

## REPORT | Dana Kathleen Redeña

### Implementation of Algorithms

#### I. Modules

- a. OpenCV2 – for image processing
- b. Heapq – for max heap priority queues
- c. Copy – for duplicating the input image object in processing the output of the 3 algorithms

#### II. class Node

- a. class Node represents each node of the generated search trees
- b. Attributes and Get-Methods
  - i. cost (get\_cost()) – essentially  $f(n)$  (from  $f(n) = g(n) + h(n)$  of informed search algorithms)
  - ii. parent (get\_parent()) – parent of the current node
  - iii. i (get\_i()) – location: ith row in a grid/2-D array
  - iv. j (get\_j()) – location: jth row in a grid/2-D array
  - v. g\_val (get\_g\_val()) – green value/intensity of a cell
  - vi. path\_cost (get\_path\_cost()) – computed path cost of a node without heuristic function; essentially  $g(n)$  (from  $f(n) = g(n) + h(n)$ ) – sum of path costs from start node to node n

#### III. Global Variables

- a. g\_grid – 2-D integer array representing the input image
- b. grid\_row – number of rows of g\_grid
- c. grid\_col – number of columns of g\_grid
- d. priorq – priority queue used for the search algorithms
- e. goal – Node object based from the blue cell
- f. start – Node object based from the red cell
- g. best\_soln – array containing the last node of the search algorithms' solutions

#### IV. Functions

- a. gen\_grid(img, width, height)
  - i. Parameters
    - 1. img – input image object (created using opencv2's imread method)

2. width – pixel-width of the input image object (using shape method)
3. height – pixel-height of the input image object (using shape method)

ii. Functionalities

1. Generates a 2-D integer array that contains the green values of each 50x50 cell from the input image object, with the black cells represented by a -1 value.
2. Creates Node objects for the start and goal nodes indicated by the red and blue colored cells (Node objects not actual search tree nodes, created just for abstraction)
3. Returns the number of rows and columns of the 2-D integer array.

b. `is_cycle(x, y, from_node)`

i. Parameters

1. x – location: xth row in a grid/2-D array
2. y – location: yth column in a grid/2-D array
3. from\_node – Node object of parent

ii. Functionalities

1. Detects if node (represented by x and y values) will create a cyclic path by going through all the ancestor nodes to check for any repetitions.
2. Returns True if node creates a cyclic path, False if otherwise.

c. `is_path(to_i, to_j, from_node, cond)`

i. Parameters

1. to\_i – location of next node: ith row in a grid/2-D array
2. to\_j – location of next node: jth column in a grid/2-D array
3. from\_node – Node object of parent
4. cond – indicates the search algorithm being implemented (0 = Uniform-Cost; 1 = Greedy-BFS; 2 = A\*)

ii. Functionalities

1. Checks for the validity of the search tree node, before it is instantiated into a Node object and pushed into the max heap priority queue (priorq). Fulfilled conditions (listed below) simply returns the function without instantiation and changes in priority queue.

2. Checks if to\_i and to\_j are still within the boundaries of the g\_grid.
3. Checks if node is an obstacle, indicated by the -1 value in the g\_grid
4. Checks if node causes a cyclic path by calling is\_cycle().
5. Computes for the f(n) according to the algorithm being implemented.
6. Checks if node causes a solution to be found already (node is equal to goal state) and if found solution is less optimal than the previous found solution, if any.
7. Checks if node exists in the priority queue already and checks whether the duplication in the priority queue has a less cost. In the case where the existing node in the priority queue has a greater cost, the node is deleted, the priority queue is heapified.
8. Checks if the node cost is greater than the found solution so far if ever there is any.
9. In the case that the node does not fulfil the conditions mentioned above, a Node object containing the node cost, location, parent and path cost is pushed into the priority queue.

d. px\_path(i, j, from\_node, cond)

i. Parameters

1. i – location: ith row in the grid
2. j – location: jth column in the grid
3. from\_node – Node object of parent
4. cond - indicates the search algorithm being implemented (0 = Uniform-Cost; 1 = Greedy-BFS; 2 = A\*)

ii. Functionalities

1. Gets the next nodes (up, down, left, right nodes) from the current node of the search tree and calls is\_path() to see the validity of those nodes.

e. pathfind(cond)

i. Parameters

1. cond - indicates the search algorithm being implemented (0 = Uniform-Cost; 1 = Greedy-BFS; 2 = A\*)

ii. Functionalities

1. From the starting node, the function will call `px_path()` and continues the expansion of nodes. The function returns that last node of the solution.
  2. Which node to expand next depends on the least cost node that is popped/dequeued from the max heap priority queue.
  3. The function will continue to loop until either the priority queue is empty, until the next popped cost has a greater cost than the best found solution so far, or in the case of Greedy-BFS, until the first solution is found.
- f. `back_draw(goal, cond, img)`
- i. Parameters
    1. `goal` – last node of solution from the search algorithm
    2. `cond` - indicates the search algorithm being implemented (0 = Uniform-Cost; 1 = Greedy-BFS; 2 = A\*)
    3. `img` – a duplicate/copy of the input image object (using `.copy()`)
  - ii. Functionalities
    1. The function traces a white line along the final solutions in the search algorithms. Outputs the generated image along with the corresponding image name based of the search algorithm implemented.

## V. Outputs

- a. Three images for each of the search algorithms with a white line trace from the start node to the goal node.
- b. The path costs for each of the search algorithms is outputted to “solutions.txt”.

## Heuristic Function

The Greedy-BFS implementation uses the Manhattan Distance as its heuristic function

For the A\* the heuristic function is an improvement of the Manhattan Distance with  $h(n) = \text{man\_d} + \text{to\_g} - 1$ , where  $\text{man\_d}$  is the Manhattan Distance value and  $\text{to\_g}$  is the green intensity value of the current node being expanded to. The idea behind this is that for every possible path from the start node to the goal node, where total path cost computed by the summation of the differences in green intensity values of adjacent nodes (with no adjacent passable nodes having the same green intensity value), the difference of taking the “first step” (the adjacent node to the start node) is always equal to the green intensity value of the node adjacent to the start node (since the start node has a 0 green intensity value) and this value will always be added to the path cost.

Let  $h_1(n)$  be the Manhattan Distance as the heuristic function

Let  $h_2(n)$  be the modification of the Manhattan Distance with the use of the node's green intensity value

### Prove by induction that $h_2(n)$ is a more informed heuristic function than $h_1(n)$

First, we need to prove that  $h_1(n) \leq h_2(n)$ , where  $h_2(n) = h_1(n) + \text{green} - 1$ . Let  $\text{green}$  be the green value of node  $n$ , where  $1 \leq \text{green} \leq 255$ .

**Base Case:**



node  $n$  and goal node are side by side,

$$h_1(n) = 1$$

$$h_2(n) = h_1(n) + \text{green} - 1, \text{ where } 1 \leq \text{green} \leq 255$$

$$h_2(n) = 1 + \text{green} - 1$$

$$h_2(n) = \text{green}$$

**Inductive Step:** Assume that  $h_1(m) \leq h_2(m)$ , given  $h_2(m) = h_1(m) + \text{green} - 1$  and  $h_1(m) = h_1(m-1) + 1$

Show that  $h_1(m+1) \leq h_2(m+1)$ , given  $h_2(m+1) = h_1(m+1) + \text{green} - 1$  and  $h_1(m+1) = h_1(m) + 1$

$$h_2(m+1) = h_1(m+1) + \text{green} - 1$$

$$h_2(m+1) = h_1(m) + 1 + \text{green} - 1$$


$$h_2(m+1) = h_1(m) + \text{green}, \text{ where } 1 \leq \text{green} \leq 255$$

$$h_1(m+1) = h_1(m) + 1$$

$$h_1(m) + 1 \leq h_1(m) + \text{green}$$

$$\therefore h_1(m+1) \leq h_2(m+1) \blacksquare$$

Then we need to show that  $h_2(n)$  is admissible given that  $h_2(n) \leq h^*(n)$  where  $h^*(n)$  is the true cost.  $h^*(n)$  is the summation of all the differences in the green intensity value of the adjacent nodes from the start node to the goal node.

**Base Case:**  node  $n$  and goal node are side by side,

$$h^*(n) = \text{green} - \text{green}_{\text{goal}}$$

$$h^*(n) = \text{green} - 0$$

$$h^*(n) = \text{green}$$

$$h_2(n) = h_1(n) + \text{green} - 1, h_2(n) = 1 + \text{green} - 1$$

$$h_2(n) = \text{green}$$

**Inductive Step:** Assume that  $h_2(m) \leq h^*(m)$ , given  $h_2(m) = h_1(m) + \text{green} - 1$  and  $h_1(m) \leq h^*(m)$

Show that  $h_2(m+1) \leq h^*(m+1)$ , given  $h_2(m+1) = h_1(m+1) + \text{green} - 1$  and  $h_1(m+1) \leq$

$h^*(m+1)$

$$h_2(m+1) = h_1(m+1) + \text{green} - 1$$

$$h_2(m+1) - \text{green} + 1 = h_1(m+1)$$

$$h_2(m+1) - \text{green} + 1 \leq h^*(m+1), \text{ where } 1 \leq \text{green} \leq 255$$

$$\therefore h_2(m+1) \leq h^*(m+1) \blacksquare$$