Inteligenta Artificiala Lab4

Am inteles cum verifica si rezolva problema EightPuzzle, in EightPuzzleMiss returnand cate elemente nu sunt pe aceeasi pozitie ca la inceput (s!=g). Dupa care am implementat distanta Manhattan astfel:

class EightPuzzleMht(EightPuzzle):

def h(self, node):

""" implement Manhattan distance. Hint! Look at

Missplaced Tiles heuristic function above """

return sum((abs(self.goal.index(s)/3 - node.state.index(s)/3) + abs(self.goal.index(s)%3 - node.state.index(s)%3)) for(s) in self.goal)

Pentru fiecare index, cauta pozitia initiate si cea finala, la care face pe rand catul in urma impartirii la 3, pentru a obtine coordonata randului, si restul impartirii la 3 pentru a obtine coordonata coloanei elementului. Scade fiecare pereche de caturi si resturi, la care punem absolut pentru a calcula distanta, iar in cele din urma le aduna. Astfel returneaza distanta Manhattan. Apelam functia si ne da rezultatul:

['DOWN', 'LEFT', 'UP', 'RIGHT', 'DOWN']

0.0006616115570068359 #timp pentru EightPuzzleMiss

['DOWN', 'LEFT', 'UP', 'RIGHT', 'DOWN']

0.0006878376007080078 #timp pentru EightPuzzleMht

Searcher A\* h1(n) A\* h2(n)

astar\_search < 9/ 11/ 26/(1, 2, 3, 4, 5, 6, 7, 8, 0)> < 5/ 7/ 15/(1, 2, 3, 4, 5, 6, 7, 8, 0)>

recursive\_best\_first\_search < 13/ 14/ 36/(1, 2, 3, 4, 5, 6, 7, 8, 0)> < 6/ 7/ 17/(1, 2, 3, 4, 5, 6, 7, 8, 0)>

In continuare am copiat si modificat problema in search.py pentru a putea rezolva problema FifteenPuzzle astfel:

class FifteenPuzzle(Problem):

""" The problem of sliding tiles numbered from 1 to 15 on a 4x4 board,

where one of the squares is a blank. A state is represented as a 4x4 list,

where element at index i,j represents the tile number (0 if it's an empty square) """

def \_\_init\_\_(self, initial, goal=(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 0)):

""" Define goal state and initialize a problem """

self.goal = goal

Problem.\_\_init\_\_(self, initial, goal)

def find\_blank\_square(self, state):

"""Return the index of the blank square in a given state"""

return state.index(0)

def actions(self, state):

""" Return the actions that can be executed in the given state.

The result would be a list, since there are only four possible actions

in any given state of the environment """

possible\_actions = ['UP', 'DOWN', 'LEFT', 'RIGHT']

index\_blank\_square = self.find\_blank\_square(state)

if index\_blank\_square % 4 == 0:

possible\_actions.remove('LEFT')

if index\_blank\_square < 4:

possible\_actions.remove('UP')

if index\_blank\_square % 4 == 3:

possible\_actions.remove('RIGHT')

if index\_blank\_square > 11:

possible\_actions.remove('DOWN')

return possible\_actions

def result(self, state, action):

""" Given state and action, return a new state that is the result of the action.

Action is assumed to be a valid action in the state """

# blank is the index of the blank square

blank = self.find\_blank\_square(state)

new\_state = list(state)

delta = {'UP':-4, 'DOWN':4, 'LEFT':-1, 'RIGHT':1}

neighbor = blank + delta[action]

new\_state[blank], new\_state[neighbor] = new\_state[neighbor], new\_state[blank]

return tuple(new\_state)

def goal\_test(self, state):

""" Given a state, return True if state is a goal state or False, otherwise """

return state == self.goal

def check\_solvability(self, state):

""" Checks if the given state is solvable """

inversion = 0

for i in range(len(state)):

for j in range(i, len(state)):

if state[i] > state[j] != 0:

inversion += 1

return inversion % 2 == 0

def h(self, node):

""" Return the heuristic value for a given state. Default heuristic function used is

h(n) = number of misplaced tiles """

return sum(s != g for (s, g) in zip(node.state, self.goal))

In fifteen\_puzzle.py am copiat eight\_puzzle.py si modificat doar valoarea pentru cat si rest pentru distanta Manhattan de la 3 la 4.

Dupa care am copiat in main si modificat numele pentru a putea face ambele probleme simultan. Am observant ca ia foarte mult sa calculeze pentru 16 valori, asa ca am facut doar cateva modificari fata de goal.

problem\_mht\_15 = FifteenPuzzleMht((1, 2, 3, 4, 5, 6, 7, 8, 0, 10, 11, 12, 9, 13, 14, 15), (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 0))

Dupa rulare pentru FifteenPuzzle am obtinut:

['DOWN', 'RIGHT', 'RIGHT', 'RIGHT']

0.0002963542938232422

['DOWN', 'RIGHT', 'RIGHT', 'RIGHT']

0.0006427764892578125

Searcher A\* h1(n) A\* h2(n)

astar\_search < 4/ 6/ 11/(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 0)> < 4/ 6/ 11/(1, 2, 3, 4, 5, 6, 7, 8,

9, 10, 11, 12, 13, 14, 15, 0)>

recursive\_best\_first\_search < 4/ 5/ 11/(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 0)> < 6/ 7/ 18/(1, 2, 3, 4, 5, 6, 7, 8,

9, 10, 11, 12, 13, 14, 15, 0)>