Artificial Intelligence Laboratory

Heuristic Search Algorithms

Kunal Kumar 170010012
Aditya Jha 170010011

---

1. Brief description about the domain:

Our Task is based on Rugby Game. We have two teams denoted by '0' and '+' as players in test cases. Our task is to reach the goal with ball based on heuristic function. We also need to save the ball from opponents team. Any Opponents can take ball if Ball passes from his neighbour state.

1. State space:

We have 4 types of state. Same team members are denoted by '0' in input matrix. We have to pass the ball to our team player. Opponents are denoted by '+' in input matrix. We have target state denoted by '*' in matrix. Rest of the free space in input matrix are free space where players can move.

2. Start node and goal node:

We are starting the game from corner of Ground. Here we have a 2-D matrix , so we are starting from position (0,0) in matrix.
Our target is to reach the Goalpost which we have denoted by '*' in input matrix. We can vary this goal state by changing position of * in input file. Our program will stop once we reached the goal state.

3. MOVEGEN and GOALTEST algorithm:

Our Movegen Function is based on path between players.
i.e. if we can direct reach to teammates without crossing any players, than those player's state is the output of our movegen function. We used *Breadth first search* Algorithm with some modification to get neighbouring state. Then we use different heuristic function to get best neighbours.

```
# MoveGen function returns all nodes around given node
adjacent = MoveGen(temp)    # getting neighbours
 # removing already visited node
adjacent= [node for node in adjacent if node.dis==-1]
for node in adjacent:
      node.dis = temp.dis + 1    # updating distance
      node.parent = temp         # assigning parent
opens = [node for node in opens if node not in close]
opens.sort(key=lambda x: x.d)   # ordering based on
heuristic function
```

As we have already marked our goal node by '*' we have exact location of goal state. So we are just comparing coordinates of every node with goal node. If they matched, we are returning *True*.

```
 # checking goal
if node.x == goal_x and node.y== goal_y:
      return True  # goal_x and goal_y is the
coordinates of goal state.
```

2. Heuristic functions considered:

We used two different heuristic function.

1. Our first heuristic function is based on Euclidean distance. We calculated distance of all child node of a parent from goal state. We choose the one with lesser Euclidean distance from goal state as our best choice.

```
for node in friends:     # list of all child
       xx = node.x    # coordinates of node
       yy = node.y
       euclidean_distance = ((goal_x-xx)**2 +
(goal_y-yy)**2)**(0.5)
       node.d = euclidean_distance    # updating value
```

2. Our second heuristic function is based on path length from goal state. We used *Breadth first search* to find the shortest path length.

```
queue = list()
   array[goal_x][goal_y].d = 0
   queue.append(array[goal_x][goal_y])# starting from goal state
   while queue:
       temp = queue.pop(0)
       adjacent = MoveGen(temp)    # getting neighbours
       # removing already visited node
       adjacent = [node for node in adjacent if node.d == -1]
       for node in adjacent:
           node.d = temp.d + 1    # updating distance
           queue.append(node)
```

3. Best First search analysis and observation:

The Best-first search uses two sets. One is open dataset (those generated but not yet selected) another one is closed dataset (already selected). Best-first search progresses by choosing a node with the best heuristic value in the Open list for expansion. The node is removed from the Open list and each of its children's are generated. For each child node that is generated, best-first search checks the Open and Closed lists to ensure that it is not a duplicate of a previously generated node or, if it is, that it represents a better path than the previously generated duplicate. If that is the case, then the node is inserted into the closed list and a new node is chosen for expansion. This continues until a goal node is chosen for expansion.

Best first search is complete, atleast for finite domains. It explore all nodes before reporting failure. The search frontier for Best first search depends upon the accuracy of the heuristic functions. Space complexity and Time complexity both depend upon accuracy of heuristic function. With a best heuristic function , the search will find the goal in linear time. Time complexity of Best first search is much less than Breadth first search.

It is more efficient than that of BFS and DFS. The Best first search allows us to switch between paths by gaining the benefits of both breadth first and depth first search. Because, depth first is good because a solution can be found without computing all nodes and Breadth first search is good because it does not get trapped in dead ends.

4. Hill Climbing and Best First search comparison:

Hill climbing Expand the node which is nearest to goal based on heuristic value. It keeps no history, the algorithm cannot recover from failures of its policy. A major problem of hill climbing is their tendency to become stuck at local data. It is neither optimal nor complete. It can be very fast if we use good heuristic function. Only one element is taken into consideration at each instance of time. It take less space than best first search.

Best first search is complete and It explore all nodes before reporting failure. But it is also not optimal. It can recover if stuck at local minima/maxima. Best First search is one such technique in which the distance from the search node is recorded to find out or assume the final estimated distance.

Heuristic type:
1 - Euclidean Distance
2 - Manhattan Distance

Red→ Goal not found

	input.txt				input2.txt				input3.txt			
	Nodes explored		Steps		Nodes explored		Steps		Nodes explored		Steps	
Heuristic function	1	2	1	2	1	2	1	2	1	2	1	2
BFS	6	6	28	25	9	9	31	30	14	14	235	134
HC	3 (X)	6	7 (X)	25	7	7	31	30	7 (X)	9	111 (X)	134

5. Beam search analysis for different beam lengths:

Beam search is an approximate search algorithm so it doesn't always output the most likely sentence. We can improve the performance of beam search by increasing beam width. Beam search is like extension of Hill climbing. Beam width is the number of maximum node that we can choose for Open list at every stage/level. Beam search is better than Best first search in term of space and Time complexity. But it is not optimal nor complete.

We observe that with increase of beam width, chances to reach goal state also increases. But No. of explored node and time taken to reach goal state increases as well. If we reached Goal state for beam width '**β**' than for all beam width greater than '**β**' we have solution. This will just take extra time to explore others state.

The below table show the number of explored state and time taken in term of path-steps to reach goal state.

Heuristic type:
1 - Euclidean Distance
2 - Manhattan Distance

Red→ Goal not found

		input.txt				input2.txt				input3.txt			
		Nodes explored		Steps		Nodes explored		Steps		Nodes explored		Steps	
	Heuristic type	1	2	1	2	1	2	1	2	1	2	1	2
Beam Search	Beam = 1	3 (X)	6	7 (X)	25	7	7	31	30	7 (X)	9	111 (X)	134
	2	6	6	28	28	8	8	46	46	11 (X)	15	198 (X)	243
	3	6	6	28	28	8	8	57	57	11 (X)	15	235 (X)	275
	4	6	6	28	28	8	8	57	57	11 (X)	15	272 (X)	307

- For input3 we reached goal state for beam width 7.

6. Tabu search for different values of tabu tenure:

The main idea in Tabu search is to augment the exploitative strategy of heuristic search with an explorative tendency that looks for new areas in search space. The Tabu search follows the diktat of the heuristic function as long as better choices are present.

		input.txt				input2.txt				input3.txt			
		Nodes explored		Steps		Nodes explored		Steps		Nodes explored		Steps	
	Heuristic type	1	2	1	2	1	2	1	2	1	2	1	2
Tabu Search	tt = 1	4	3	28	25	4	4	31	30	inf	5	inf	134
	2	4	3	28	25	4	4	31	30	inf	5	inf	134
	3	4	3	28	25	4	4	31	30	inf	5	inf	134
	4	4	3	28	25	4	4	31	30	8	5	235	134

7. Comparison of Variable neighborhood descent, Beam Search,Tabu Search:

Heuristic type:
1 - Euclidean Distance
2 - Manhattan Distance

	input.txt				input2.txt				input3.txt			
	Nodes explored		Steps		Nodes explored		Steps		Nodes explored		Steps	
Heuristic type	1	2	1	2	1	2	1	2	1	2	1	2
BFS	6	6	28	25	9	9	31	30	14	14	235	134
HC	3	6	7 (x)	25	7	7	31	30	7 (x)	9	111	134
VND	4	3	28	25	4	4	31	30	8	5	235	134

Red→ Goal not found

1 - Euclidean Distance
2 - Manhattan Distance

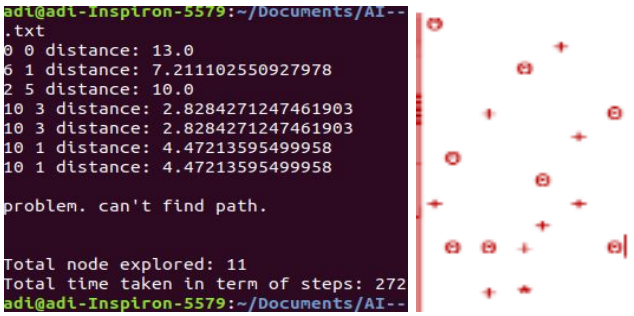
		input.txt				input2.txt				input3.txt			
		Nodes explored		Steps		Nodes explored		Steps		Nodes explored		Steps	
	Heuristic type	1	2	1	2	1	2	1	2	1	2	1	2
Beam Search	Beam = 1	3 (X)	6	7	25	7	7	31	30	7 (X)	9	111	134
	2	6	6	28	28	8	8	46	46	11 (X)	15	198	243
	3	6	6	28	28	8	8	57	57	11 (X)	15	235	275
	4	6	6	28	28	8	8	57	57	11 (X)	15	272	307

- For input3 we reached goal state for beam width 7.

		input.txt				input2.txt				input3.txt			
		Nodes explored		Steps		Nodes explored		Steps		Nodes explored		Steps	
	Heuristic type	1	2	1	2	1	2	1	2	1	2	1	2
Tabu Search	tt = 1	4	3	28	25	4	4	31	30	inf	5	inf	134
	2	4	3	28	25	4	4	31	30	inf	5	inf	134
	3	4	3	28	25	4	4	31	30	inf	5	inf	134
	4	4	3	28	25	4	4	31	30	8	5	235	134

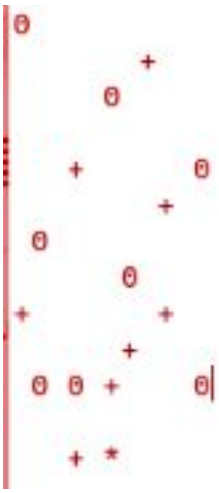
VND: is able to find the path for all 3 inputs.

Beam Search: fails of ß = 1 in input1 but succeeds for ß>1.
In input3 it fails for Heuristic 1[Euclidean Distance] for all test cases even at ß =4 because the neighbours reachable because of [10 10]being covered by enemy players as shown below , So the control keeps revisiting the easier back nodes and fails.



Tabu Search: is able to find solution for input1 & input2 because at even tt = 1 because Hill Climb was also able to find a solution for it. Now, for input3 it doesn't give a Solution till tt=4 [tabu tenure] because the short tabu list is unable to stop it from re-visiting and ending up in a loop.

```
10 3 distance: 2.8284271247461903
10 1 distance: 4.47213595499958
7 6 distance: 5.0990195135927845
10 3 distance: 2.8284271247461903
10 1 distance: 4.47213595499958
7 6 distance: 5.0990195135927845
10 3 distance: 2.8284271247461903
10 1 distance: 4.47213595499958
7 6 distance: 5.0990195135927845
^CTraceback (most recent call last):
  File "tabu.py", line 175, in <module>
    opens.sort(key=lambda x: x.d)  # sorting opens based on distance
KeyboardInterrupt
adi@adi-Inspiron-5579:~/Documents/AI--Lab/AI_Lab/Assignment2$
```



It loops [10 3] -> [10 1] -> [7 6]->[10 3] … because until the tt <= 4 at the cell [7 6] it always finds [10 3] nearer than [10 10] because [10 3] is not included in the tabu list. At tt = 4 [10 3] is taboo and it has to go to [10 10] eventually reaching the goal at [12 5].

In conclusion VND is finds a solution in all cases followed by Tabu which fails for tt = {1,2,3} in input3 but passes at tt = 4. Whereas beam search a non-complete algo like tabu search but fails in <input,ß =1> and <input3,ß={1,2,3,4}>.

Variable Neighbourhood Descent :

In VND the Hill Climb method is called With incremental neighbourhood range until a non-empty set containing the best neighbour better than the current is returned.

```
def VND(node):
    adj = []
    ind = 0
    while(len(adj) < 1 or MoveGen(node) == None ):
        adj = (best(MoveGen(node),node))
        # print("MM ",len(adj))

        temp = MoveGen(node)
        temp.sort(key=lambda x: x.d)
        node = temp[0]
        ind = ind + 1
        # print("||||||||||||",len(adj)," ",adj[0].
    return adj
```

Tabu Search :Here a circular queue with a given size [tabu tenure (tt)]
Is used to store the list of nodes regarded as taboo. We have implemented
2 conditions for taboo list inclusion:
>distance-to-goal more than the best-till-time
>node explored
Aspiration Criterion: If the set of neighbours turns out empty after comparing it with tabu list then the best of the neighbours is taken to explore.

loop:
temp = open.pop()
open <= open U {best(allowed(movegen(temp)))}

CONCLUSION:

1.Our heuristic function 2 is better than 1. It gives solution for every type of inputs.
2.Heuristic function based search algorithms are better than Breadth first search and Depth first search. If we have good heuristic function we can get solution in linear space and linear time.