Subway Itinerary Planner

Erdal Sidal Dogan
Computer Engineering Dept. MEF University
January 18, 2021

***Abstract*—Artificial Intelligence depends on advanced algorithms to solve various kinds of problems such as searching, optimizing, decision making etc. This paper introduces the implementation of a popular search algorithm, A* Search. The algorithm is applied for finding the shortest path between two subway stations in Istanbul. Given a station and destination from the user, the program returns the shortest itinerary.**

***Index Terms*—Artificial Intelligence, Shortest Path Problem, Algorithm, Search Algorithm**

I. Introduction

Search problems are being studied by mathematicians and computers scientist for decades. Consequently, there are numerous of algorithms that excel at certain aspects or problems but perform average for other tasks. In this paper, our problem is shortest path finding. Most commonly used algorithms for this problem are:

- Depth/Breadth First
- Uniform Cost
- A* Star Search
- Graph Search

and many other customized algorithms. For the path finding problem, we use *A* Search* in our implementation.

All of this methods share a common terminology which is used in this paper also.

- State Space: Set of all possible states the agent can be. For our problem, set of subway stations
- Node: A state on the state space, corresponds to a station in this problem.
- Cost: Cost of moving between two nodes in the state space. Higher distance means higher cost
- Heuristic: Estimation of the cost between two nodes.

II. Algorithm & Implementation

A Search* algorithm simple yet powerful. It is classified under a group of search algorithms named *Informed Search*. Apart from the *Uninformed Search* algorithms where only the state space, starting node and destination node is known by the agent, informed search has an additional information that makes the searching more efficient. This additional piece of information is called a *heuristic*. [1]

A. Heuristic

In general terms, heuristic is a broad estimation of the likelihood of the present node. For our problem, it is the estimation of the distance between the current node and the destination. Heuristic function is denoted with $h(n)$

One simple rule about the heuristics is that *it must not overestimate*. In order to satisfy such requirement, we used the *Euclidean Distance* between two nodes in the state spaces, which guarantees the absence of overestimation by resulting in absolute shortest distance.

$$h(n) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2} \tag{1}$$

B. Algorithm

A Search* uses some data structures while conducting the search. Usually, graphs and priority queues. *Priority Queue* is used to get the nodes with the minimum cost available.

Cost is calculated as $f(n) = g(n) + h(n)$ where the $f(n)$ is the total cost, $g(n)$ is the current cost of until reaching the node and $h(n)$ estimated distance to goal (heuristic function)

Algorithm keeps track of the nodes that as been traversed so far. Previous node to each node is known at any time. Also, cost of reaching to that node is stored as well. Traversed nodes are stored in a `Priority Queue` with their total cost $f(n)$. Retrieval of an element from a `Priority Queue` results with the retrieval of the element with the least cost. Therefore, when we retrieve an element from Queue, we get the node that is closest to destination according to our knowledge. Each element in this `Queue` is retrieved and its neighbors are added to the queue until either the queue is empty or the retrieved node is the destination node.

C. Implementation Details

Object-Oriented approach has been used for the implementation of this program for convenience. There are `Station`, `Map` and `Priority Queue` objects which provides attributes and methods for the corresponding matter. Also `Utils.py` file contains the utility functions such as formatted printing, reading the user input etc.

D. Input Data

Stations are given to the progam in *JSON*[1] format. Each subway line is a key and values are the list of stations in that line.

```
{"Line": [list_of_stations]}
```

Also, `Stations` are given in a list where the first element is the name and second is a tuple that contains the location

[1] JavaScript Object Notation

of the station. Example station can be seen below.

```
["ITU Ayazaga", [883, 335]]
```

E. Pseudo-code

Algorithm 1 A* Search

```
function GET_PATH)(start, destination)
    frontier ← Priority Queue
    frontier.put(node=start, cost=0)
    came_from ← dictionary
    cost_so_far ← dictionary
    cost_so_far.add(start)
    came_from[start] ← NULL
    cost_so_far[start] ← 0
    while frontier is not empty do
        current ← frontier.pop()
        if current is destination then
            break
        end if
        for c in current.get_neighbors() do
            _cost ← cost_so_far[current] + 10
            if c not in cost_so_far or
_cost < cost_so_far[c] then
                cost_so_far[c] ← _cost
                p ← _cost + heuristic(c, destination)
                frontier.put(node=c, cost=p)
                came_from[c] ← current
            end if
        end for

    end while
     return came_from
end function
```

III. Conclusion & Future Work

This paper present the implementation of *A* Search Algorithm* on a program that can be used on daily basis. The program is capable of returning the shortest path between two subway stations.

As mentioned earlier, there are numerous search algorithms that could have been used for this task. Experiment on these algorithms for similar problems in terms of performance, accuracy, efficient use of computing resources might be conducted in the future.

References

[1] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.