**Artificial and Computational Intelligence**

**Assignment 1**

**Names of team members-**

| **Sl Number** | **Student Name** | **BITS ID** | **Contribution Percentage** |
| --- | --- | --- | --- |
| 1 | Chandupatla Anirudh Reddy | 2022DA04387 | 100% |
| 2 | Aditya Ruhela | 2022DA04599 | 100% |
| 3 | Manikandan S | 2022DA04396 | 100%S |
| 4 | Sheik Rizwan | NA | 0% |

**PROBLEM STATEMENT**

The problem statement informs that the rabbit is trapped at the cave and it has to reach the ‘goal’ state in order to exit from the cave. The cave is depicted in the form of a maze. The solution has to be achieved through a Robot. The following 2 algorithms are to be used to achieve the goal-

1. A\* Algorithm and
2. Random Restart Hill Climbing Algorithm

**Explanation of the PEAS:**

The PEAS (Performance measure, Environment, Actuator and Sensor) for the case are as follows-

* **Performance Measure**- The performance efficiency and effectiveness of the robot is measured on the basis of the ability to find the optimal path from the start position to the finish position, termed as ‘Goal’, while avoiding and minimizing the encounter with obstacles such as the Fire, bush and walls. The robot should be able to complete its task in a timely and efficient manner.
* **Environment-** The environment of the rescue robot is the cave represented as a maze. The cave contains obstacles such as walls, bush and the fire. The task of the robot is to move through the maze from the starting point till the end point, that is the goal, and rescue the rabbit from the cave. While trying to navigate through the cave, the robot must find the optimal path, avoiding any penalties awarded for crossing the bush and the fire and at least a number of steps.
* **Actuators**- The actuators of the robot that rescues the rabbit are its movement capabilities. The robot can move in four directions, North, South, East and West. It uses these movements to navigate through the cave and find the optimal path to the finish position.
* **Sensors-** The rescue robot has sensors that allows it to perceive the environment around it and helps it more in legal directions. The robot can use its sensors to detect obstacles such as the walls and penalty imposing cells of bush and fire. The sensor also allows the robot to determine its present position and helps it in moving towards the goal position.

**Code and Its Output:**

The code has been executed in Python and the solution file is uploaded for the reference.

#Code Block : Set Initial State (Must handle dynamic inputs)

start = (0, 2)

#Code Block : Set the matrix for transition & cost (as relevant for the given problem)

maze = [[25, 12, 5, 8, 12, 37],

[3, 1, 11, 12, 21, 3],

[3, 10, 7, 40, 14, 7],

[11, 36, 3, 9, 12, 6],

[3, 25, 14, 14, 13, 21],

[2, 11, 12, 5, 3, 3],

[8, 6, 8, 14, 7, 2]]

#Code Block : Write function to design the Transition Model/Successor function. Ideally this would be called while search algorithms are implemented

def can\_walk\_from\_current(cell, to\_new\_position):

return ((to\_new\_position == UP and cell & 2 == 2) or

(to\_new\_position == DOWN and cell & 1 == 1) or

(to\_new\_position == LEFT and cell & 4 == 4) or

(to\_new\_position == RIGHT and cell & 8 == 8))

def does\_not\_cross\_maze\_border\_from(maze,node\_position):

return (0 <= node\_position[0] < (len(maze)) and

0 <= node\_position[1] < (len(maze[0])))

def manhattan\_distance(p1, p2):

return (abs(p1[0] - p2[0]) +

abs(p1[1] - p2[1]))

bush\_in = lambda cell: cell & 16 == 16

fire\_at = lambda cell: cell & 32 == 32

LEFT, RIGHT, UP, DOWN = (0, -1), (0, 1), (-1, 0), (1, 0)

#Code block : Write function to handle goal test (Must handle dynamic inputs). Ideally this would be called while search algorithms are implemented

def is\_goal\_state(state, goal= (6,4)):

return state == goal

**A Star:**

Theory/pseudocode explanation:

function A\_STAR(problem) returns a path if exists

initialize start and goal nodes

initialize open\_list/to\_be\_searched list and closed\_list/searched\_list

add start node to open\_list

loop until open\_list is not empty:

pick the node from open\_list with lowest f(n) value as current

remove current from open\_list

add current to closed\_list , indicating we’re searching current

if current node is goal return path from start

else keep searching neighbors/childrens of current node

check if neighbors are reachable or not and add them to neighbors/children's list along with their parent info

loop over neighbors/children’s list

if child is not searched before

calculate f = g + h + penalty for fire & bush

if this child’s g value is least among existing open\_list node then add this child to open\_list

return failure if goal not reached even after searching all possible neighbors along the path

#Code Block : Function for algorithm 1 implementation

class Node():

def \_\_init\_\_(self, parent=None, position=None):

self.parent = parent

self.position = position

self.g = 0

self.h = 0

self.f = 0

def \_\_eq\_\_(self, other):

return self.position == other.position

def A\_STAR(maze, start, end):

start\_node = Node(None, start)

start\_node.g = start\_node.h = start\_node.f = 0

end\_node = Node(None, end)

end\_node.g = end\_node.h = end\_node.f = 0

open\_list = []

closed\_list = []

open\_list.append(start\_node)

while len(open\_list) > 0:

current\_node = open\_list[0]

current\_index = 0

for index, item in enumerate(open\_list):

if item.f < current\_node.f:

current\_node = item

current\_index = index

open\_list.pop(current\_index)

closed\_list.append(current\_node)

if current\_node == end\_node:

path = []

current = current\_node

total\_cost = current.g

while current is not None:

path.append(current.position)

current = current.parent

return {'path\_taken':'->'.join([str(e) for e in path[::-1]]), 'total\_cost':total\_cost}

children = []

for new\_position in [LEFT, RIGHT, UP, DOWN]:

node\_position = (current\_node.position[0] + new\_position[0], current\_node.position[1] + new\_position[1])

if does\_not\_cross\_maze\_border\_from(maze,node\_position):

cell = maze[current\_node.position[0]][current\_node.position[1]]

if(can\_walk\_from\_current(cell, new\_position)):

children.append(Node(current\_node, node\_position))

for child in children:

if child not in closed\_list:

child\_cell = maze[child.position[0]][child.position[1]]

step\_cost = 3

if(fire\_at(child\_cell)): step\_cost += 5

if(bush\_in(child\_cell)): step\_cost += 1

child.g = current\_node.g + step\_cost

child.h = manhattan\_distance(child.position, end\_node.position)

child.f = child.g + child.h

for open\_node in open\_list:

if child == open\_node and child.g > open\_node.g:

continue

open\_list.append(child)

return 'not able to find path'

**Entire code execution with clear output flow:**

visited this node: (0, 2) as it has least cost 0 and added it to closed list:[ [{'cell': (0, 2), 'cost': 0}] ]

checking neighbor: (0, 1)

neighbor (0, 1) is reachable so adding to children list

checking neighbor: (0, 3)

checking neighbor: (-1, 2)

checking neighbor: (1, 2)

neighbor (1, 2) is reachable so adding to children list

child (0, 1) is not visited before so checking it

child has f(n): 12 g(n): 3 h(n): 9

since this child (0, 1) has best g value out of existing open list nodes adding it to open\_list/to\_be\_searched list [{'cell': (0, 1), 'cost': 12}]

child (1, 2) is not visited before so checking it

child has f(n): 10 g(n): 3 h(n): 7

since this child (1, 2) has best g value out of existing open list nodes adding it to open\_list/to\_be\_searched list [{'cell': (0, 1), 'cost': 12}, {'cell': (1, 2), 'cost': 10}]

----------

visited this node: (1, 2) as it has least cost 10 and added it to closed list:[ [{'cell': (0, 2), 'cost': 0}, {'cell': (1, 2), 'cost': 10}] ]

checking neighbor: (1, 1)

checking neighbor: (1, 3)

neighbor (1, 3) is reachable so adding to children list

checking neighbor: (0, 2)

neighbor (0, 2) is reachable so adding to children list

checking neighbor: (2, 2)

neighbor (2, 2) is reachable so adding to children list

child (1, 3) is not visited before so checking it

child has f(n): 12 g(n): 6 h(n): 6

since this child (1, 3) has best g value out of existing open list nodes adding it to open\_list/to\_be\_searched list [{'cell': (0, 1), 'cost': 12}, {'cell': (1, 3), 'cost': 12}]

child (2, 2) is not visited before so checking it

child has f(n): 12 g(n): 6 h(n): 6

since this child (2, 2) has best g value out of existing open list nodes adding it to open\_list/to\_be\_searched list [{'cell': (0, 1), 'cost': 12}, {'cell': (1, 3), 'cost': 12}, {'cell': (2, 2), 'cost': 12}]

----------

visited this node: (0, 1) as it has least cost 12 and added it to closed list:[ [{'cell': (0, 2), 'cost': 0}, {'cell': (1, 2), 'cost': 10}, {'cell': (0, 1), 'cost': 12}] ]

checking neighbor: (0, 0)

neighbor (0, 0) is reachable so adding to children list

checking neighbor: (0, 2)

neighbor (0, 2) is reachable so adding to children list

checking neighbor: (-1, 1)

checking neighbor: (1, 1)

child (0, 0) is not visited before so checking it

child has f(n): 17 g(n): 7 h(n): 10

since this child (0, 0) has best g value out of existing open list nodes adding it to open\_list/to\_be\_searched list [{'cell': (1, 3), 'cost': 12}, {'cell': (2, 2), 'cost': 12}, {'cell': (0, 0), 'cost': 17}]

----------

visited this node: (1, 3) as it has least cost 12 and added it to closed list:[ [{'cell': (0, 2), 'cost': 0}, {'cell': (1, 2), 'cost': 10}, {'cell': (0, 1), 'cost': 12}, {'cell': (1, 3), 'cost': 12}] ]

checking neighbor: (1, 2)

neighbor (1, 2) is reachable so adding to children list

checking neighbor: (1, 4)

neighbor (1, 4) is reachable so adding to children list

checking neighbor: (0, 3)

checking neighbor: (2, 3)

child (1, 4) is not visited before so checking it

child has f(n): 15 g(n): 10 h(n): 5

since this child (1, 4) has best g value out of existing open list nodes adding it to open\_list/to\_be\_searched list [{'cell': (2, 2), 'cost': 12}, {'cell': (0, 0), 'cost': 17}, {'cell': (1, 4), 'cost': 15}]

----------

visited this node: (2, 2) as it has least cost 12 and added it to closed list:[ [{'cell': (0, 2), 'cost': 0}, {'cell': (1, 2), 'cost': 10}, {'cell': (0, 1), 'cost': 12}, {'cell': (1, 3), 'cost': 12}, {'cell': (2, 2), 'cost': 12}] ]

checking neighbor: (2, 1)

neighbor (2, 1) is reachable so adding to children list

checking neighbor: (2, 3)

checking neighbor: (1, 2)

neighbor (1, 2) is reachable so adding to children list

checking neighbor: (3, 2)

neighbor (3, 2) is reachable so adding to children list

child (2, 1) is not visited before so checking it

child has f(n): 16 g(n): 9 h(n): 7

since this child (2, 1) has best g value out of existing open list nodes adding it to open\_list/to\_be\_searched list [{'cell': (0, 0), 'cost': 17}, {'cell': (1, 4), 'cost': 15}, {'cell': (2, 1), 'cost': 16}]

child (3, 2) is not visited before so checking it

child has f(n): 14 g(n): 9 h(n): 5

since this child (3, 2) has best g value out of existing open list nodes adding it to open\_list/to\_be\_searched list [{'cell': (0, 0), 'cost': 17}, {'cell': (1, 4), 'cost': 15}, {'cell': (2, 1), 'cost': 16}, {'cell': (3, 2), 'cost': 14}]

----------

visited this node: (3, 2) as it has least cost 14 and added it to closed list:[ [{'cell': (0, 2), 'cost': 0}, {'cell': (1, 2), 'cost': 10}, {'cell': (0, 1), 'cost': 12}, {'cell': (1, 3), 'cost': 12}, {'cell': (2, 2), 'cost': 12}, {'cell': (3, 2), 'cost': 14}] ]

checking neighbor: (3, 1)

checking neighbor: (3, 3)

checking neighbor: (2, 2)

neighbor (2, 2) is reachable so adding to children list

checking neighbor: (4, 2)

neighbor (4, 2) is reachable so adding to children list

child (4, 2) is not visited before so checking it

child has f(n): 16 g(n): 12 h(n): 4

since this child (4, 2) has best g value out of existing open list nodes adding it to open\_list/to\_be\_searched list [{'cell': (0, 0), 'cost': 17}, {'cell': (1, 4), 'cost': 15}, {'cell': (2, 1), 'cost': 16}, {'cell': (4, 2), 'cost': 16}]

----------

visited this node: (1, 4) as it has least cost 15 and added it to closed list:[ [{'cell': (0, 2), 'cost': 0}, {'cell': (1, 2), 'cost': 10}, {'cell': (0, 1), 'cost': 12}, {'cell': (1, 3), 'cost': 12}, {'cell': (2, 2), 'cost': 12}, {'cell': (3, 2), 'cost': 14}, {'cell': (1, 4), 'cost': 15}] ]

checking neighbor: (1, 3)

neighbor (1, 3) is reachable so adding to children list

checking neighbor: (1, 5)

checking neighbor: (0, 4)

checking neighbor: (2, 4)

neighbor (2, 4) is reachable so adding to children list

child (2, 4) is not visited before so checking it

child has f(n): 17 g(n): 13 h(n): 4

since this child (2, 4) has best g value out of existing open list nodes adding it to open\_list/to\_be\_searched list [{'cell': (0, 0), 'cost': 17}, {'cell': (2, 1), 'cost': 16}, {'cell': (4, 2), 'cost': 16}, {'cell': (2, 4), 'cost': 17}]

----------

visited this node: (2, 1) as it has least cost 16 and added it to closed list:[ [{'cell': (0, 2), 'cost': 0}, {'cell': (1, 2), 'cost': 10}, {'cell': (0, 1), 'cost': 12}, {'cell': (1, 3), 'cost': 12}, {'cell': (2, 2), 'cost': 12}, {'cell': (3, 2), 'cost': 14}, {'cell': (1, 4), 'cost': 15}, {'cell': (2, 1), 'cost': 16}] ]

checking neighbor: (2, 0)

checking neighbor: (2, 2)

neighbor (2, 2) is reachable so adding to children list

checking neighbor: (1, 1)

neighbor (1, 1) is reachable so adding to children list

checking neighbor: (3, 1)

child (1, 1) is not visited before so checking it

child has f(n): 20 g(n): 12 h(n): 8

since this child (1, 1) has best g value out of existing open list nodes adding it to open\_list/to\_be\_searched list [{'cell': (0, 0), 'cost': 17}, {'cell': (4, 2), 'cost': 16}, {'cell': (2, 4), 'cost': 17}, {'cell': (1, 1), 'cost': 20}]

----------

visited this node: (4, 2) as it has least cost 16 and added it to closed list:[ [{'cell': (0, 2), 'cost': 0}, {'cell': (1, 2), 'cost': 10}, {'cell': (0, 1), 'cost': 12}, {'cell': (1, 3), 'cost': 12}, {'cell': (2, 2), 'cost': 12}, {'cell': (3, 2), 'cost': 14}, {'cell': (1, 4), 'cost': 15}, {'cell': (2, 1), 'cost': 16}, {'cell': (4, 2), 'cost': 16}] ]

checking neighbor: (4, 1)

neighbor (4, 1) is reachable so adding to children list

checking neighbor: (4, 3)

neighbor (4, 3) is reachable so adding to children list

checking neighbor: (3, 2)

neighbor (3, 2) is reachable so adding to children list

checking neighbor: (5, 2)

child (4, 1) is not visited before so checking it

child has f(n): 21 g(n): 16 h(n): 5

since this child (4, 1) has best g value out of existing open list nodes adding it to open\_list/to\_be\_searched list [{'cell': (0, 0), 'cost': 17}, {'cell': (2, 4), 'cost': 17}, {'cell': (1, 1), 'cost': 20}, {'cell': (4, 1), 'cost': 21}]

child (4, 3) is not visited before so checking it

child has f(n): 18 g(n): 15 h(n): 3

since this child (4, 3) has best g value out of existing open list nodes adding it to open\_list/to\_be\_searched list [{'cell': (0, 0), 'cost': 17}, {'cell': (2, 4), 'cost': 17}, {'cell': (1, 1), 'cost': 20}, {'cell': (4, 1), 'cost': 21}, {'cell': (4, 3), 'cost': 18}]

----------

visited this node: (0, 0) as it has least cost 17 and added it to closed list:[ [{'cell': (0, 2), 'cost': 0}, {'cell': (1, 2), 'cost': 10}, {'cell': (0, 1), 'cost': 12}, {'cell': (1, 3), 'cost': 12}, {'cell': (2, 2), 'cost': 12}, {'cell': (3, 2), 'cost': 14}, {'cell': (1, 4), 'cost': 15}, {'cell': (2, 1), 'cost': 16}, {'cell': (4, 2), 'cost': 16}, {'cell': (0, 0), 'cost': 17}] ]

checking neighbor: (0, -1)

checking neighbor: (0, 1)

neighbor (0, 1) is reachable so adding to children list

checking neighbor: (-1, 0)

checking neighbor: (1, 0)

neighbor (1, 0) is reachable so adding to children list

child (1, 0) is not visited before so checking it

child has f(n): 19 g(n): 10 h(n): 9

since this child (1, 0) has best g value out of existing open list nodes adding it to open\_list/to\_be\_searched list [{'cell': (2, 4), 'cost': 17}, {'cell': (1, 1), 'cost': 20}, {'cell': (4, 1), 'cost': 21}, {'cell': (4, 3), 'cost': 18}, {'cell': (1, 0), 'cost': 19}]

----------

visited this node: (2, 4) as it has least cost 17 and added it to closed list:[ [{'cell': (0, 2), 'cost': 0}, {'cell': (1, 2), 'cost': 10}, {'cell': (0, 1), 'cost': 12}, {'cell': (1, 3), 'cost': 12}, {'cell': (2, 2), 'cost': 12}, {'cell': (3, 2), 'cost': 14}, {'cell': (1, 4), 'cost': 15}, {'cell': (2, 1), 'cost': 16}, {'cell': (4, 2), 'cost': 16}, {'cell': (0, 0), 'cost': 17}, {'cell': (2, 4), 'cost': 17}] ]

checking neighbor: (2, 3)

neighbor (2, 3) is reachable so adding to children list

checking neighbor: (2, 5)

neighbor (2, 5) is reachable so adding to children list

checking neighbor: (1, 4)

neighbor (1, 4) is reachable so adding to children list

checking neighbor: (3, 4)

child (2, 3) is not visited before so checking it

child has f(n): 26 g(n): 21 h(n): 5

since this child (2, 3) has best g value out of existing open list nodes adding it to open\_list/to\_be\_searched list [{'cell': (1, 1), 'cost': 20}, {'cell': (4, 1), 'cost': 21}, {'cell': (4, 3), 'cost': 18}, {'cell': (1, 0), 'cost': 19}, {'cell': (2, 3), 'cost': 26}]

child (2, 5) is not visited before so checking it

child has f(n): 21 g(n): 16 h(n): 5

since this child (2, 5) has best g value out of existing open list nodes adding it to open\_list/to\_be\_searched list [{'cell': (1, 1), 'cost': 20}, {'cell': (4, 1), 'cost': 21}, {'cell': (4, 3), 'cost': 18}, {'cell': (1, 0), 'cost': 19}, {'cell': (2, 3), 'cost': 26}, {'cell': (2, 5), 'cost': 21}]

----------

visited this node: (4, 3) as it has least cost 18 and added it to closed list:[ [{'cell': (0, 2), 'cost': 0}, {'cell': (1, 2), 'cost': 10}, {'cell': (0, 1), 'cost': 12}, {'cell': (1, 3), 'cost': 12}, {'cell': (2, 2), 'cost': 12}, {'cell': (3, 2), 'cost': 14}, {'cell': (1, 4), 'cost': 15}, {'cell': (2, 1), 'cost': 16}, {'cell': (4, 2), 'cost': 16}, {'cell': (0, 0), 'cost': 17}, {'cell': (2, 4), 'cost': 17}, {'cell': (4, 3), 'cost': 18}] ]

checking neighbor: (4, 2)

neighbor (4, 2) is reachable so adding to children list

checking neighbor: (4, 4)

neighbor (4, 4) is reachable so adding to children list

checking neighbor: (3, 3)

neighbor (3, 3) is reachable so adding to children list

checking neighbor: (5, 3)

child (4, 4) is not visited before so checking it

child has f(n): 20 g(n): 18 h(n): 2

since this child (4, 4) has best g value out of existing open list nodes adding it to open\_list/to\_be\_searched list [{'cell': (1, 1), 'cost': 20}, {'cell': (4, 1), 'cost': 21}, {'cell': (1, 0), 'cost': 19}, {'cell': (2, 3), 'cost': 26}, {'cell': (2, 5), 'cost': 21}, {'cell': (4, 4), 'cost': 20}]

child (3, 3) is not visited before so checking it

child has f(n): 22 g(n): 18 h(n): 4

since this child (3, 3) has best g value out of existing open list nodes adding it to open\_list/to\_be\_searched list [{'cell': (1, 1), 'cost': 20}, {'cell': (4, 1), 'cost': 21}, {'cell': (1, 0), 'cost': 19}, {'cell': (2, 3), 'cost': 26}, {'cell': (2, 5), 'cost': 21}, {'cell': (4, 4), 'cost': 20}, {'cell': (3, 3), 'cost': 22}]

----------

visited this node: (1, 0) as it has least cost 19 and added it to closed list:[ [{'cell': (0, 2), 'cost': 0}, {'cell': (1, 2), 'cost': 10}, {'cell': (0, 1), 'cost': 12}, {'cell': (1, 3), 'cost': 12}, {'cell': (2, 2), 'cost': 12}, {'cell': (3, 2), 'cost': 14}, {'cell': (1, 4), 'cost': 15}, {'cell': (2, 1), 'cost': 16}, {'cell': (4, 2), 'cost': 16}, {'cell': (0, 0), 'cost': 17}, {'cell': (2, 4), 'cost': 17}, {'cell': (4, 3), 'cost': 18}, {'cell': (1, 0), 'cost': 19}] ]

checking neighbor: (1, -1)

checking neighbor: (1, 1)

checking neighbor: (0, 0)

neighbor (0, 0) is reachable so adding to children list

checking neighbor: (2, 0)

neighbor (2, 0) is reachable so adding to children list

child (2, 0) is not visited before so checking it

child has f(n): 21 g(n): 13 h(n): 8

since this child (2, 0) has best g value out of existing open list nodes adding it to open\_list/to\_be\_searched list [{'cell': (1, 1), 'cost': 20}, {'cell': (4, 1), 'cost': 21}, {'cell': (2, 3), 'cost': 26}, {'cell': (2, 5), 'cost': 21}, {'cell': (4, 4), 'cost': 20}, {'cell': (3, 3), 'cost': 22}, {'cell': (2, 0), 'cost': 21}]

----------

visited this node: (1, 1) as it has least cost 20 and added it to closed list:[ [{'cell': (0, 2), 'cost': 0}, {'cell': (1, 2), 'cost': 10}, {'cell': (0, 1), 'cost': 12}, {'cell': (1, 3), 'cost': 12}, {'cell': (2, 2), 'cost': 12}, {'cell': (3, 2), 'cost': 14}, {'cell': (1, 4), 'cost': 15}, {'cell': (2, 1), 'cost': 16}, {'cell': (4, 2), 'cost': 16}, {'cell': (0, 0), 'cost': 17}, {'cell': (2, 4), 'cost': 17}, {'cell': (4, 3), 'cost': 18}, {'cell': (1, 0), 'cost': 19}, {'cell': (1, 1), 'cost': 20}] ]

checking neighbor: (1, 0)

checking neighbor: (1, 2)

checking neighbor: (0, 1)

checking neighbor: (2, 1)

neighbor (2, 1) is reachable so adding to children list

----------

visited this node: (4, 4) as it has least cost 20 and added it to closed list:[ [{'cell': (0, 2), 'cost': 0}, {'cell': (1, 2), 'cost': 10}, {'cell': (0, 1), 'cost': 12}, {'cell': (1, 3), 'cost': 12}, {'cell': (2, 2), 'cost': 12}, {'cell': (3, 2), 'cost': 14}, {'cell': (1, 4), 'cost': 15}, {'cell': (2, 1), 'cost': 16}, {'cell': (4, 2), 'cost': 16}, {'cell': (0, 0), 'cost': 17}, {'cell': (2, 4), 'cost': 17}, {'cell': (4, 3), 'cost': 18}, {'cell': (1, 0), 'cost': 19}, {'cell': (1, 1), 'cost': 20}, {'cell': (4, 4), 'cost': 20}] ]

checking neighbor: (4, 3)

neighbor (4, 3) is reachable so adding to children list

checking neighbor: (4, 5)

neighbor (4, 5) is reachable so adding to children list

checking neighbor: (3, 4)

checking neighbor: (5, 4)

neighbor (5, 4) is reachable so adding to children list

child (4, 5) is not visited before so checking it

child has f(n): 25 g(n): 22 h(n): 3

since this child (4, 5) has best g value out of existing open list nodes adding it to open\_list/to\_be\_searched list [{'cell': (4, 1), 'cost': 21}, {'cell': (2, 3), 'cost': 26}, {'cell': (2, 5), 'cost': 21}, {'cell': (3, 3), 'cost': 22}, {'cell': (2, 0), 'cost': 21}, {'cell': (4, 5), 'cost': 25}]

child (5, 4) is not visited before so checking it

child has f(n): 22 g(n): 21 h(n): 1

since this child (5, 4) has best g value out of existing open list nodes adding it to open\_list/to\_be\_searched list [{'cell': (4, 1), 'cost': 21}, {'cell': (2, 3), 'cost': 26}, {'cell': (2, 5), 'cost': 21}, {'cell': (3, 3), 'cost': 22}, {'cell': (2, 0), 'cost': 21}, {'cell': (4, 5), 'cost': 25}, {'cell': (5, 4), 'cost': 22}]

----------

visited this node: (4, 1) as it has least cost 21 and added it to closed list:[ [{'cell': (0, 2), 'cost': 0}, {'cell': (1, 2), 'cost': 10}, {'cell': (0, 1), 'cost': 12}, {'cell': (1, 3), 'cost': 12}, {'cell': (2, 2), 'cost': 12}, {'cell': (3, 2), 'cost': 14}, {'cell': (1, 4), 'cost': 15}, {'cell': (2, 1), 'cost': 16}, {'cell': (4, 2), 'cost': 16}, {'cell': (0, 0), 'cost': 17}, {'cell': (2, 4), 'cost': 17}, {'cell': (4, 3), 'cost': 18}, {'cell': (1, 0), 'cost': 19}, {'cell': (1, 1), 'cost': 20}, {'cell': (4, 4), 'cost': 20}, {'cell': (4, 1), 'cost': 21}] ]

checking neighbor: (4, 0)

checking neighbor: (4, 2)

neighbor (4, 2) is reachable so adding to children list

checking neighbor: (3, 1)

checking neighbor: (5, 1)

neighbor (5, 1) is reachable so adding to children list

child (5, 1) is not visited before so checking it

child has f(n): 23 g(n): 19 h(n): 4

since this child (5, 1) has best g value out of existing open list nodes adding it to open\_list/to\_be\_searched list [{'cell': (2, 3), 'cost': 26}, {'cell': (2, 5), 'cost': 21}, {'cell': (3, 3), 'cost': 22}, {'cell': (2, 0), 'cost': 21}, {'cell': (4, 5), 'cost': 25}, {'cell': (5, 4), 'cost': 22}, {'cell': (5, 1), 'cost': 23}]

----------

visited this node: (2, 5) as it has least cost 21 and added it to closed list:[ [{'cell': (0, 2), 'cost': 0}, {'cell': (1, 2), 'cost': 10}, {'cell': (0, 1), 'cost': 12}, {'cell': (1, 3), 'cost': 12}, {'cell': (2, 2), 'cost': 12}, {'cell': (3, 2), 'cost': 14}, {'cell': (1, 4), 'cost': 15}, {'cell': (2, 1), 'cost': 16}, {'cell': (4, 2), 'cost': 16}, {'cell': (0, 0), 'cost': 17}, {'cell': (2, 4), 'cost': 17}, {'cell': (4, 3), 'cost': 18}, {'cell': (1, 0), 'cost': 19}, {'cell': (1, 1), 'cost': 20}, {'cell': (4, 4), 'cost': 20}, {'cell': (4, 1), 'cost': 21}, {'cell': (2, 5), 'cost': 21}] ]

checking neighbor: (2, 4)

neighbor (2, 4) is reachable so adding to children list

checking neighbor: (2, 6)

checking neighbor: (1, 5)

neighbor (1, 5) is reachable so adding to children list

checking neighbor: (3, 5)

neighbor (3, 5) is reachable so adding to children list

child (1, 5) is not visited before so checking it

child has f(n): 25 g(n): 19 h(n): 6

since this child (1, 5) has best g value out of existing open list nodes adding it to open\_list/to\_be\_searched list [{'cell': (2, 3), 'cost': 26}, {'cell': (3, 3), 'cost': 22}, {'cell': (2, 0), 'cost': 21}, {'cell': (4, 5), 'cost': 25}, {'cell': (5, 4), 'cost': 22}, {'cell': (5, 1), 'cost': 23}, {'cell': (1, 5), 'cost': 25}]

child (3, 5) is not visited before so checking it

child has f(n): 23 g(n): 19 h(n): 4

since this child (3, 5) has best g value out of existing open list nodes adding it to open\_list/to\_be\_searched list [{'cell': (2, 3), 'cost': 26}, {'cell': (3, 3), 'cost': 22}, {'cell': (2, 0), 'cost': 21}, {'cell': (4, 5), 'cost': 25}, {'cell': (5, 4), 'cost': 22}, {'cell': (5, 1), 'cost': 23}, {'cell': (1, 5), 'cost': 25}, {'cell': (3, 5), 'cost': 23}]

----------

visited this node: (2, 0) as it has least cost 21 and added it to closed list:[ [{'cell': (0, 2), 'cost': 0}, {'cell': (1, 2), 'cost': 10}, {'cell': (0, 1), 'cost': 12}, {'cell': (1, 3), 'cost': 12}, {'cell': (2, 2), 'cost': 12}, {'cell': (3, 2), 'cost': 14}, {'cell': (1, 4), 'cost': 15}, {'cell': (2, 1), 'cost': 16}, {'cell': (4, 2), 'cost': 16}, {'cell': (0, 0), 'cost': 17}, {'cell': (2, 4), 'cost': 17}, {'cell': (4, 3), 'cost': 18}, {'cell': (1, 0), 'cost': 19}, {'cell': (1, 1), 'cost': 20}, {'cell': (4, 4), 'cost': 20}, {'cell': (4, 1), 'cost': 21}, {'cell': (2, 5), 'cost': 21}, {'cell': (2, 0), 'cost': 21}] ]

checking neighbor: (2, -1)

checking neighbor: (2, 1)

checking neighbor: (1, 0)

neighbor (1, 0) is reachable so adding to children list

checking neighbor: (3, 0)

neighbor (3, 0) is reachable so adding to children list

child (3, 0) is not visited before so checking it

child has f(n): 23 g(n): 16 h(n): 7

since this child (3, 0) has best g value out of existing open list nodes adding it to open\_list/to\_be\_searched list [{'cell': (2, 3), 'cost': 26}, {'cell': (3, 3), 'cost': 22}, {'cell': (4, 5), 'cost': 25}, {'cell': (5, 4), 'cost': 22}, {'cell': (5, 1), 'cost': 23}, {'cell': (1, 5), 'cost': 25}, {'cell': (3, 5), 'cost': 23}, {'cell': (3, 0), 'cost': 23}]

----------

visited this node: (3, 3) as it has least cost 22 and added it to closed list:[ [{'cell': (0, 2), 'cost': 0}, {'cell': (1, 2), 'cost': 10}, {'cell': (0, 1), 'cost': 12}, {'cell': (1, 3), 'cost': 12}, {'cell': (2, 2), 'cost': 12}, {'cell': (3, 2), 'cost': 14}, {'cell': (1, 4), 'cost': 15}, {'cell': (2, 1), 'cost': 16}, {'cell': (4, 2), 'cost': 16}, {'cell': (0, 0), 'cost': 17}, {'cell': (2, 4), 'cost': 17}, {'cell': (4, 3), 'cost': 18}, {'cell': (1, 0), 'cost': 19}, {'cell': (1, 1), 'cost': 20}, {'cell': (4, 4), 'cost': 20}, {'cell': (4, 1), 'cost': 21}, {'cell': (2, 5), 'cost': 21}, {'cell': (2, 0), 'cost': 21}, {'cell': (3, 3), 'cost': 22}] ]

checking neighbor: (3, 2)

checking neighbor: (3, 4)

neighbor (3, 4) is reachable so adding to children list

checking neighbor: (2, 3)

checking neighbor: (4, 3)

neighbor (4, 3) is reachable so adding to children list

child (3, 4) is not visited before so checking it

child has f(n): 24 g(n): 21 h(n): 3

since this child (3, 4) has best g value out of existing open list nodes adding it to open\_list/to\_be\_searched list [{'cell': (2, 3), 'cost': 26}, {'cell': (4, 5), 'cost': 25}, {'cell': (5, 4), 'cost': 22}, {'cell': (5, 1), 'cost': 23}, {'cell': (1, 5), 'cost': 25}, {'cell': (3, 5), 'cost': 23}, {'cell': (3, 0), 'cost': 23}, {'cell': (3, 4), 'cost': 24}]

----------

visited this node: (5, 4) as it has least cost 22 and added it to closed list:[ [{'cell': (0, 2), 'cost': 0}, {'cell': (1, 2), 'cost': 10}, {'cell': (0, 1), 'cost': 12}, {'cell': (1, 3), 'cost': 12}, {'cell': (2, 2), 'cost': 12}, {'cell': (3, 2), 'cost': 14}, {'cell': (1, 4), 'cost': 15}, {'cell': (2, 1), 'cost': 16}, {'cell': (4, 2), 'cost': 16}, {'cell': (0, 0), 'cost': 17}, {'cell': (2, 4), 'cost': 17}, {'cell': (4, 3), 'cost': 18}, {'cell': (1, 0), 'cost': 19}, {'cell': (1, 1), 'cost': 20}, {'cell': (4, 4), 'cost': 20}, {'cell': (4, 1), 'cost': 21}, {'cell': (2, 5), 'cost': 21}, {'cell': (2, 0), 'cost': 21}, {'cell': (3, 3), 'cost': 22}, {'cell': (5, 4), 'cost': 22}] ]

checking neighbor: (5, 3)

checking neighbor: (5, 5)

checking neighbor: (4, 4)

neighbor (4, 4) is reachable so adding to children list

checking neighbor: (6, 4)

neighbor (6, 4) is reachable so adding to children list

child (6, 4) is not visited before so checking it

child has f(n): 24 g(n): 24 h(n): 0

since this child (6, 4) has best g value out of existing open list nodes adding it to open\_list/to\_be\_searched list [{'cell': (2, 3), 'cost': 26}, {'cell': (4, 5), 'cost': 25}, {'cell': (5, 1), 'cost': 23}, {'cell': (1, 5), 'cost': 25}, {'cell': (3, 5), 'cost': 23}, {'cell': (3, 0), 'cost': 23}, {'cell': (3, 4), 'cost': 24}, {'cell': (6, 4), 'cost': 24}]

----------

visited this node: (5, 1) as it has least cost 23 and added it to closed list:[ [{'cell': (0, 2), 'cost': 0}, {'cell': (1, 2), 'cost': 10}, {'cell': (0, 1), 'cost': 12}, {'cell': (1, 3), 'cost': 12}, {'cell': (2, 2), 'cost': 12}, {'cell': (3, 2), 'cost': 14}, {'cell': (1, 4), 'cost': 15}, {'cell': (2, 1), 'cost': 16}, {'cell': (4, 2), 'cost': 16}, {'cell': (0, 0), 'cost': 17}, {'cell': (2, 4), 'cost': 17}, {'cell': (4, 3), 'cost': 18}, {'cell': (1, 0), 'cost': 19}, {'cell': (1, 1), 'cost': 20}, {'cell': (4, 4), 'cost': 20}, {'cell': (4, 1), 'cost': 21}, {'cell': (2, 5), 'cost': 21}, {'cell': (2, 0), 'cost': 21}, {'cell': (3, 3), 'cost': 22}, {'cell': (5, 4), 'cost': 22}, {'cell': (5, 1), 'cost': 23}] ]

checking neighbor: (5, 0)

checking neighbor: (5, 2)

neighbor (5, 2) is reachable so adding to children list

checking neighbor: (4, 1)

neighbor (4, 1) is reachable so adding to children list

checking neighbor: (6, 1)

neighbor (6, 1) is reachable so adding to children list

child (5, 2) is not visited before so checking it

child has f(n): 25 g(n): 22 h(n): 3

since this child (5, 2) has best g value out of existing open list nodes adding it to open\_list/to\_be\_searched list [{'cell': (2, 3), 'cost': 26}, {'cell': (4, 5), 'cost': 25}, {'cell': (1, 5), 'cost': 25}, {'cell': (3, 5), 'cost': 23}, {'cell': (3, 0), 'cost': 23}, {'cell': (3, 4), 'cost': 24}, {'cell': (6, 4), 'cost': 24}, {'cell': (5, 2), 'cost': 25}]

child (6, 1) is not visited before so checking it

child has f(n): 25 g(n): 22 h(n): 3

since this child (6, 1) has best g value out of existing open list nodes adding it to open\_list/to\_be\_searched list [{'cell': (2, 3), 'cost': 26}, {'cell': (4, 5), 'cost': 25}, {'cell': (1, 5), 'cost': 25}, {'cell': (3, 5), 'cost': 23}, {'cell': (3, 0), 'cost': 23}, {'cell': (3, 4), 'cost': 24}, {'cell': (6, 4), 'cost': 24}, {'cell': (5, 2), 'cost': 25}, {'cell': (6, 1), 'cost': 25}]

----------

visited this node: (3, 5) as it has least cost 23 and added it to closed list:[ [{'cell': (0, 2), 'cost': 0}, {'cell': (1, 2), 'cost': 10}, {'cell': (0, 1), 'cost': 12}, {'cell': (1, 3), 'cost': 12}, {'cell': (2, 2), 'cost': 12}, {'cell': (3, 2), 'cost': 14}, {'cell': (1, 4), 'cost': 15}, {'cell': (2, 1), 'cost': 16}, {'cell': (4, 2), 'cost': 16}, {'cell': (0, 0), 'cost': 17}, {'cell': (2, 4), 'cost': 17}, {'cell': (4, 3), 'cost': 18}, {'cell': (1, 0), 'cost': 19}, {'cell': (1, 1), 'cost': 20}, {'cell': (4, 4), 'cost': 20}, {'cell': (4, 1), 'cost': 21}, {'cell': (2, 5), 'cost': 21}, {'cell': (2, 0), 'cost': 21}, {'cell': (3, 3), 'cost': 22}, {'cell': (5, 4), 'cost': 22}, {'cell': (5, 1), 'cost': 23}, {'cell': (3, 5), 'cost': 23}] ]

checking neighbor: (3, 4)

neighbor (3, 4) is reachable so adding to children list

checking neighbor: (3, 6)

checking neighbor: (2, 5)

neighbor (2, 5) is reachable so adding to children list

checking neighbor: (4, 5)

child (3, 4) is not visited before so checking it

child has f(n): 25 g(n): 22 h(n): 3

since this child (3, 4) has best g value out of existing open list nodes adding it to open\_list/to\_be\_searched list [{'cell': (2, 3), 'cost': 26}, {'cell': (4, 5), 'cost': 25}, {'cell': (1, 5), 'cost': 25}, {'cell': (3, 0), 'cost': 23}, {'cell': (3, 4), 'cost': 24}, {'cell': (6, 4), 'cost': 24}, {'cell': (5, 2), 'cost': 25}, {'cell': (6, 1), 'cost': 25}, {'cell': (3, 4), 'cost': 25}]

----------

visited this node: (3, 0) as it has least cost 23 and added it to closed list:[ [{'cell': (0, 2), 'cost': 0}, {'cell': (1, 2), 'cost': 10}, {'cell': (0, 1), 'cost': 12}, {'cell': (1, 3), 'cost': 12}, {'cell': (2, 2), 'cost': 12}, {'cell': (3, 2), 'cost': 14}, {'cell': (1, 4), 'cost': 15}, {'cell': (2, 1), 'cost': 16}, {'cell': (4, 2), 'cost': 16}, {'cell': (0, 0), 'cost': 17}, {'cell': (2, 4), 'cost': 17}, {'cell': (4, 3), 'cost': 18}, {'cell': (1, 0), 'cost': 19}, {'cell': (1, 1), 'cost': 20}, {'cell': (4, 4), 'cost': 20}, {'cell': (4, 1), 'cost': 21}, {'cell': (2, 5), 'cost': 21}, {'cell': (2, 0), 'cost': 21}, {'cell': (3, 3), 'cost': 22}, {'cell': (5, 4), 'cost': 22}, {'cell': (5, 1), 'cost': 23}, {'cell': (3, 5), 'cost': 23}, {'cell': (3, 0), 'cost': 23}] ]

checking neighbor: (3, -1)

checking neighbor: (3, 1)

neighbor (3, 1) is reachable so adding to children list

checking neighbor: (2, 0)

neighbor (2, 0) is reachable so adding to children list

checking neighbor: (4, 0)

neighbor (4, 0) is reachable so adding to children list

child (3, 1) is not visited before so checking it

child has f(n): 30 g(n): 24 h(n): 6

since this child (3, 1) has best g value out of existing open list nodes adding it to open\_list/to\_be\_searched list [{'cell': (2, 3), 'cost': 26}, {'cell': (4, 5), 'cost': 25}, {'cell': (1, 5), 'cost': 25}, {'cell': (3, 4), 'cost': 24}, {'cell': (6, 4), 'cost': 24}, {'cell': (5, 2), 'cost': 25}, {'cell': (6, 1), 'cost': 25}, {'cell': (3, 4), 'cost': 25}, {'cell': (3, 1), 'cost': 30}]

child (4, 0) is not visited before so checking it

child has f(n): 25 g(n): 19 h(n): 6

since this child (4, 0) has best g value out of existing open list nodes adding it to open\_list/to\_be\_searched list [{'cell': (2, 3), 'cost': 26}, {'cell': (4, 5), 'cost': 25}, {'cell': (1, 5), 'cost': 25}, {'cell': (3, 4), 'cost': 24}, {'cell': (6, 4), 'cost': 24}, {'cell': (5, 2), 'cost': 25}, {'cell': (6, 1), 'cost': 25}, {'cell': (3, 4), 'cost': 25}, {'cell': (3, 1), 'cost': 30}, {'cell': (4, 0), 'cost': 25}]

----------

visited this node: (3, 4) as it has least cost 24 and added it to closed list:[ [{'cell': (0, 2), 'cost': 0}, {'cell': (1, 2), 'cost': 10}, {'cell': (0, 1), 'cost': 12}, {'cell': (1, 3), 'cost': 12}, {'cell': (2, 2), 'cost': 12}, {'cell': (3, 2), 'cost': 14}, {'cell': (1, 4), 'cost': 15}, {'cell': (2, 1), 'cost': 16}, {'cell': (4, 2), 'cost': 16}, {'cell': (0, 0), 'cost': 17}, {'cell': (2, 4), 'cost': 17}, {'cell': (4, 3), 'cost': 18}, {'cell': (1, 0), 'cost': 19}, {'cell': (1, 1), 'cost': 20}, {'cell': (4, 4), 'cost': 20}, {'cell': (4, 1), 'cost': 21}, {'cell': (2, 5), 'cost': 21}, {'cell': (2, 0), 'cost': 21}, {'cell': (3, 3), 'cost': 22}, {'cell': (5, 4), 'cost': 22}, {'cell': (5, 1), 'cost': 23}, {'cell': (3, 5), 'cost': 23}, {'cell': (3, 0), 'cost': 23}, {'cell': (3, 4), 'cost': 24}] ]

checking neighbor: (3, 3)

neighbor (3, 3) is reachable so adding to children list

checking neighbor: (3, 5)

neighbor (3, 5) is reachable so adding to children list

checking neighbor: (2, 4)

checking neighbor: (4, 4)

----------

visited this node: (6, 4) as it has least cost 24 and added it to closed list:[ [{'cell': (0, 2), 'cost': 0}, {'cell': (1, 2), 'cost': 10}, {'cell': (0, 1), 'cost': 12}, {'cell': (1, 3), 'cost': 12}, {'cell': (2, 2), 'cost': 12}, {'cell': (3, 2), 'cost': 14}, {'cell': (1, 4), 'cost': 15}, {'cell': (2, 1), 'cost': 16}, {'cell': (4, 2), 'cost': 16}, {'cell': (0, 0), 'cost': 17}, {'cell': (2, 4), 'cost': 17}, {'cell': (4, 3), 'cost': 18}, {'cell': (1, 0), 'cost': 19}, {'cell': (1, 1), 'cost': 20}, {'cell': (4, 4), 'cost': 20}, {'cell': (4, 1), 'cost': 21}, {'cell': (2, 5), 'cost': 21}, {'cell': (2, 0), 'cost': 21}, {'cell': (3, 3), 'cost': 22}, {'cell': (5, 4), 'cost': 22}, {'cell': (5, 1), 'cost': 23}, {'cell': (3, 5), 'cost': 23}, {'cell': (3, 0), 'cost': 23}, {'cell': (3, 4), 'cost': 24}, {'cell': (6, 4), 'cost': 24}] ]

finally visited goal!

Output: {'path\_taken': '(0, 2)->(1, 2)->(2, 2)->(3, 2)->(4, 2)->(4, 3)->(4, 4)->(5, 4)->(6, 4)',

'total\_cost': 24}

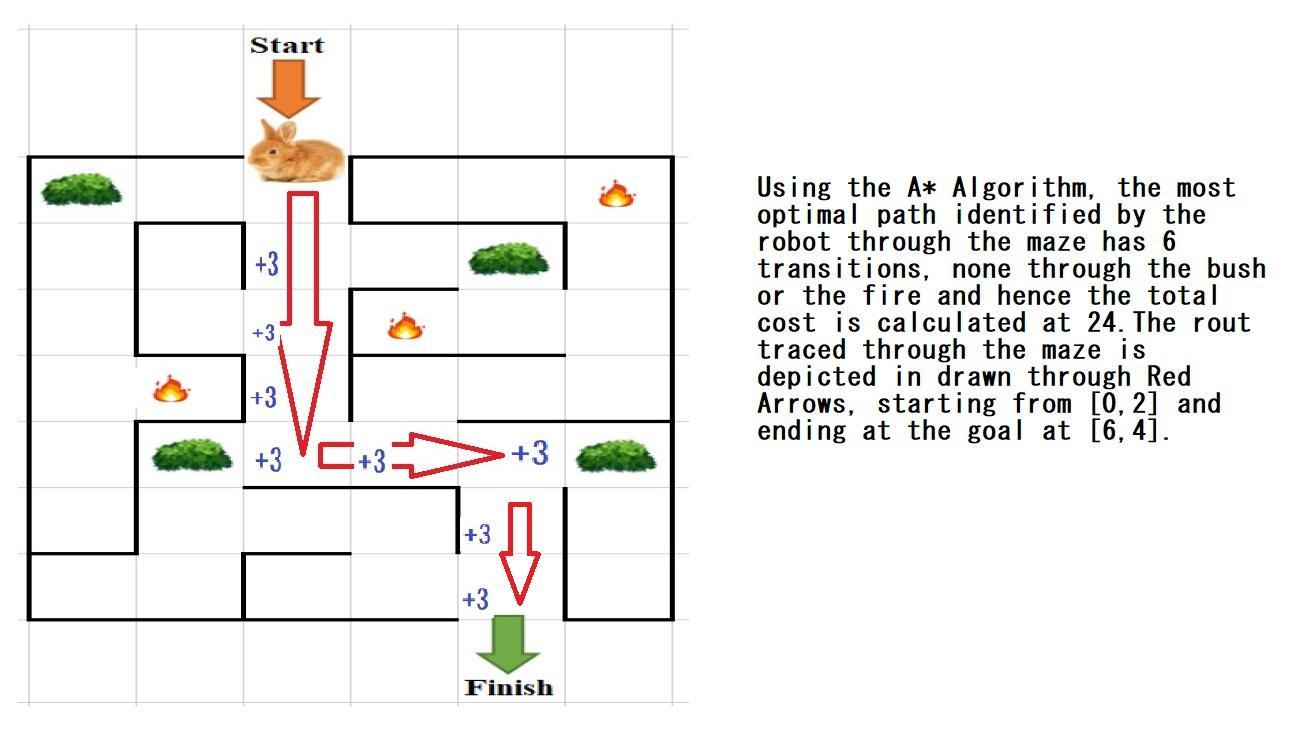


Fig 1. Optimal path found by A\* Algorithm

**Random Restart Hill Climbing:**

Theory/pseudocode explanation:

function HILL-CLIMBING(problem) returns a state that is a local maximum

current ← MAKE-NODE(problem.INITIAL-STATE)

loop do

neighbor ← a highest-valued successor of current

if neighbor.VALUE ≤ current.VALUE then return current.STATE

current ← neighbor

Note: Random-restart hill climbing adopts the well-known adage, “If at first you don’t succeed, try, try again.” It conducts a series of hill-climbing searches from randomly generated initial states, until a goal is found.

function RANDOM-RESTART-HILL-CLIMBING(problem) returns a state that is a local maximum

searched-states = hash-set()

loop until all possible nodes are selected as initial-state but goal was not found using it

random-initial-state ← randomly pick a suitable state from set of available states

loop until initial-state selected above was not used before

if random-initial-state not in searched-nodes

problem.INITIAL-STATE = random-initial-state

searched-states.add(random-initial-state)

break

random-initial-state ← randomly pick a suitable state from set of available states

final-state = HILL-CLIMBING(problem)

if final-state is GOAL then return final-state

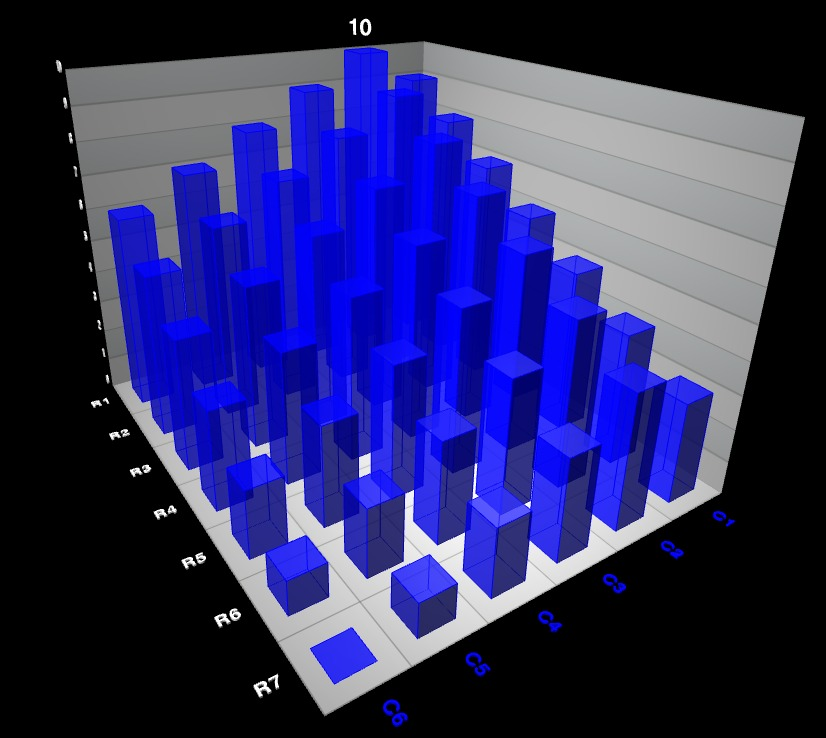


Fig 2. 3D Hill Graph for cost functions,calculated by subtracting the Manhattan distance of each node to the goal from a numerical constant (here it was 10 but can be any) where the highest peak is the goal.

#Code Block : Function for algorithm 2 implementation

from random import randint

from queue import PriorityQueue

hill\_state\_value = []

goal\_state = (6, 4)

for row in range(len(maze)):

hill\_state\_value.append([])

for column in range(len(maze[row])):

hill\_state\_value[row].append(10 - manhattan\_distance(goal\_state, (row, column)))

def HILL\_CLIMBING(maze, initial\_state, iter\_limit=50):

current\_state = initial\_state

walked\_path = [current\_state]

variable\_cost = 0

while iter\_limit != 0:

cell = maze[current\_state[0]][current\_state[1]]

neighbors = PriorityQueue()

for to\_new\_position in [LEFT, RIGHT, UP, DOWN]:

neighbor\_position = current\_state[0] + to\_new\_position[0], current\_state[1] + to\_new\_position[1]

if (does\_not\_cross\_maze\_border\_from(maze,neighbor\_position)):

if can\_walk\_from\_current(cell, to\_new\_position) and not fire\_at(cell) and neighbor\_position not in walked\_path:

neighbors.put\_nowait((-hill\_state\_value[neighbor\_position[0]][neighbor\_position[1]], neighbor\_position))

if (neighbors.empty()): return None, None, None

best\_neighbor\_value, best\_neighbor\_state = neighbors.get\_nowait()

best\_cell = maze[best\_neighbor\_state[0]][best\_neighbor\_state[1]]

variable\_cost += 3

if (fire\_at(best\_cell)): variable\_cost += 5

if (bush\_in(best\_cell)): variable\_cost += 1

walked\_path += [best\_neighbor\_state]

if (-best\_neighbor\_value <= hill\_state\_value[current\_state[0]][current\_state[1]]) or best\_neighbor\_state == goal\_state:

return best\_neighbor\_state, walked\_path, variable\_cost

current\_state = best\_neighbor\_state

iter\_limit -= 1

def RANDOM\_RESTART\_HILL\_CLIMBING(maze, goal\_state, iter\_limit=50):

searched\_states = set()

if (len(maze) > 0):

vertical\_length = len(maze[0]) - 1

horizontal\_length = len(maze) - 1

while iter\_limit != 0:

random\_initial\_state = randint(0, horizontal\_length), randint(0, vertical\_length)

while True:

if (not random\_initial\_state in searched\_states and

not fire\_at(maze[random\_initial\_state[0]][random\_initial\_state[1]])):

searched\_states.add(random\_initial\_state)

break

random\_initial\_state = randint(0, horizontal\_length), randint(0, vertical\_length)

print('selected random\_initial\_state:',random\_initial\_state)

final\_state, walked\_path, final\_cost = HILL\_CLIMBING(maze, random\_initial\_state)

if final\_state == goal\_state:

print('found global maxima!')

return {'path\_taken':'->'.join([str(e) for e in walked\_path]), 'total\_cost':final\_cost}

else: print('got stuck at local maxima')

iter\_limit -= 1

return 'not able to find path'

**Entire code execution with clear output flow:**

----------

selected random\_initial\_state: (6, 1)

checking neighbor: (6, 0)

neighbor (6, 0) is reachable so adding to neighbor list

checking neighbor: (6, 2)

checking neighbor: (5, 1)

neighbor (5, 1) is reachable so adding to neighbor list

checking neighbor: (7, 1)

picked neighbor cell (5, 1) since it is closer to goal

path walked so far [(6, 1), (5, 1)] with cost 3

got stuck at local maxima

----------

selected random\_initial\_state: (1, 5)

checking neighbor: (1, 4)

checking neighbor: (1, 6)

checking neighbor: (0, 5)

neighbor (0, 5) is reachable so adding to neighbor list

checking neighbor: (2, 5)

neighbor (2, 5) is reachable so adding to neighbor list

picked neighbor cell (2, 5) since it is closer to goal

path walked so far [(1, 5), (2, 5)] with cost 3

checking neighbor: (2, 4)

neighbor (2, 4) is reachable so adding to neighbor list

checking neighbor: (2, 6)

checking neighbor: (1, 5)

checking neighbor: (3, 5)

neighbor (3, 5) is reachable so adding to neighbor list

picked neighbor cell (2, 4) since it is closer to goal

path walked so far [(1, 5), (2, 5), (2, 4)] with cost 6

checking neighbor: (2, 3)

neighbor (2, 3) is reachable so adding to neighbor list

checking neighbor: (2, 5)

checking neighbor: (1, 4)

neighbor (1, 4) is reachable so adding to neighbor list

checking neighbor: (3, 4)

picked neighbor cell (1, 4) since it is closer to goal

encountered BUSH

path walked so far [(1, 5), (2, 5), (2, 4), (1, 4)] with cost 10

got stuck at local maxima

----------

selected random\_initial\_state: (5, 1)

checking neighbor: (5, 0)

checking neighbor: (5, 2)

neighbor (5, 2) is reachable so adding to neighbor list

checking neighbor: (4, 1)

neighbor (4, 1) is reachable so adding to neighbor list

checking neighbor: (6, 1)

neighbor (6, 1) is reachable so adding to neighbor list

picked neighbor cell (5, 2) since it is closer to goal

path walked so far [(5, 1), (5, 2)] with cost 3

checking neighbor: (5, 1)

checking neighbor: (5, 3)

neighbor (5, 3) is reachable so adding to neighbor list

checking neighbor: (4, 2)

checking neighbor: (6, 2)

picked neighbor cell (5, 3) since it is closer to goal

path walked so far [(5, 1), (5, 2), (5, 3)] with cost 6

checking neighbor: (5, 2)

checking neighbor: (5, 4)

checking neighbor: (4, 3)

checking neighbor: (6, 3)

neighbor (6, 3) is reachable so adding to neighbor list

picked neighbor cell (6, 3) since it is closer to goal

path walked so far [(5, 1), (5, 2), (5, 3), (6, 3)] with cost 9

checking neighbor: (6, 2)

neighbor (6, 2) is reachable so adding to neighbor list

checking neighbor: (6, 4)

neighbor (6, 4) is reachable so adding to neighbor list

checking neighbor: (5, 3)

checking neighbor: (7, 3)

picked neighbor cell (6, 4) since it is closer to goal

path walked so far [(5, 1), (5, 2), (5, 3), (6, 3), (6, 4)] with cost 12

found global maxima!

Output: {'path\_taken': '(5, 1)->(5, 2)->(5, 3)->(6, 3)->(6, 4)', 'total\_cost': 12}

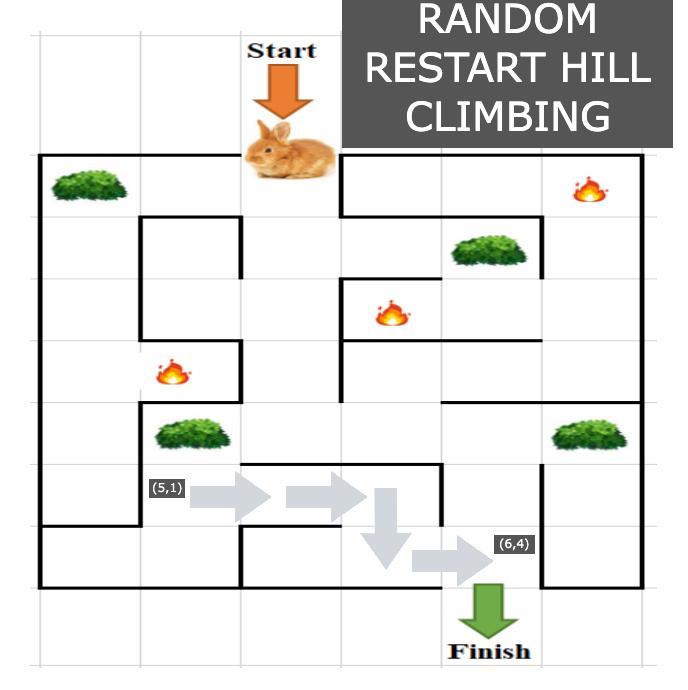
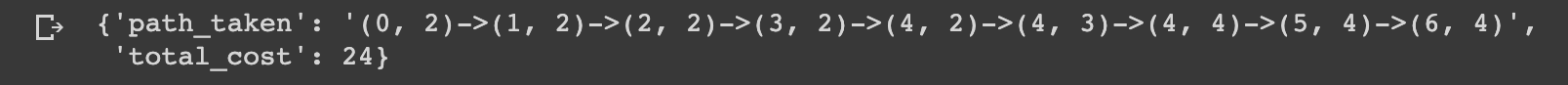


Fig 3. Optimal path found by Random Restart Hill Climbing algorithm

In the Random Restart Hill Climbing algorithm, the starting cell is selected at random and the target is to reach the end cell. In the current scenario, the PSA started from cell (5,1) and moved towards the goal cell that is (6,4). While moving towards the goal cell, it transitioned 4 times and hence the total cost was 12, since each transition cost is 3.

**Optimal Path sequence and costs (with screenshots)**

**A\* Algorithm**

****

As it can be seen from the above image, the total cost of the path is 24.

**Random Restart Hill Climbing Algorithm**

****

As it can be seen from the above image, the total cost for the Random Restart Hill climbing in this scenario was 12

**Space and Time Complexity Comparison** :

* A\* algorithm time complexity is O(b^d), where b is the branching factor and d is the length of the solution path. space complexity is also O(b^d), as it stores all generated nodes in memory
* Random Restart Hill Climbing algorithm time and space complexity of it depends on problem space, evaluation function, and number of restarts. Worst case, the time complexity is O(∞), as it may never find the global optimum. or we can put limit on number of iterations(n\_iter) so that it doesn't go to O(∞) and stops after that if global optimum is not found space complexity is O(b), where b is the branching factor, as it discards old nodes
* A\* uses cost function f(n) = g(n){distance from start to current} + h(n){from current to goal here it's Manhattan distance} whereas Random Restart Hill Climbing uses unit step distance from goal and its graph looks like hill
* A\* gives the same path every time since the starting and end goal nodes are predefined whereas Random Restart Hill Climbing gives a different path based on the start cell selected randomly.
* Hence, overall the Random Restart Hill Climbing would take more time to find a path as it might walk from the middle/part of the same route through which it had already passed in the previous iterations but was unable to find a solution.