# **ICE -3**

# **INFORMED SEARCH**

## OVERVIEW

Search algorithms can be classified into two types:

* Informed search algorithms: These type of algorithms leverage any information (heuristics, path cost) on the problem to search through the search space to find the solution efficiently.
  + Examples:
    1. Best First Search
    2. Uniform Cost Search
    3. A\* Search
    4. Recursive Best First Search

**CONTENTS**

Problem

Node

Greedy Best First Search

A\* Search

## PROBLEM

psource(Problem)

class Problem:

\_\_init\_\_(self, initial, goal=None):

actions(self, state):

result(self, state, action):

goal\_test(self, state):

path\_cost(self, c, state1, action, state2):

value(self, state):

## NODE

psource(Node)

class Node:

\_\_init\_\_(self, state, parent=None, action=None, path\_cost=0):

expand(self, problem):

child\_node(self, problem, action):

solution(self):

**def** path(self):

These 4 methods override standards Python functionality for representing an object as a string, the less-than (<) operator, the equal-to (=) operator, and the hash function.

repr\_\_(self):

\_\_lt\_\_(self, node):

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

\_\_hash\_\_(self):

\*Abstract class Problem to define our real **problem** named GraphProblem. You can see how we define GraphProblem by running the next cell.

psource(GraphProblem)

**class** **GraphProblem**(Problem):

\_\_init\_\_(self, initial, goal, graph):

actions(self, A):

result(self, state, action):

path\_cost(self, cost\_so\_far, A, action, B):

find\_min\_edge(self):

h(self, node):

## 

## GREEDY BEST FIRST SEARCH:

## F(n) = estimate of cost from n to goal.

## F(n) is the straight line distance from Bucharest.

## It expands the node that appears to be the closest to goal.

Greedy Best-first graph search is an informative searching algorithm with f(n) = h(n).

You need to specify the h function when you call best\_first\_search, or

else in your Problem subclass.

## Diagram Description automatically generated

## Step 1: Add the first node to the OPEN list.

## Step 2: Stop and return failure if the OPEN list is empty.

## Step 3: Remove the node n from the OPEN list with the lowest f(n) value and transfer it to the CLOSED list.

## Step 4: Extend node n and produce node n's successors.

## Step 5: Examine each successor of node n to see whether any of them is a goal node. If any of the successor nodes is a goal node, return success and end the search; otherwise, move to Step 6.

## Step 6: For each successor node, the algorithm looks for the evaluation function f(n) and then determines if the node has been in the OPEN or CLOSED list. If the node hasn't already been in both lists, add it to the OPEN list.

## A\* SEARCH:

## Avoids expanding paths that are already expansive

A\* search is best-first graph search with f(n) = g(n)+h(n).

Evaluation function f(n) = g(n)+h(n)

g(n)= cost so far to reach n

h(n)= estimated cost from n to goal

f(n)= estimated total cost of path through n to goal

Diagram

Description automatically generated

You need to specify the h function when you call astar\_search, or else in your Problem subclass.

Node class defines the structure of the state(configuration) and also provides functions to move the empty space and generate child states from the current state. Puzzle class accepts the initial and goal states of the N-Puzzle problem and provides functions to calculate the **f-score** of any given node(state).

Different heuristics provide different efficiency in solving A\* problems which are generally defined by the number of explored nodes as well as the branching factor. With the classic 8 puzzle we can show the efficiency of different heuristics through the number of explored nodes.

### 8 Puzzle Problem

The *8 Puzzle Problem* consists of a 3x3 tray in which the goal is to get the initial configuration to the goal state by shifting the numbered tiles into the blank space.

 Initial State                        Goal State  
          | 7 | 2 | 4 |                       | 1 | 2 | 3 |  
          | 5 | 0 | 6 |                       | 4 | 5 | 6 |  
          | 8 | 3 | 1 |                       | 7 | 8 | 0 |

We have a total of 9 blank tiles giving us a total of 9! initial configuration but not all of these are solvable. The solvability of a configuration can be checked by calculating the Inversion Permutation. If the total Inversion Permutation is even then the initial configuration is solvable else the initial configuration is not solvable which means that only 9!/2 initial states lead to a solution.  
Let's define our goal state.

#### **Heuristics :-**

1) Manhattan Distance:- For the 8 puzzle problem Manhattan distance is defined as the distance of a tile from its goal state( for the tile numbered '1' in the initial configuration Manhattan distance is 4 "2 for left and 2 for upward displacement").

2) No. of Misplaced Tiles:- The heuristic calculates the number of misplaced tiles between the current state and goal state.

3) Sqrt of Manhattan Distance:- It calculates the square root of Manhattan distance.

4) Max Heuristic:- It assign the score as the maximum between "Manhattan Distance" and "No. of Misplaced Tiles".

## Step 1: A set of prospective states we might be in

## Step 2: A beginning and end state

## Step 3: way to decide if we’ve reached the endpoint

## Step 4: A set of actions in case of possible direction/path changes

## Step 5: A function that advises us about the result of an action

## Step 6: A set of costs incurring in different states/paths of movementGraphical user interface, text, application, email Description automatically generated

## VISUALIZATION:

## Un-explored: The which we did not approach

## Frontier: Successor nodes to currently exploring node

## Currently exploring: The node which is to be compared with

## Explored: node which is already explored and visited

## Final Solution: The goal of getting the shortest distance.

## WE OBSERVE THERE ARE CHANGES IN ITERATIONS IN BOTH THE SEARCH

**TASK1: SANTA\_BARBARA MAP**

Chart, diagram, radar chart

Description automatically generated

## GREEDY BEST FIRST SEARCH

## F(n) = estimate of cost from n to goal.

## F(n) is the straight line distance from the starting node

## It expands the node that appears to be the closest to goal.

## Chart, radar chart Description automatically generated

**EXPLAINATION:**

Greedy search takes 16 iterations for the shortest path

* 1. Start from SantaBarbara
  2. Check from SantaBarbara LosAngeles – 30
* Check from SantaBarbara - Barstow -45
* Check from SantaBarbara -Malibu -45
  1. In greedy search it takes the lowest f(n ) value
  2. So it chooses path from SantaBarbara LosAngeles – 30
  3. Then it compares LosAngeles – SanDiego -100
* LosAngeles –riverside -25
* Riverside - SanDiego -90
* 100<115 (100+25)
  1. It also checks near paths and selects the lesser f(n) values
  2. Finally it chooses from SanDiego to ELCanjo.
  3. The Total distance id 145.

## A\* SEARCH

## Avoids expanding paths that are already expansive

A\* search is best-first graph search with f(n) = g(n)+h(n).

Evaluation function f(n) = g(n)+h(n)

g(n)= cost so far to reach n

h(n)= estimated cost from n to goal

f(n)= estimated total cost of path through n to goal

Chart, radar chart

Description automatically generated

A\*search takes 19 iterations for the shortest path

IT takes an imaginary route path to the ELCajon

1.Start from SantaBarbara

2.Check from SantaBarbara LosAngeles – 30

* Check from SantaBarbara - Barstow -45
* Check from SantaBarbara -Malibu -45

3.In greedy search it takes the lowest f(n ) value

4.So it chooses path from SantaBarbara LosAngeles – 30

5.Then it compares LosAngeles – SanDiego -100

* LosAngeles –riverside -25
* Riverside - SanDiego -90
* 100<115 (100+25)

6.It also checks near paths and selects the lesser f(n) values

7.Finally it chooses from SanDiego to ELCanjo.

8.The Total distance is 145.

**Task 5. – BREST\_MAP**

Chart, radar chart

Description automatically generated

## A\* SEARCH

## Avoids expanding paths that are already expansive

A\* search is best-first graph search with f(n) = g(n)+h(n).

Evaluation function f(n) = g(n)+h(n)

g(n)= cost so far to reach n

h(n)= estimated cost from n to goal

f(n)= estimated total cost of path through n to goal

Chart, radar chart

Description automatically generated

**RED:**

brest\_map.locations:

Brest=(25, 77), Rennes=(58, 88), Nantes=(59, 133),

Bordeaux=(64, 199), Toulouse=(108, 232), Montpellier=(156, 234),

Avignon=(198, 230), Grenoble=(227, 205), Lyon=(203, 176),

Dijon=(209, 117), Strasbourg=(238, 65), Nancy=(210, 84),

Calais=(124, 35), Caen=(89, 44), Paris=(137, 97),

Limoges=(119, 164), Marseille=(213, 267), Nice=(255, 259))

1. Bordeaux
2. Limoges
3. Toulouse
4. Nantes
5. Rennes
6. Montpellier
7. Paris
8. Lyon
9. Caen
10. Avignon
11. Calais
12. Grenoble
13. Dijon
14. Brest
15. Marseille
16. Nice
17. Nancy
18. Strasbourg

**Greedy best first search:**

## F(n) = estimate of cost from n to goal.

## F(n) is the straight line distance from the starting node

## It expands the node that appears to be the closest to goal.

**Chart, radar chart

Description automatically generated**

1. Bordeaux
2. Limoges
3. Paris
4. Nancy
5. Strasbourg

**---------------------------------------------------------------------------------------------------------------------**