

Game Playing Agent — Minimax — Alpha-Beta Pruning

Systematic adversarial search can lead to savings in terms of pruning of sub-trees resulting in lesser node evaluations

Ankur Singh
MTECH CSE (2020-22)
IIIT Vadodara
Gandhinagar, India
202061001@iiitvadodara.ac.in

Anand Mundhe
MTECH CSE (2020-22)
IIIT Vadodara
Gandhinagar, India
202061006@iiitvadodara.ac.in

I. PROBLEMS AND SOLUTIONS

- 1) What is the size of the game tree for Noughts and Crosses? Sketch the game tree.

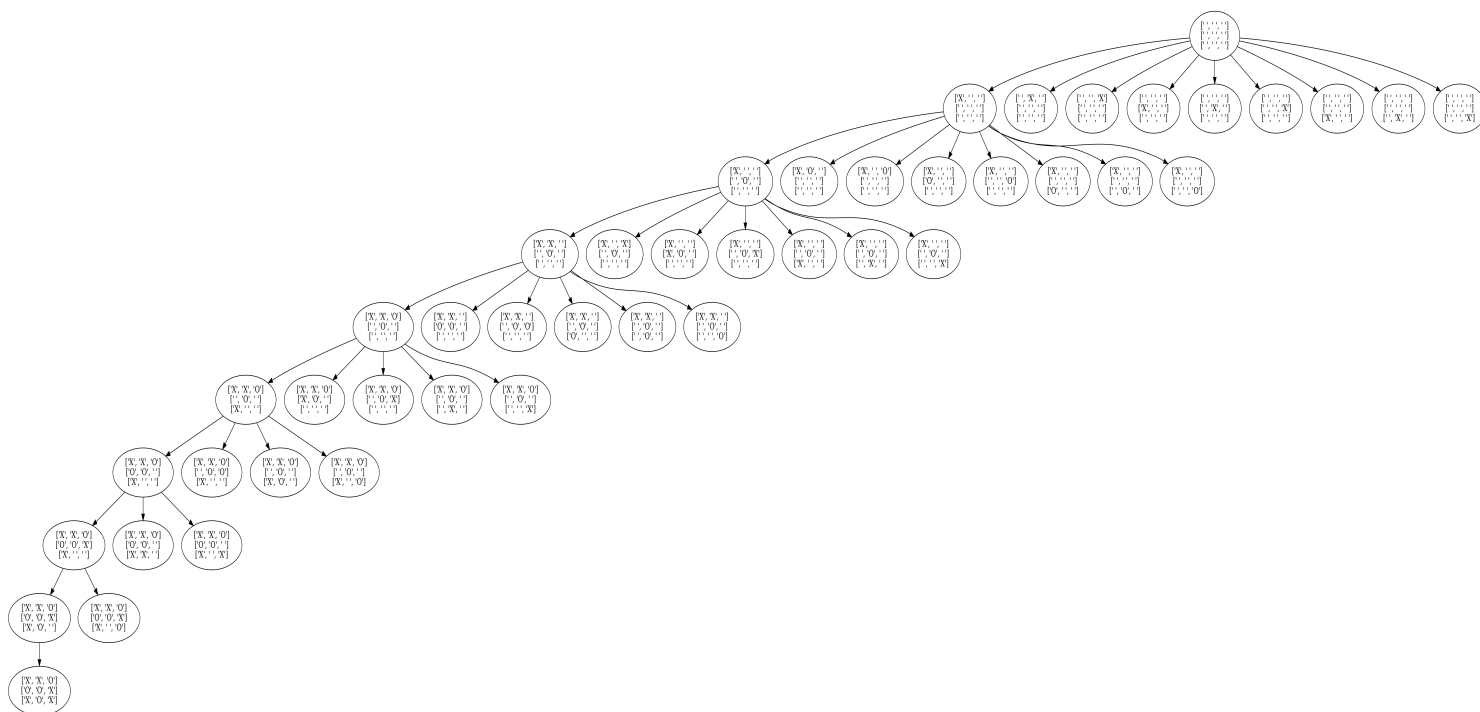


Fig. 1. Game tree of Noughts and Crosses

- 2) Read about the game of Nim (a player left with no move losing the game). For the initial configuration of the game with three piles of objects as shown in Figure, show that regardless of the strategy of player-1, player-2 will always win. Try to explain the reason with the MINIMAX value backup argument on the game tree.



Fig. 2. Pile of coins in game of Nim

The Game of Nim is defined by rules given as- Given a number of piles and each pile contains some number of coins. In each turn, a player can choose any number of coins (at least one) from the pile. The player with no move loses the game and the one who takes the last coin is the winner. The game depends on the factors-

- The player who starts first.
- The initial configuration of the piles

To understand the winning and losing condition of this game, it is necessary to get familiar about **Nim-Sum**. A Nim-Sum is the cumulative XOR value of the number of coins in each pile at any point in the game. The following properties of XOR sum makes the game interesting

- If the XOR sum is zero, then it is impossible to make the XOR sum stay zero by single reduction of a number.
- If the XOR sum is non zero, there exist at least one way by which the XOR sum can be made zero by reducing the number.

It is clear that the optimal strategy for each player is to make the Nim-Sum for the opponent zero in each turn. The given problem contains 10 coins in pile 1, 7 coins in pile 2 and 9 coins in pile 3. $10 \oplus 7 \oplus 9$ gives 4 which is a non zero value. Thus if player 2 is taking the first move then regardless the strategy adopted by player 1, player 2 will win because whatever move the player 1 takes it will definitely make the nim sum non zero only. [2]

- 3) Implement MINIMAX and alpha-beta pruning agents. Report on number of evaluated nodes for Noughts and Crosses game tree.

```
1  #Pruned AI
2  import math
3  class Node:
4      def __init__(self, state=[' ']*9, p1='X', p2='O', alpha=-math.inf, beta=-math.inf):
```

```
5      self.state=state
6      self.p=[p1,p2]
7      self.child=None
8      self.children=[]
9      self.ready=False
10     self.alpha=alpha
11     self.beta=beta
12     def winner(self, Type):
13         #Player 1 maximizer
14         x=lambda t:-1 if Type==0 else 1
15         #Row win for given player number
16         for i in range(3):
17             for j in range(3):
18                 if self.state[i*3+j]!=self.p[
Type]:break
19             elif j==2:return x(Type)
20         #Col win
21         for i in range(3):
22             for j in range(3):
23                 if self.state[j*3+i]!=self.p[
Type]:break
24             elif j==2:return x(Type)
25         #Diagonal win
26         if self.state[4]==self.p[Type]:
27             for i in [0,8]:
28                 if self.state[i]!=self.p[Type]:
29                     break
30             elif i==8:return x(Type)
31             for i in [2,6]:
32                 if self.state[i]!=self.p[Type]:
33                     break
34             elif i==6:return x(Type)
35         #no winner and no move mean Tie
36         if self.remaining_moves()==0:return 0
37         #otherwise None, not final state
38         return None
39     def remaining_moves(self):
40         return self.state.count(' ')
41     def getChildren(self):
42         out=[]
43         for i in range(len(self.state)):
44             if self.state[i]!=' ':
45                 out.append(i)
46         return out
47 class Agent:
48     def __init__(self):
49         self.NodesCount=0
50     def mm(self, node, Type):
51         #Type 0 min player
52         #Type 1 max player
53         if Type==0:cond=lambda a,b: a>b if a!=None else True
54         else:cond=lambda a,b: a<b if a!=None else True
55         #Check if already win
56         tmp=node.winner((Type+1)%2)
57         if tmp!=None:
58             return node, tmp*(node.remaining_moves()+1) #Utility
59         #Not win yet so check children
60         tnode=None
61         tmp=None
62         for i in node.getChildren():
63             #get child and make move
64             node.state[i]=node.p[Type]
65             #run same for child
66             a,b=self.mm(Node(node.state[:], node.p[0], node.p[1], node.alpha, node.beta), (Type+1)%2)
67             #Node count increases
68             self.NodesCount+=1
69             #Append child in node's children
70             node.children.append(a)
71             #b is Utility and is none when at
```

```

leaf node (Win Draw or Lose)
70         if b!=None and cond(tmp,b):
71             #Assign alpha and beta values
to node accordingly
72         if Type==0:
73             if node.beta>b:node.beta=b
74         else:
75             if node.alpha<b:node.alpha=
b
76             tmp=b
77             tnode=a
78             #Undo the last move
79             node.state[i]=' '
80             #Prune next children if alpha>beta
81             if node.alpha>node.beta:break
82             node.child=tnode
83             node.ready=True
84             return node,tmp
85
86         #Driver for Pruned AI
87 class Game:
88     #initilise game
89     def __init__(self,p1,p2):
90         self.node=Node([' ']*9,p1=p1,p2=p2)
91     #Print state of game
92     def pstate(self):
93         for i in range(3):
94             for j in range(3):
95                 print(self.node.state[i*3+j],
end="\t|\t")
96                 print()
97     #play a move either player 0 or 1
98     def play(self,Type):
99         #Type 0 min player, User player
100         #Type 1 max player, Computer player
101         #User input variable
102         ip=-1
103         if Type==0:
104             #Check if ip is valid, and take
input till valid input is there
105             while not ip in self.node.
getChildren():
106                 ip=int(input("Enter next move :
"+str(self.node.getChildren())))
107                 if(not ip in self.node.
getChildren()):print("Invalid move")
108                 #Apply input to state
109                 self.node.state[ip]=self.node.p[
Type]
110                 #Computer's move
111             else:
112                 #Initially agent is not executed,
so run once
113                 if not self.node.ready:
114                     #return node with children and
utility value
115                     obj=Agent()
116                     self.node.alpha=-math.inf
117                     self.node.beta+=math.inf
118                     self.node.win=obj.mm(self.node
,1)
119                     print("Smart AI is ready with
total node exploration :",obj.NodesCount,"
With result : ",win)
120                     #Make 1 move by heading to
child
121                     self.node=self.node.child
122                     #Now agent is ready
123                     self.node.ready=True
124                     return
125                 #from second time onward, Check
children for user move's node
126                 for i in self.node.children:
127                     #check till user move node is

```

```

found
128         if i.state==self.node.state:
129             #from that user move node,
if move is possible then make move
130             if i.child!=None:
131                 self.node=i.child
132                 #again agent is still
ready
133                 self.node.ready=True
134                 return
135     #Execution of game from here
136     def drive(self):
137         #initial player -1
138         turn=-1
139         #Play untill no more moves are possible
or node is None
140         while self.node!=None and self.node.
remaining_moves()!=0:
141             #This shows utility value for final
state
142             print("Player 1 win : ",self.node.
winner(0))
143             print("Player 2 win : ",self.node.
winner(1))
144             #display current state
145             self.pstate()
146             #User's move
147             self.play((turn+1)%2)
148             #Check if user win
149             if self.node.winner(0)==-1:
150                 print("You win")
151                 break
152             #Check if game tie
153             if self.node.winner(0)==0:
154                 print("Game is tie")
155                 break
156
157             #Computer's move
158             self.play((turn+2)%2)
159             #Check if computer wins
160             if self.node!=None and self.node.
winner(1)==1:
161                 print("Computer wins...")
162                 break
163             #Check if no child mean computer
cant make move
164             if self.node==None:break
165             #Check if game tie
166             if self.node.winner(1)==0:
167                 print("Game is tie")
168                 break
169             #Show's final states when game is
finished
170             if self.node!=None:
171                 self.pstate()
172                 print("Player 1 win : ",self.node.
winner(0))
173                 print("Player 2 win : ",self.node.
winner(1))
174             #When no child availabe, computer is
stuck
175             else:
176                 print("Computer cannot make move")
177
178             Game(0,1).drive()
179
180

```

```

1     #Pruned AI computer vs computer
2 class Game:
3     #initilise game
4     def __init__(self,p1,p2):
5         self.node=Node([' ']*9,p1=p1,p2=p2)
6     #Print state of game

```

```

Player 1 win : None
Player 2 win : None

Enter next move : [0, 1, 2, 3, 4, 5, 6, 7, 8]0
Smart AI is ready with total node exploration : 10556 With result : 0
Player 1 win : None
Player 2 win : None
0
1
Enter next move : [1, 2, 3, 5, 6, 7, 8]8
Player 1 win : None
Player 2 win : None
0
1
1
0
Enter next move : [2, 3, 5, 6, 7]7
Player 1 win : None
Player 2 win : None
0
1
1
0
1
0
Enter next move : [2, 3, 5]2
Player 1 win : None
Player 2 win : None
0
1
1
0
1
0
Enter next move : [3]3
Game is tie
0
0
1
0
1
0
Player 1 win : 0
Player 2 win : 0

```

Fig. 3. Output of game

```

7 def pstate(self):
8     for i in range(3):
9         for j in range(3):
10            print(self.node.state[i*3+j],
11                end="\t\t")
12            print()
13 #play a move either player 0 or 1
14 def play(self,Type):
15     #Type 0 min player, User player
16     #Type 1 max player, Computer player
17     #User input variable
18     ip=-1
19     if Type==0:
20         #Check if ip is valid, and take
21         input till valid input is there
22         while not ip in self.node.
23         getChildren():
24             ip=int(input("Enter next move :
25             "+str(self.node.getChildren())))
26             if(not ip in self.node.
27             getChildren()):print("Invalid move")
28             #Apply input to state
29             self.node.state[ip]=self.node.p[
30             Type]
31             #Computer's move
32     else:
33         #Initially agent is not executed,
34         so run once
35         if not self.node.ready:
36             #return node with children and
37             utility value
38             obj=Agent()
39             self.node.alpha=-math.inf

```

```

32 self.node.beta=math.inf
33 self.node.win=obj.mm(self.node
34 ,0)
35 print("Smart AI is ready with
36 total node exploration :",obj.NodesCount)
37 #Make 1 move by heading to
38 child
39 self.node=self.node.child
40 #Now agent is ready
41 self.node.ready=True
42 return
43 #from second time onward, Check
44 children for user move's node
45 self.node=self.node.child
46 #Execution of game from here
47 def drive(self):
48     #initial player -1
49     turn=-1
50     #Play untill no more moves are possible
51     or node is None
52     while self.node!=None and self.node.
53     remaining_moves()!=0:
54         #This shows utility value for final
55         state
56         print("Player 1 win : ",self.node.
57         winner(0))
58         print("Player 2 win : ",self.node.
59         winner(1))
60         #display current state
61         self.pstate()
62         #Computer 1's move
63         self.play((turn+2)%2)
64         #Check if computer win
65         if self.node.winner(0)==-1:
66             print("You win")
67             break
68         #Check if game tie
69         if self.node.winner(0)==0:
70             print("Game is tie")
71             break
72         #This shows utility value for final
73         state
74         print("Player 1 win : ",self.node.
75         winner(0))
76         print("Player 2 win : ",self.node.
77         winner(1))
78         #display current state
79         self.pstate()
80         #Computer 2's move
81         self.play((turn+2)%2)
82         #Check if computer wins
83         if self.node!=None and self.node.
84         winner(1)==1:
85             print("Computer wins...")
86             break
87         #Check if no child mean computer
88         cant make move
89         if self.node==None:break
90         #Check ig game tie
91         if self.node.winner(1)==0:
92             print("Game is tie")
93             break
94         #Show's final states when game is
95         finished
96         if self.node!=None:
97             self.pstate()
98             print("Player 1 win : ",self.node.
99             winner(0))
100            print("Player 2 win : ",self.node.
101            winner(1))
102            #When no child availabe, computer is
103            stuck

```



```

61         #get child and make move
62         node.state[i]=node.p[Type]
63         #run same for child
64         a,b=self.mm(Node(node.state[:],node
        .p[0],node.p[1],node.alpha,node.beta),(Type
        +1)%2)
65         #Node count increases
66         self.NodesCount+=1
67         #Append child in node's children
68         node.children.append(a)
69         #b is Utility and is none when at
        leaf node (Win Draw or Lose)
70         if b!=None and cond(tmp,b):
71             tmp=b
72             tnode=a
73             node.state[i]=' '
74             node.child=tnode
75             node.ready=True
76             return node,tmp
77
78     #Driver game for Ai vs pruned AI
79 class Game:
80     #initilise game
81     def __init__(self,p1,p2):
82         self.node=Node([' ']*9,p1=p1,p2=p2)
83     #Print state of game
84     def pstate(self):
85         for i in range(3):
86             for j in range(3):
87                 print(self.node.state[i*3+j],
88 end="\t|\t")
89         print()
90     #play a move either player 0 or 1
91     def play(self,Type):
92         #Type 0 min player, User player
93         #Type 1 max player, Computer player
94         #Computer's input variable
95         if Type==0:
96             #Initially agent is not executed,
97             so run once
98             if not self.node.ready:
99                 #return node with children and
100                 utility value
101                 obj=AgentUnprune()
102                 self.node1,win=obj.mm(self.node
103                 ,0)
104                 print("Smart AI is ready with
105                 total node exploration :",obj.NodesCount)
106                 #Make 1 move by heading to
107                 child
108                 self.node=self.node1.child
109                 self.node1=self.node1.child
110                 #Now agent is ready
111                 self.node.ready=False
112                 return
113                 #from second time onward, Check
114                 children for user move's node
115                 for i in self.node1.children:
116                     #check till user move node is
117                     found
118                     if i.state==self.node.state:
119                         #from that user move node,
120                         if move is possible then make move
121                         if i.child!=None:
122                             self.node=i.child
123                             self.node1=i.child
124                             #again agent is still
125                             ready
126                             self.node.ready=True
127                             return
128                 #Computer's move
129                 else:
130                     #Initially agent is not executed,
131                     so run once

```

```

121         if not self.node.ready:
122             #return node with children and
123             utility value
124             obj=Agent()
125             self.node.alpha=-math.inf
126             self.node.beta=math.inf
127             self.node2,win=obj.mm(self.node
128             ,1)
129             print("Smart AI is ready with
130             total node exploration :",obj.NodesCount)
131             #Make 1 move by heading to
132             child
133             self.node=self.node2.child
134             self.node2=self.node2.child
135             #Now agent is ready
136             self.node.ready=True
137             return
138             #from second time onward, Check
139             children for user move's node
140             for i in self.node2.children:
141                 #check till user move node is
142                 found
143                 if i.state==self.node.state:
144                     #from that user move node,
145                     if move is possible then make move
146                     if i.child!=None:
147                         self.node=i.child
148                         self.node2=i.child
149                         #again agent is still
150                         ready
151                         self.node.ready=True
152                         return
153             #Execution of game from here
154             def drive(self):
155                 #initial player -1
156                 turn=-1
157                 #Play untill no more moves are possible
158                 or node is None
159                 while self.node!=None and self.node.
160                 remaining_moves()!=0:
161                     #This shows utility value for final
162                     state
163                     print("Player 1 win : ",self.node.
164                     winner(0))
165                     print("Player 2 win : ",self.node.
166                     winner(1))
167                     #display current state
168                     self.pstate()
169                     #User's move
170                     self.play((turn+1)%2)
171                     #Check if user win
172                     if self.node.winner(0)==-1:
173                         print("Computer 1 win")
174                         break
175                     #Check if game tie
176                     if self.node.winner(0)==0:
177                         print("Game is tie")
178                         break
179                     #This shows utility value for final
180                     state
181                     print("Player 1 win : ",self.node.
182                     winner(0))
183                     print("Player 2 win : ",self.node.
184                     winner(1))
185                     #display current state
186                     self.pstate()
187                     #Computer's move
188                     self.play((turn+2)%2)
189                     #Check if computer wins
190                     if self.node!=None and self.node.
191                     winner(1)==1:
192                         print("Computer 2 wins...")
193                         break

```


- 4) Using recurrence relation show that under perfect ordering of leaf nodes, the alpha-beta pruning time complexity is $O(b^{m/2})$, where b is the effective branching factor and m is the depth of the tree.

Let $T(m)$ be the search complexity for depth m . Normally under minimax algorithm without alpha beta pruning, the recurrence relation will be

$$T(m) = b.T(m-1) + c$$

The time complexity would be $O(b^m)$. For alpha-beta pruning the recurrence relation would be

$$T(m) = T(m-1) + (b-1).T(m-2) + c$$

because it is necessary to know the value of first child i.e. $T(m-1)$ and this child has sub-tree of height $m-1$. For the other children, it is required to compute value at one of their children at depth $m-2$. Solving this,

$$T(m) = T(m-2) + (b-1).T(m-3) + (b-1).T(m-2) + c$$

$$T(m) = b.T(m-2) + (b-1).T(m-3) + c$$

It is evident that $T(m-3) < T(m-2)$, so:

$$T(m) < (2b-1).T(m-2)$$

$$T(m) < 2b.T(m-2)$$

i.e., the branching factor every two levels is less than $2b$ which means that the effective branching factor is less than $\sqrt{2b}$. This is not too far off the asymptotic upper bound of $(\sqrt{b}+1)^{m+1}$ hence, giving the final time complexity as $O(b^{m/2})$. This is a substantial improvement. It does not affect the final result. With perfect ordering, it doubles the depth of search. [3]

REFERENCES

- [1] Artificial Intelligence: a Modern Approach, Russell and Norvig (Fourth edition)
- [2] Combinatorial Game Theory — Set 2 (Game of Nim). <https://www.geeksforgeeks.org/combinatorial-game-theory-set-2-game-nim/>
- [3] Best-Case Analysis of Alpha-Beta Pruning
<http://www.cs.utsa.edu/~bylander/cs5233/a-b-analysis.pdf>