Artificial Intelligence Lab

AL-2002

Lab 11

Instructor: Hurmat Hidayat

Semester: Spring 2023

Artificial Intelligence Lab 11

Objective

The objective of this lab is to understand and apply informed search algorithms in problem-solving.

Learning Outcomes

1. Understand the concept of heuristics and their function in informed search algorithms.
2. Describe what the A* search algorithm is, and how it works.
3. Implementation of A* search algorithm.

Table of Contents

Informed Searches

So far we have talked about the uninformed search algorithms which looked throughsearch space for all possible solutions of the problem without having any additional knowledge about search space. But informed search algorithm **contains an array ofknowledge** such as how far we are from the goal, path cost, how to reach to goal node, etc. This knowledge help agents to explore less to the search space and find more efficiently the goal node.

The informed search algorithm is more useful for large search space. Informed searchalgorithm uses the idea of heuristic, so it is also called Heuristic search.

Heuristics function: Heuristic is a function which is used in Informed Search, and itfinds the most promising path. It takes the current state of the agent as its input and produces the estimation of how close agent is from the goal. The heuristic method, however, might not always give the best solution, but it guaranteed to find a good solution in reasonable time. Heuristic function estimates how close a state is to the goal. It is represented by **h(n)**, and it calculates the **cost of an optimal path between the pair of states**. The value of the heuristic function is always positive.

Admissibility of the heuristic function is given as:

$h(n) <= h^*(n)$

Pure Heuristic Search

Pure heuristic search is the simplest form of heuristic search algorithms. It expands nodes based on their heuristic value h(n). It maintains two lists, OPEN and CLOSEDlist. In the CLOSED list, it places those nodes which have already expanded and in the OPEN list, it places nodes which have yet not been expanded.

On each iteration, each node n with the lowest heuristic value is expanded and generates all its successors and n is placed to the closed list. The algorithm continuesunit a goal state is found.

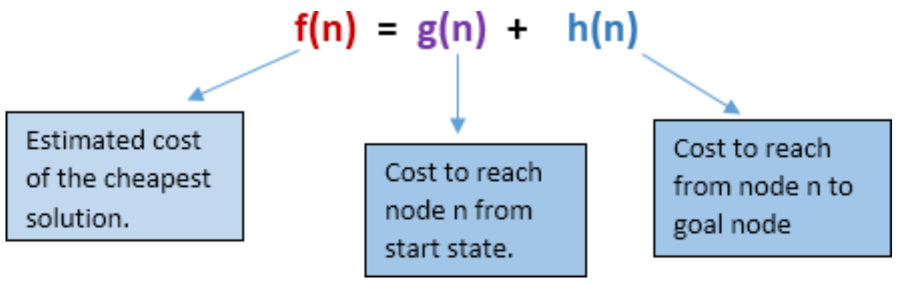
In the informed search we will discuss two main algorithms which are given below:

- **Best First Search Algorithm(Greedy search)**
- **A* Search Algorithm**

A* Search Algorithm

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). It has combined features of UCS and greedy best-first search, by which it solve the problem efficiently. A* search algorithm finds the shortest path through the search space using the heuristic function. This search algorithm expands less search tree and provides optimal result faster. A* algorithm is similar to UCS except that it uses g(n)+h(n) instead of g(n).

In A* search algorithm, we use search heuristic as well as the cost to reach the node. Hence we can combine both costs as following, and this sum is called as a **fitness number**.


f(n) = g(n) + h(n)
Estimated cost of the cheapest solution.
Cost to reach node n from start state.
Cost to reach from node n to goal node

Algorithm of A* search

Step1: Place the starting node in the OPEN list.

Step 2: Check if the OPEN list is empty or not, if the list is empty then return failure and stops.

Step 3: Select the node from the OPEN list which has the smallest value of evaluation function (g+h), if node n is goal node then return success and stop, otherwise

Step 4: Expand node n and generate all of its successors, and put n into the closed list. For each successor n', check whether n' is already in the OPEN or CLOSED list, if not then compute evaluation function for n' and place into Open list.

Step 5: Else if node n' is already in OPEN and CLOSED, then it should be attached to the back pointer which reflects the lowest g(n') value.

Step 6: Return to **Step 2**

Advantages:

- A* search algorithm is the best algorithm than other search algorithms.
- A* search algorithm is optimal and complete.
- This algorithm can solve very complex problems.

Disadvantages:

- It does not always produce the shortest path as it mostly based on heuristics and approximation.
- A* search algorithm has some complexity issues.
- The main drawback of A* is memory requirement as it keeps all generated nodes in the memory, so it is not practical for various large-scale problems.

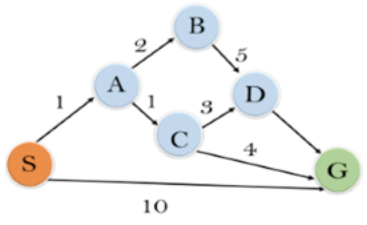
Example:

In this example, we will traverse the given graph using the A* algorithm. The heuristic value of all states is given in the below table so we will calculate the f(n) of each state using the formula

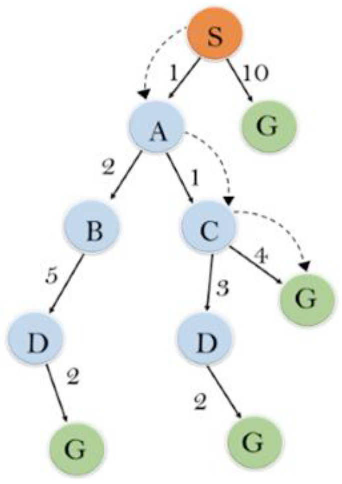
$$f(n)= g(n) + h(n)$$, where g(n) is the cost to reach any node from start state.

Here we will use OPEN and CLOSED list.

Problem **Solution**



State	h(n)
S	5
A	3
B	4
C	2
D	6
G	0



Initialization: {(S, 5)}

Iteration1: {(S--> A, 4), (S-->G, 10)}

Iteration2: {(S--> A-->C, 4), (S--> A-->B, 7), (S-->G, 10)}

Iteration3: {(S--> A-->C--->G, 6), (S--> A-->C--->D, 11), (S--> A-->B, 7), (S-->G, 10)}

Iteration 4 will give the final result, as S--->A--->C--->G it provides the optimal path with cost 6.

Points to remember

- A* algorithm returns the path which occurred first, and it does not search for all remaining paths.
- The efficiency of A* algorithm depends on the quality of heuristic.
- A* algorithm expands all nodes which satisfy the condition f(n)

Complete: A* algorithm is complete as long as:

- Branching factor is finite.
- Cost at every action is fixed.

Optimal: A* search algorithm is optimal if it follows below two conditions:

- Admissible: the first condition requires for optimality is that h(n) should be an admissible heuristic for A* tree search. An admissible heuristic is optimistic in nature.
- Consistency: Second required condition is consistency for only A* graph-search.

If the heuristic function is admissible, then A* tree search will always find the least cost path.

Time Complexity: The time complexity of A* search algorithm depends on heuristic function, and the number of nodes expanded is exponential to the depth of solution d. So the time complexity is O(b^d), where b is the branching factor.

Space Complexity: The space complexity of A* search algorithm is O(b^d)

Lab Task

A* Search Algorithm Implementation for Shortest Path Finding in Romania Map

Problem Statement: Implement the A* search algorithm in Python to find the shortest path between two cities on the map of Romania. The heuristic function used should be the straight-line distance between two cities.

Heuristic Function: The straight-line distance between two cities can be calculated using the latitude and longitude of the cities. This can be obtained using the geopy library in Python. Or use the following

```
lat_long = {
        'Arad': (46.1667, 21.3167), 'Bucharest': (44.4167, 26.1000),
        'Craiova': (44.3333, 23.8167), 'Drobeta': (44.6259, 22.6566),
        'Eforie': (44.0667, 28.6333), 'Fagaras': (45.8416, 24.9730),
        'Giurgiu': (43.9037, 25.9699), 'Hirsova': (44.6833, 27.9500),
        'Iasi': (47.1585, 27.6014), 'Lugoj': (45.6904, 21.9033),
        'Neamt': (46.9283, 26.3705), 'Oradea': (47.0553, 21.9214),
        'Pitesti': (44.8565, 24.8697), 'Rimnicu Vilcea': (45.1042, 24.3758),
        'Sibiu': (45.7977, 24.1521), 'Timisoara': (45.7489, 21.2087),
        'Urziceni': (44.7167, 26.6333), 'Vaslui': (46.6333, 27.7333),
        'Zerind': (46.6225, 21.5174)
}
```

Deliverables:

1. Python code for A* search algorithm implementation using a node class.
2. The shortest path and its distance between two cities.

   ```
   Shortest path from Arad to Bucharest: ['Arad', 'Sibiu', 'Rimnicu Vilcea', 'Pitesti', 'Bucharest']
   Distance: 418
   ```