Search Algorithm for 8 puzzles

Seyyed Ali Shohadaalhosseini

Implementation

This module will have the search algorithms methods. Each of this method takes an initial state and the Goal states and returns the solution to reach the goals.

DFS – Depth First Search

In the graph search we have both frontier and explored list, to prevent exploring a same state.

Steps

1- We put the initial state in the explored list.
2- We delete this state from the frontier
3- We put the initial state's child in the frontier
4- We choose the last child if it is not in the explored list, then we explore it and put it in the explored list.
5- We do this to reach the goal state

Note: the strategy is last in first out

Seyyed Ali Shohadaalhosseini Computer Engineering Student **@University of Mazandaran**
Email: Ali.shhde00@gmail.com **Id:** Alishhde

BFS – Breadth First Search

In the graph search we have both frontier and explored list, to prevent exploring a same state.

Steps

1- We put the initial state in the explored list.
2- We delete this state from the frontier
3- We put the initial state's child in the frontier
4- We choose the First child if it is not in the explored list, then we explore it and put it in the explored list.
5- We do this to reach the goal state

Note: the strategy is first in first out

DLS – Depth limited Search

In the graph search we have both frontier and explored list, to prevent exploring a same state.

Steps

1- We put the initial state in the explored list.
2- We delete this state from the frontier
3- We put the initial states child in the frontier
4- We choose the last child.
5- Check if this child has the depth limit or not, if not then step 6
6- if the child is not in the explored list, then we explore it and put it in the explored list.
7- We do this until we reach to the limit depth

Note: the strategy is the same strategy as the dfs, because this algorithm is dfs but with a limitation on depth.

Seyyed Ali Shohadaalhosseini Computer Engineering Student **@University of Mazandaran**
Email: Ali.shhde00@gmail.com **Id:** Alishhde

IDS – Iterative Deepening Search

In the graph search we have both frontier and explored list, to prevent exploring a same state.

Steps

1- We call DLS algorithm in a loop each time with a limit
2- The limit starts from zero to end of the space graph

UCS

In the graph search we have both frontier and explored list, to prevent exploring a same state.

Steps

1- We put the initial state in the explored list.
2- We delete this state from the frontier
3- We put the initial states child in the frontier
4- We calculate each child cost
   In the UCS the cost is $G_{(n)}$ which is the actual cost we spend to reach to the current node from the start node, or the actual cost it would be to reach to the next cost from the start to that node
5- We choose the child with the minimum cost, if it is not in the explored list, then we explore it and put it in the explored list.
6- We do this to reach the goal state

Seyyed Ali Shohadaalhosseini Computer Engineering Student **@University of Mazandaran**
Email: Ali.shhde00@gmail.com **Id:** Alishhde

Greedy

In the graph search we have both frontier and explored list, to prevent exploring a same state.

Steps

1- We put the initial state in the explored list.
2- We delete this state from the frontier
3- We put the initial states child in the frontier
4- We calculate each child cost
   In the Greedy the cost is $H_{(n)}$ which is the actual cost we spend to reach to the Goal node from the Current node.
5- We choose the child with the minimum cost, if it is not in the explored list, then we explore it and put it in the explored list.
6- We do this to reach the goal state

A*

In the graph search we have both frontier and explored list, to prevent exploring a same state.

Steps

1- We put the initial state in the explored list.
2- We delete this state from the frontier
3- We put the initial states child in the frontier
4- We calculate each child cost
   In the A* the cost is $F_{(n)}$ which is the **sum** of the actual cost we spend to reach to the current node from the start node and the actual cost we spend to reach to the Goal node from the Current node.
5- We choose the child with the minimum cost, if it is not in the explored list, then we explore it and put it in the explored list.
6- We do this to reach the goal state

Seyyed Ali Shohadaalhosseini Computer Engineering Student **@University of Mazandaran**
Email: Ali.shhde00@gmail.com **Id:** Alishhde

Hill Climbing

In the graph search we have both frontier and explored list, to prevent exploring a same state.

Steps

1- Put initial state into the current state
2- While current state is not equal the goal state does
    a. Find all the neighbor of the current state
    b. Choose the neighbor with the best value
    c. Compare this neighbor's value with the current state's value
    d. If the value is better, choose this neighbor as the current neighbor
    e. Else, we are in the local maximum, and in our problem, the solution for this local maximum is to starting again from a random state.
    f. And loop

In the next page you can see the code that has implemented these steps.

Seyyed Ali Shohadaalhosseini Computer Engineering Student **@University of Mazandaran**
Email: Ali.shhde00@gmail.com **Id:** Alishhde

```python
def searchHillClimbing(self, initialState, GoalState, spaceGraph=[]):
    ## Done successfully
    """This algorithm will return the solution path.

    Args:
        initialState (List): Start state.
        GoalState (list): End state.
        spaceGraph (list): For the time user enter his graph states.
    """
    # Step 1
    currentState = deepcopy(initialState)
    explored = [] # We don't use this in hillClimbing. it is for not getting error and avoid coding more
    counter = 0

    # Step 2 - loop
    while currentState != GoalState:
        if counter == 1000 and self.autoFinish:
            print("\nunfortunately we couldn't find answer after 5000-th round.\n")
            break

        # Step 3
        self.fringe = []
        self.updateFringe(currentState, explored, GoalState, algorithmType="")

        # Step 4
        statesCost = []
        for state in self.fringe:
            cst = self.heuristics(state, GoalState)
            statesCost.append(cst)
        minValue = min(statesCost)
        minValueIndex = statesCost.index(minValue)


        currentValue = self.heuristics(currentState, GoalState)

        # print(self.fringe, self.fringe[minValueIndex], minValue, currentState, currentValue, sep="\n")
        # raise NotImplementedError

        # Step 5, 6
        if currentValue < minValue: # Lower because, lower heuristic means we are closer to the goal
            # we are trap into the local Maximum
            # now we must have random restart
            # we do this by creating a random state
            print(currentValue)
            print("we are restarting..")
            sleep(3)
            currentState = self.random8PuzzleCreator()
        else:
            currentState = self.fringe[minValueIndex]

        counter += 1

    else:
        return currentState
```


Seyyed Ali Shohadaalhosseini Computer Engineering Student **@University of Mazandaran**
Email: Ali.shhde00@gmail.com **Id:** Alishhde

The notation about the hill climbing is that most time, it traps in the local maximum. We have a couple of solution for this problem. Here we choose random restart for this problem about the local maximum. Each time we recognize the local maximum we start randomly again. In this random restart, we create a random state and we choose it as our current state and we try to reach the goal from this random state by choosing the best neighbor, we do this until we reach to the goal.

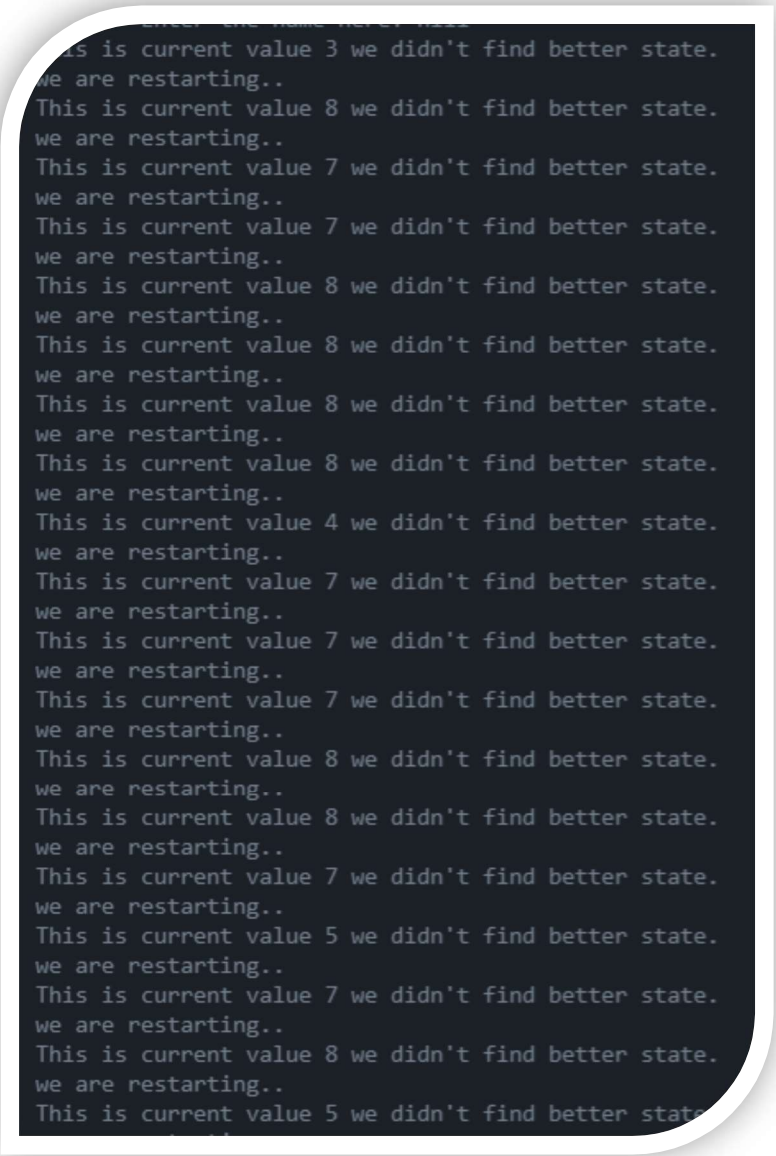
In the below picture you can see the function that creates the random states and returns this new state.

```
def random8PuzzleCreator(self):
    # Done successfully
    """
        we want a list, consist of three list that
        each list has three value

        """
    states = [[1, 2, 3], [4, 5, 6], [7, 8, " "]]
    shuffle(states)
    for row in states:
        shuffle(row)
    return states
```

In continue you can see the output. What we understand from the output is that trapping into the local maximum and the solution, restarting again doesn't help us good enough to reach the goal. That's because of the size of the space graph. Because the space of the 8 puzzle is very large, finding the goal and the good state is very hard and its probability is very less. You can see the current value that is some times more and sometimes low, that's because of the random restart we have in our problem.

Seyyed Ali Shohadaalhosseini Computer Engineering Student **@University of Mazandaran**
Email: Ali.shhde00@gmail.com **Id:** Alishhde



Thanks,

Seyyed Ali shohadaalhosseini

Seyyed Ali Shohadaalhosseini Computer Engineering Student **@University of Mazandaran**
Email: Ali.shhde00@gmail.com **Id:** Alishhde