DATA130008 Introduction to Artificial Intelligence

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

高亦照

Lab 2

April 3rd 2019

Outline

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

Outline

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

Gomoku

- **Gomoku Rule**
- **Solve Gomoku**
  - **Genetic Algorithm**
  - **Monte Carlo Tree Search**
  - **Reinforcement Learning**
    - **Adaptive Dynamic Programming**
  - **Dependency-Based Search**
    - **Threat-Space Search**
  - **Proof-Number Search**

- **Gomoku Rule**
- Solve Gomoku
  - Genetic Algorithm
  - Monte Carlo Tree Search
  - Reinforcement Learning
    - Adaptive Dynamic Programming
  - Dependency-Based Search
    - Threat-Space Search
  - Proof-Number Search

Gomoku Rule

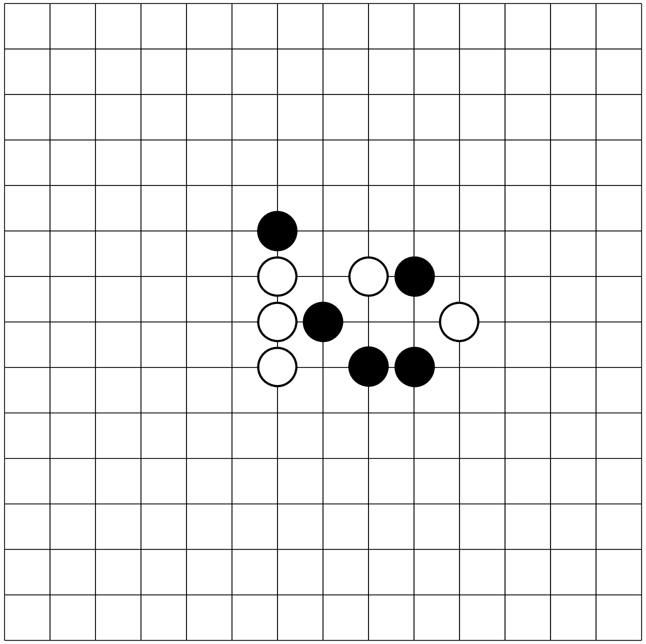
- Goal: Five in a row, 15 × 15
- Proved: Black first leads to win (1899)
- Gomoku:
  - Free/Standard Gomoku Rule: Victoria 15 × 15
  - Swap 2 Rule
- Renju:
  - Free Renju Rule/Soosyrv-8 Rule

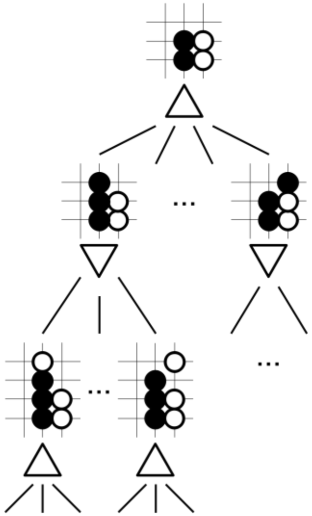
- Focus on: Free Gomoku

- **Gomoku Rule**
- **Solve Gomoku**
  - Genetic Algorithm
  - Monte Carlo Tree Search
  - Reinforcement Learning
    - Adaptive Dynamic Programming
  - Dependency-Based Search
    - Threat-Space Search
  - Proof-Number Search

Solve Gomoku

- State representation
  - Board representation
    - (x, y, ●/○/ ·)
  - Features
    - Patterns
    - Turns
    - Offensive/Defensive
    - ……
  - Axiom/Structured
- Search for next step
  - Single agent
  - Adaptive agent





- **Gomoku Rule**
- **Solve Gomoku**
  - Genetic Algorithm
  - Monte Carlo Tree Search
  - Reinforcement Learning
    - Adaptive Dynamic Programming
  - Dependency-Based Search
    - Threat-Space Search
  - Proof-Number Search

Solve Gomoku: Genetic Algorithm

- Initialization: Coding Scheme
- Selection
- Crossover
- Mutation

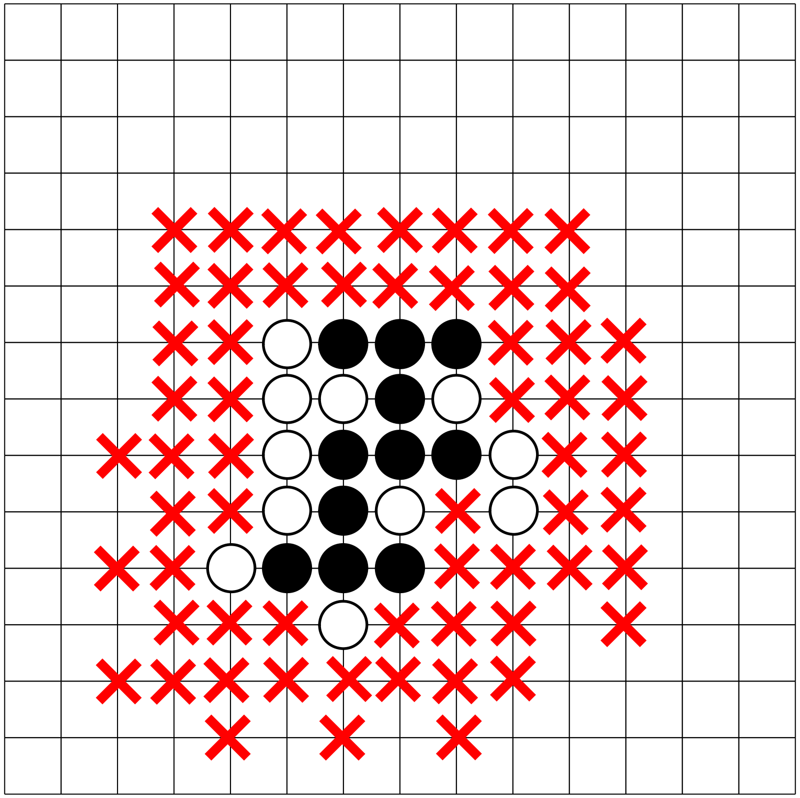
- Fitness function

Solve Gomoku: Genetic Algorithm

- 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

- Selection
  - Fitness function
    - Example：

      f(s) = 4800 * (number of four structures in neighborhood)
      + 97 * (number of three structures in neighborhood)
      + 17 * (number of two structures in neighborhood)

Solve Gomoku: Genetic Algorithm

- Crossover
  - Parents:

    $A_2A_1A_7A_3A_8A_5A_6$

    $A_5A_3A_1A_4A_9A_2A_8$
  - Children:

    $A_2A_1A_7A_3A_9A_2A_8$

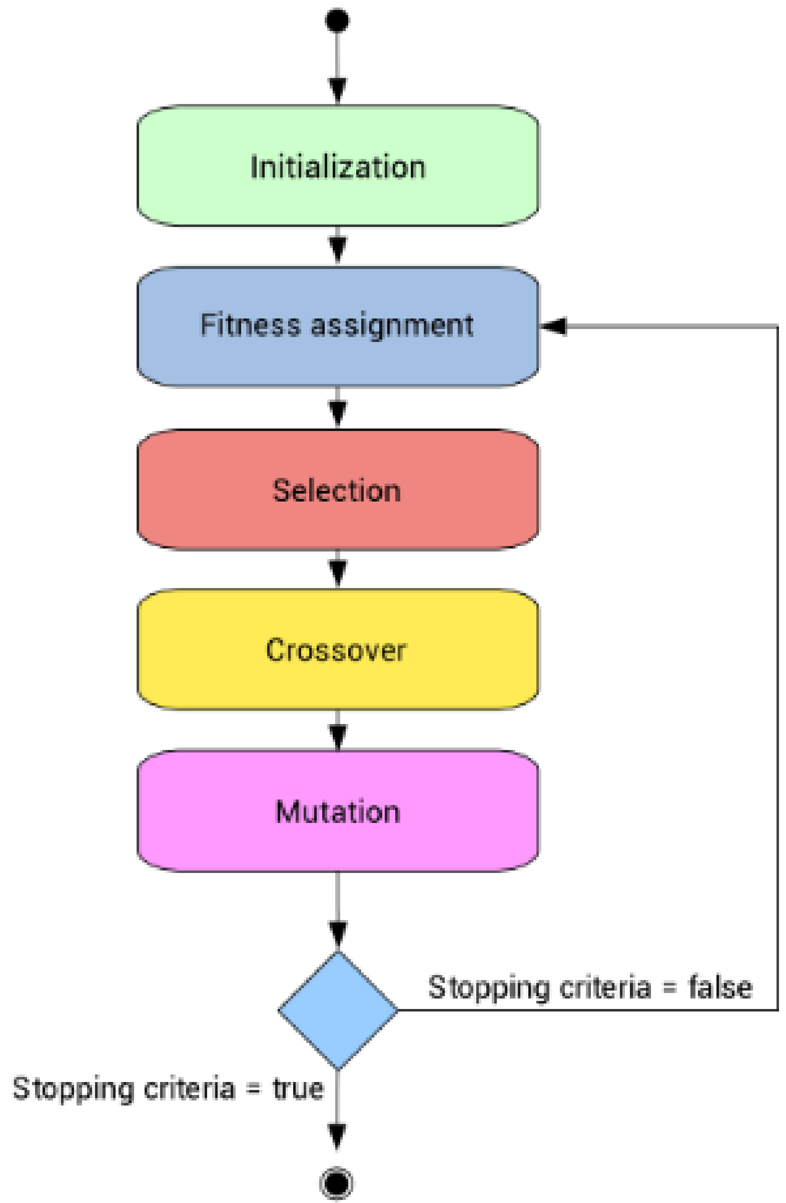
    $A_5A_3A_1A_4A_8A_5A_6$
- Mutation

  $A_2A_9A_3A_4A_7A_8A_6$

  $A_2A_9A_7A_8A_3A_4A_6$

Solve Gomoku: Genetic Algorithm

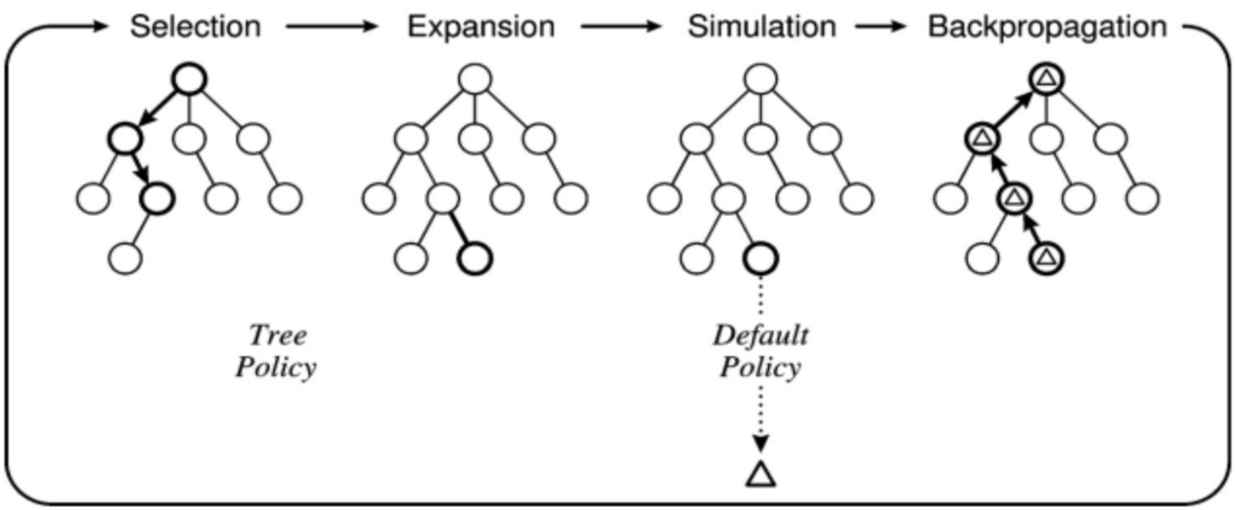
- Procedure



Gomoku

- Gomoku Rule
- Solve Gomoku
  - Genetic Algorithm
  - Monte Carlo Tree Search
  - Reinforcement Learning
    - Adaptive Dynamic Programming
  - Dependency-Based Search
    - Threat-Space Search
  - Proof-Number Search

- Monte Carlo Tree Search: Simulation, Expectation
  - 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*)

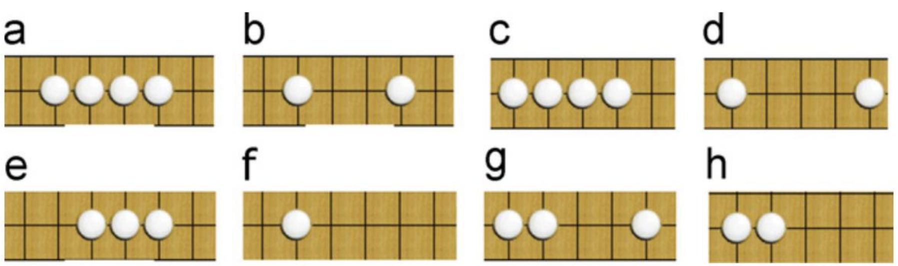
Simulation(state s_t)
  if (s_t is win **and** s_t is terminal) **then return** 1.0;
    else return 0.0;
  end if
  if (s_t satisfied with Heuristic Knowledge)
    then obtain forced action a_f;
      new state $s_{t+1} \leftarrow f(s_t, a_f)$;
    else choose random action $a_r \in$ untried actions;
      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 Rule
- Solve Gomoku
  - Genetic Algorithm
  - Monte Carlo Tree Search
  - Reinforcement Learning
    - Adaptive Dynamic Programming
  - Dependency-Based Search
    - Threat-Space Search
  - Proof-Number Search

- State representation: Board situation
  - Patterns (Structured)
    - 5 or 6 adjacent positions
  - Turns
    - Whose turn to move
  - Offensive/Defensive
  - Winning probability



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.

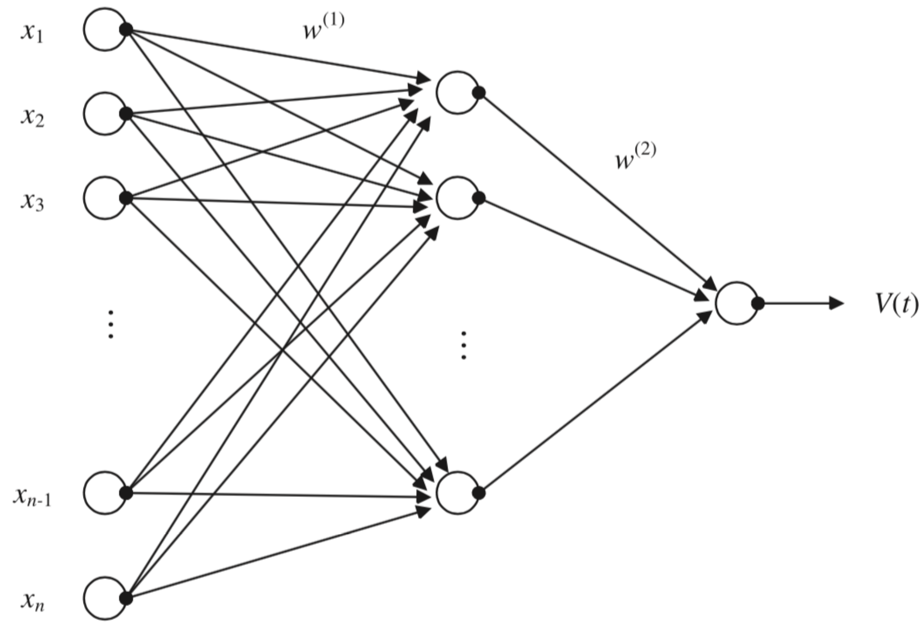
- Value function
  - Critics network

$$h_i(t) = \sum_{j=1}^{n} x_j(t) w_{ji}^{(1)}(t)$$

$$g_i(t) = \frac{1}{1+\exp^{-h_i(t)}}$$

$$p(t) = \sum_{i=1}^{m} w_i^{(2)}(t) g_i(t)$$

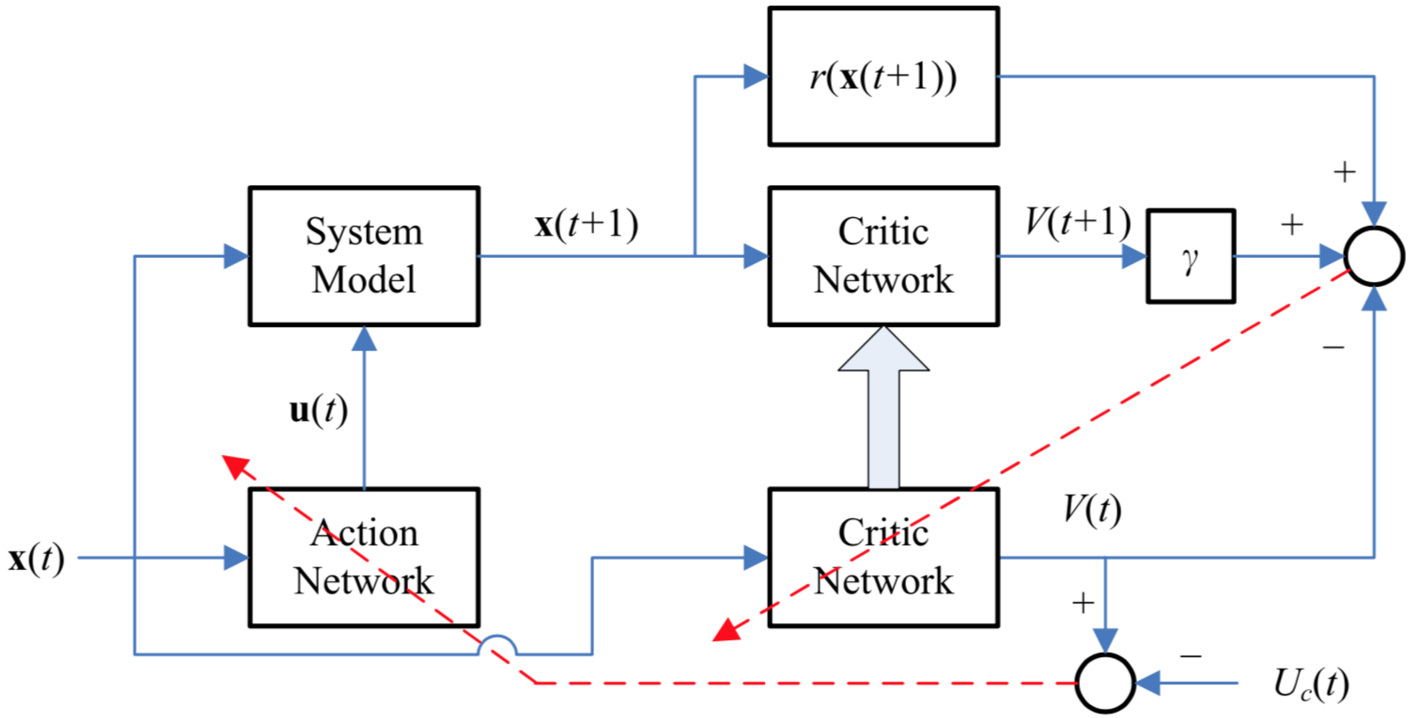
$$v(t) = \frac{1}{1+\exp^{-p(t)}}$$



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: Adaptive Dynamic Programming



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.

- Gomoku Rule
- Solve Gomoku
  - Genetic Algorithm
  - Monte Carlo Tree Search
  - Reinforcement Learning
    - Adaptive Dynamic Programming
  - Dependency-Based Search
    - Threat-Space Search
  - Proof-Number Search

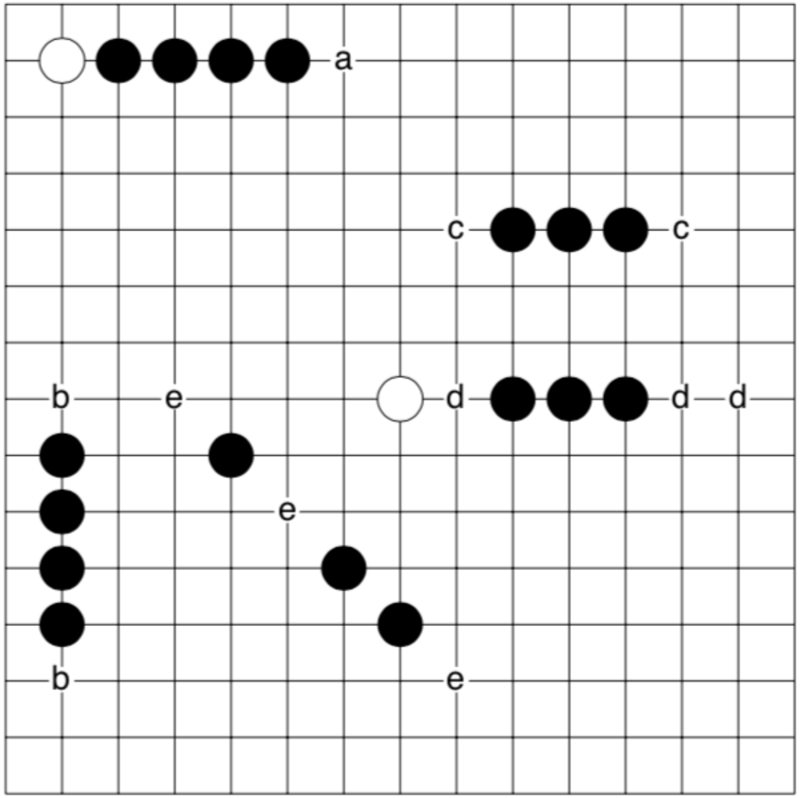
Solve Gomoku: Threat-Space Search

- Goal: Five in a row

Four in a row
Three in a row
…

Threat

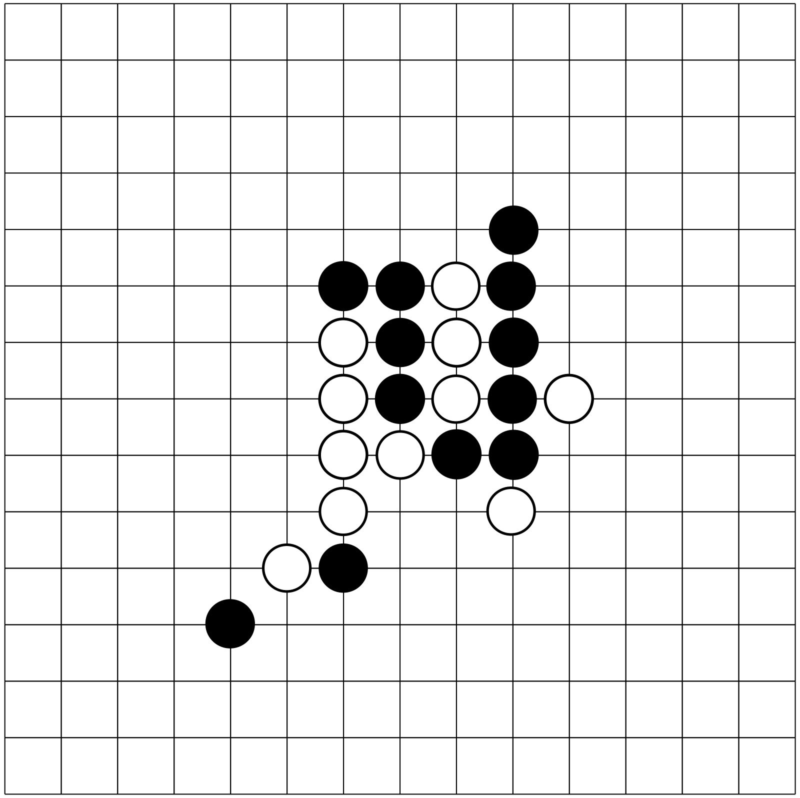
Threat Sequence



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

Solve Gomoku: Threat-Space Search

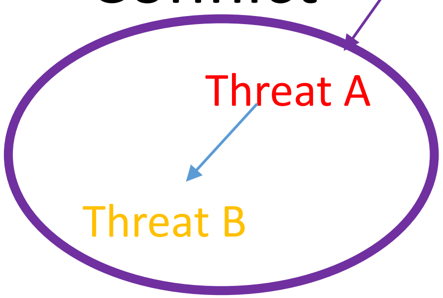
- Threat Sequence
- Winning Threat Sequence

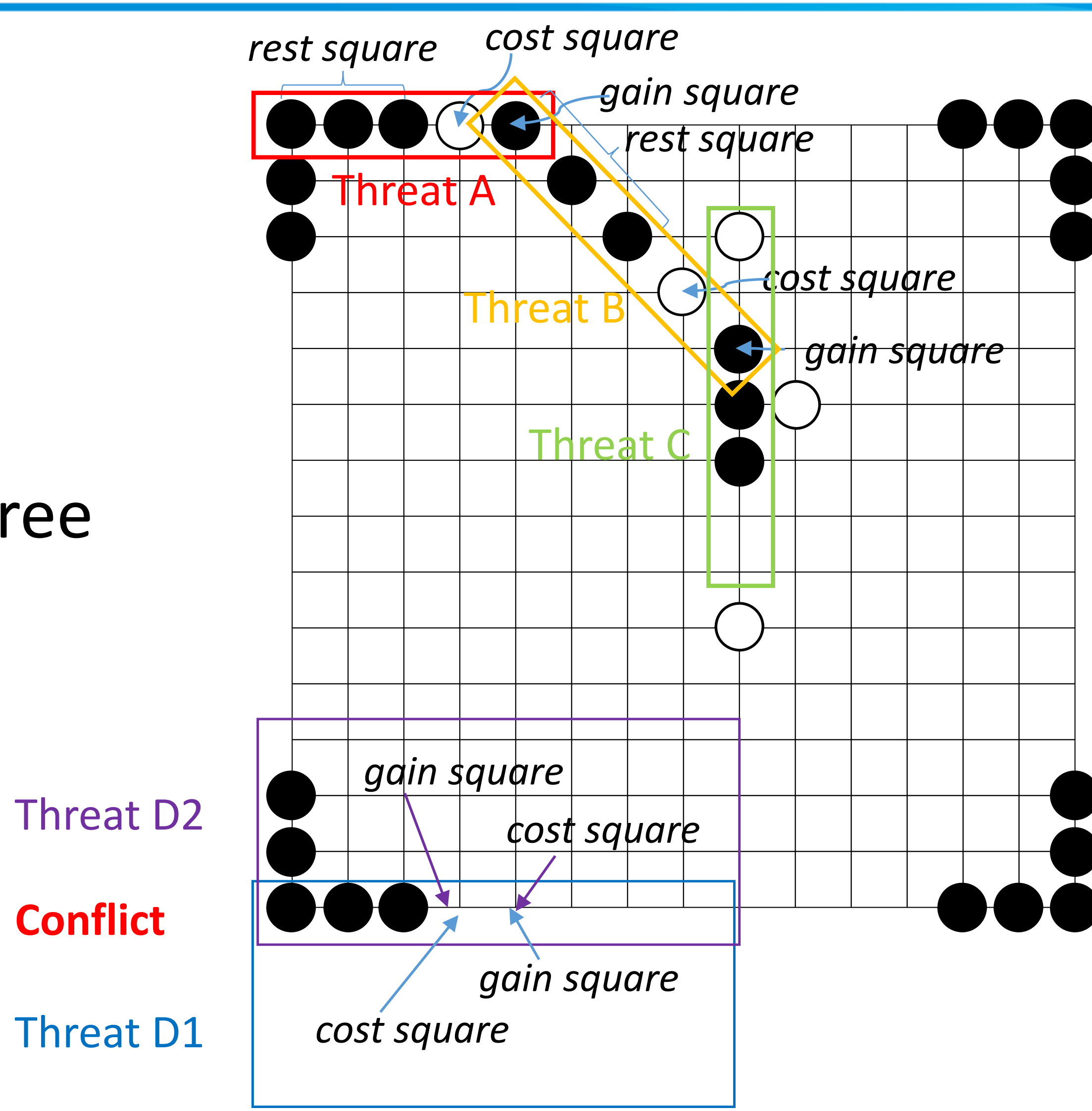


Louis Victor Allis and Hj Van Den Herik.
Go-moku and threat-space search.
. Ist. Psu. Edu/, 1993.

Solve Gomoku: Threat-Space Search

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



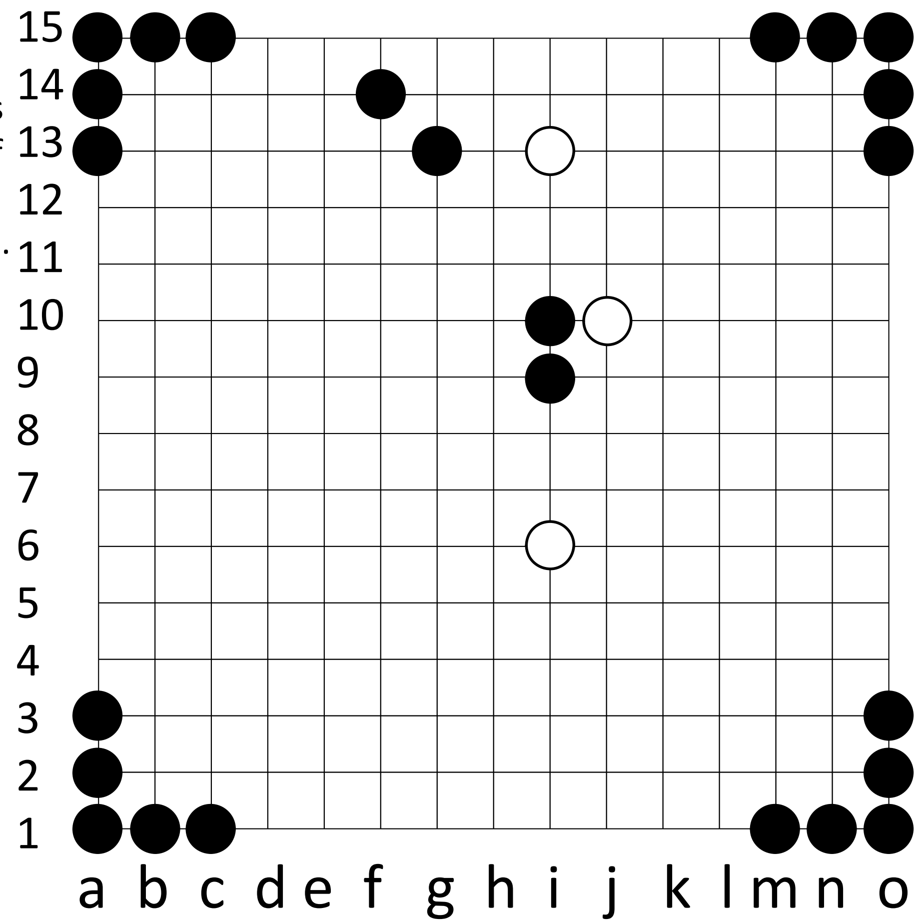


Louis Victor Allis and Hj Van Den Herik.
Go-moku and threat-space search.
. Ist. Psu. Edu/, 1993.

Solve Gomoku: Threat-Space Search

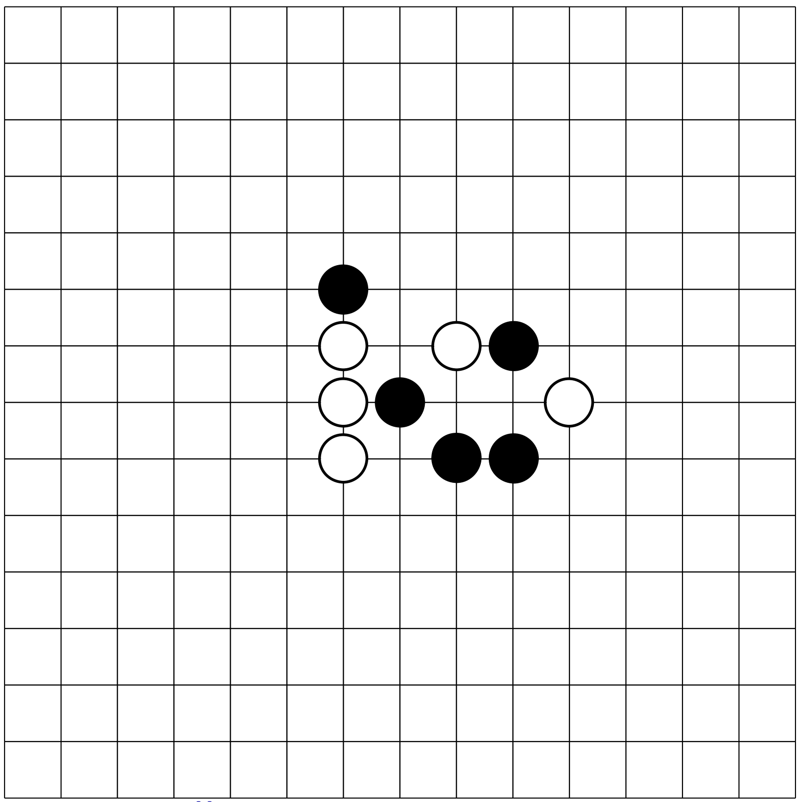
- Search tree:
  - Threat A being independent of threat B is not allowed to occur in the search tree of threat B.
  - Only threats for the attacker are included.

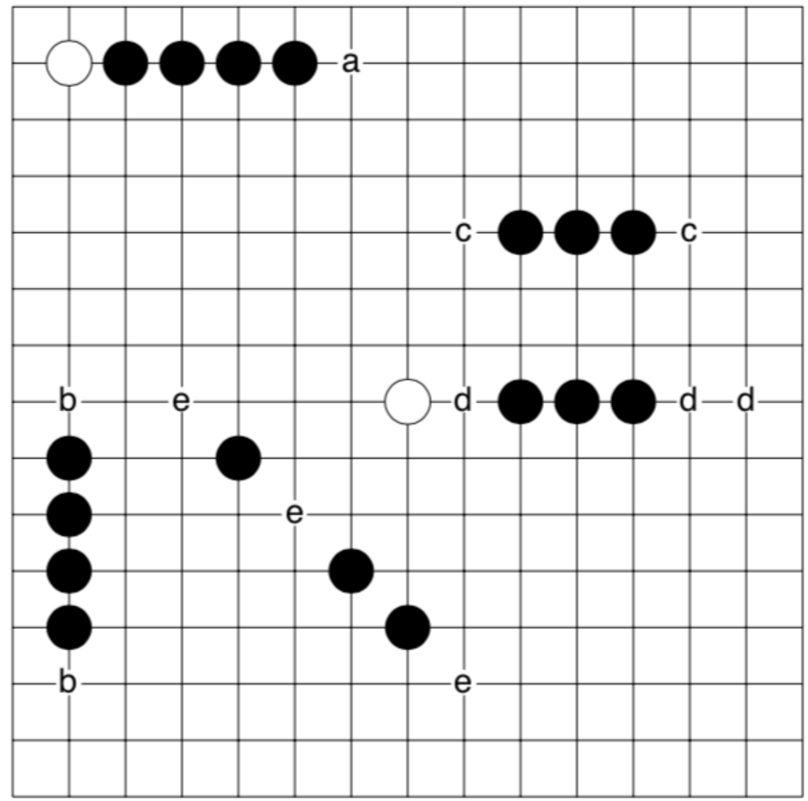
Depth	Type of threat	Gain square	Cost squares
1	Four	*l15*	*k15*
1	Four	*k15*	*l15*
1	Four	*e15*	*d15*
2	Four	*i11*	*h12*
3	Straight Four	*i8*	*i7*
2	Four	*h12*	*i11*
1	Four	*d15*	*e15*
1	Four	*o12*	*o11*
1	Four	*o11*	*o12*
1	Four	*a12*	*a11*
1	Four	*a11*	*a12*
1	Three	*i11*	*i7,i8,i12*
2	Four	*h12*	*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.

- Representation
  - Axiom
  - Structure





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

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
  - Genetic Algorithm
  - Monte Carlo Tree Search
  - Reinforcement Learning
    - Adaptive Dynamic Programming
  - Dependency-Based Search
    - Threat-Space Search
  - Proof-Number Search

Solve Gomoku: Proof-Number Search

- Board situation
  - 3 types: Win (1), Lose (0), unknown
- 2 players:
  - Black turn
    - Win if there is an action leading to Black win
    - Lose if all actions leading to Black lose
  - White turn
    - Win if all actions leading to Black win
    - Lose if there is an action leading to Black lose
  - Black: OR
  - White: AND

Win or Lose: BLACK's View

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

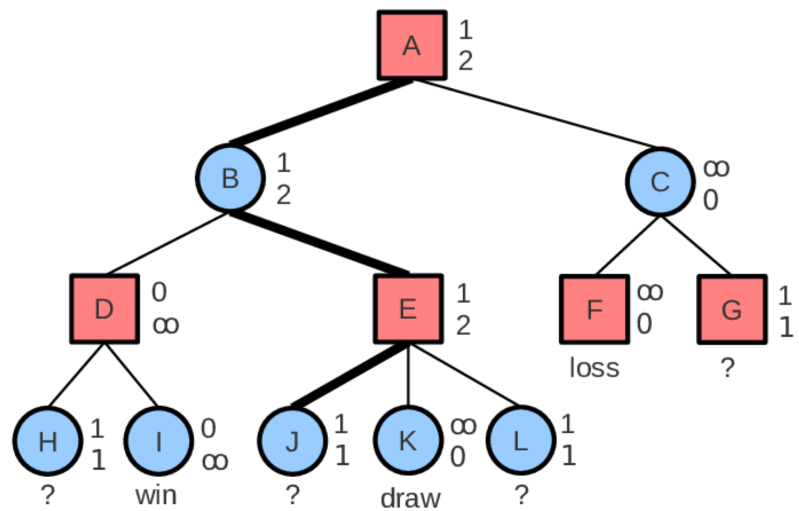
Solve Gomoku: Proof-Number Search

- AND/OR Tree
  - 3 Values: true, false, unknown
    - Terminal node: true, false
    - Frontier node: unknown
  - 2 Nodes: AND, OR
    - Black: OR
      - True if one child is true
      - Unknown if no true and has unknown
      - False if all children are false
    - White: AND
      - False if one child is false
      - Unknown if no false and has unknown
      - True if all children are true

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

Solve Gomoku: Proof-Number Search

- AND/OR Tree   AND: circle; OR: square
  - Proof number
    - 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 number
  - State of leaf nodes
    - Win: 0, ∞
    - Lose: 0, ∞
    - Unknown: 1, 1

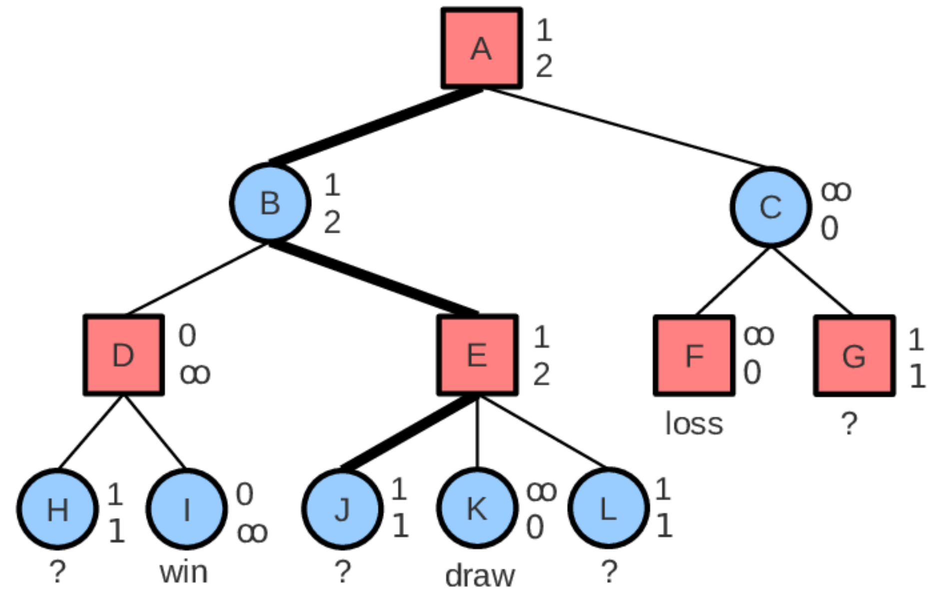


Louis Victor Allis.
Searching for Solutions in Games and Artificial Intelligence.
1994.

Solve Gomoku: Proof-Number Search

- AND/OR Tree
  - Proof number
    - AND: sum
    - OR: min
  - Disproof number
    - AND: min
    - OR: sum
- Proof-Number Search
  - 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.

- 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
```

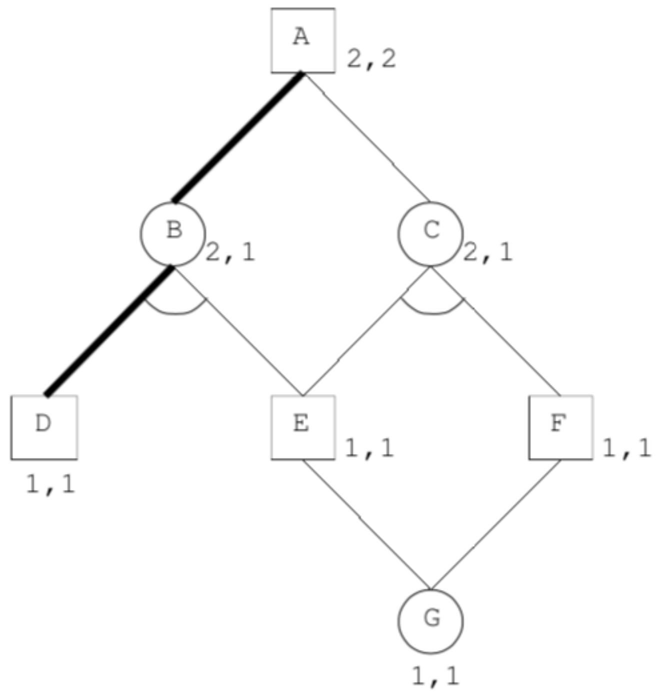
```
procedure DevelopNode(node);
  GenerateAllChildren(node);
   for i := 1 to node.numberOfChildren do
      Evaluate(node.children[i]);
      SetProofAndDisproofNumbers(node.children[i])
   od
end
```

```
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: Complementary

- Transposition
  - Hash table
  - Directed Acyclic Graphs



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

- 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/

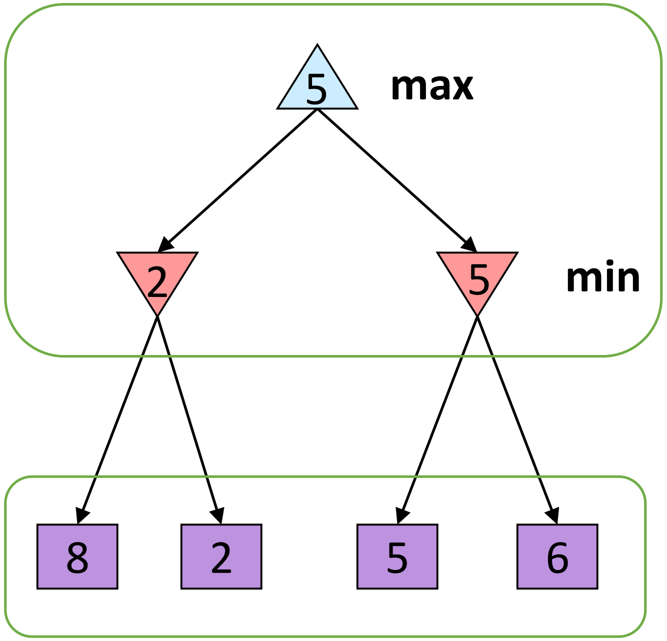
Outline

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

Lab2: Adversarial Search (Minimax)

- Deterministic, zero-sum games:
  - Tic-tac-toe, chess
  - One player maximizes result
  - The other minimizes result
- Minimax search:
  - A state-space search tree
  - Players alternate turns
  - Compute each node's minimax value: the best achievable utility against a rational (optimal) adversary

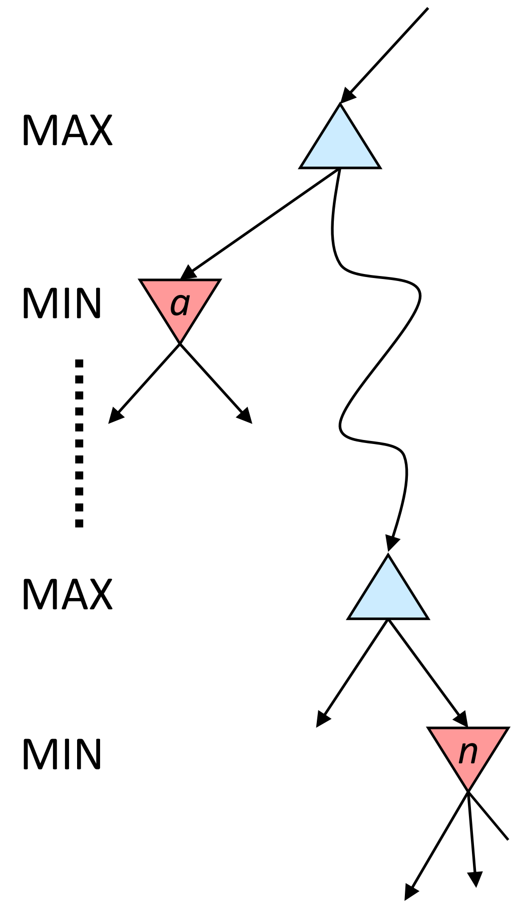
Minimax values: computed recursively



Terminal values: part of the game

Alpha-Beta Pruning

- General configuration (MIN version)
  - We're computing the MIN-VALUE at some node *n*
  - We're looping over *n*'s children
  - *n*'s estimate of the childrens' min is dropping
  - Who cares about *n*'s value? MAX
  - Let *a* be the best value that MAX can get at any choice point along the current path from the root
  - If *n* becomes worse than *a*, MAX will avoid it, so we can stop considering *n*'s other children (it's already bad enough that it won't be played)



Alpha-Beta Implementation

```
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).

Alpha-Beta Example

For MAX node

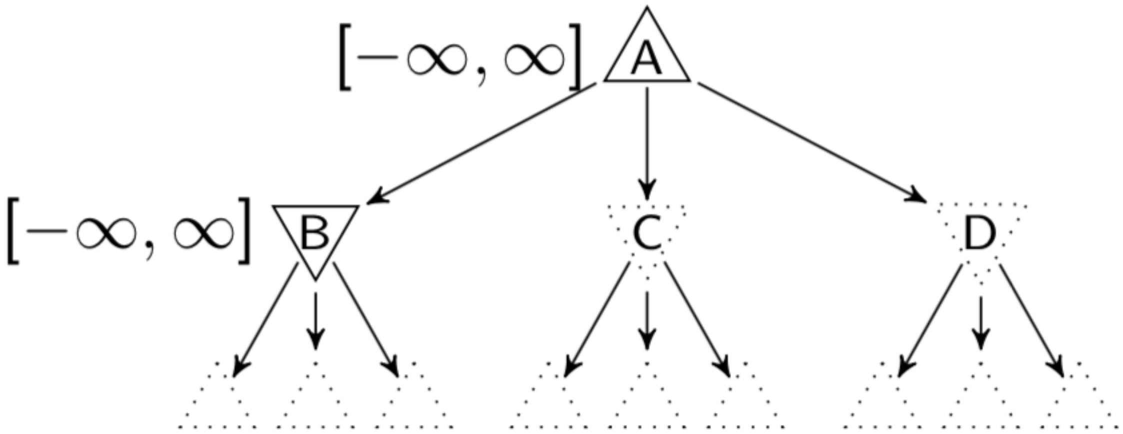
β is fixed as β_{parent}

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

For MAX node

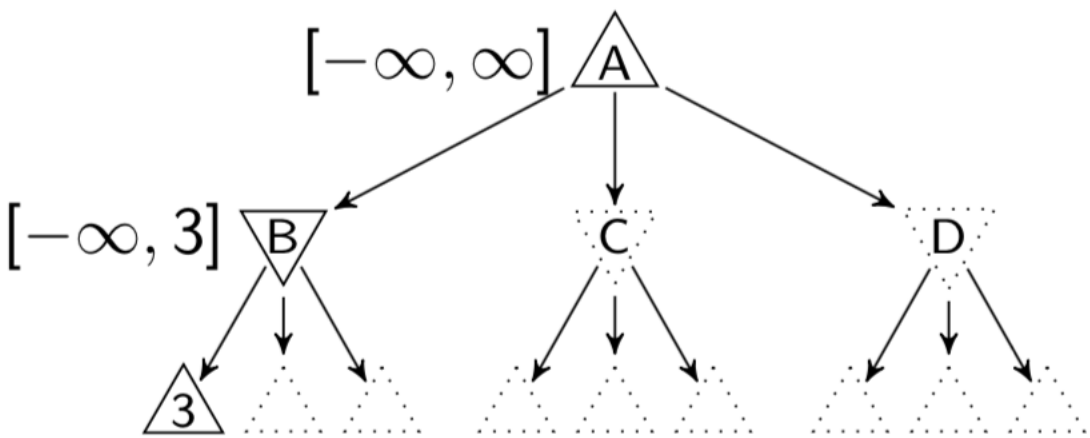
β is fixed as β_{parent}

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

For MAX node

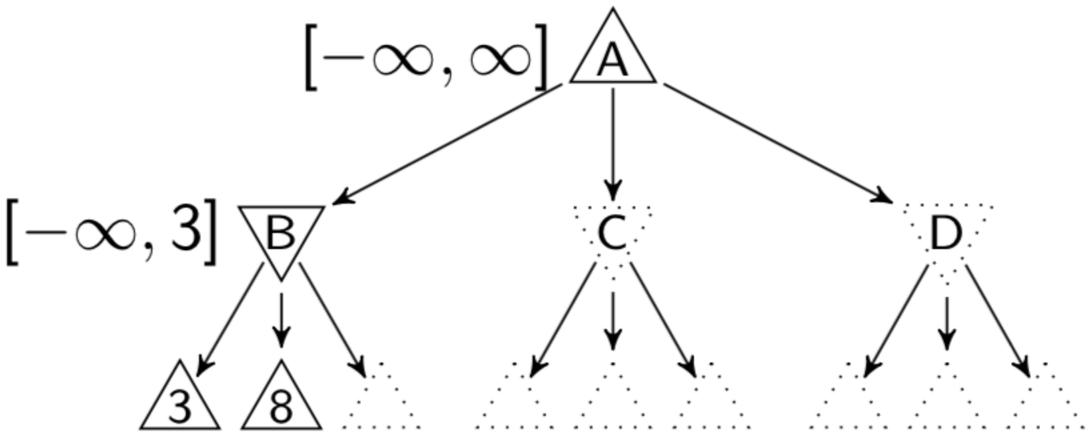
β is fixed as β_{parent}

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

For MAX node

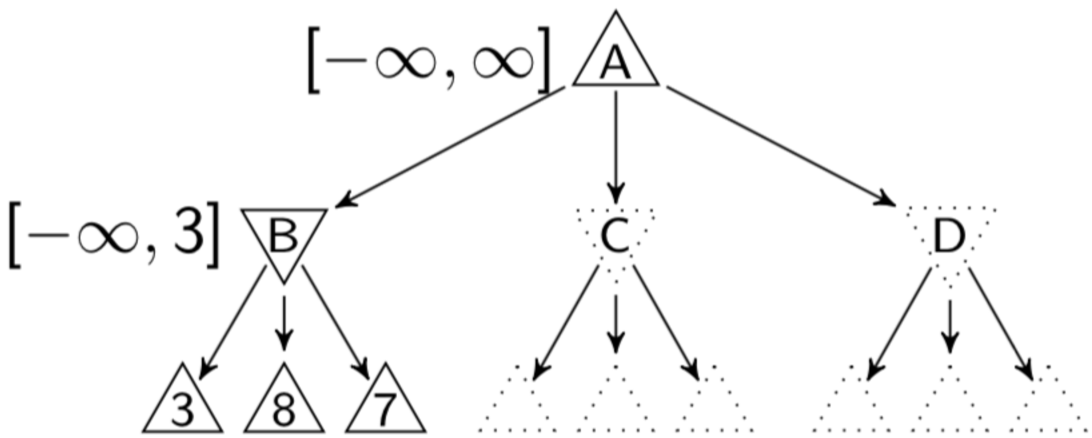
β is fixed as β_{parent}

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

For MAX node

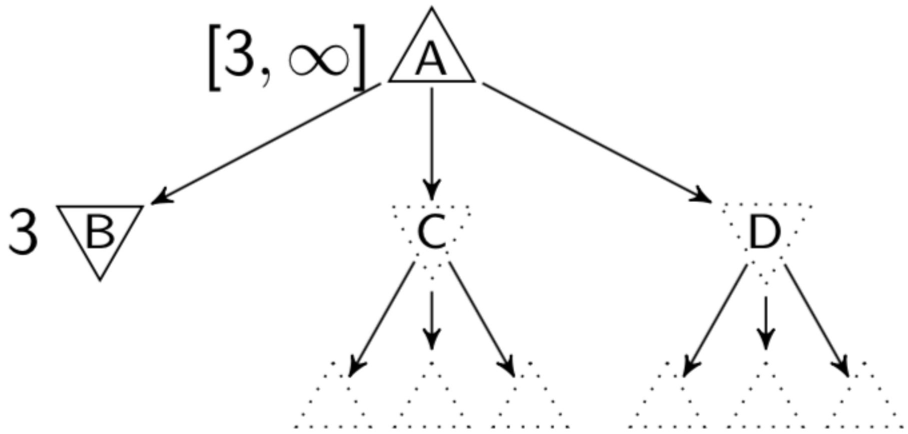
β is fixed as β_{parent}

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

For MAX node

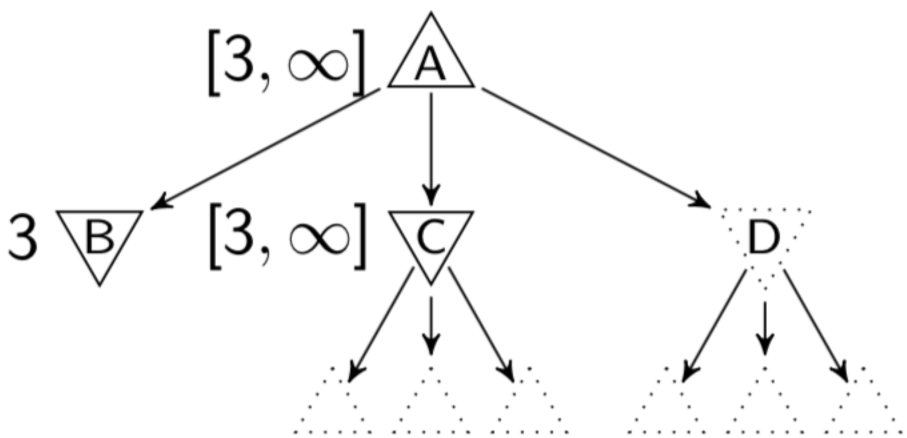
β is fixed as β_{parent}

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

For MAX node

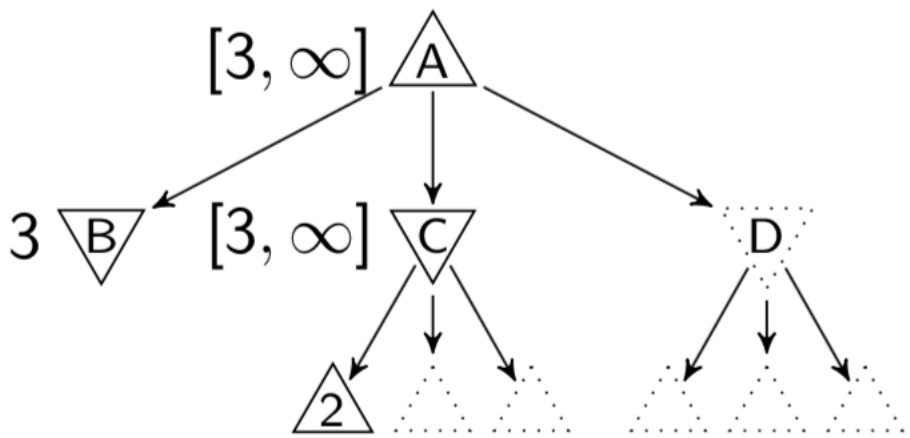
β is fixed as β_{parent}

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

For MAX node

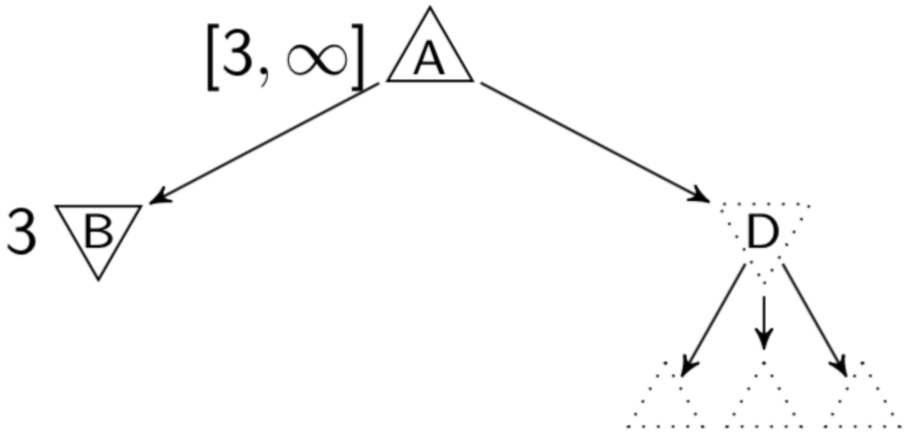
β is fixed as β_{parent}

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

For MAX node

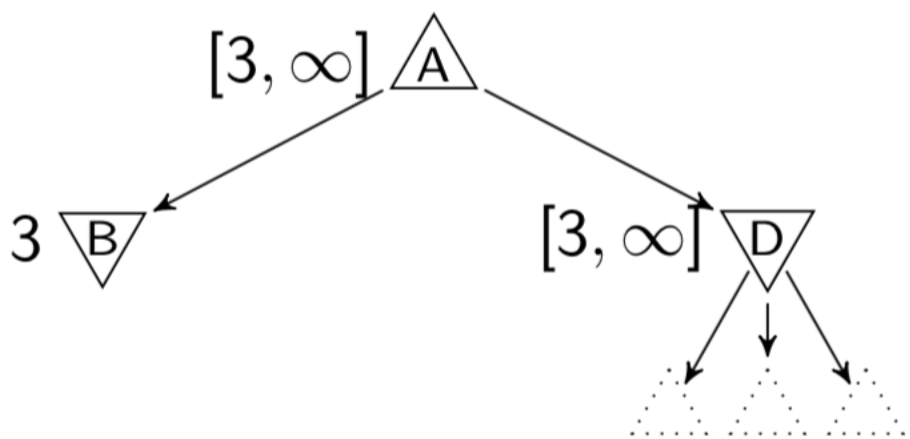
β is fixed as β_{parent}

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

For MAX node

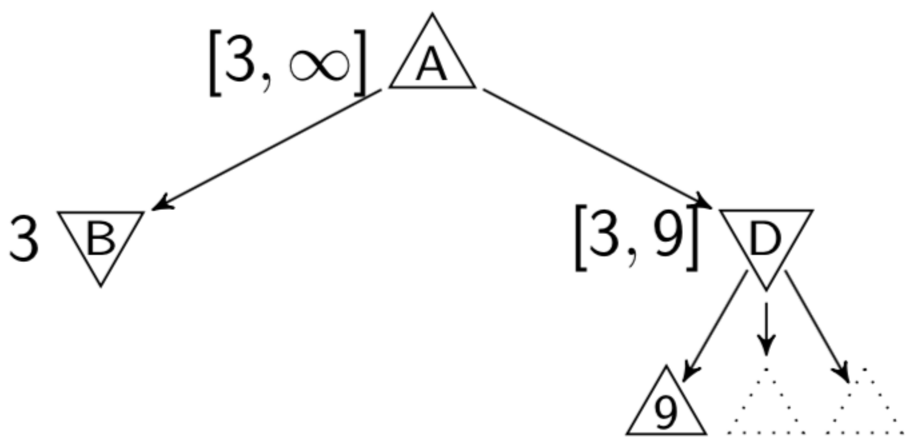
β is fixed as β_{parent}

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

For MAX node

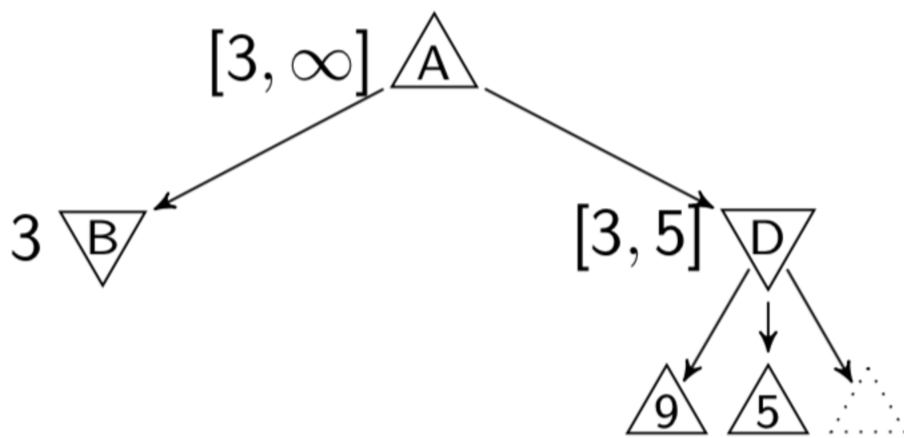
β is fixed as β_{parent}

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

For MAX node

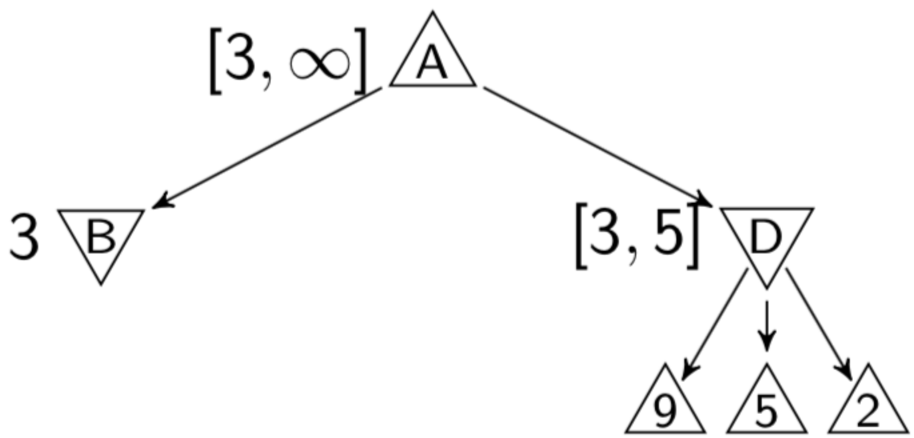
β is fixed as β_{parent}

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

For MAX node

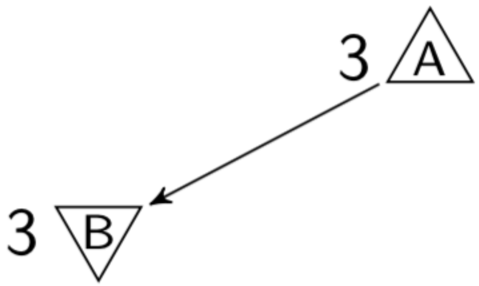
β is fixed as β_{parent}

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}

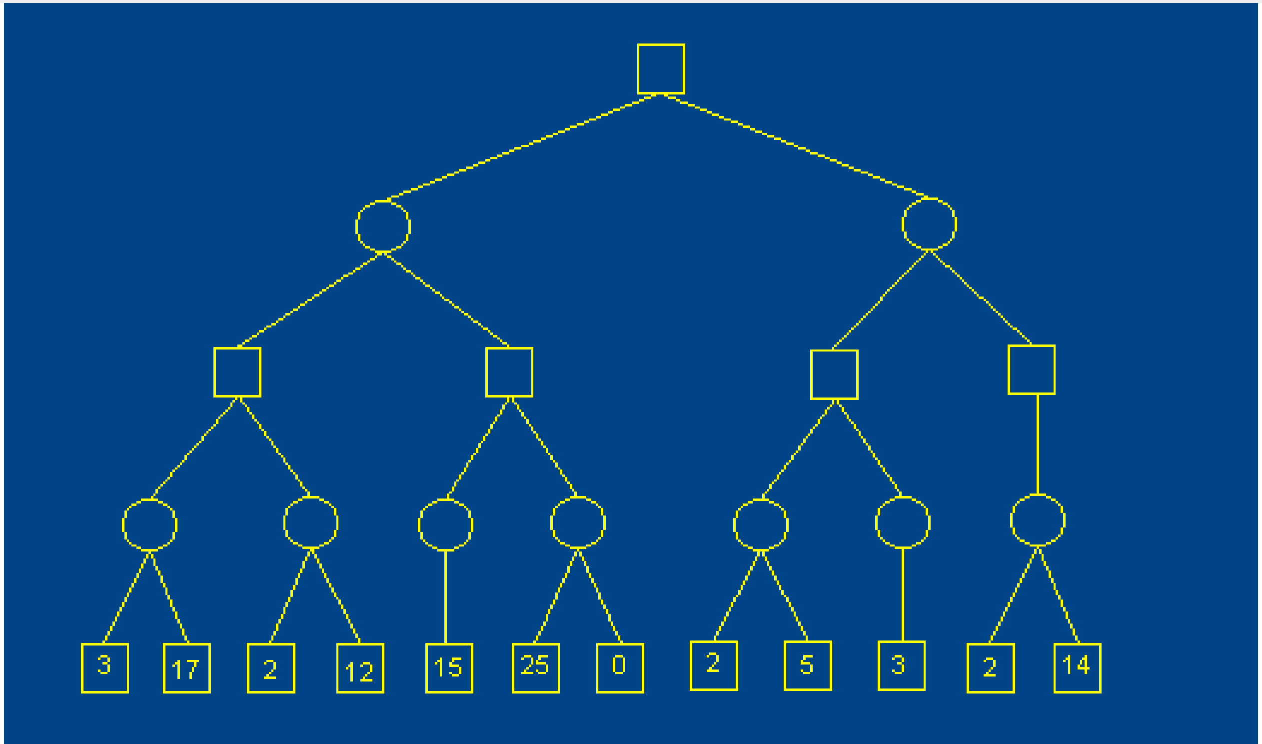


Practice for Alpha-beta Pruning

Input Example:

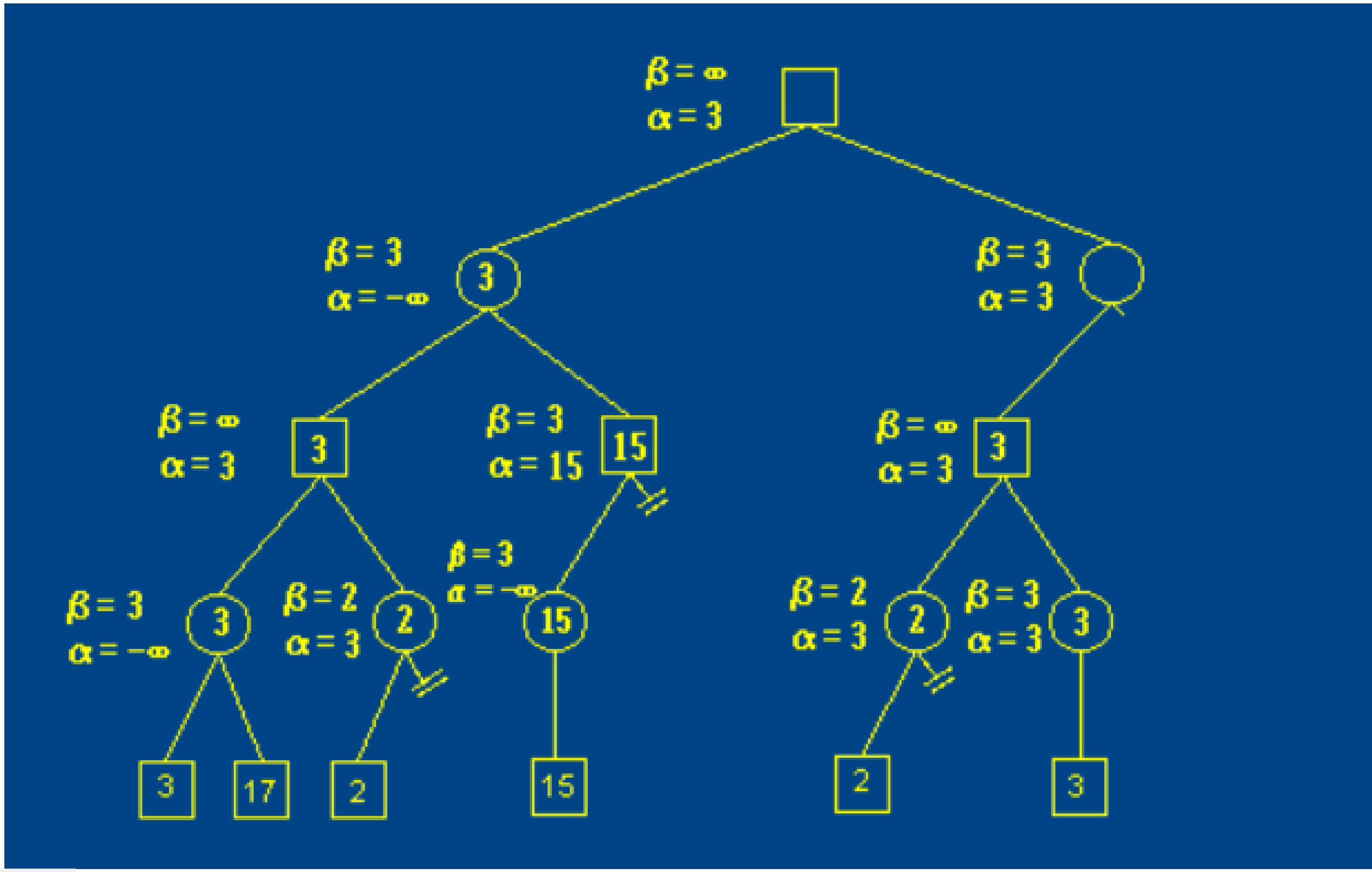
Square represents MAX node while circle stands for MIN node. For each test case, it contains two lines. The first line consists of two integers, the role of the root node (1 for MAX node and 0 for MIN node) and the depth of the tree. The second line is a nested list which stands for the game tree.

```
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

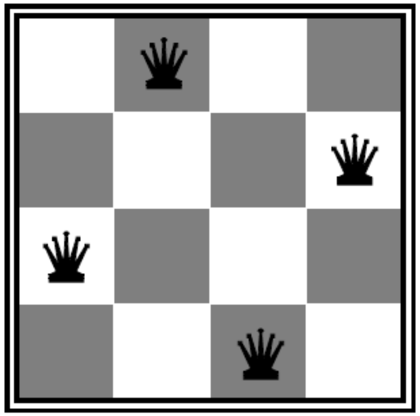
- Implement the following 3 functions:
  - **def** get_value(node, alpha, beta)
    - Choose which function to call
  - **def** max_value(node, alpha, beta)
  - **def** min_value(node, alpha, beta)

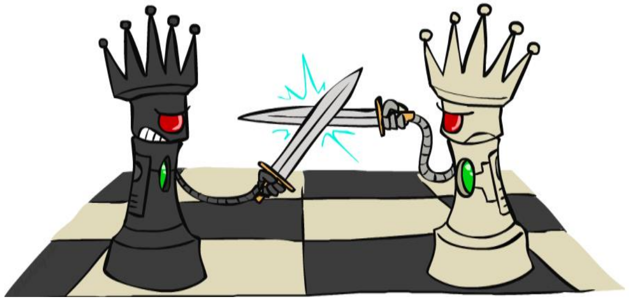
Outline

- Gomoku
  - Final project to be released in May
- Alpha-Beta Pruning
  - Submit in class via OJ
- **Constraint Satisfaction Problems**
  - **Take home as an assignment (Project 2)**
  - **20 points in total (it accounts for 7% in the course)**

CSP for N-Queens

- Formulation 1:
  - Variables: X_{ij}
  - Domains: $\{0, 1\}$
  - Constraints:



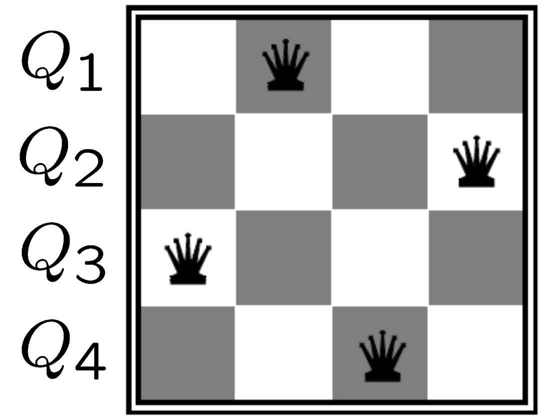


$$\forall i, j, k \ \ (X_{ij}, X_{ik}) \in \{(0,0), (0,1), (1,0)\}$$
$$\forall i, j, k \ \ (X_{ij}, X_{kj}) \in \{(0,0), (0,1), (1,0)\}$$
$$\forall i, j, k \ \ (X_{ij}, X_{i+k,j+k}) \in \{(0,0), (0,1), (1,0)\}$$
$$\forall i, j, k \ \ (X_{ij}, X_{i+k,j-k}) \in \{(0,0), (0,1), (1,0)\}$$

$$\sum_{i,j} X_{ij} = N$$

CSP for N-Queens

- Formulation 2:
  - Variables: Q_k
  - Domains: $\{1, 2, 3, \ldots N\}$
  - Constraints:



Implicit: $\forall i, j \;\; \text{non-threatening}(Q_i, Q_j)$

Explicit: $(Q_1, Q_2) \in \{(1,3), (1,4), \ldots\}$

$\cdots$

Code implement for CSP (csp.py)


```
class CSP:
    def __init__(self):
        self.vars_num = 0
        self.variables = []
        self.values = {}
        self.unary_factors = {}
        self.binary_factors = {}
    def add_variable(self, var, domain)
    def get_neighbor_vars(self, var)
    def add_unary_factor(self, var, factor_function)
    def add_binary_factor(self, var1, var2, factor_function)
    def _update_binary_factor_table(self, var1, var2, table)
```

- Problem a
  - Create an N-Queen problem on the board of size n * n.
    - O(n)

submission.py

```
class BacktrackingSearch:
        def __init__(self):
            self.num_assignments = 0
            self.num_operations = 0
            self.first_assignment_num_operations = 0
            self.all_assignments = []

            self.csp = None
            self.mcv = False
            self.ac3 = False
            self.domains = {}

        def reset_results(self)
        def check_factors(self, assignment, var, val)
        def solve(self, csp, mcv=False, ac3=False)
        def backtrack(self, assignment)
        def get_unassigned_variable(self, assignment)
        def arc_consistency_check(self, var)
```

Usage:
```
search = BacktrackingSearch()
search.solve(csp)
```

Code implement for Backtracking Search


```
def Backtracking(assignment):
    if all variables assigned:
        add current assignment to assignments
    else:
        var < - choose an unassigned variable
    for value in domain of var:
        add {var = value} to assignment
        Backtracking(assignment)
        delete var from assignment
```

Improve Backtracking Search

- You can improve the performance of CSP by 3 ways:
  - Ordering:
    - Which variable should be assigned next?
      - *Most Constrained Variable (MCV)*
        - *Also called Minimum remaining values (MRV)*
    - In what order should its values be tried? (LCV)
- Filtering: Can we detect inevitable failure early?
  - Arc-consistency checking (AC -3)
- Structure: Can we exploit the problem structure?
  - Tree structure
  - Cutting set conditioning

Improve Backtracking Search

- You can improve the performance of CSP by 3 ways:
  - Ordering:
    - Which variable should be assigned next?
      - *Most Constrained Variable (MCV)* Problem b
        - *Also called Minimum remaining values (MRV)*
    - In what order should its values be tried? (LCV)
- Filtering: Can we detect inevitable failure early?
  - Arc-consistency checking (AC -3). Problem c
- Structure: Can we exploit the problem structure?
  - Tree structure
  - Cutting set conditioning

Code implement for Backtracking Search


```
def Backtracking(assignment):
    if all variables assigned:
        add current assignment to assignments
    else:
        if MRC not implemented:
            randomly choose var from unassigned variables
        else(MRC implemented):
            choose variable by MCV
    for value in domain of var:
        add{var, value} to assignment
        if AC-3 not implemented:
            Backtracking(assignment)
        else(AC-3 implemented):
            localCopy <- store a deep copy for domains
            arc consistency checking
            Backtracking(assignment)
            use pre-stored localCopy to undo modify on domain
        delete assigned value for var
```

Problem b & c

- Problem b
  - Select a variable with the least number of remaining domain values.
  - Modify function *get_unassigned_variable* *such that* when mcv is *True,* it returns a variable with the least number of remaining domain values.
- Problem c
  - AC-3 algorithm: reduce the size of the domain values for the unassigned variables based on arc consistency.
  - Fill in function arc_consistency_check such that when ac3 is *True*, AC-3 will be used after each variable is made.

Problem b —— Most Constrained Variable

- **Select a variable with the least number of remaining domain**

Hint: self.domains[var] gives you all the possible values
Hint: get_delta_weight gives the change in weights given a partial assignment, a variable, and a proposed value to this variable
Hint: for ties, choose the variable with lowest index in self.csp.variables

Problem c —— AC-3

- After assigning new value to var, update domain for var's neighbors with AC-3.

Hint: get variables neighboring variable var:

```
self.csp.get_neighbor_vars(var)
```

Hint: check if a value or two values are inconsistent:

For unary factors

```
self.csp.unary_factors[var1][val1] == 0
```

For binary factors

```
self.csp.binary_factors[var1][var2][val1][val2] == 0
```

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 April 23rd, Tuesday, 11:55pm, 2019.**