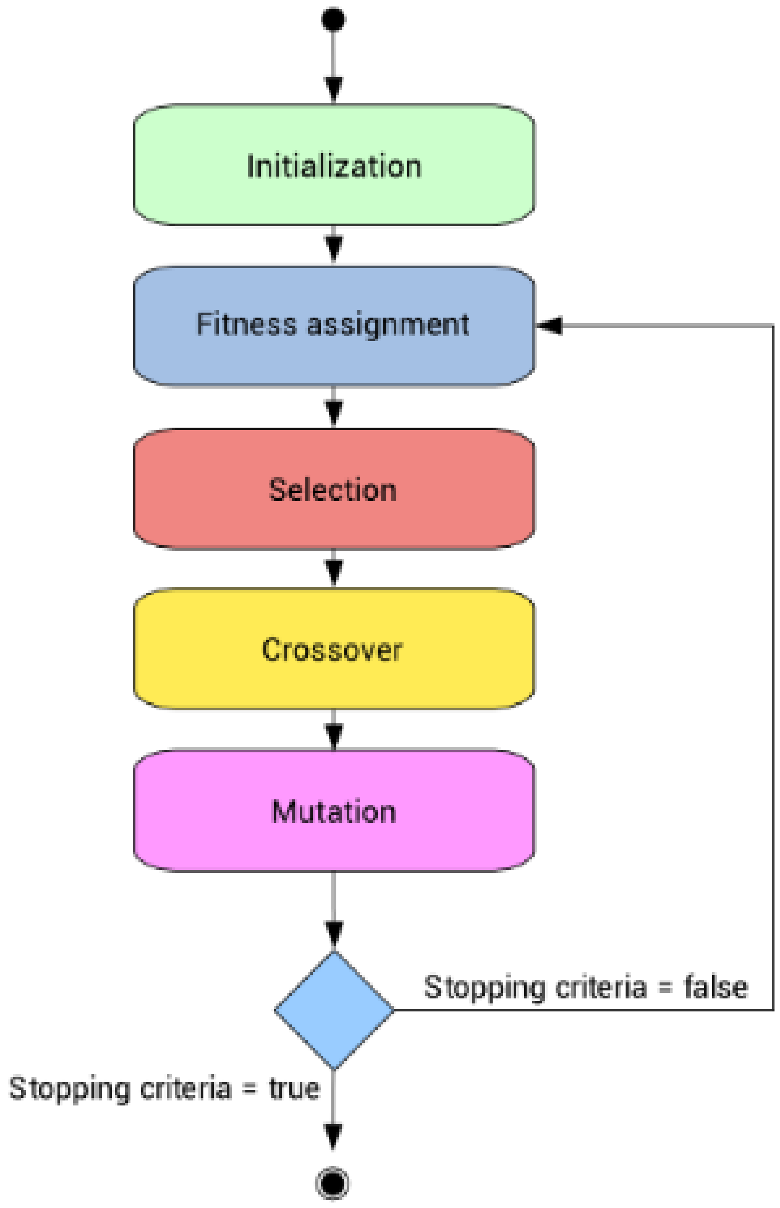
  $A_2A_9A_3A_4A_7A_8A_6$

  $A_2A_9A_7A_8A_3A_4A_6$



Solve Gomoku: Genetic Algorithm

- Initialization: Coding Scheme
- Fitness assignment
- Selection
- Crossover
- Mutation



Gomoku Competition

- Gomoku manager
  - http://gomocup.org/download-gomocup-manager/
- AI
  - http://gomocup.org/download-gomoku-ai/
- Python Template
  - https://github.com/stranskyjan/pbrain-pyrandom
- Gomocup
  - http://gomocup.org/

Outline

- Gomoku
  - Final project
- Alpha-Beta Pruning
  - Submit in class via OJ
- Constraint Satisfaction Problems
  - Take home as an assignment (Project 2)

Lab2: Adversarial Search (Minimax)

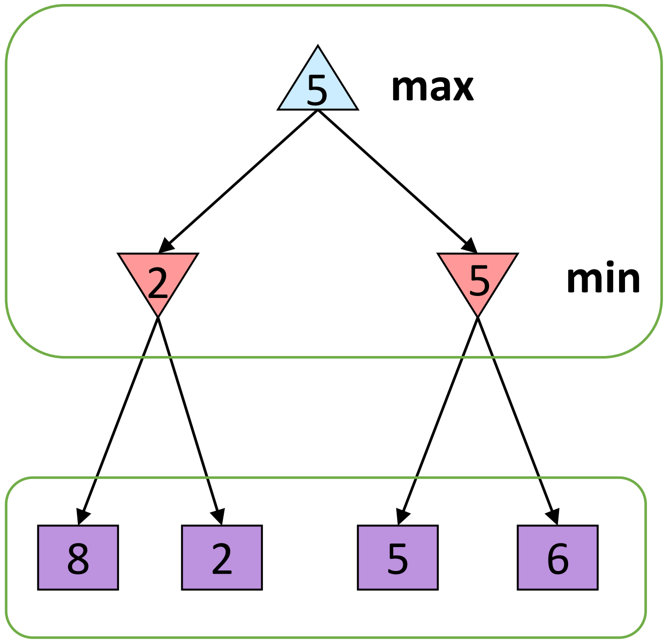

```
function ALPHA-BETA-SEARCH(state) returns an action
   v ← MAX-VALUE(state, −∞, +∞)
   return the action in ACTIONS(state) with value v

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
```

Figure 5.7 The alpha–beta search algorithm. Notice that these routines are the same as the MINIMAX functions in Figure 5.3, except for the two lines in each of MIN-VALUE and MAX-VALUE that maintain α and β (and the bookkeeping to pass these parameters along).

Minimax values: computed recursively



Terminal values: part of the game

Alpha-Beta 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(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
```

For MAX node

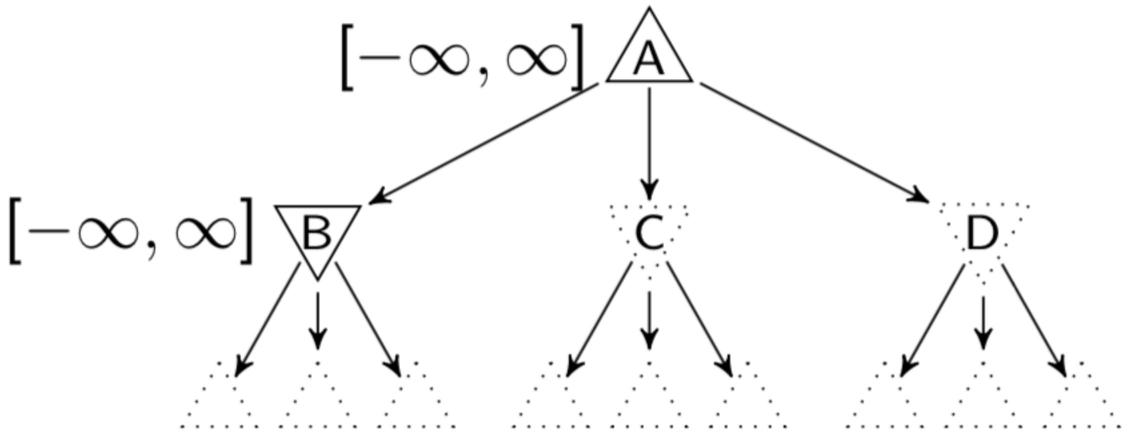
β is fixed as β_{parent}

v is used to update α initialized as α_{parent}

For MIN node

α is fixed as α_{parent}

v is used to update β initialized as β_{parent}



Alpha-Beta 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(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
```

For MAX node

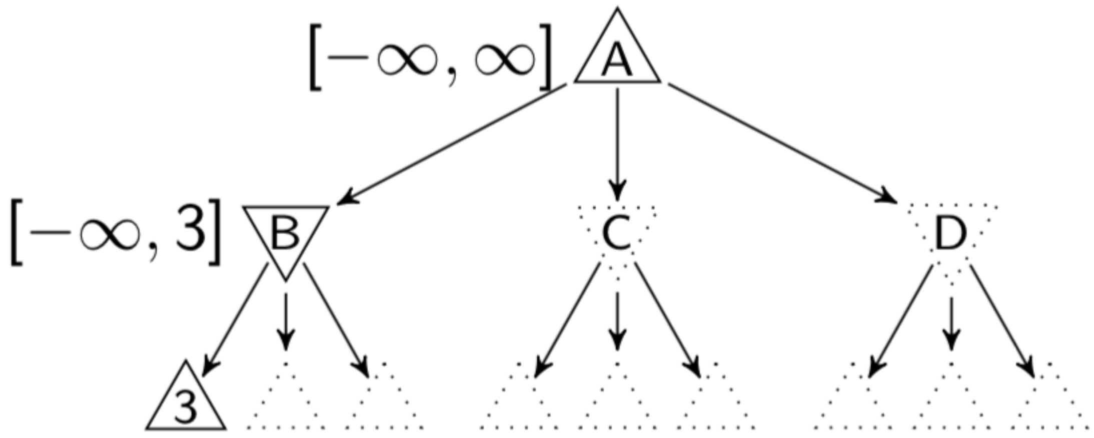
β is fixed as β_{parent}

v is used to update α initialized as α_{parent}

For MIN node

α is fixed as α_{parent}

v is used to update β initialized as β_{parent}



Alpha-Beta Example

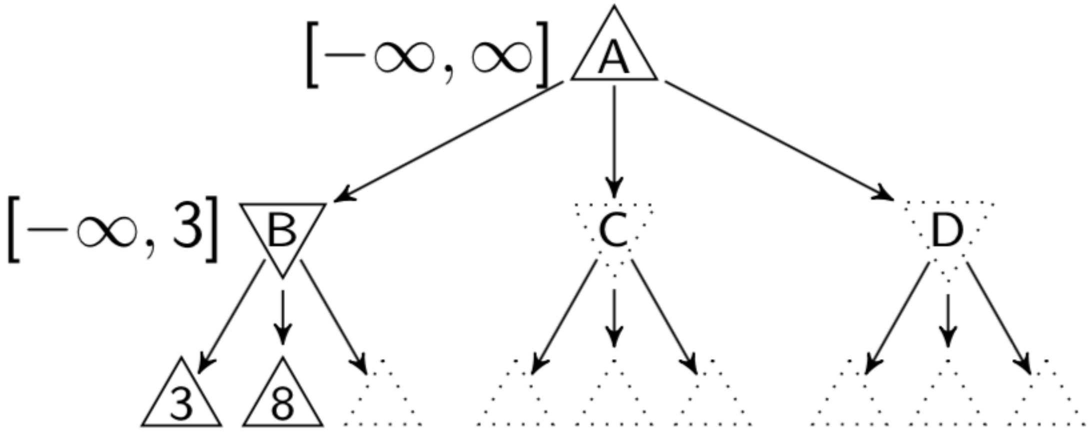
function ALPHA-BETA-SEARCH(*state*) **returns** an action
  $v \leftarrow$ MAX-VALUE(*state*, $-\infty$, $+\infty$)
  return the *action* in ACTIONS(*state*) with value v

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

For MAX node
  β is fixed as β_{parent}
  v is used to update α initialized as α_{parent}

For MIN node
  α is fixed as α_{parent}
  v is used to update β initialized as β_{parent}



Alpha-Beta 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(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
```

For MAX node

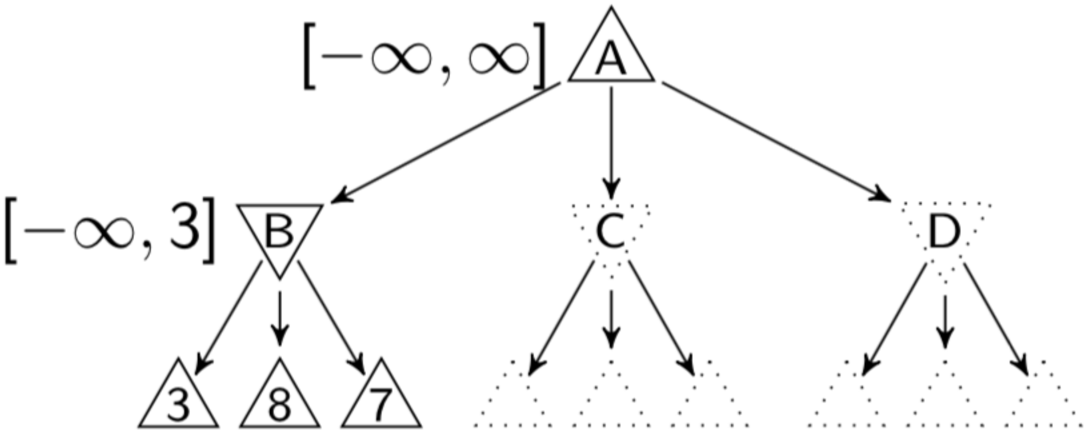
β is fixed as β_{parent}

v is used to update α initialized as α_{parent}

For MIN node

α is fixed as α_{parent}

v is used to update β initialized as β_{parent}



Alpha-Beta Example

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

For MAX node

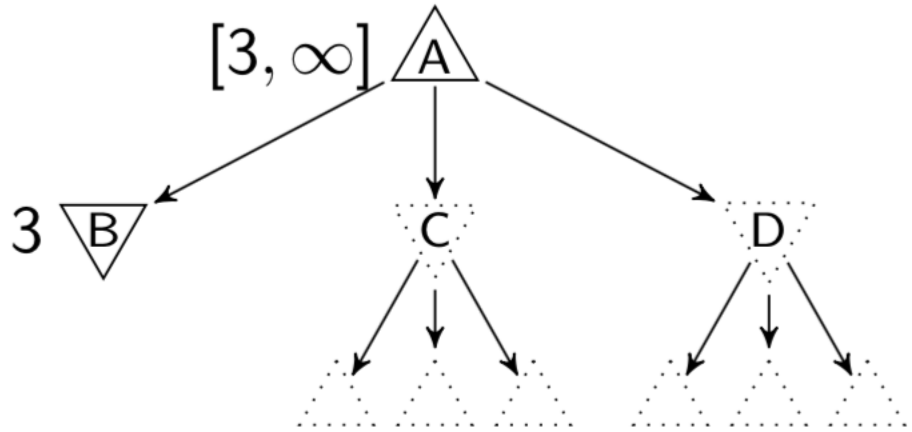
  β is fixed as β_{parent}

  v is used to update α initialized as α_{parent}

For MIN node

  α is fixed as α_{parent}

  v is used to update β initialized as β_{parent}



Alpha-Beta 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(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
```

For MAX node

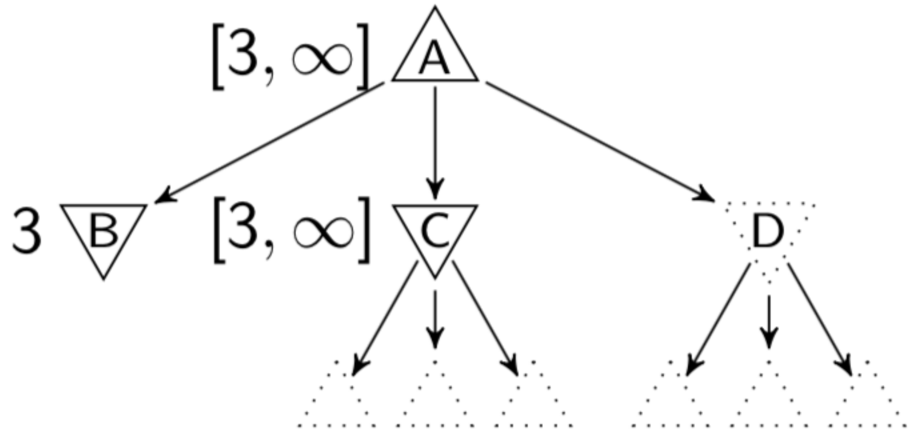
β is fixed as β_{parent}

v is used to update α initialized as α_{parent}

For MIN node

α is fixed as α_{parent}

v is used to update β initialized as β_{parent}



Alpha-Beta 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(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
```

For MAX node

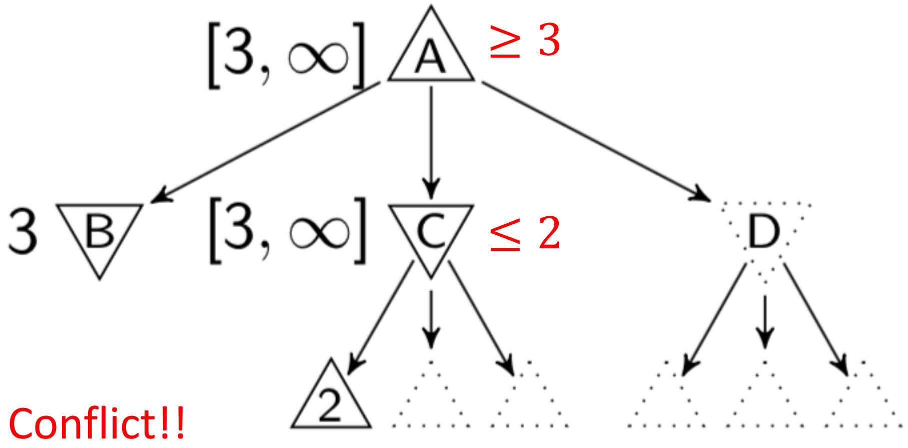
β is fixed as β_{parent}

v is used to update α initialized as α_{parent}

For MIN node

α is fixed as α_{parent}

v is used to update β initialized as β_{parent}



Alpha-Beta Example

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($state, \alpha, \beta$) **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($state, \alpha, \beta$) **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

For MAX node

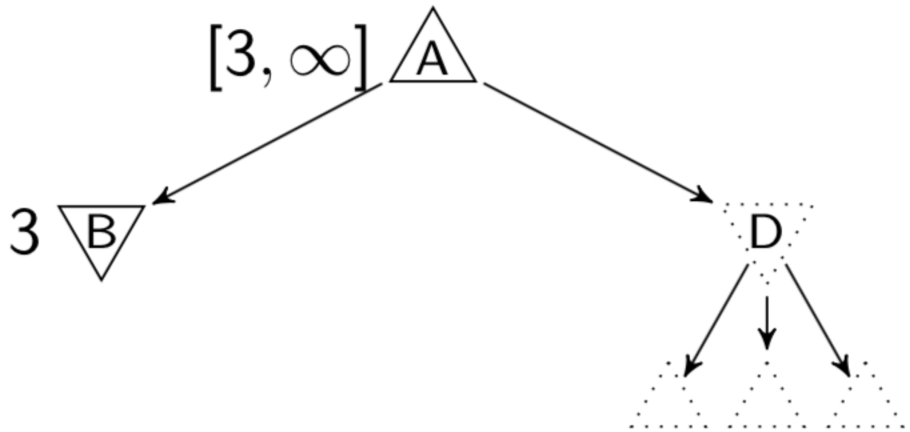
β is fixed as β_{parent}

v is used to update α initialized as α_{parent}

For MIN node

α is fixed as α_{parent}

v is used to update β initialized as β_{parent}

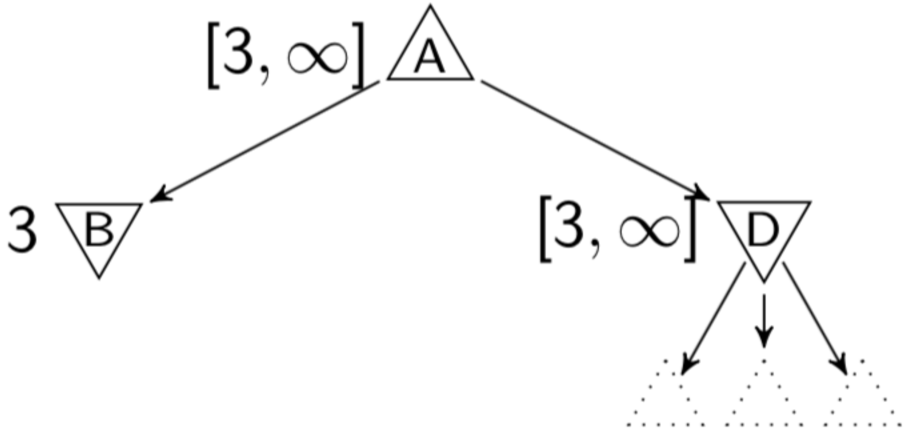


function ALPHA-BETA-SEARCH(*state*) **returns** an action
  $v \leftarrow$ MAX-VALUE($state, -\infty, +\infty$)
  return the *action* in ACTIONS(*state*) with value v

function MAX-VALUE($state, \alpha, \beta$) **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($state, \alpha, \beta$) **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

For MAX node
  β is fixed as β_{parent}
  v is used to update α initialized as α_{parent}
For MIN node
  α is fixed as α_{parent}
  v is used to update β initialized as β_{parent}



Alpha-Beta 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(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
```

For MAX node

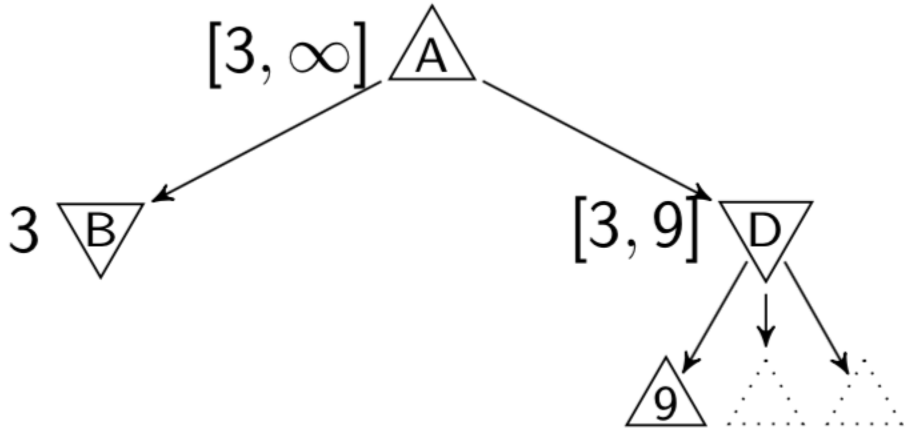
β is fixed as β_{parent}

v is used to update α initialized as α_{parent}

For MIN node

α is fixed as α_{parent}

v is used to update β initialized as β_{parent}



Alpha-Beta Example

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

For MAX node

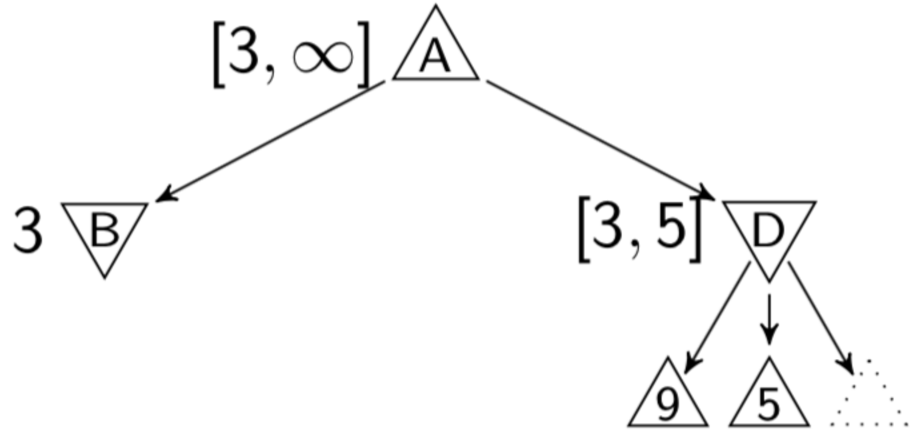
  β is fixed as β_{parent}

  v is used to update α initialized as α_{parent}

For MIN node

  α is fixed as α_{parent}

  v is used to update β initialized as β_{parent}



Alpha-Beta 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(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
```

For MAX node

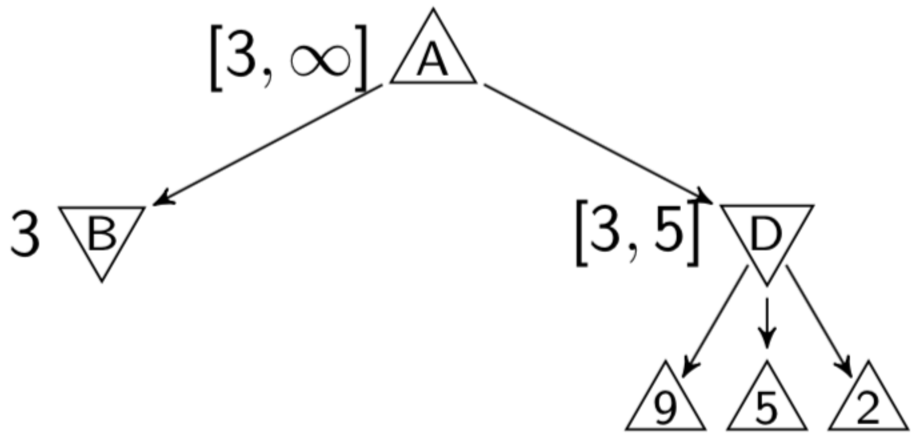
β is fixed as β_{parent}

v is used to update α initialized as α_{parent}

For MIN node

α is fixed as α_{parent}

v is used to update β initialized as β_{parent}



Alpha-Beta 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(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
```

For MAX node

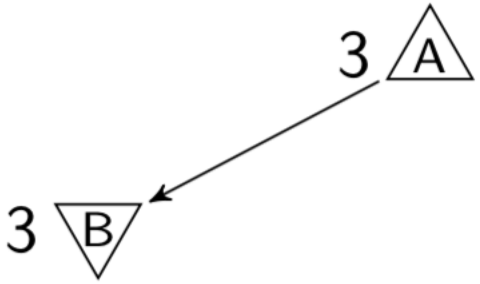
β is fixed as β_{parent}

v is used to update α initialized as α_{parent}

For MIN node

α is fixed as α_{parent}

v is used to update β initialized as β_{parent}

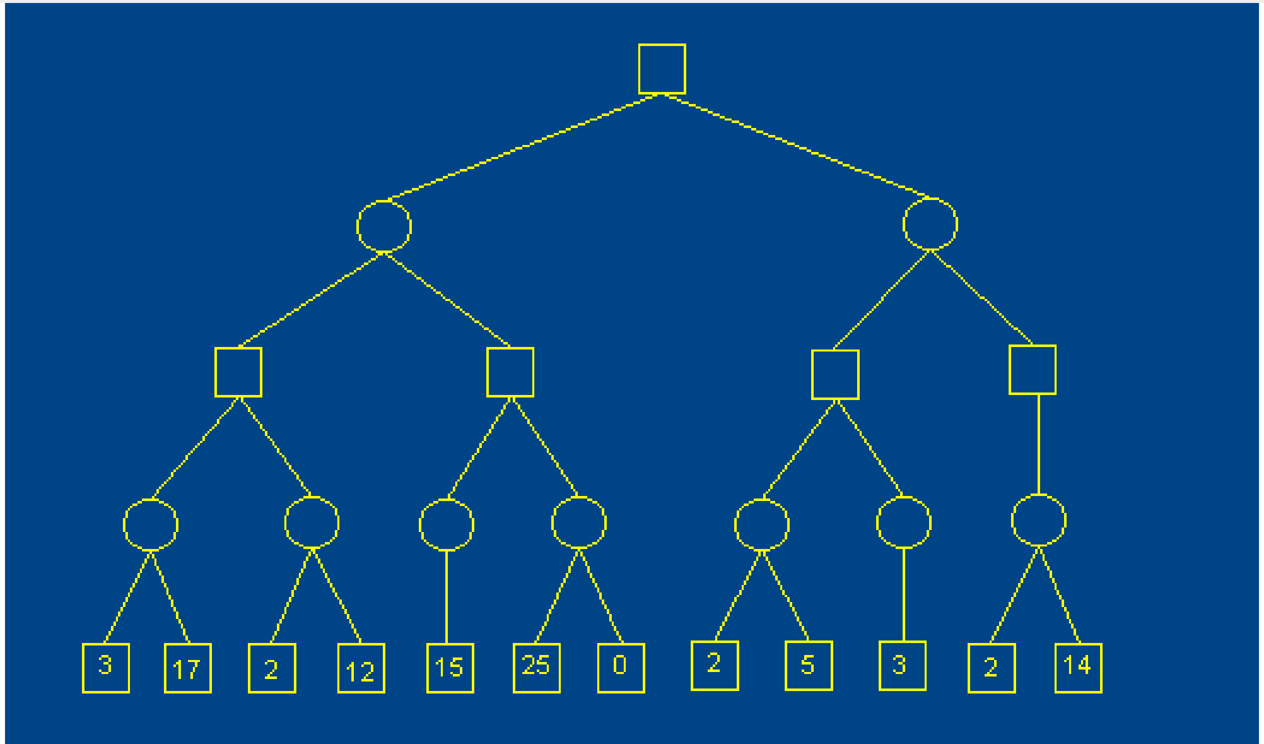


Practice for Alpha-beta Pruning

Input Example:

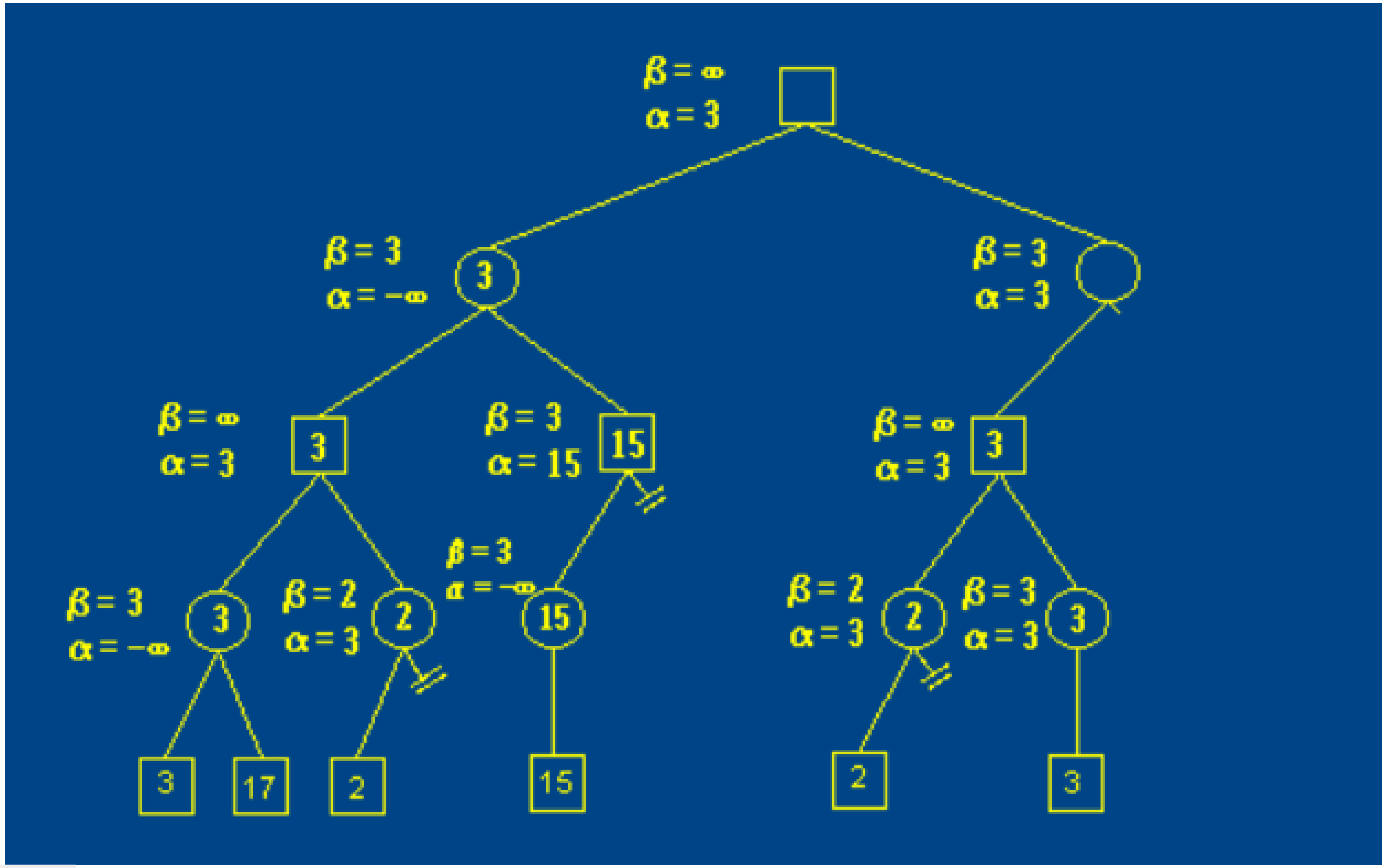
Square represents MAX node while circle stands for MIN node. For each test case, it contains two lines. The first line consists of two integers, the role of the root node (1 for MAX node and 0 for MIN node) and the depth of the tree. The second line is a nested list which stands for the game tree.

```
1 5
[[[[3,17],[2,12]],[[15],[25,0]]],[[[2,5],[3]],[[2,14]]]]
```



Practice for Alpha-beta Pruning

For each test case, the output should include two lines. The first line contains the result for minimax search. The second line should consist of **pruned nodes** in order.




```
3
12 25 0 5 2 14
```

Alpha-Beta Pruning Template


```
rule, n = map(int, input().strip().split())
tree = eval(input().strip())
root_node = construct_tree(n-1, tree, rule)
print(get_value(root_node, float('-inf'), float('inf')))
# print out unvisited nodes
print(' '.join( [str(node) for node in get_unvisited_nodes(root_node)]))
```

- **def** get_value(node, alpha, beta)
  - Choose which function to call
- **def** max_value(node, alpha, beta)
- **def** min_value(node, alpha, beta)

Alpha-Beta Pruning Template

复旦大学大数据学院
School of Data Science, Fudan University

```python
def construct_tree(n, tree, rule):
    """Construct a tree using given information and return the root node.

    Args:
        n: int, the height of tree
        tree: the input tree described with list nested structure
        rule: int, root node's type, 1 for max, 0 for min

    Returns:
        root node

    Hint: tree structure example
        root_node:
            rule: 1 (MAX node)
            is_leaf: False
            value: 5
            visited: bool, visited or not
            successor: [child1, child2, child3, ...]
                and each child has similar structure of root_node
    """
    node = Node(rule=rule)
    successors = []
    if n == 1:  # leaf
        for t in tree:
            successors.append(Node(rule=1-rule, is_leaf=True, value=t))
    else:  # sub-tree
        for t in tree:
            successors.append(construct_tree(n-1, t, 1-rule))
    node.successor = successors
    return node
```

```python
class Node:
    """Node of the tree.

    Attributes:
        rule: int, 0 or 1, 1 for MAX node and 0 for MIN node
        successor: list of Node representing children of the current node
        is_leaf: bool, whether the node is a leaf or not
        value: value of the node
        visited: bool, visited or not

    Hint:
        We use this class to construct a tree in construct_tree method.
    """
    def __init__(self, rule=0, successor=None, is_leaf=False, value=None):
        if successor is None:
            successor = []
        self.rule = 'max' if rule == 1 else 'min'
        self.successor = successor
        self.is_leaf = is_leaf
        self.value = value
        self.visited = False

def get_unvisited_nodes(node):
    """Get unvisited nodes for the tree.

    Args:
        node: class Node object, root node of the current tree (or leaf)

    Returns:
        float list of values of the unvisited nodes.
    """
    unvisited = []
    if node.successor:
        for successor in node.successor:
            unvisited += get_unvisited_nodes(successor)
    else:
        if not node.visited:
            unvisited.append(node.value)
    return unvisited
```

What You Need to Do for Alpha-Beta Pruning

- Implement the following 3 functions:
  - **def** get_value(node, alpha, beta)
    - Choose which function to call
  - **def** max_value(node, alpha, beta)
  - **def** min_value(node, alpha, beta)

Outline

- Gomoku
  - Final project
- Alpha-Beta Pruning
  - Submit in class via OJ
- Constraint Satisfaction Problems
  - Take home as an assignment (Project 2)

Project 2

- You need to submit your own version of code.
- You are encouraged to discuss with your group members. It might take some time to get familiar with all the supportive codes.
- **Homework 2 is due on Nov 18rd, Wednesday, 11:55pm, 2020.**