DATA130008 Introduction to Artificial Intelligence

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

张霁雯

Gomoku Tutorial 1

November 02th 2021

Gomoku

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

Gomoku

- **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 × 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

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

Solve Gomoku

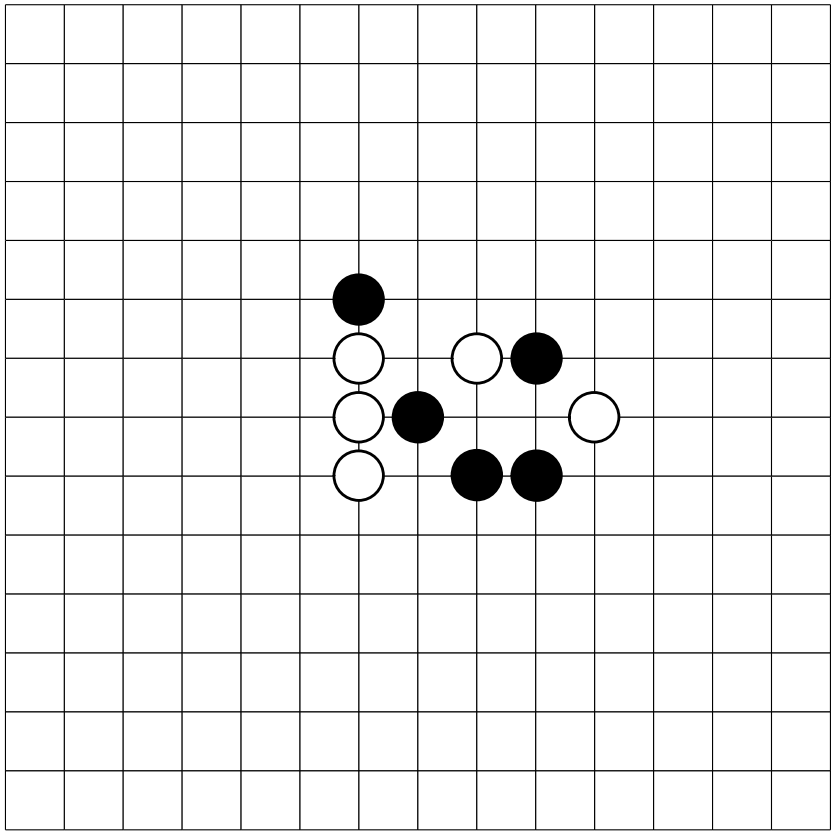
- Input
  - Current board state
- Goal
  - Search for next step

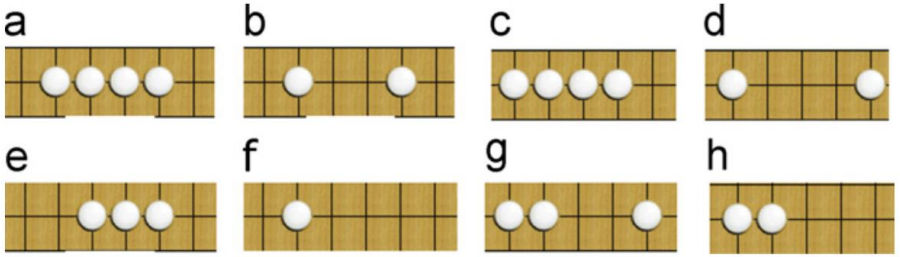
Gomoku

- **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



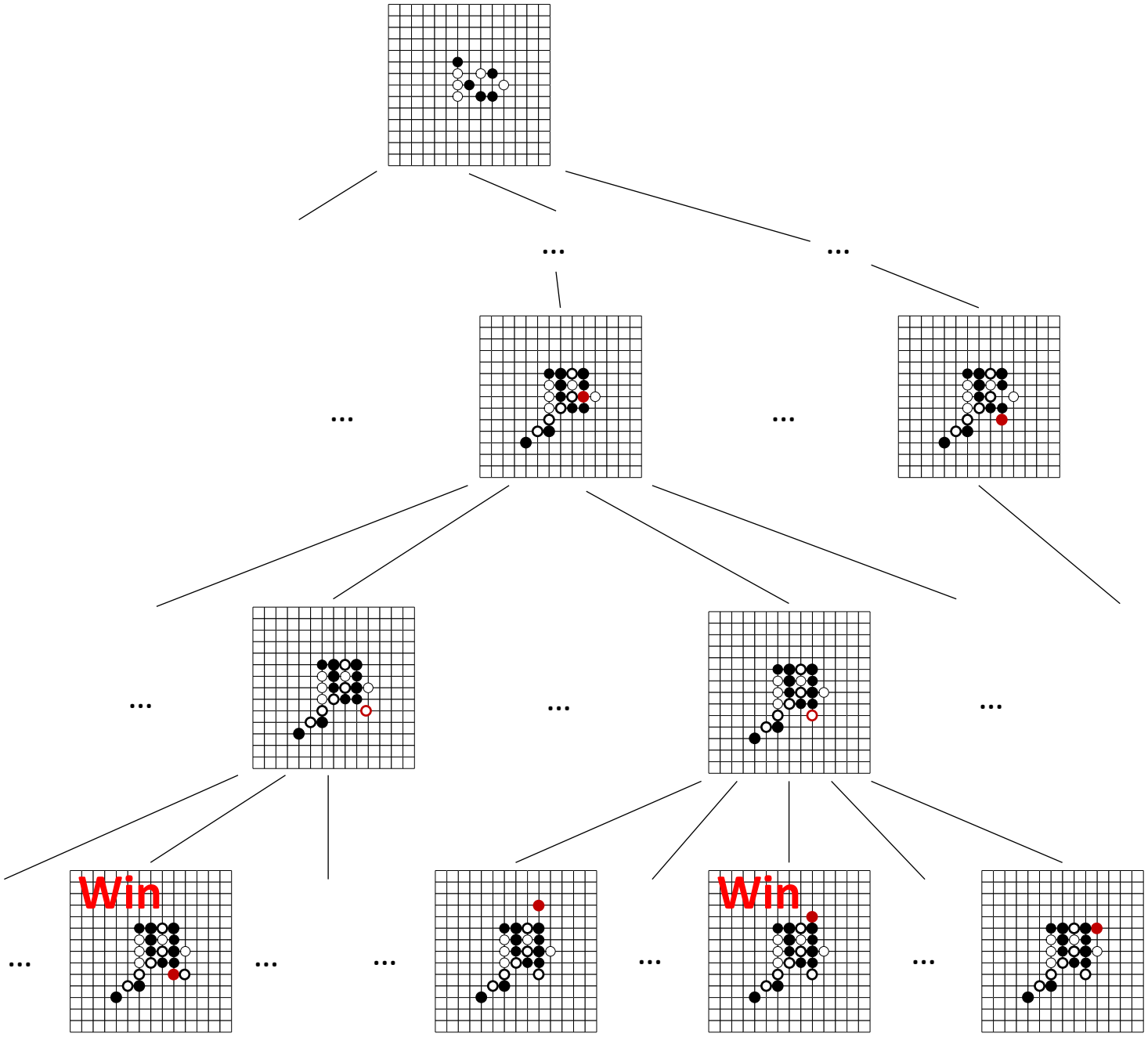


Dongbin Zhao, Zhen Zhang, and Yujie Dai.
Self-teaching adaptive dynamic programming for Gomoku.
Neurocomputing, 78(1):23–29, 2012.

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

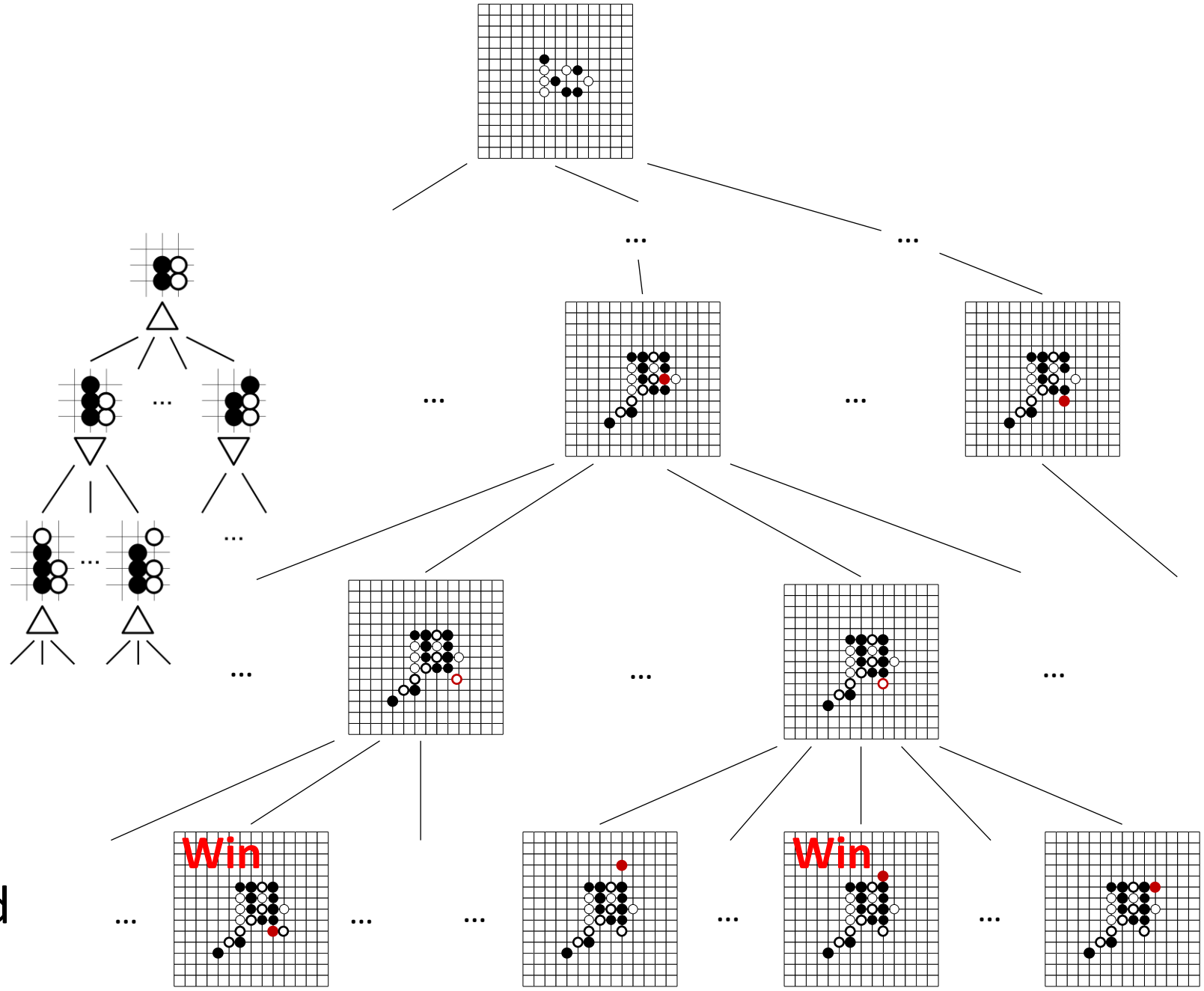
Solve Gomoku

- 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

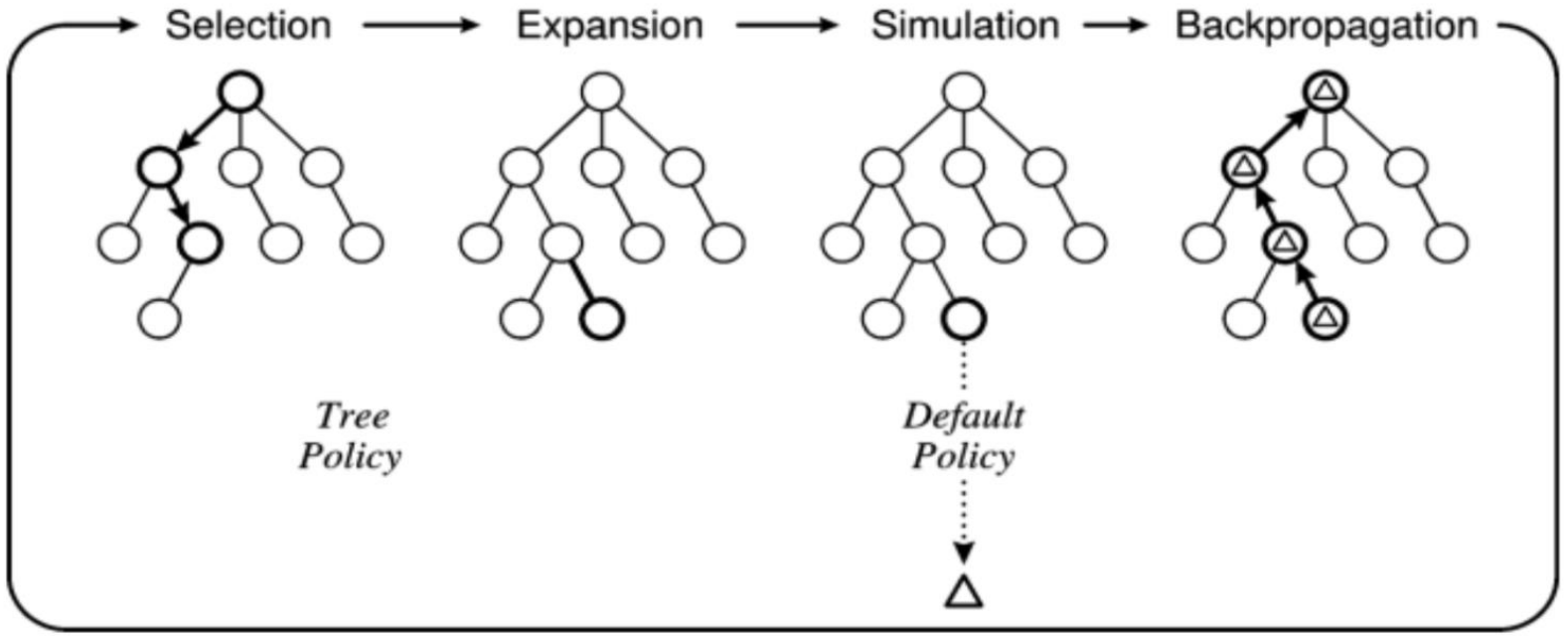


Gomoku

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

Solve Gomoku: Monte Carlo Tree Search

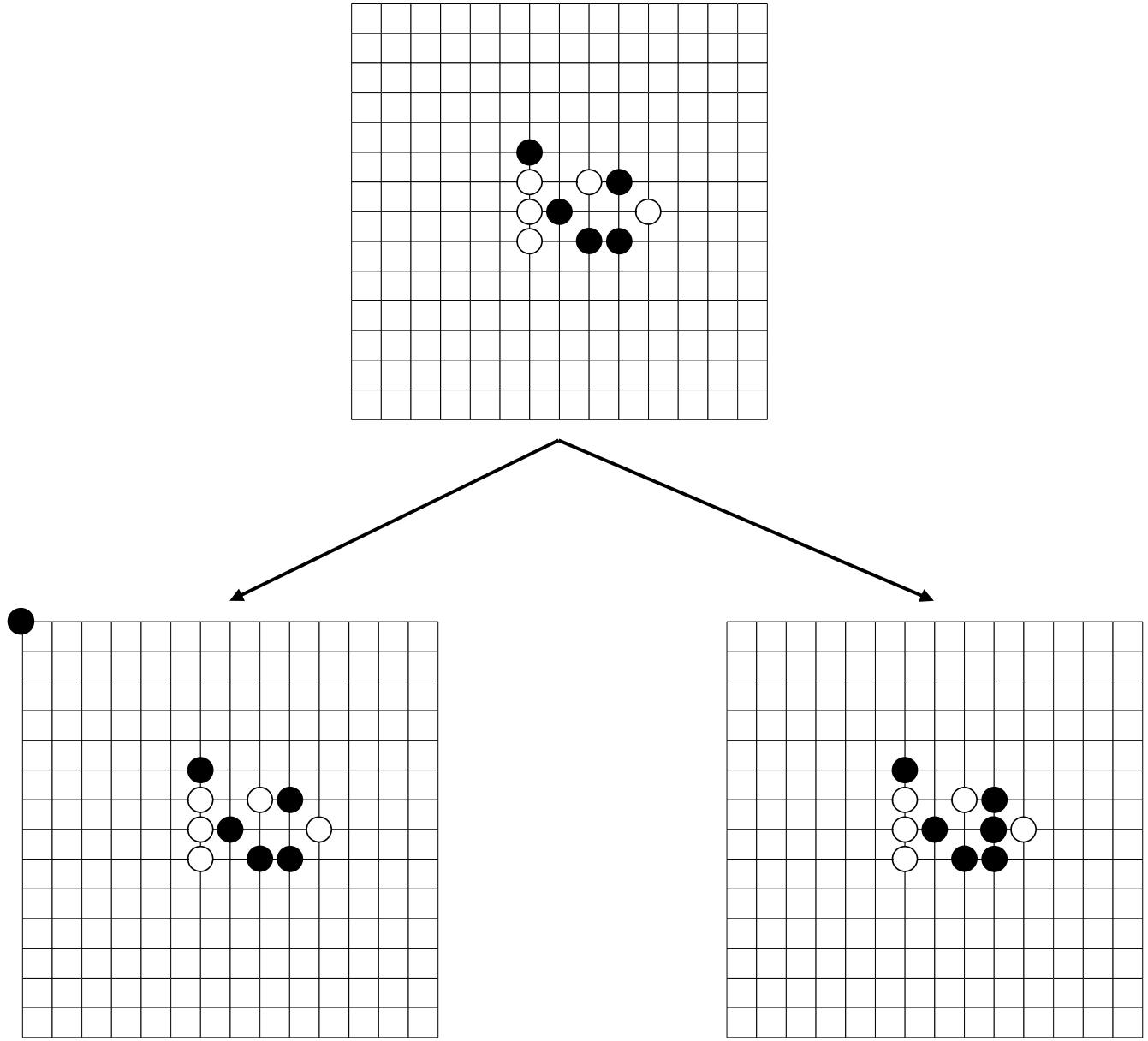
- Monte Carlo Tree Search
  - Simulation, Expectation
  - Steps
    - Selection
    - Expansion
    - Simulation
    - Backpropagation



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

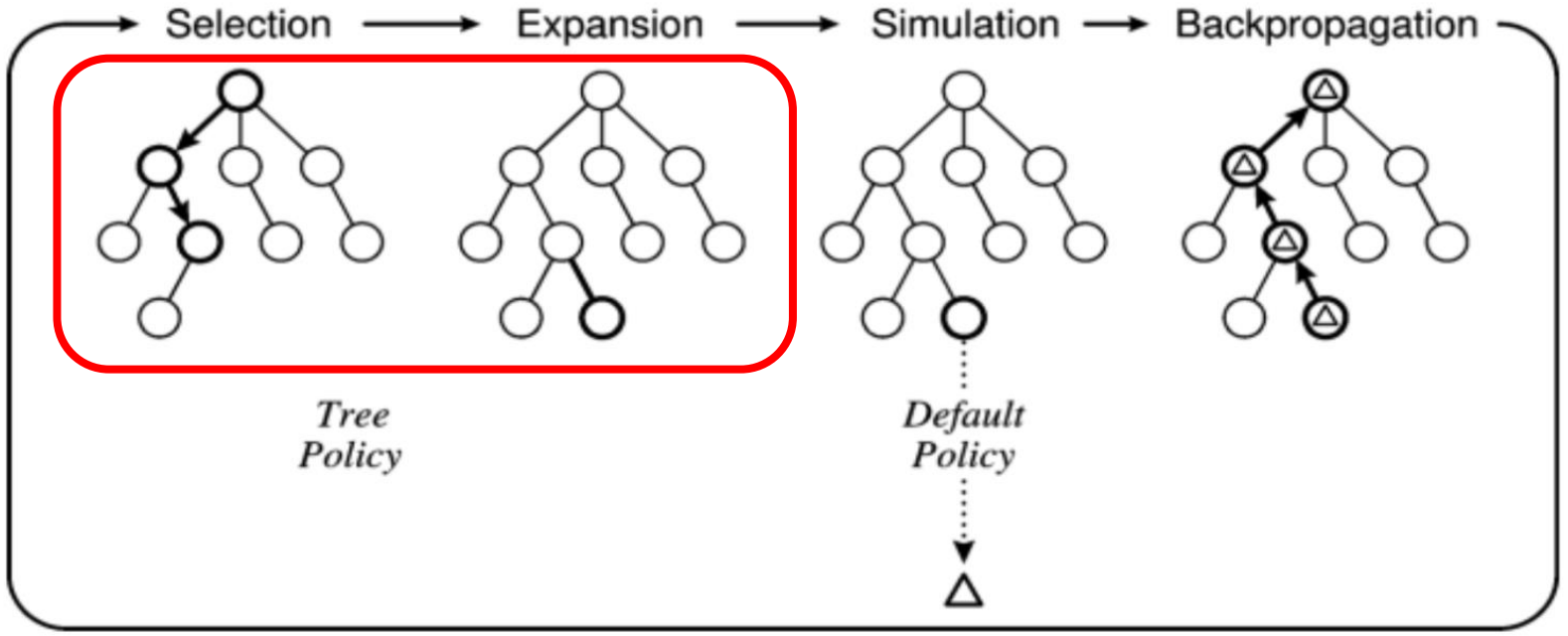
- Monte Carlo Tree Search
  - Simulation, Expectation
  - Steps
    - Selection
    - Expansion
    - Simulation
    - Backpropagation



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

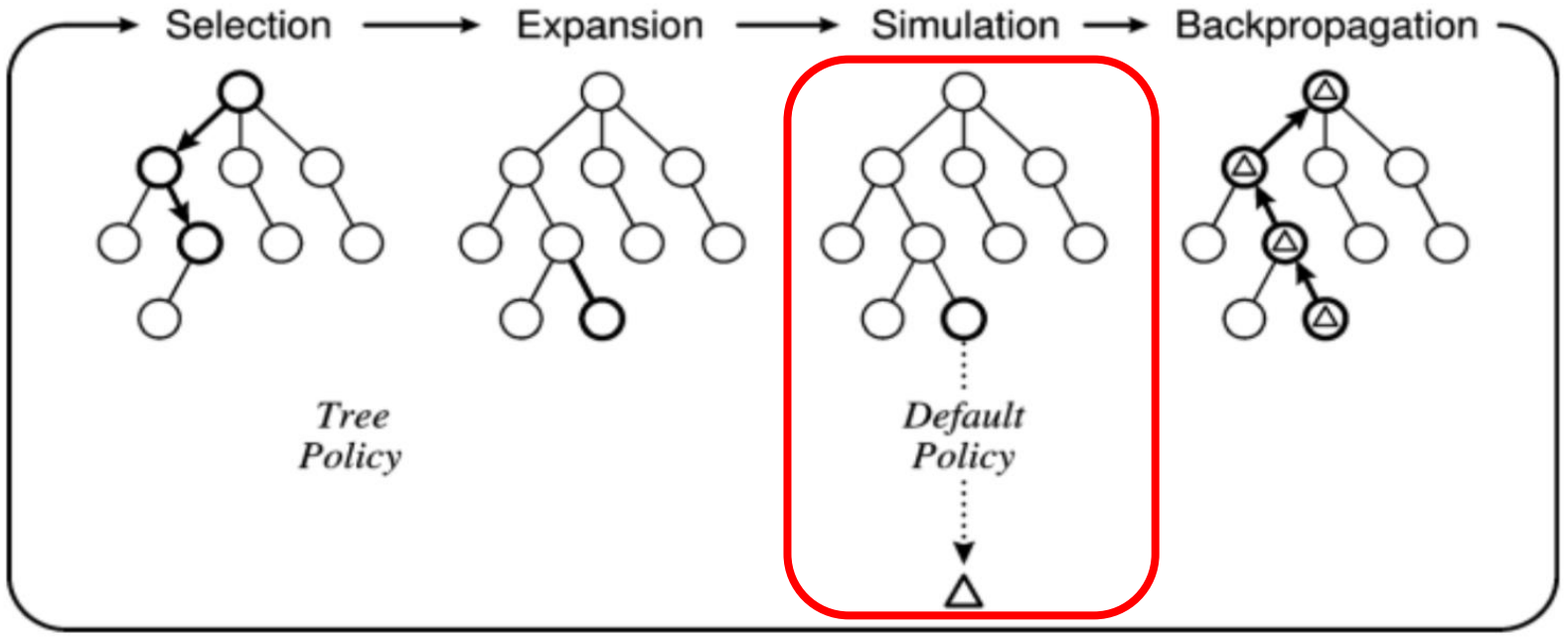
- Monte Carlo Tree Search
  - Simulation, Expectation
  - Steps
    - Selection
    - Expansion
    - Simulation
    - Backpropagation



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

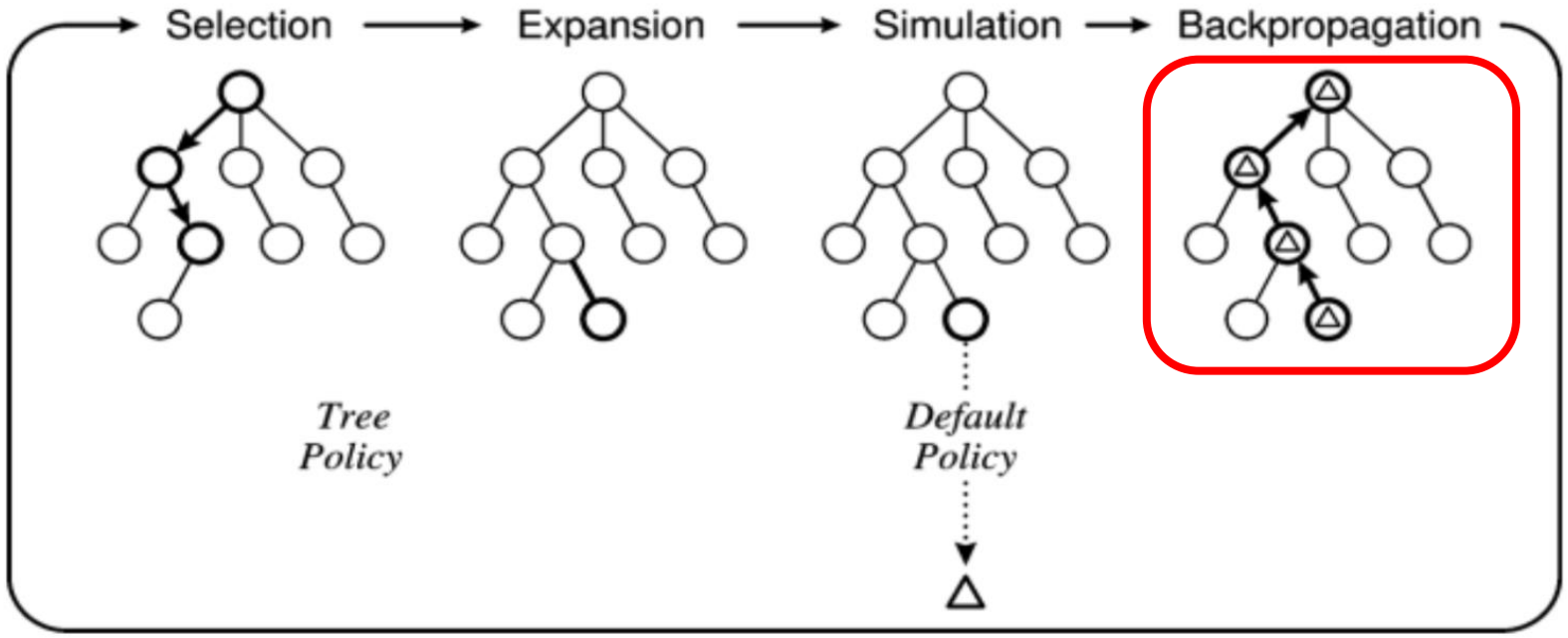
- Monte Carlo Tree Search
  - Simulation, Expectation
  - Steps
    - Selection
    - Expansion
    - Simulation
    - Backpropagation



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.

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

- **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

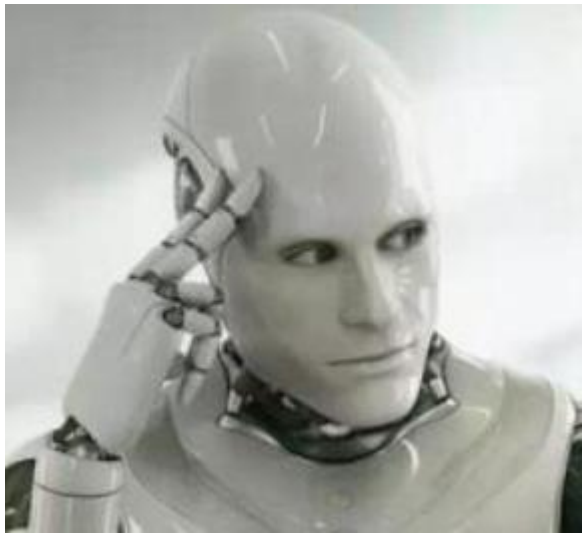
Zhentao Tang, Dongbin Zhao, Kun Shao, and Le Lv.
ADP with MCTS algorithm for Gomoku.
2016 IEEE Symp. Ser. Comput. Intell. SSCI 2016, (61273136), 2017.

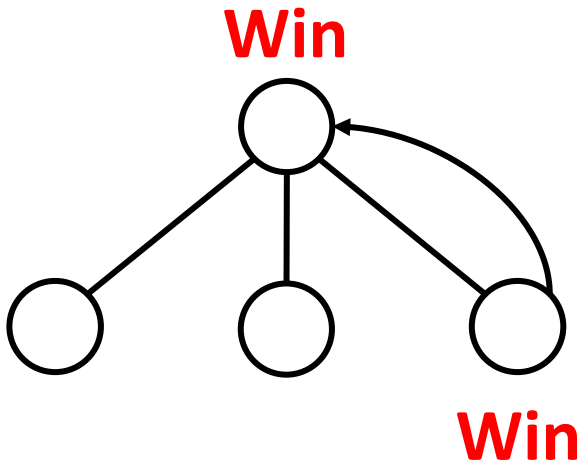
Gomoku

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

Solve Gomoku: Proof-Number Search

- When will the Black win?

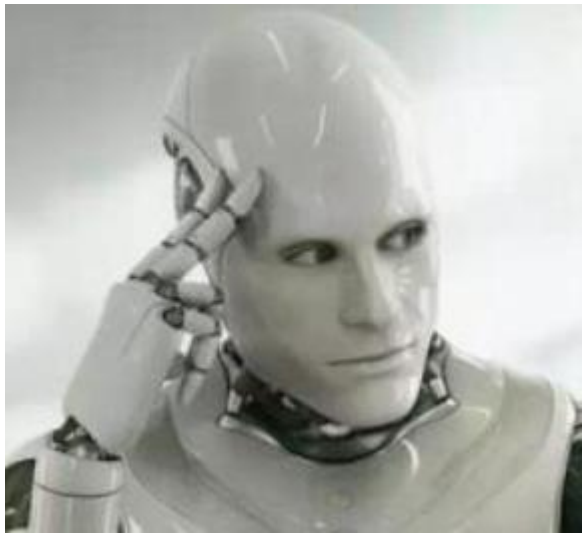


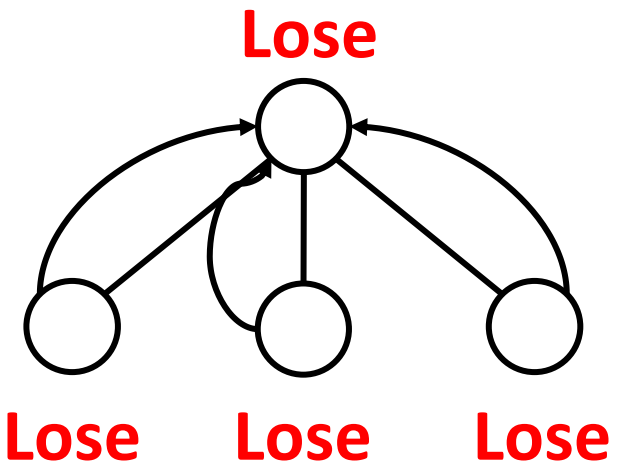


Louis Victor Allis.
Searching for Solutions in Games and Artificial Intelligence.
1994.

Solve Gomoku: Proof-Number Search

- 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.

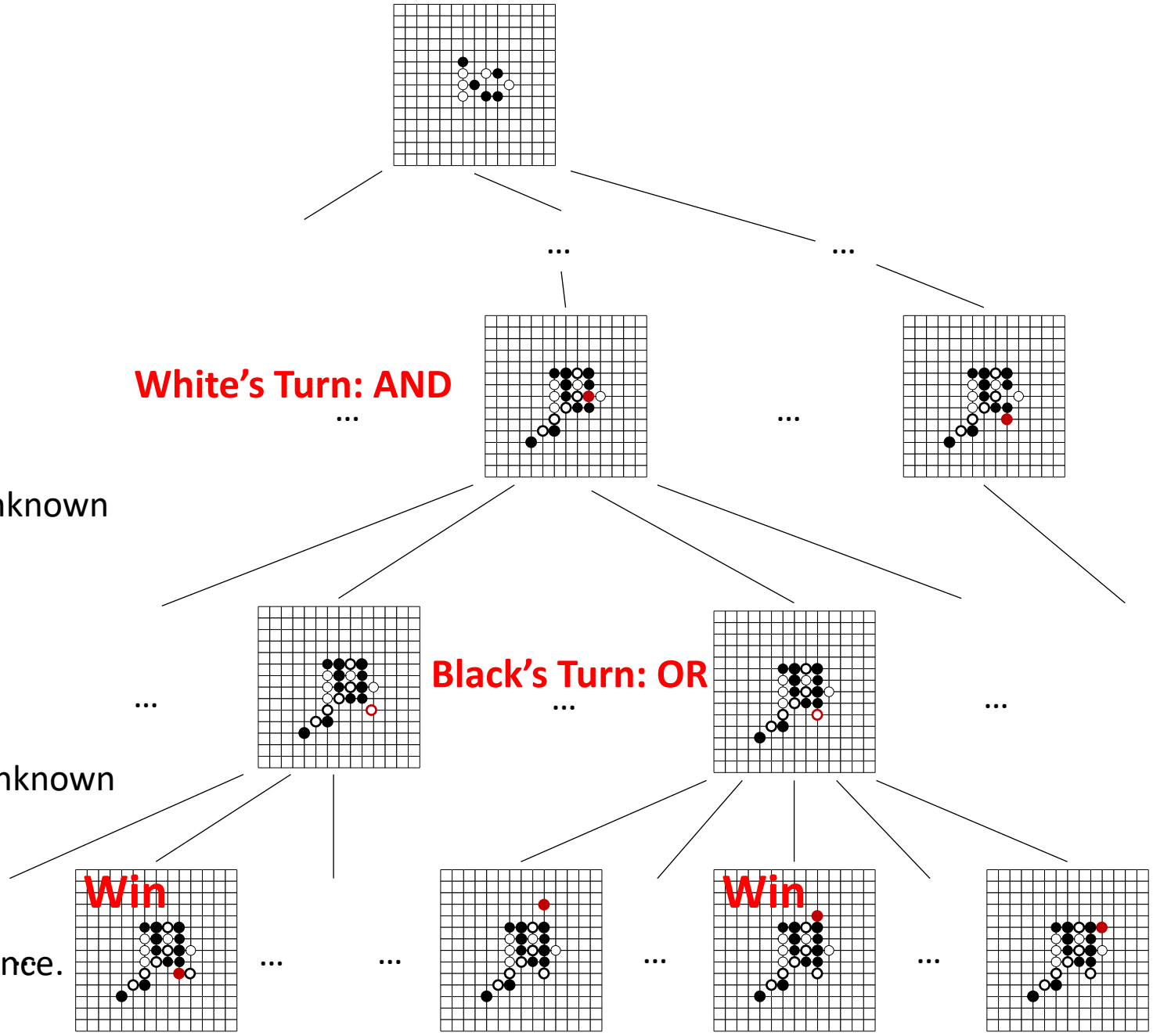
Solve Gomoku: Proof-Number Search

- 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

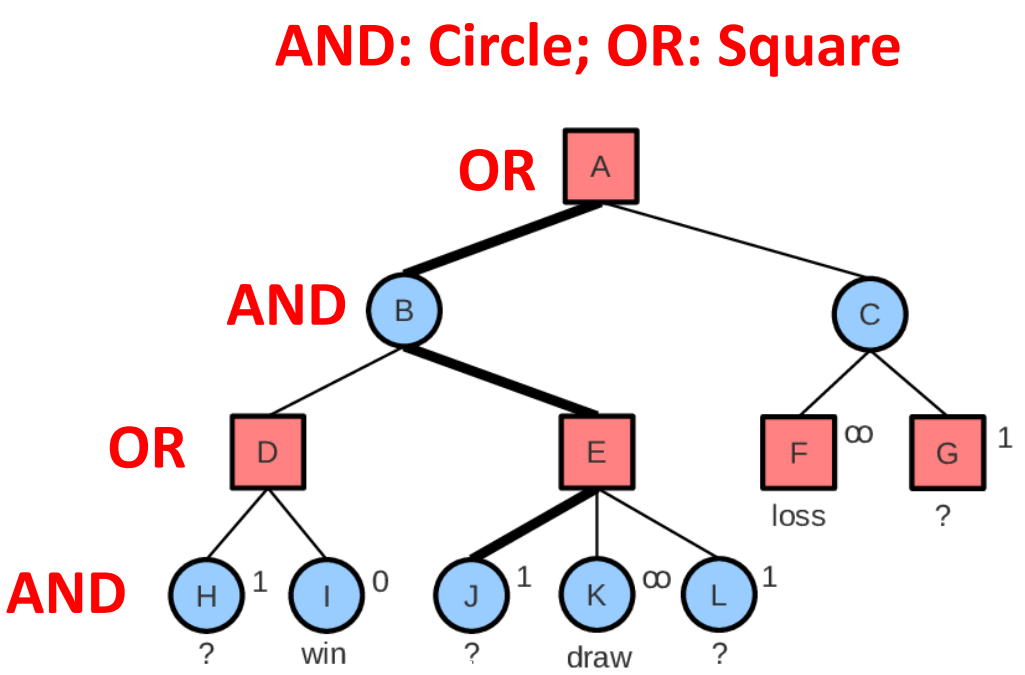
- AND/OR Tree
  - 3 Values: true, false, unknown
  - 2 Nodes: AND, OR
    - Black: OR
      - Win if one child is win
      - Unknown if no win and has unknown
      - Lose if all children are lose
    - White: AND
      - Lose if one child is lose
      - Unknown if no lose and has unknown
      - Win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



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

Solve Gomoku: Proof-Number Search

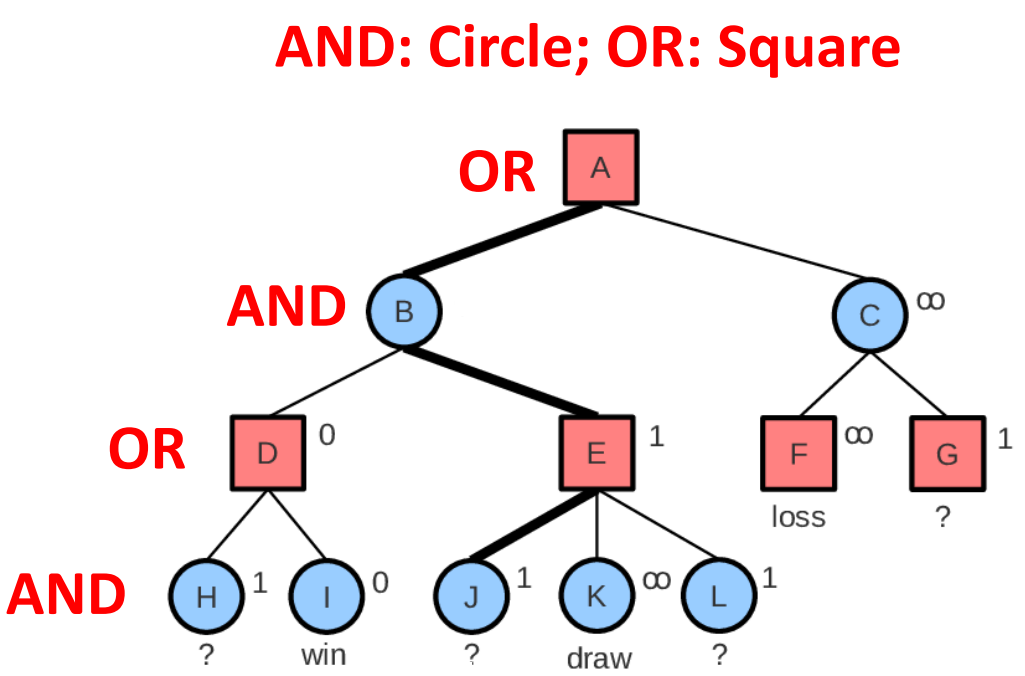
- AND/OR Tree
  - Proof set
    - Proof set: a set of frontier nodes S is a proof set if proving all nodes within S proves T
    - The proof number of T is defined as the cardinality of the smallest proof set of T
  - Disproof set
  - State of leaf nodes
    - Win: 0, ∞
    - Lose: ∞, 0
    - Unknown: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



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

Solve Gomoku: Proof-Number Search

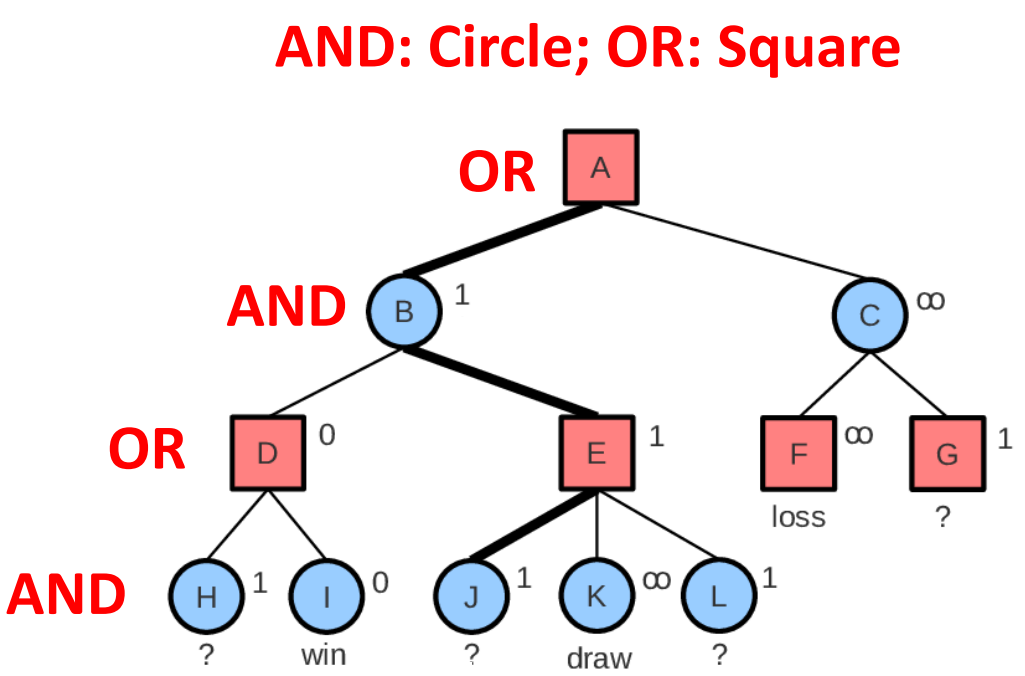
- AND/OR Tree
  - Proof set
    - Proof set: a set of frontier nodes S is a proof set if proving all nodes within S proves T
    - The proof number of T is defined as the cardinality of the smallest proof set of T
  - Disproof set
  - State of leaf nodes
    - Win: 0, ∞
    - Lose: ∞, 0
    - Unknown: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



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

Solve Gomoku: Proof-Number Search

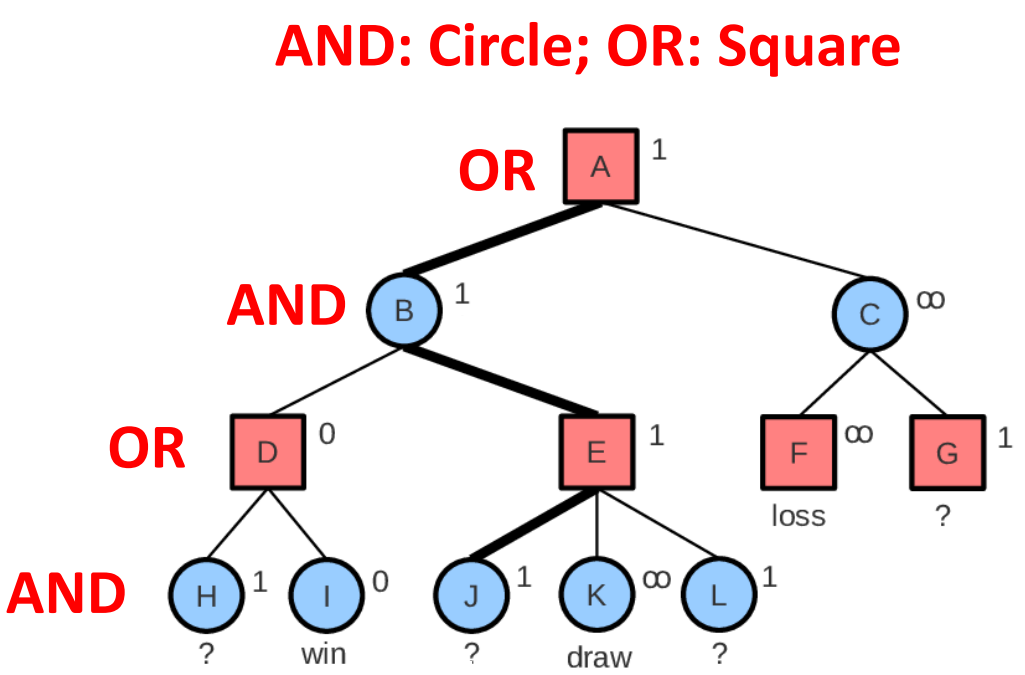
- AND/OR Tree
  - Proof set
    - Proof set: a set of frontier nodes S is a proof set if proving all nodes within S proves T
    - The proof number of T is defined as the cardinality of the smallest proof set of T
  - Disproof set
  - State of leaf nodes
    - Win: 0, ∞
    - Lose: ∞, 0
    - Unknown: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



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

Solve Gomoku: Proof-Number Search

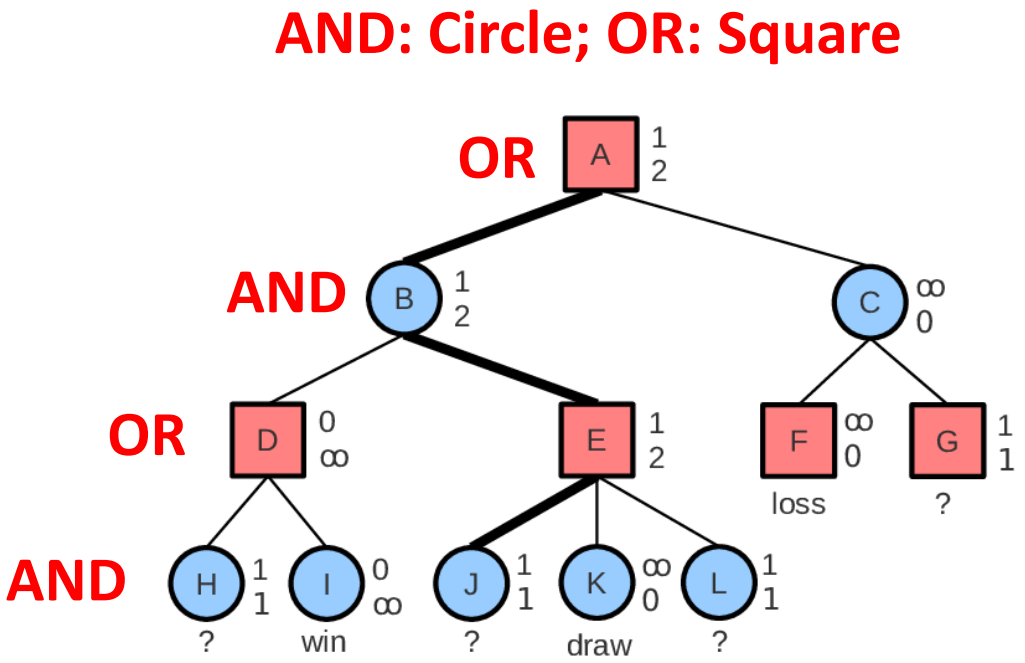
- AND/OR Tree
  - Proof set
    - Proof set: a set of frontier nodes S is a proof set if proving all nodes within S proves T
    - The proof number of T is defined as the cardinality of the smallest proof set of T
  - Disproof set
  - State of leaf nodes
    - Win: 0, ∞
    - Lose: ∞, 0
    - Unknown: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



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

Solve Gomoku: Proof-Number Search

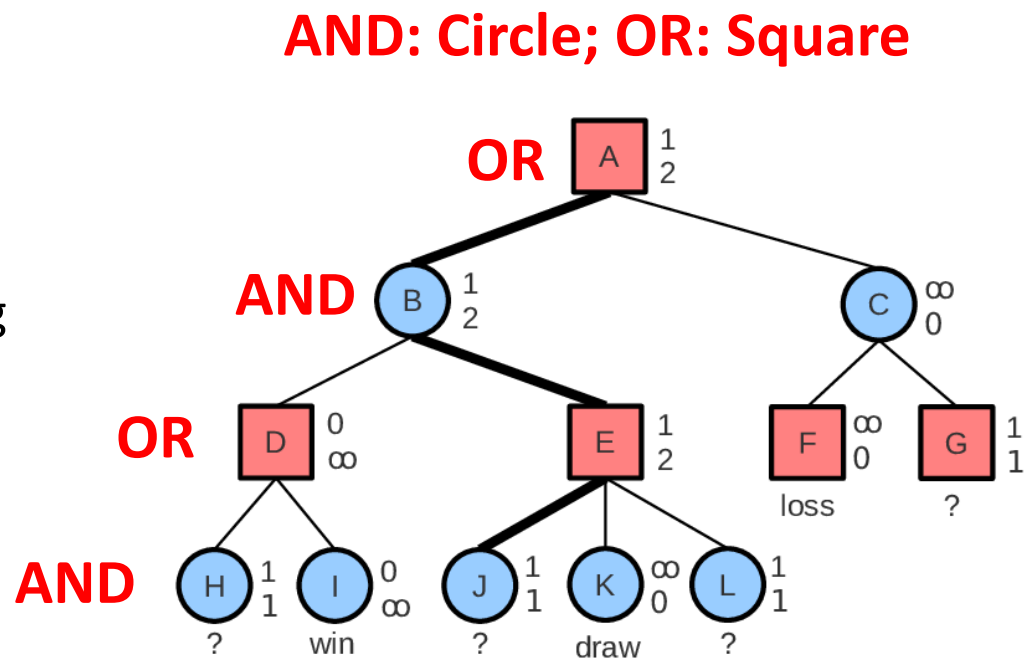
- AND/OR Tree
  - Set proof number
    - AND: sum OR: min
  - Set disproof number
    - AND: min OR: sum



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

Solve Gomoku: Proof-Number Search

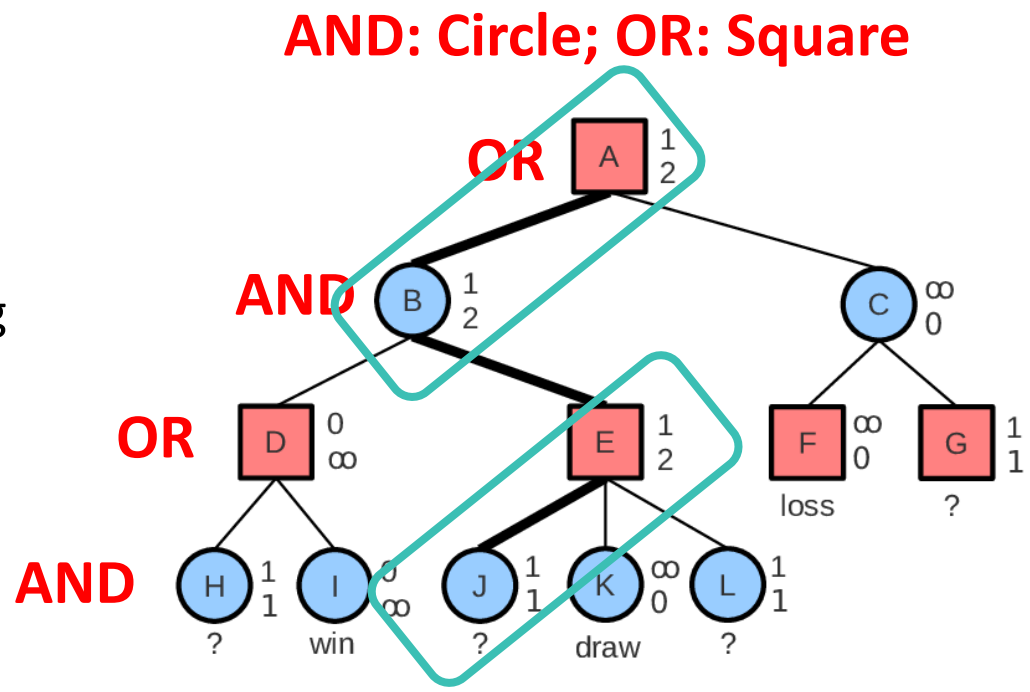
- AND/OR Tree
  - Most-proving node
    - Proved: each pair consisting of a smallest proof set and a smallest disproof set has a non-empty intersection.
    - i.e. There must exist at least one most-proving node.



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

Solve Gomoku: Proof-Number Search

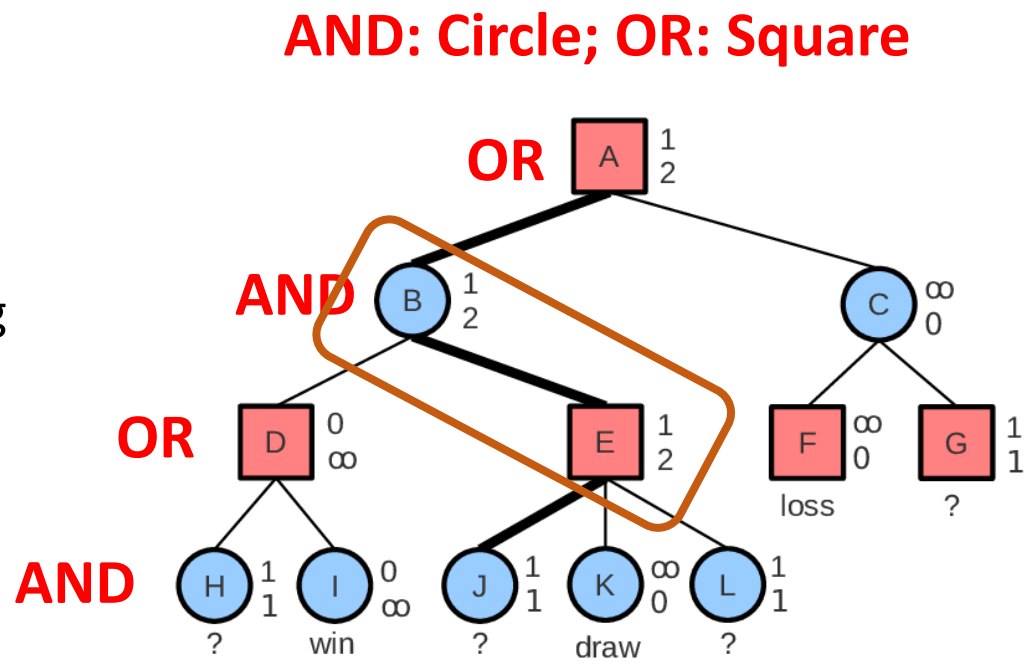
- AND/OR Tree
  - Most-proving node
    - Proved: each pair consisting of a smallest proof set and a smallest disproof set has a non-empty intersection.
    - i.e. There must exist at least one most-proving node.



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.



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

Solve Gomoku: Proof-Number Search

- 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
```

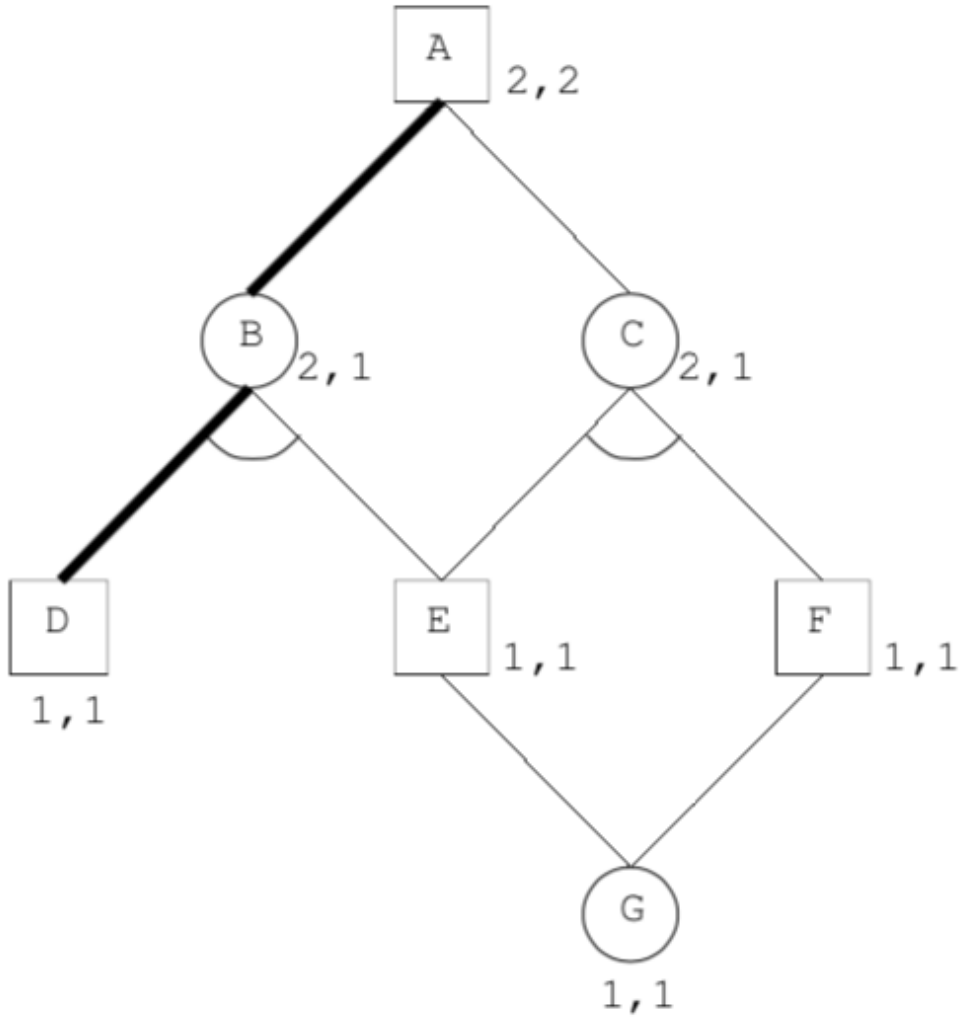
```
procedure DevelopNode(node);
    GenerateAllChildren(node);
    for i := 1 to node.numberOfChildren do
        Evaluate(node.children[i]);
        SetProofAndDisproofNumbers(node.children[i])
    od
end
```

```
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

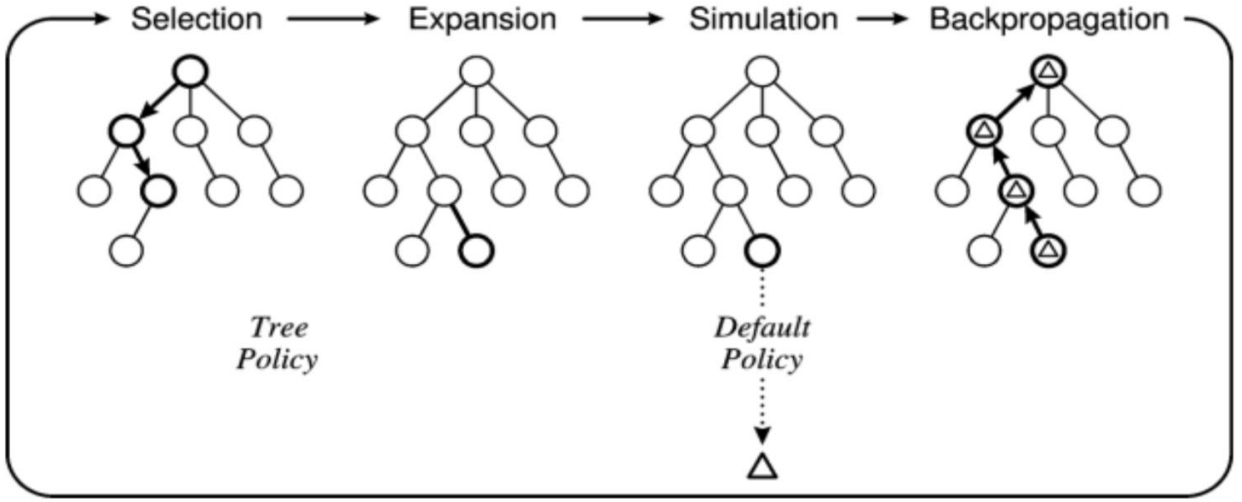
- Transposition
  - Hash table
  - Directed Acyclic Graphs



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

Solve Gomoku: Monte Carlo Tree Search

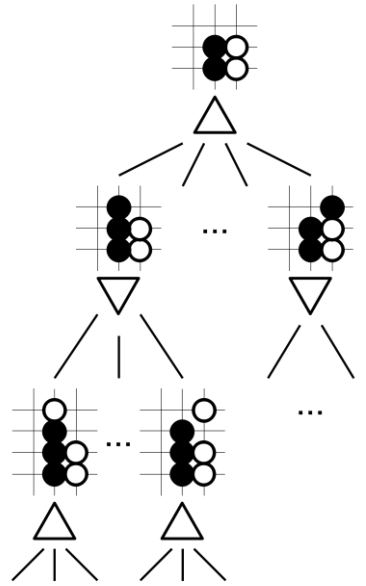
复旦大学大数据学院
School of Data Science, Fudan University



- 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, (61273136), 2017.

Gomoku

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

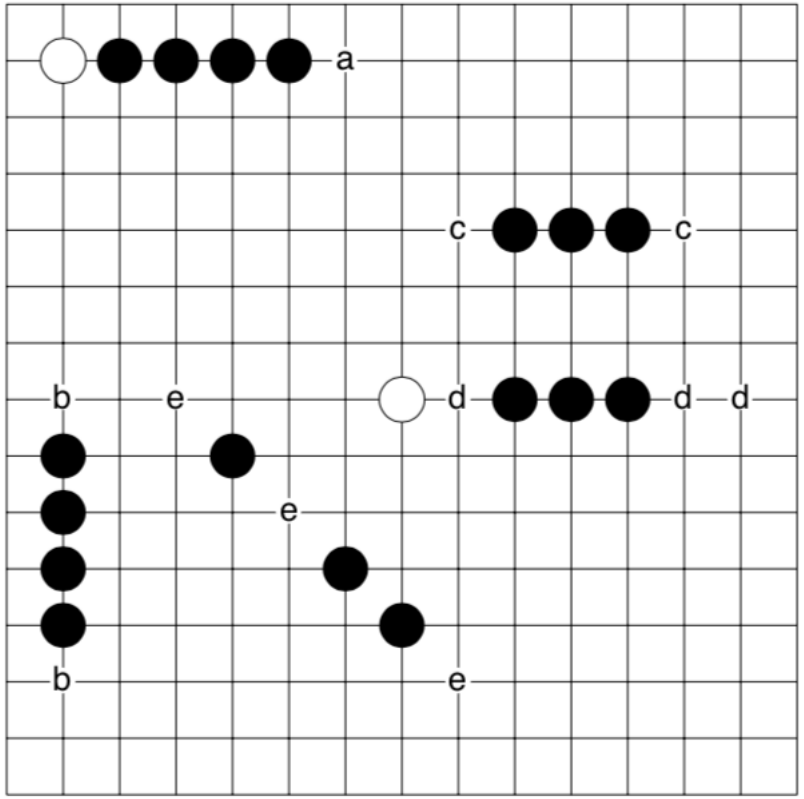
Solve Gomoku: Threat-Space Search

- Goal: Five in a row

Four in a row
Three in a row
…

Threat

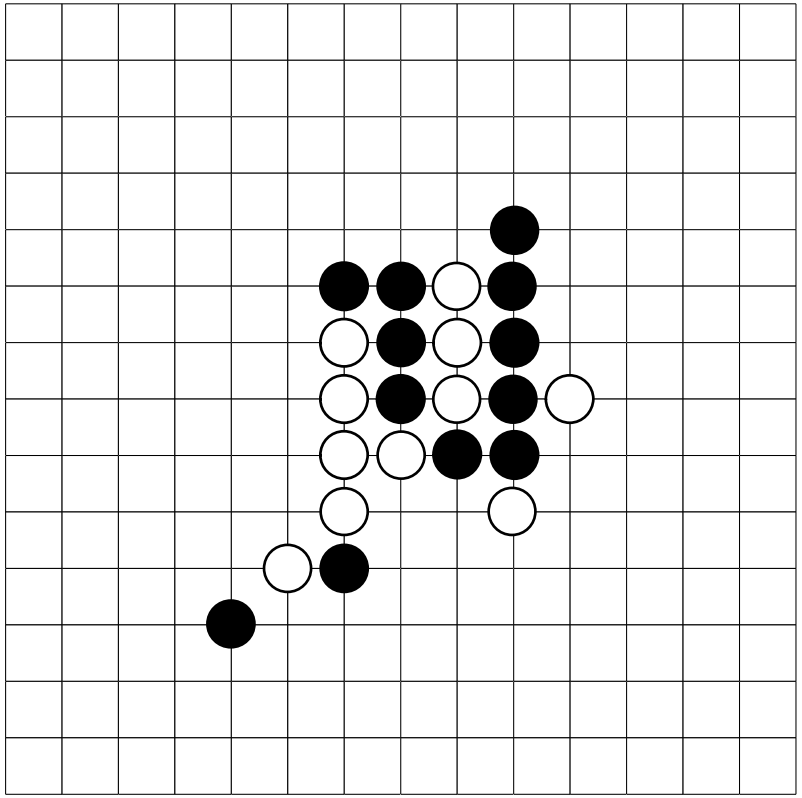
Threat Sequence



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

Solve Gomoku: Threat-Space Search

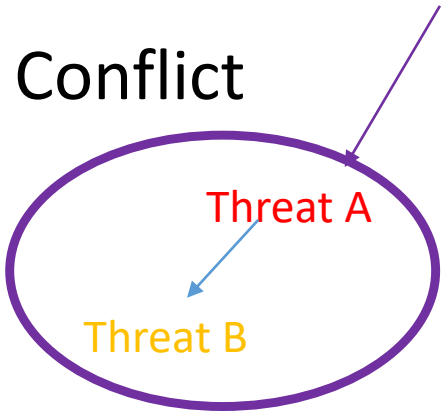
- Threat Sequence
- Winning Threat Sequence

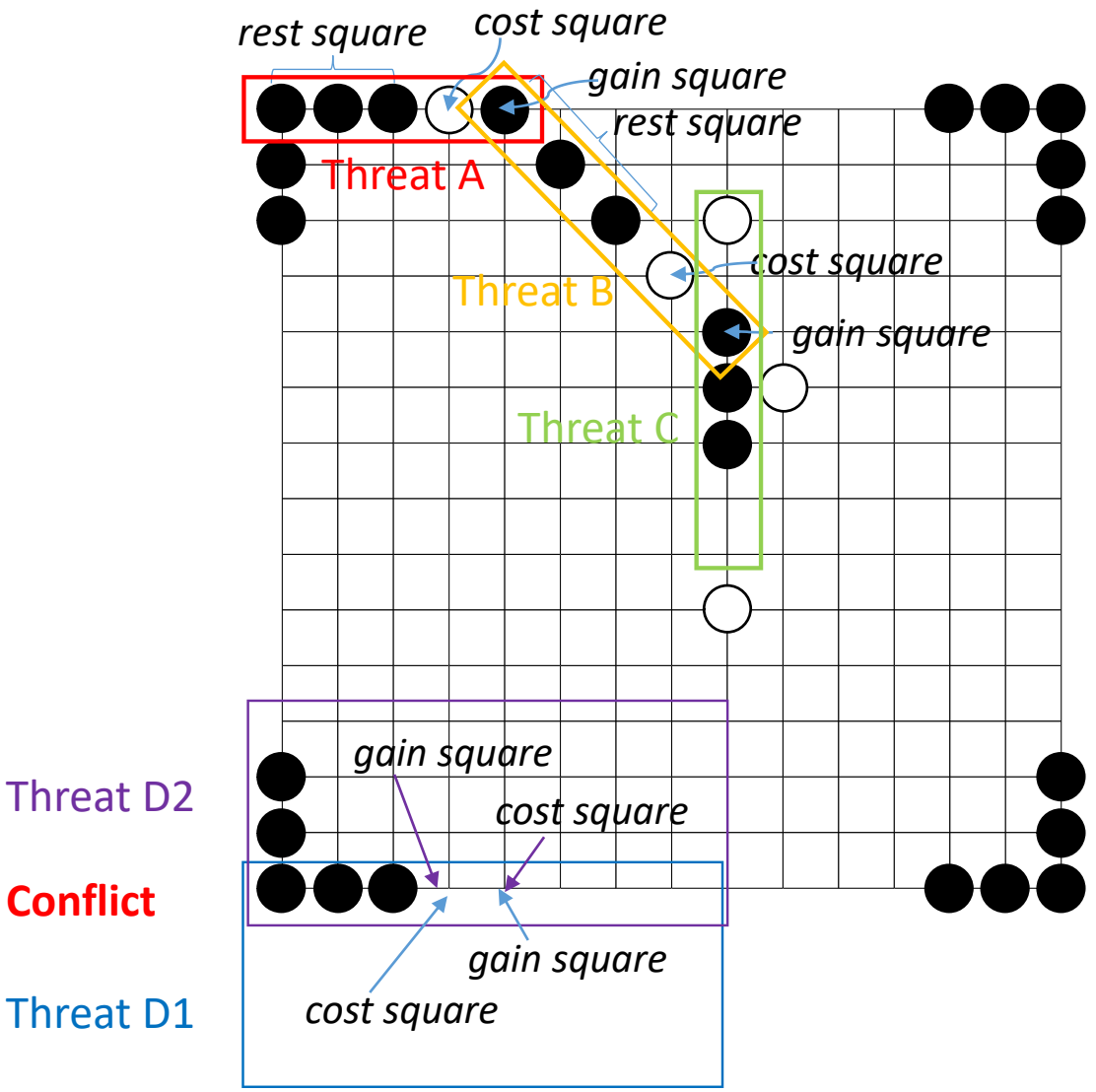


Louis Victor Allis and Hj Van Den Herik.
Go-moku and threat-space search.
. Ist. Psu. Edu/, 1993.

Solve Gomoku: Threat-Space Search

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



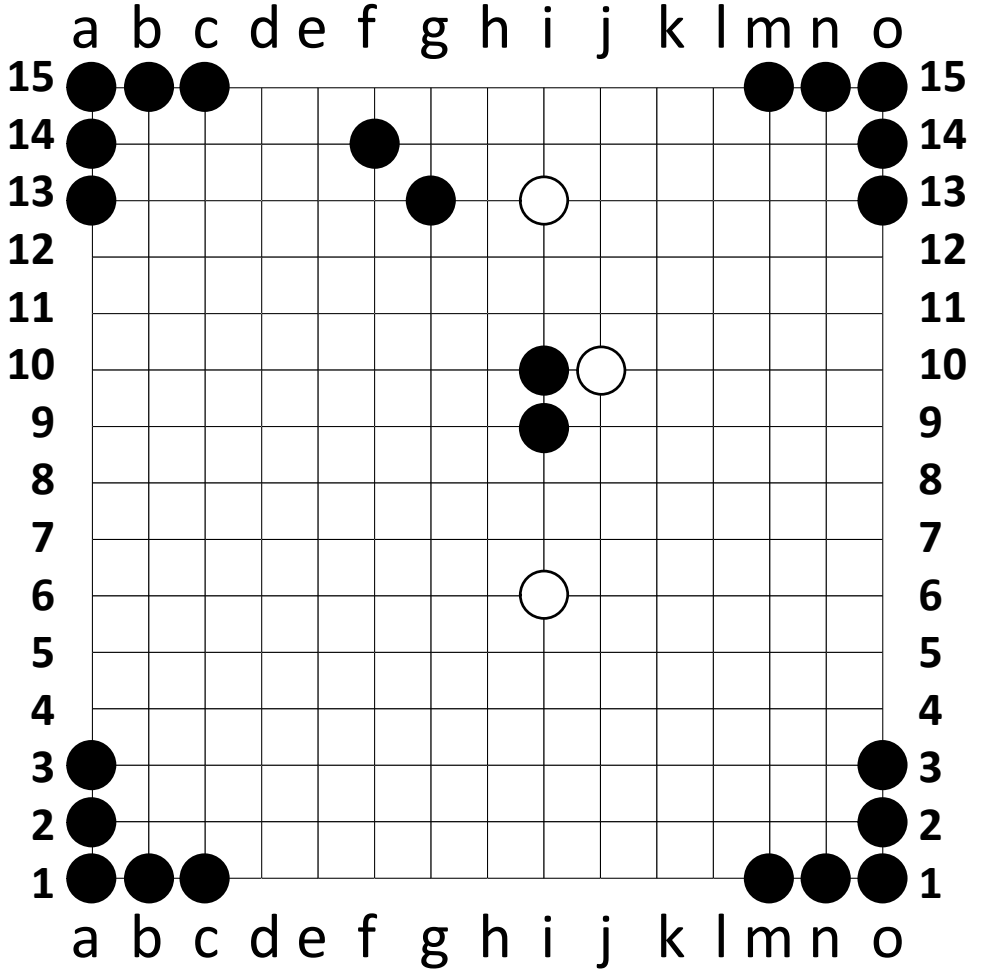


Louis Victor Allis and Hj Van Den Herik.
Go-moku and threat-space search.
. Ist. Psu. Edu/, 1993.

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

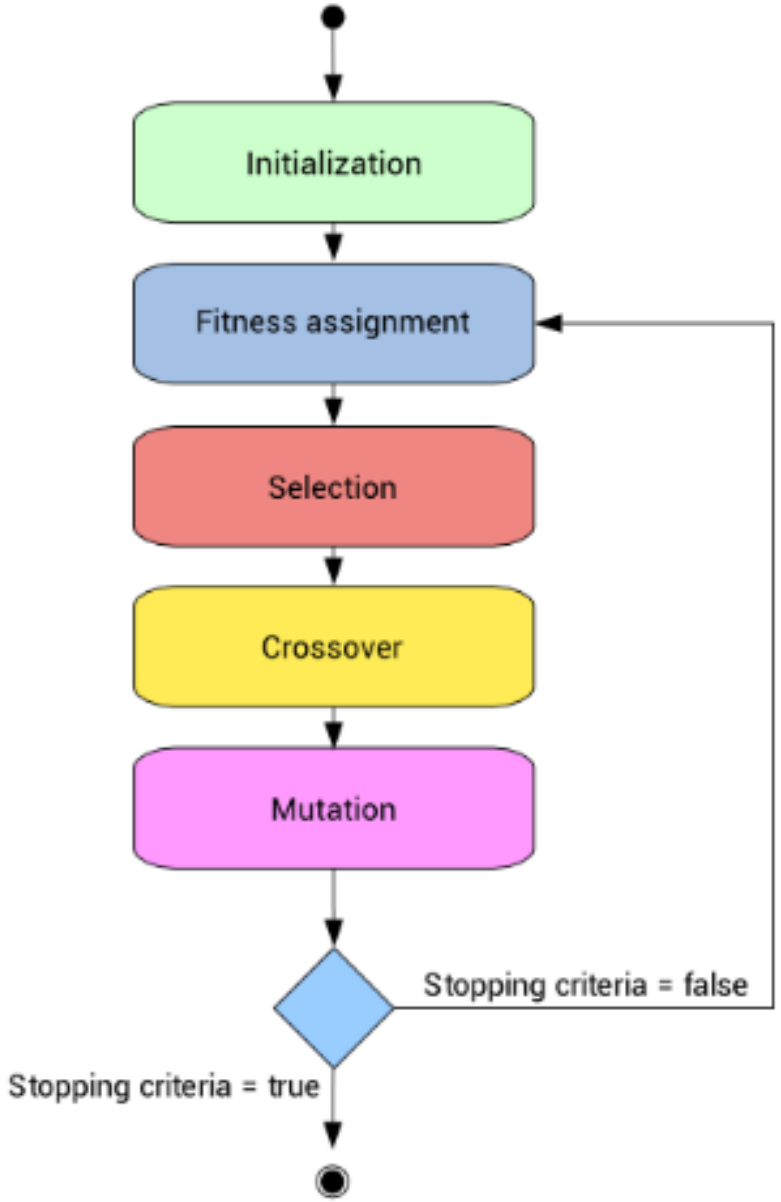
Louis Victor Allis and Hj Van Den Herik.
Go-moku and threat-space search.
. Ist. Psu. Edu/, 1993.

Gomoku

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

Solve Gomoku: Genetic Algorithm

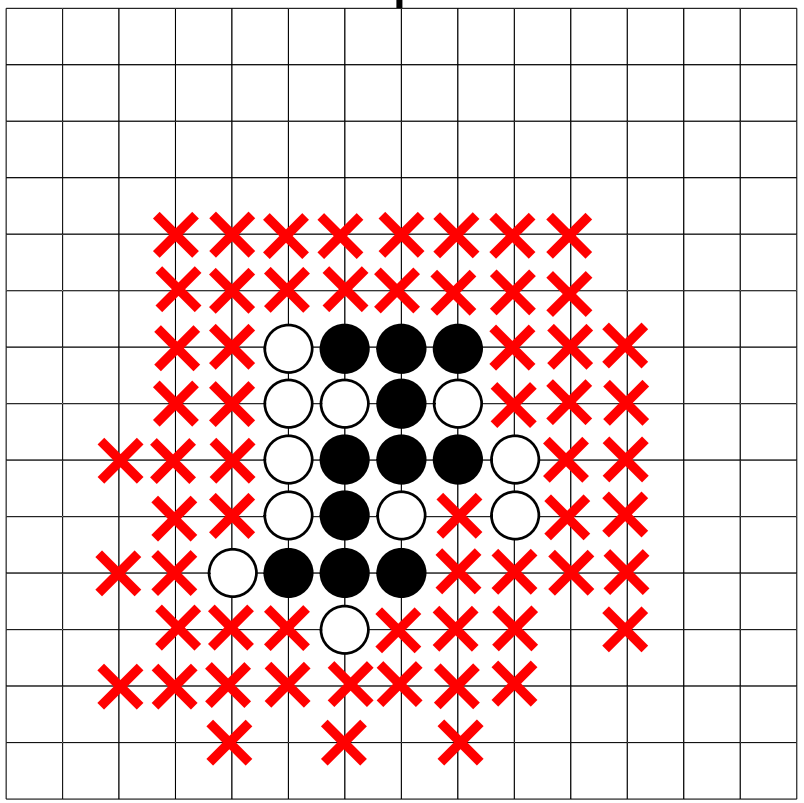
- Initialization: Coding Scheme
- Fitness assignment
- Selection
- Crossover
- Mutation

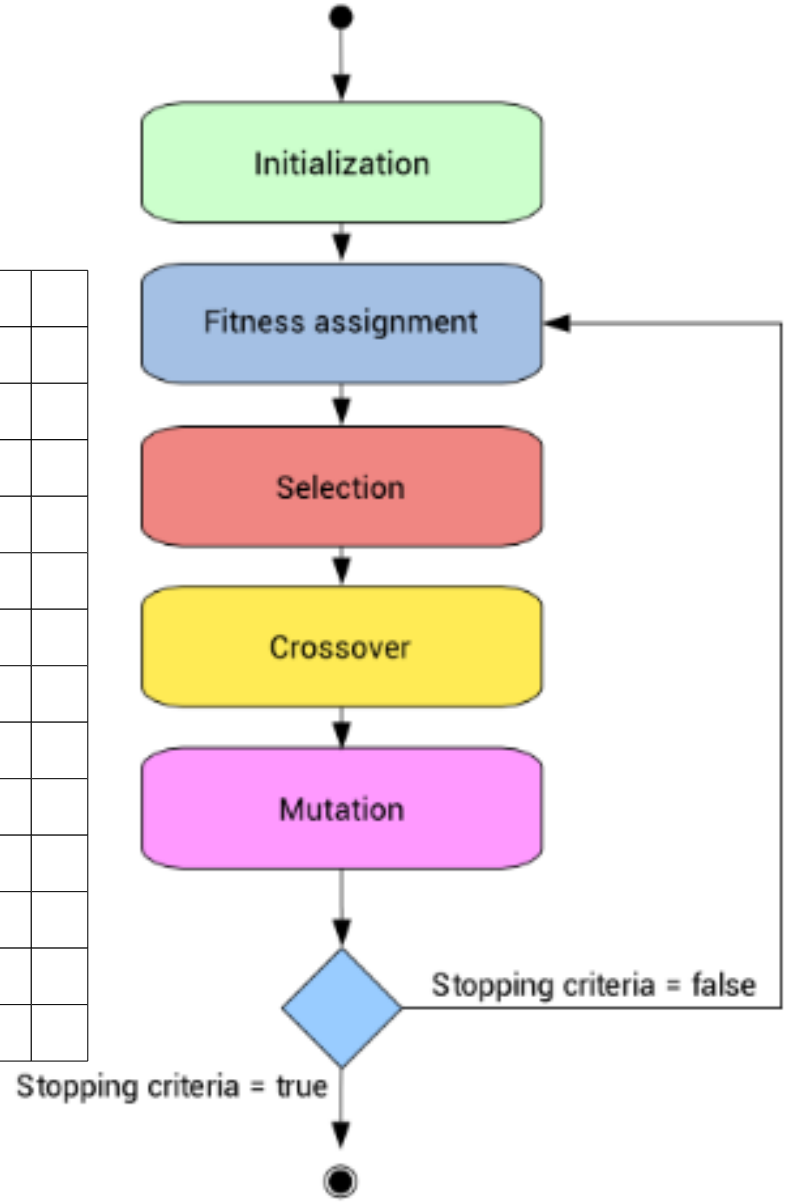


Solve Gomoku: Genetic Algorithm

- 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





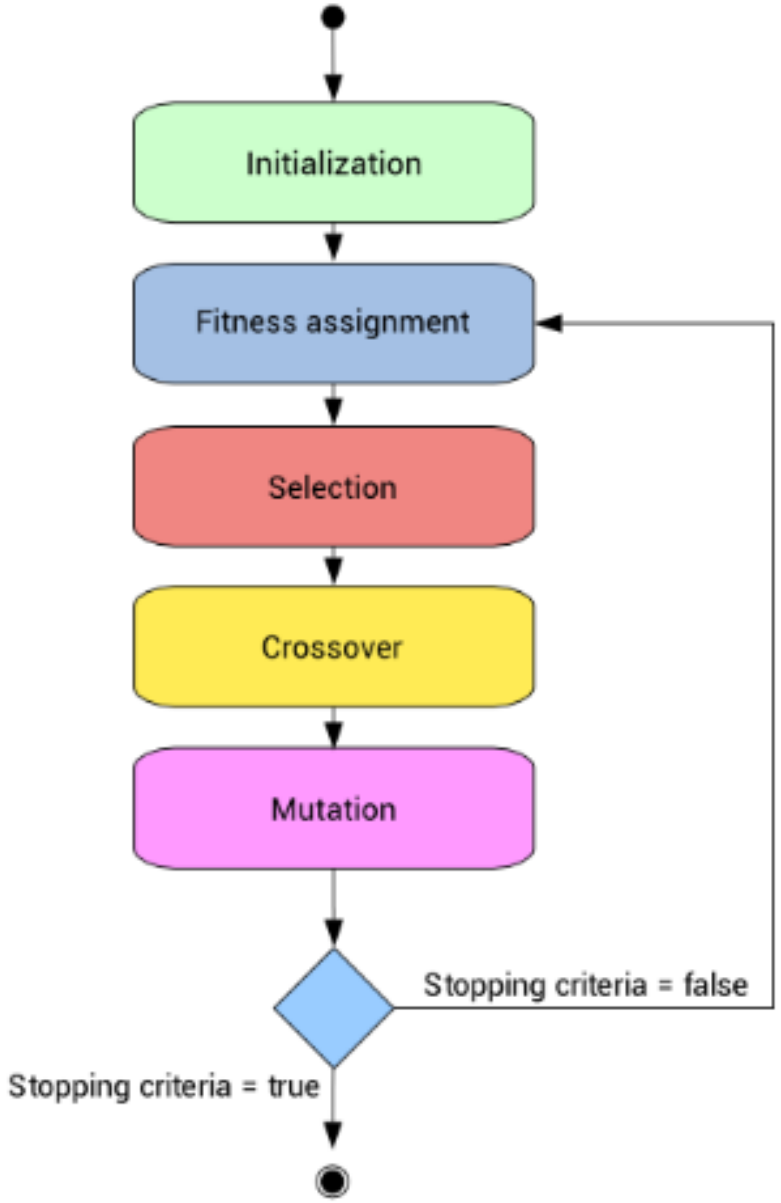
Solve Gomoku: Genetic Algorithm

- Initialization: Coding Scheme
- Fitness assignment
  - Example：

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

    + 97 * (number of three structures in neighborhood)

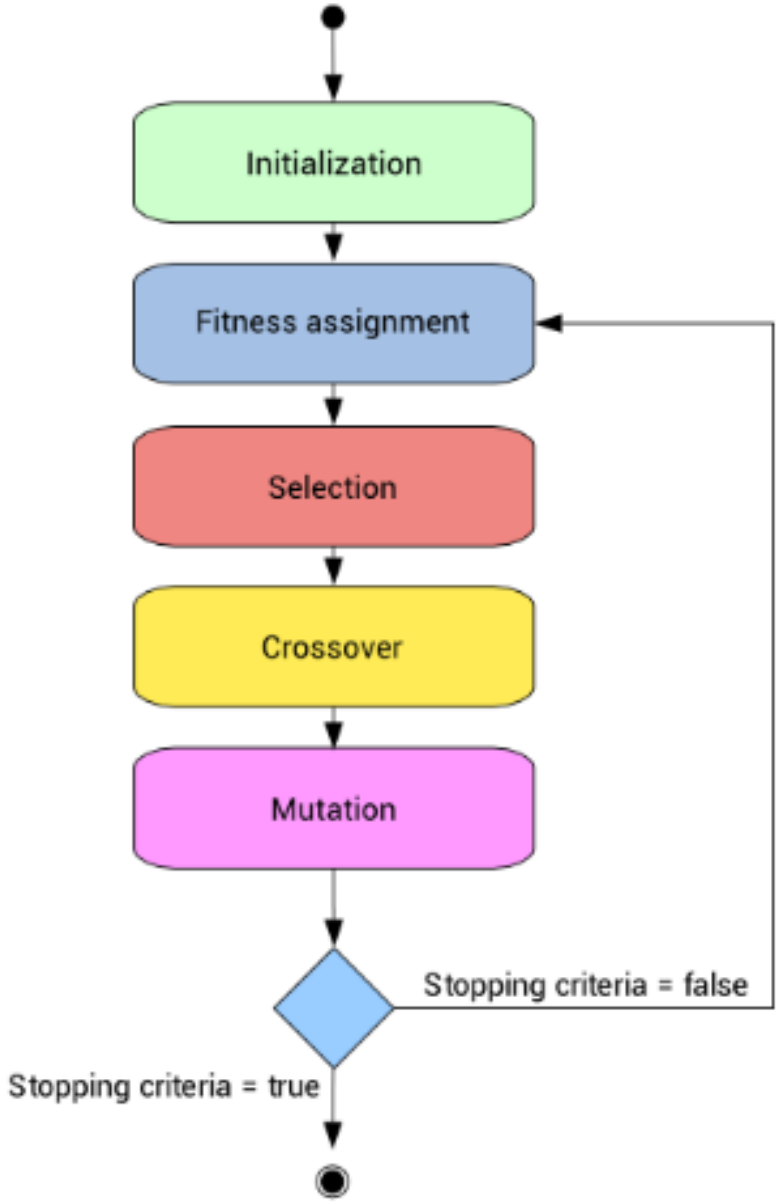
    + 17 * (number of two structures in neighborhood)
- Selection



Solve Gomoku: Genetic Algorithm

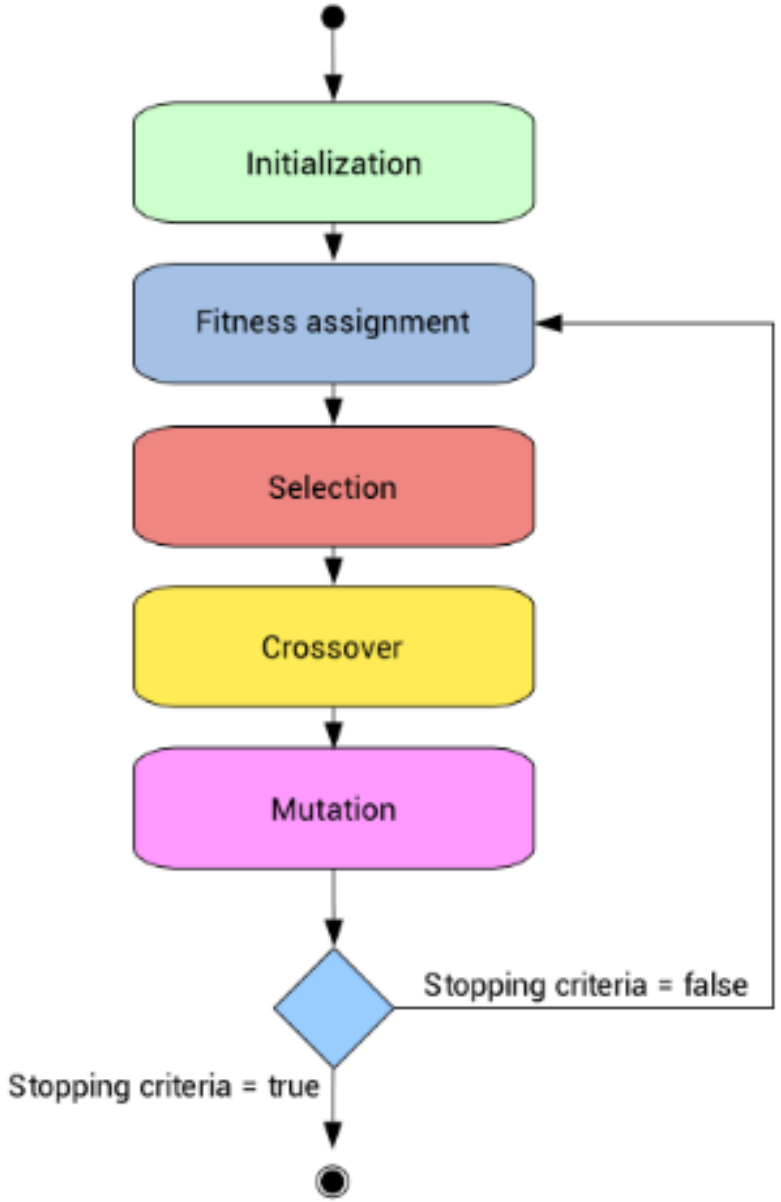
- Initialization: Coding Scheme
- Fitness assignment
- Selection
- Crossover
  - Parents: $A_2A_1A_7A_3A_8A_5A_6$
    $A_5A_3A_1A_4A_9A_2A_8$
  - Children: $A_2A_1A_7A_3A_9A_2A_8$
    $A_5A_3A_1A_4A_8A_5A_6$
- Mutation

  $A_2A_9A_3A_4A_7A_8A_6$
  $A_2A_9A_7A_8A_3A_4A_6$



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/

DATA130008 Introduction to Artificial Intelligence

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

Lab 2 利用对抗搜索构建黑白棋AI

大纲

- 黑白棋介绍
- Alpha-Beta 剪枝
- 代码细节

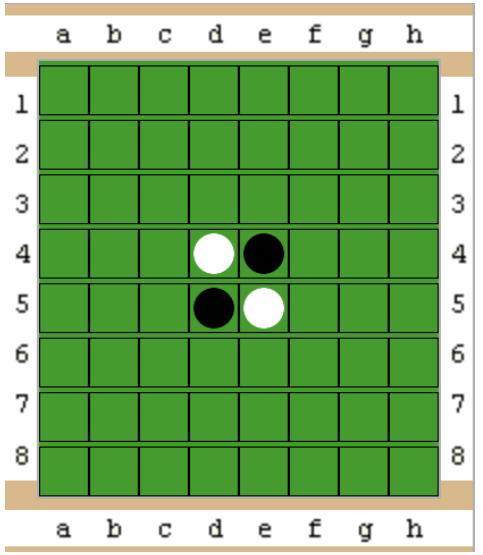
黑白棋介绍

规则简介

- 棋盘共有8行8列共64格。开局时，棋盘正中央的4格先置放黑白相隔的4枚棋子（亦有求变化相邻放置）。

- 通常黑子先行。双方轮流落子。只要落子和棋盘上任一枚己方的棋子在一条线（横、直、斜线皆可）夹着对方棋子，就能将对方的这些棋子转变为我己方（翻面即可）。

- 如果在任一位置落子都不能夹住对手的任一颗棋子，就要让对手下子。当双方皆不能下子时，游戏就结束，子多的一方胜。



大纲

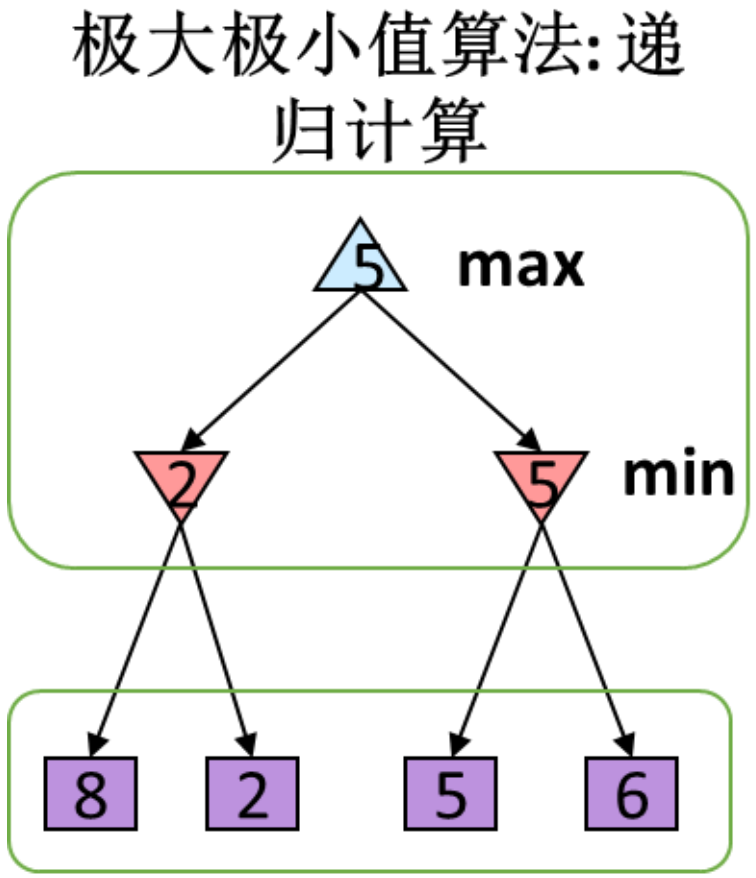
- 黑白棋介绍
- **Alpha-Beta 剪枝**
- 代码细节

Alpha-Beta剪枝


```
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 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
```

对于极大值节点

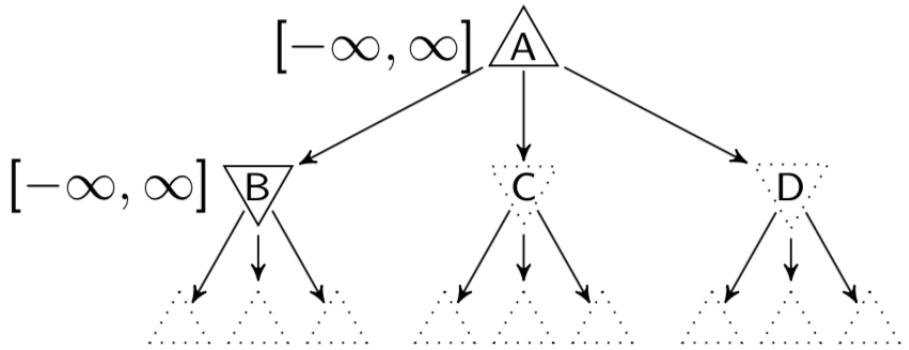
$\beta = \beta_{parent}$

v 用于更新α， 初始$\alpha = \alpha_{parent}$

对于极小值节点

$\alpha = \alpha_{parent}$

v 用于更新β， 初始$\beta = \beta_{parent}$



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
```

对于极大值节点

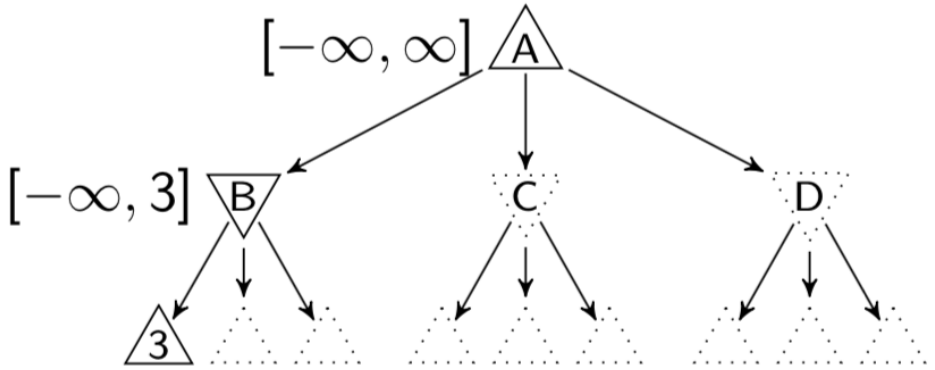
$\beta = \beta_{parent}$

v 用于更新α， 初始$\alpha = \alpha_{parent}$

对于极小值节点

$\alpha = \alpha_{parent}$

v 用于更新β， 初始$\beta = \beta_{parent}$



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
```

对于极大值节点

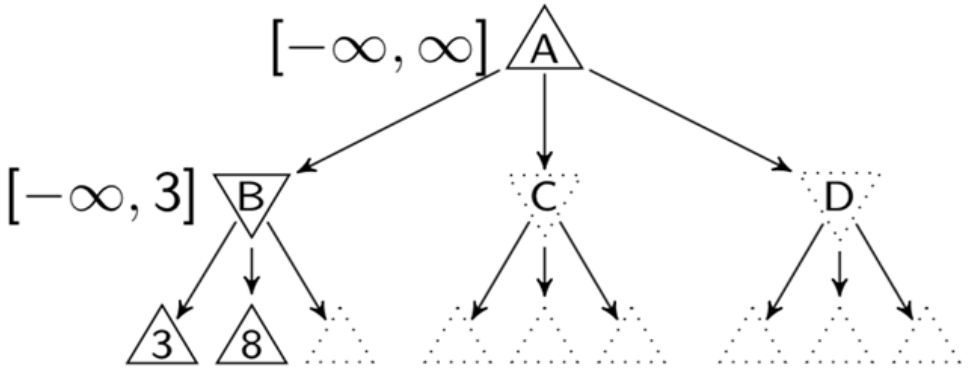
$\beta = \beta_{parent}$

v 用于更新α， 初始$\alpha = \alpha_{parent}$

对于极小值节点

$\alpha = \alpha_{parent}$

v 用于更新β， 初始$\beta = \beta_{parent}$



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
```

对于极大值节点

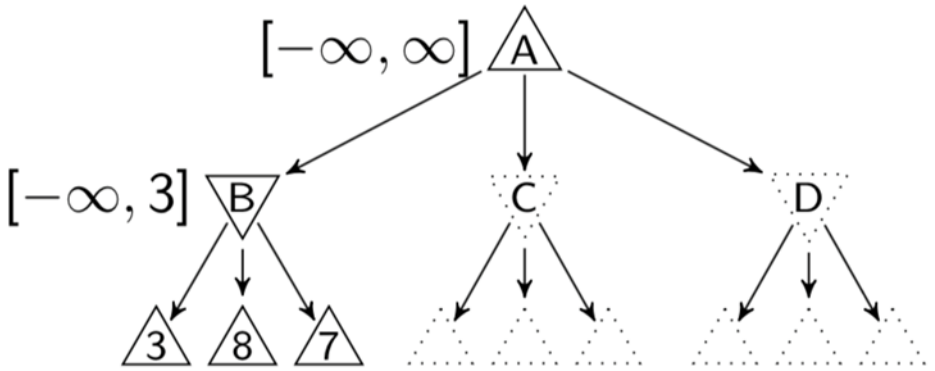
$\beta = \beta_{parent}$

v 用于更新α， 初始$\alpha = \alpha_{parent}$

对于极小值节点

$\alpha = \alpha_{parent}$

v 用于更新β， 初始$\beta = \beta_{parent}$



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
```

对于极大值节点

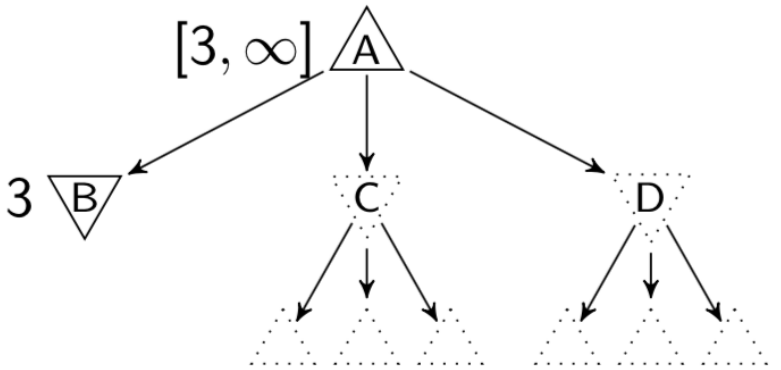
$\beta = \beta_{parent}$

v 用于更新α， 初始$\alpha = \alpha_{parent}$

对于极小值节点

$\alpha = \alpha_{parent}$

v 用于更新β， 初始$\beta = \beta_{parent}$



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
```

对于极大值节点

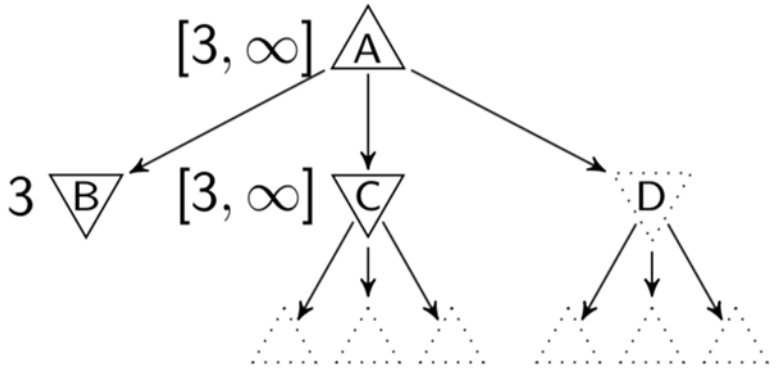
$\beta = \beta_{parent}$

v 用于更新α， 初始$\alpha = \alpha_{parent}$

对于极小值节点

$\alpha = \alpha_{parent}$

v 用于更新β， 初始$\beta = \beta_{parent}$



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
```

对于极大值节点

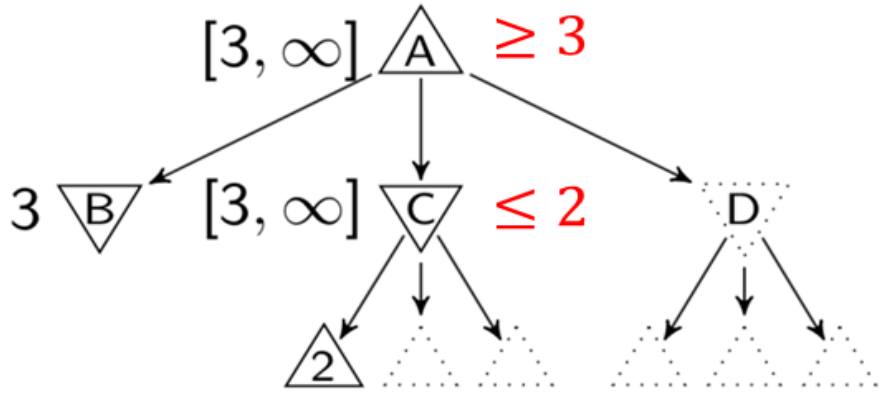
$\beta = \beta_{parent}$

v 用于更新α， 初始$\alpha = \alpha_{parent}$

对于极小值节点

$\alpha = \alpha_{parent}$

v 用于更新β， 初始$\beta = \beta_{parent}$



Conflict!!

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
```

对于极大值节点

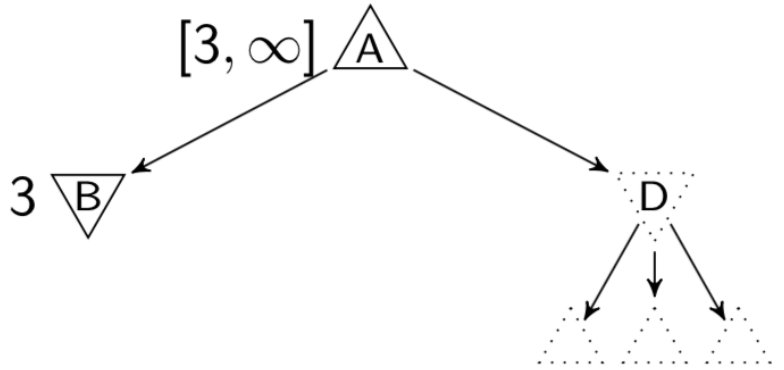
$\beta = \beta_{parent}$

v 用于更新α， 初始$\alpha = \alpha_{parent}$

对于极小值节点

$\alpha = \alpha_{parent}$

v 用于更新β， 初始$\beta = \beta_{parent}$



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
```

对于极大值节点

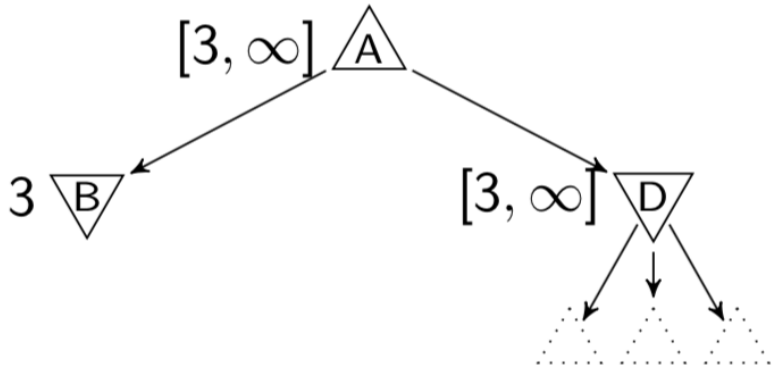
$\beta = \beta_{parent}$

v 用于更新α， 初始$\alpha = \alpha_{parent}$

对于极小值节点

$\alpha = \alpha_{parent}$

v 用于更新β， 初始$\beta = \beta_{parent}$



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
```

对于极大值节点

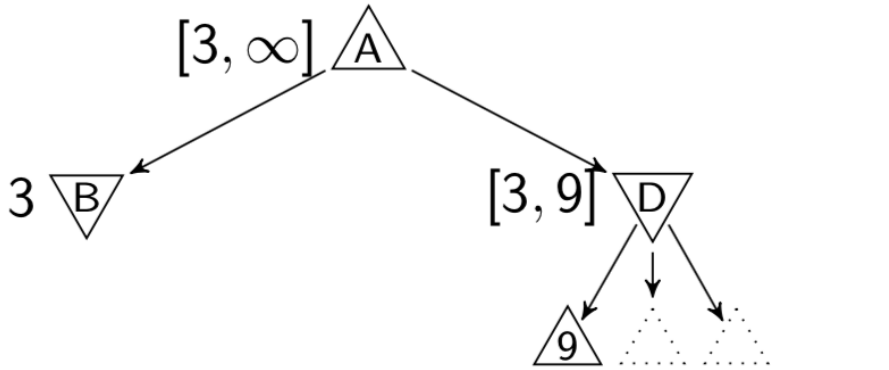
$\beta = \beta_{parent}$

v 用于更新α， 初始$\alpha = \alpha_{parent}$

对于极小值节点

$\alpha = \alpha_{parent}$

v 用于更新β， 初始$\beta = \beta_{parent}$



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
```

对于极大值节点

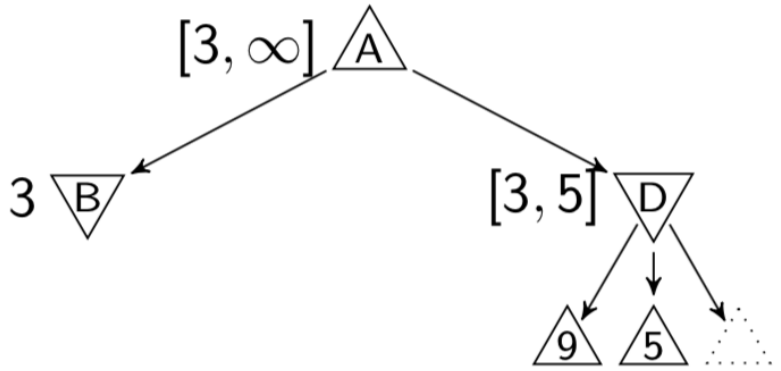
$\beta = \beta_{parent}$

v 用于更新α， 初始$\alpha = \alpha_{parent}$

对于极小值节点

$\alpha = \alpha_{parent}$

v 用于更新β， 初始$\beta = \beta_{parent}$



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
```

对于极大值节点

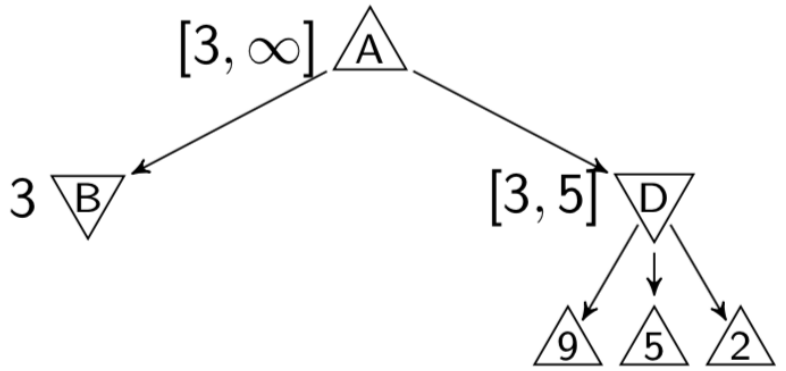
$\beta = \beta_{parent}$

v 用于更新α， 初始$\alpha = \alpha_{parent}$

对于极小值节点

$\alpha = \alpha_{parent}$

v 用于更新β， 初始$\beta = \beta_{parent}$



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
```

对于极大值节点

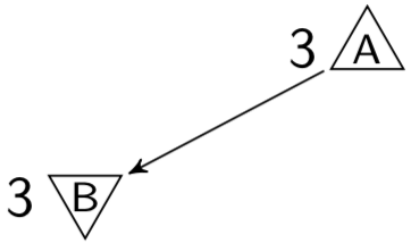
$\beta = \beta_{parent}$

v 用于更新α，初始$\alpha = \alpha_{parent}$

对于极小值节点

$\alpha = \alpha_{parent}$

v 用于更新β，初始$\beta = \beta_{parent}$



大纲

- 黑白棋介绍
- Alpha-Beta 剪枝
- 代码细节

代码细节

文件介绍

alpha_beta_templ
ate.py

需要填充的文件，其中 board 类不需要关注！！！
直接找到Aiplayer类进行填充
提交时，将此文件全部复制到OJ的代码框中

game.py

本地游戏文件，写完AI后可以本地与自己写的AI进行对弈
（不需要提交，不需要填充）

testcase

测试用例文件夹，其中有一个 1.in 和 1.out，写完alpha_beta_template.py，运行alpha_beta_template.py，将1.in 复制并输入，检查输出是否与1.out相同

代码细节


```
function ALPHA-BETA_SEARCH( board, color ) returns an action
    depth ← 0
    v, action ← MAX-VALUE( board, color, -∞, ∞, depth)
    return v and action
```

初始深度为0

当搜索深度大于规定最大深度时进行截断

注意！这里需要将board deepcopy 成新变量，否则会造成bug

```
function MAX-VALUE( board, color, α, β, depth) returns an action
    if TERMINAL-TEST(depth) then return UTILITY(board,color)
    v ← - ∞
    for each a in get_actions( color ) do
        make_move( a, color )
        move_v, move_action ← MIN-VALUE( board, color, α, β, depth+1)
        if move_v > v then  v ← move_v, action ← a
        if  v >= β then return v and action
        α ← MAX(α, v)
return v and action
```

迭代深度+1

```
function MIN-VALUE( board, color, α, β, depth) returns an action
    if TERMINAL-TEST(depth) then return UTILITY(board,color)
    v ← + ∞
    for each a in board.get_actions( opposite_color ) do
        make_move(a,opposite_color)
        move_v, move_action ← MAX-VALUE( board, color, α, β, depth+1)
        if move_v < v then  v ← move_v, action ← a
        if  v <= α then return v and action
        β ← MIN(β, v)
return v and action
```

由于min层需要考虑对手的策略，所以这里的移动应该是对手的棋子颜色