Gavin Boin

CS 470

Deterministic Search Lab

Part 1

1.3) Node format = (average node expanded, average search time, average solution length)

Averages / Difficulty	**Uniform Cost Search**	**Greedy**	**A***
Easy.txt	143.7/0.0037/7	52.675/0.0015/10.7	13.35/0.0006/7
Medium.txt	6407.25/0.1831/15	624.8/0.0147/75	341.175/0.0084/15
Hard.txt	72320/3.0481/21	716.2/0.0171/94	4887.875/0.1351/21

1.5) All tests run on hard.txt to be consistent but also show difference between each.

Search method / Heuristic	**Uniform Cost Search**	**Greedy**	**A***
Tiles_out_of_place	72320/2.97/21	716/0.01/94	4887/0.13/21
Tiles_out_of_row_and_col	72320/4.48/21	695113/12.8/21	35073/1.81/21
Manhattan_distance	72320/4.41/21	15443/0.54/96	36551/1.75/21

I learned a lot about how each of the search methods affect the time and efficiency of finding the goal. For instance, the uniform cost searches every time expanded 72320 nodes meaning that every time it would expand the same amount for hard mode and always finds the optimal solution of 21 average path length for each because it is the optimal solution. However, the uniform cost is not using any heuristic and is only made off the cost of the values to their goal The greedy algorithm would always run much faster than any other most of the time per puzzle but would never get the optimal solution. The number of nodes that need to be expanded each time is a lot more variable as greedy tries to find the solution as fast as possible and will get lucky if it finds the solution with low nodes expanded but sometimes takes much longer than others. The A* used the principles of both and used the heuristic with the cost to find the best value to expand off of and is able to take the strengths of both search functions. We see this because overall the A* doesn't always beat either but is steadier as it always finds the optimal solution but also runs much faster than the uniform cost.

Part 2 values(ave nodes expanded / ave time / ave best path)

Search / Heuristic	**Uniform Cost Search**	**Greedy**	**A***
Top	28270/0.31/15	28015/0.29/15	28015/0.33/15
Torc	28270/0.49/15	28265/0.48/15	28265/0.48/15
Md	28270/0.49/15	28267/0.48/15	28267/0.50/15

I chose to implement the same functions of the dfs and iterative deepening search for the IDA function so that I wouldn't get it confused and would still be able to test the past sections after changing it. It was surprising that many of the different heuristics acted very similar even though in past circumstances it was much drastically different. I chose to use the medium.txt for the testing because it would run faster and would be a way to test the slight discrepancies between all the different settings. The lowest times were for the first heuristic where the out of place tiles are added up and gave the lowest run times. Each of these were very similar in that each was able to find the optimal solution within around 28000 nodes expanded each time. The uniform cost search had the same number of nodes expanded for each as

before but for the greedy and A* they both had the same number of nodes expanded per heuristic. We can see that the IDA* algorithm runs similarly regardless of the f value of the search because it will run very similarly every time.

Index

Part 1

```
def is_solved(self):
    """
    Returns True if the puzzle is solved, False otherwise
    """
    ######## TASK 1.1 BEGIN ##########
    for i in self.state:
        if self.state[i] != i:
            return False
    return True
    ######## TASK 1.1 END   ##########
```

```
def compute_f_value(self):
    """
    Compute the f-value for this node
    """
    ######## TASK 1.3 BEGIN ##########

    #Modify these lines to implement the search algorithms (greedy, Uniform-cost or A*)
    self.h = heuristic(self, self.options)
    self.f_value = 0

    if self.options.type == 'g':
        #greedy search algorithm
        self.f_value = self.h

    elif self.options.type == 'u':
        #uniform cost search algorithm
        self.f_value = self.cost

    elif self.options.type == 'a':
        #A* search algorithm
        self.f_value = self.h + self.cost

    else:
        print('Invalid search type (-t) selected: Valid options are g, u, and a')
        sys.exit()

    ######## TASK 1.3 END   ##########
```

```python
def tiles_out_of_row_column(puzzle):
    """
    This heuristic counts the number of tiles that are in the wrong row,
    the number of tiles that are in the wrong column
    and returns the sum of these two numbers.
    Remember not to count the blank tile as being out of place, or the heuristic is inadmissible
    """
    ######## TASK 1.4.1 BEGIN   ##########

    null_row = 0
    null_col = 0
    for i in range(1,len(puzzle.state)):
        if get_tile_row(i) - get_tile_row(puzzle.state[i]) != 0:
            null_row = 1
        if get_tile_column(i) - get_tile_column(puzzle.state[i]) != 0:
            null_col = 1
    return null_row + null_col

    ######## TASK 1.4.1 END   ##########
```

```python
def manhattan_distance_to_goal(puzzle):
    """
    This heuristic should calculate the sum of all the manhattan distances for each tile to get to
    its goal position.  Again, make sure not to include the distance from the blank to its goal.
    """

    ######## TASK 1.4.2 BEGIN   #########
    total = 0
    for i in range(1, len(puzzle.state)):
        total += get_tile_row(i) - get_tile_row(puzzle.state[i])
        total += get_tile_column(i) - get_tile_column(puzzle.state[i])
    return total
    ######## TASK 1.4.2 END   ##########
```

Part 2

```python
def run_IDA(start_node):
    max_depth_limit = 40
    start_node.compute_f_value()
    cutoff = start_node.f_value
    total_expanded = 0
    print(start_node.puzzle)
    while cutoff < max_depth_limit:
        visited = dict()
        visited['N'] = 0
        visited[start_node.puzzle.id()] = True

        path_length = run_dfs(start_node, cutoff, visited)
        total_expanded += visited['N']

        if path_length is not None:
            print('Expanded ', total_expanded, 'nodes')
            print('IDA Found solution at depth', cutoff)
            return total_expanded, path_length
        cutoff += 1
    return None
```

```python
def run_IDA_dfs(node, depth_limit, visited):
    visited['N'] = visited['N'] + 1

    if node.puzzle.is_solved():
        print('Iterative Deepening A* SOLVED THE PUZZLE! SOLUTION = ', node.path)
        return len(node.path)
    if node.compute_f_value() > depth_limit:
        return None

    moves = node.puzzle.get_moves()
    for m in moves:
        node.puzzle.do_move(m)
        node.compute_f_value()

        node.path = node.path + m
        node.cost = node.cost + 1
        if node.puzzle.id() not in visited:
            visited[node.puzzle.id()] = True
            path_length = run_IDA_dfs(node, depth_limit, visited)

            if path_length is not None:
                return path_length

            del visited[node.puzzle.id()]
        node.puzzle.undo_move(m)
        node.path = node.path[0:-1]
        node.cost = node.cost - 1
    return None
```