TRIBHUVAN UNIVERSITY
INSTITUTE OF ENGINEERING
PURWANCHAL CAMPUS
DHARAN

ARTIFICIAL INTELLIGENCE

Lab Report IV

Submitted by:

Gokarna Baskota
PUR075BEI013

Submitted To:

Department of Electronics and Computer Engineering

Lab 4: Problem Solving by Searching: A* Search

Informed Search: Informed Search algorithms have information on the goal state which helps in more efficient searching. This information is obtained by a function that estimates how close a state is to the goal state. In this lab, we are going to implement A* search.

A* search is the most commonly known form of best-first search. It uses heuristic function h(n), and cost to reach the node n from the start state g(n). A* search algorithm finds the shortest path through the search space using the heuristic function

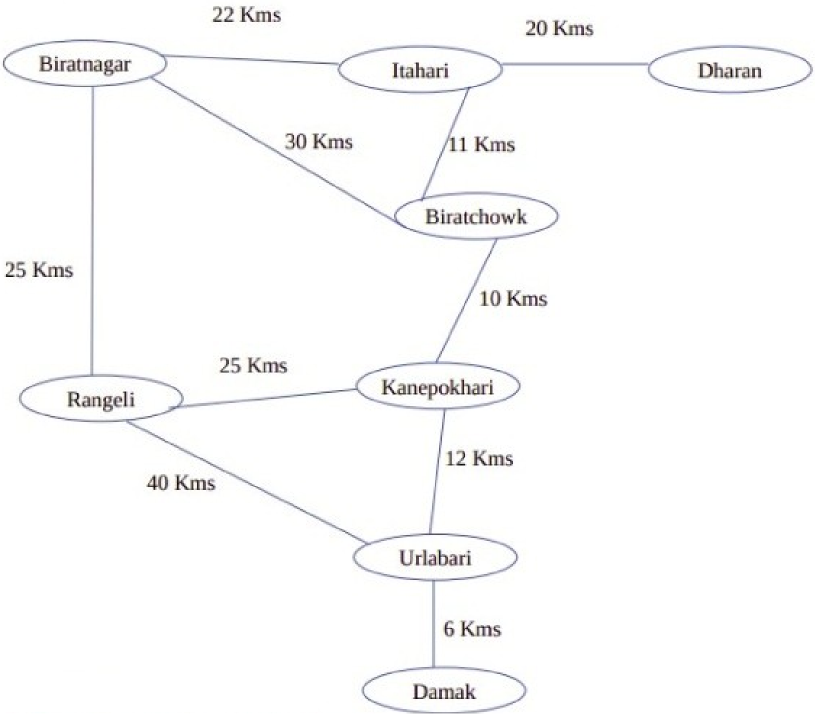
$$f(n) = g(n) + h(n).$$

This search algorithm expands less search tree and provides optimal result faster.

Algorithm

```
1. Declare the visited list
2. Declare the unvisited list
3. For each node in graph:
      – Add the node to the unvisited list with a g-score of infinity, an f-score of
        infinity and previous node of null.
4. Set the start node's g-score to 0 in the unvisited list
5. Set the start node's f-score to its h-score in the unvisited list
6. Set finished to False
7. While finished is False:
   ○ Set current node to the node in the unvisited list with the lowest f-score
   ○ If the current node is the target node
      ▪ Set finished to True
      ▪ Copy the values for the current node from the unvisited list to the
        visited list
   ○ Else
      ▪ For each neighbor of current node:
         • If neighbor is not in the visited list
            ○ Calculate new g-score = weight of edge + g-score of current
              node
            ○ If new g-score is less than neighbor's g-score in unvisited list
               ▪ Update the neighbor's g-score with the new g-score
               ▪ Update the neighbor's f-score to new g-score + h_score
               ▪ Update the neighbor's previous node to the current node
      ▪ Copy the values for the current node from the unvisited list to the
        visited list
      ▪ Remove the current node from the unvisited list

8. Return the visited list
```

We will use the A* algorithm to find the best path from Biratnagar to Damak in the following map.



The straight line distance from each of the city to Damak is shown below:

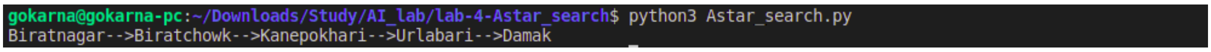
Heuristic function h(n)	
Biratnagar	46 Kms
Itahari	39 Kms
Dharan	41 Kms
Rangeli	28 Kms
Biratchowk	29 Kms
Kanepokhari	17 Kms
Urlabari	6 Kms
Damak	0 Kms

Program:

```
import numpy as np

g_index = 0
f_index = 1
prev = 2

def get_minimun(unvisited):
    fval = np.inf
    place = ""

    for city in unvisited.keys():
        if unvisited[city][1] < fval:
            fval = unvisited[city][1]
            place = city
    return place
```

```
def Astar(graph, start, goal):
    visited = {}
    unvisited = {}
    done = False

    for place in graph.keys():
        #                   g-score, f-score, previous
        unvisited[place] = [np.inf, np.inf, ""]

    unvisited[start] = [0, h[start], ""]

    while not done:
        if False:
            done = True
        else:
            current_node = get_minimun(unvisited)
            if current_node == goal:
                done = True
                visited[current_node] = unvisited[current_node]
            else:
                for neighbor in graph[current_node]:
                    if neighbor not in visited:
                        g_score =  unvisited[current_node][g_index] +
graph[current_node][neighbor]

                        if g_score < unvisited[neighbor][g_index]:
                            unvisited[neighbor][g_index] = g_score
                            unvisited[neighbor][f_index] = g_score +
h[neighbor]
                            unvisited[neighbor][prev] = current_node

                visited[current_node] = unvisited[current_node]
                del unvisited[current_node]

    return visited

graph = {
            "Biratnagar": {"Itahari":22, "Biratchowk":30, "Rangeli":25},
            "Itahari": {"Dharan":20, "Biratchowk":11, "Biratnagar":22},
            "Dharan": {"Itahari":20},
            "Rangeli": {"Kanepokhari":25, "Urlabari":40, "Biratnagar":25},
            "Biratchowk" : {"Kanepokhari":10, "Itahari":11,
"Biratnagar":30},
            "Kanepokhari": {"Urlabari":12, "Rangeli":25, "Biratchowk":10},
            "Urlabari": {"Damak":6, "Kanepokhari":12, "Rangeli":40},
            "Damak": {"Urlabari":6}
        }
# heuristic function h(n)
h = {
            "Biratnagar": 46,
            "Itahari": 39,
            "Dharan": 41,
            "Rangeli": 28,
            "Biratchowk" : 29,
            "Kanepokhari": 17,
            "Urlabari": 6,
            "Damak": 0
         }

def make_map(dict, goal):
    path = goal
    temp = goal
    while dict[temp][prev] != "":
        path = dict[temp][prev] + "-->" + path
        temp = dict[temp][prev]

    return path

if __name__ == "__main__":

    start = "Biratnagar"
    goal = "Damak"

    visit_node = Astar(graph, start, goal)

    print(make_map(visit_node, goal))
```

Output:



```
gokarna@gokarna-pc:~/Downloads/Study/AI_lab/lab-4-Astar_search$ python3 Astar_search.py
Biratnagar-->Biratchowk-->Kanepokhari-->Urlabari-->Damak
```