Optimal Path Finder.

May 7, 2024

SIRIPURAPU MADHU SUDHANA RAO (2022CSB1127) ,
MURRU SAI YASWANTH (2022MCB1271) ,
VANJIVAKA SAIRAM (2022MCB1363)

Instructor:
Dr. Anil Shukla

Teaching Assistant:
Sravanthi Chede

Abstract: This research project focuses on developing a Minimum Distance Route Finder application that utilizes the A* algorithm to find the shortest path between two locations on a map. The application allows users to input starting and destination points, efficiently calculating the shortest route while considering a 'less-hop' distance metric. The A* algorithm implementation incorporates factors such as node scores, heuristic functions, and open and closed sets, successfully achieving the goal of providing an efficient shortest route between cities.

---

1. Introduction

The A* algorithm, often pronounced as "A star," is a widely used pathfinding algorithm in computer science and artificial intelligence. It is specifically designed for finding the most efficient path between two nodes in a graph, such as a map or a game world. A* combines the best features of two other popular algorithms, Dijkstra's algorithm and Greedy Best-First Search, to provide an optimal and heuristic-driven solution.
A* is used in a wide range of applications, including robotics, video games, route planning, and even natural language processing. The key idea behind A* is to explore the graph by considering both the actual cost to reach a node from the starting point and an estimate of the cost to reach the destination from that node. This estimate is often referred to as the "heuristic" and is used to prioritize the nodes that are most likely to lead to the optimal path.
A* is known for its efficiency and ability to find the shortest path, provided that certain conditions are met, such as having an admissible heuristic (a heuristic that never overestimates the true cost) and a graph without negative edge weights. It uses a priority queue to keep track of the nodes to be explored, ensuring that nodes with lower estimated costs are explored first.
The A* algorithm has a robust and well-documented implementation, making it a valuable tool for solving real-world problems where efficient pathfinding is crucial. It has proven to be a fundamental algorithm in computer science, enabling applications to find optimal routes, plan movements, and make informed decisions, all by efficiently navigating the network of nodes in a graph.

2. Mechanism :

Cost to reach a node (g(n)): This equation represents the cost of the path from the start node to the current node.

Heuristic function (h(n)): This is an estimate of the cost to reach the goal from the current node. The choice of heuristic is critical and should be admissible, which means that it should never overestimate the true cost.

Total estimated cost (f(n)): This is the sum of the cost to reach the current node and the heuristic estimate to the goal, that is, f (n) = g (n) + h (n).

In this algorithm, the goal is to expand nodes with the lowest ‘f(n)‘ value. Here’s how the algorithm works in a nutshell:

1. Start with the initial node and set its ‘g(n)‘ value 0.
2. Create an open list (often implemented as a priority queue) to store nodes to be explored.
3. Put the initial node on the open list with ‘f(n)‘ equal to its ‘g(n)‘ plus the heuristic estimate ‘h(n)‘ to the goal.
4. While the open list is not empty:
   (a) Pop the node with the lowest $f(n)$ value from the open list.
   (b) If it is the goal node, you have found the path.
   (c) Expand the current node by considering its neighbors.
   (d) For each neighbor, calculate its $g(n)$ value (cost to reach the neighbor from the current node) and $h(n)$ value (heuristic estimate to the goal from the neighbor).
   (e) Calculate the neighbor’s $f(n)$ value and add it to the open list.
5. If the open list becomes empty and the goal has not been reached, there is no path to the goal.

The choice of heuristic and how you calculate ‘g(n)‘ depends on the specific problem and the representation of your graph or grid. The key to the A* algorithm’s efficiency is using a good heuristic that guides the search toward the goal while also considering the actual path cost.

3. Implementation Detail-

3.1. Figures

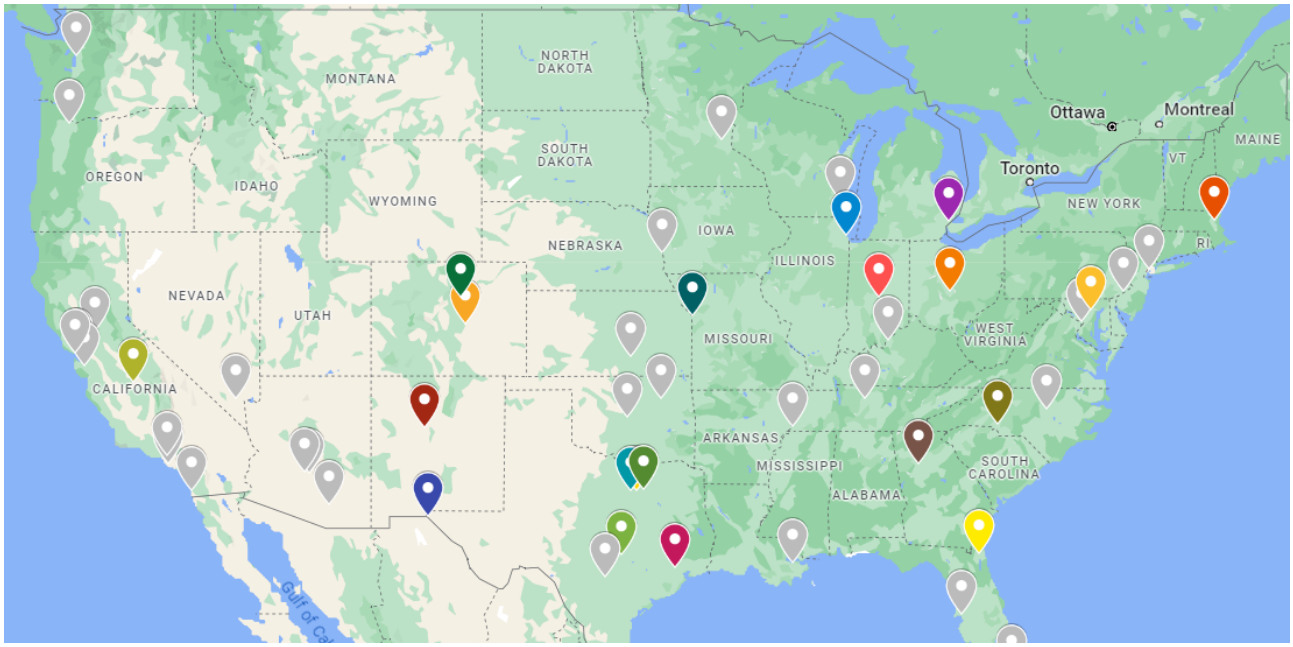

Figure 1: The real time cities that we have used in the input

Thanks to Google API, helped in getting real-time data.

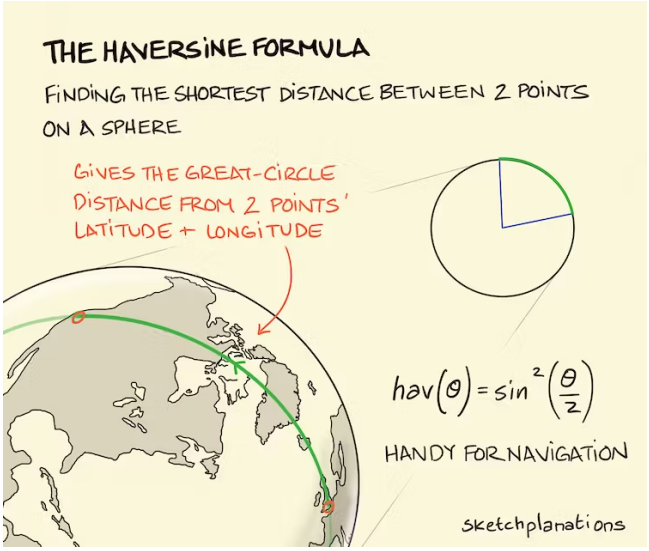

Figure 2: the formula used to calculate the haversine distance for calculating the heuristic function

3.2. Algorithms

Algorithm 1 Generate Distance Matrix in excel

Require: A range of cells in Excel representing a 2D matrix
Ensure: A distance matrix filled with values

1: **for** each cell in the range from B2 to AY51 **do**
2: Set $B54$ to the value of the cell's row index
3: Set $B55$ to the value of the cell's column index
4: Set the value of the current cell to the value of $B60$
5: **end for**

Algorithm 2 Get Distance using Bing Maps API

Require: A start location, a destination location, and an API key
Ensure: The driving distance between the start and destination in miles

1: Initialize three variables: $firstVal$, $secondVal$, and $lastVal$
2: Set $firstVal$ to "https://dev.virtualearth.net/REST/v1/Routes/DistanceMatrix?origins="
3: Set $secondVal$ to "&destinations="

```
\STATE Set $lastVal$ to "&travelMode=driving&o=xml&key="+ $key$ + "&distanceUnit=mi"
```

4: Create an object $objHTTP$ of type "MSXML2.ServerXMLHTTP"
5: Build the URL by concatenating $firstVal$, $start$, $secondVal$, $dest$, and $lastVal$
6: Open a GET request to the constructed URL using $objHTTP$
7: Set the request header to "User-Agent" for the request
8: Send an empty request body using $objHTTP$
9: Parse the XML response using FilterXML and extract the "TravelDistance" element
10: Round the travel distance to three decimal places and convert it to kilometers
11: Round the converted value to the nearest whole number (in kilometers)
12: **return** The rounded travel distance in miles

here is the pseudo code of astar algorithm that we have used in the project

Algorithm 3 A* Search Algorithm

Require: A graph $G(V, E)$ with a source node *start* and a goal node *end*
Ensure: The least cost path from *start* to *end*

Initialize
$open\_list = \{start\}$
$closed\_list = \{\}$
$g(start) = 0$
$h(start) = \text{heuristic\_function}(start, end)$
$f(start) = g(start) + h(start)$
while *open_list* is not empty **do**
  $m =$ Node on top of *open_list* with the least f
  if $m == end$ **then**
    return // Path found
  end if
  Remove m from *open_list*
  Add m to *closed_list*
  for each n in children(m) **do**
    if n is in *closed_list* **then**
      Continue
    end if
    $cost = g(m) + \text{distance}(m, n)$
    if n is in *open_list* and $cost < g(n)$ **then**
      Remove n from *open_list* as the new path is better
    end if
    if n is in *closed_list* and $cost < g(n)$ **then**
      Remove n from *closed_list*
    end if
    if n is not in *open_list* and n is not in *closed_list* **then**
      Add n to *open_list*
      $g(n) = cost$
      $h(n) = \text{heuristic\_function}(n, end)$
      $f(n) = g(n) + h(n)$
    end if
  end for
end while
return Failure // No path found

4. Conclusions

Utilizing this algorithm, we successfully determined the optimal route for cities by leveraging real-time road network data. This analysis allowed us to identify the optimal path (might not be the shortest). Also discovering more efficient alternatives through a matrix-based input of city locations.

5. Bibliography and citations

Acknowledgements

-> We have referenced the algorithm from an article published in internet archeive by Peter Hart, Nils Nilsson and Bertram Raphael who published a star algorithm[2]
-> Article published in the research gate which is used for **Determining similarity in histological images**

using graph theoretic description and matching methods for content based image retrieval in medical diagnostics helped us understanding a star algorithm better [3]
-> We have read the Haversine formula that helped in finding the Haversine distance for heuristic function [1]

References

[1] Peter Adams. The title of the work. *The name of the journal*, 4(2):201–213, 7 1993. An optional note.

[2] Nils Nilsson. MS Windows NT kernel a* algorithm: A formal basis for the heuristic determination of minimum cost paths, 2005.

[3] Harshita Sharma. Determining similarity in histological images using graph-theoretic description and matching methods for content-based image retrieval in medical diagnostics.

6. Appendix A

Manhattan Distance Heuristic: The Manhattan distance heuristic is commonly used in grid-based pathfinding, such as in games or robotics. It calculates the sum of the horizontal and vertical distances from the current node to the goal node, assuming only orthogonal movements are allowed.
Euclidean Distance Heuristic: The Euclidean distance heuristic is used in situations where diagonal movements are allowed. It calculates the straight-line (Euclidean) distance between the current node and the goal node.
Diagonal Distance Heuristic: This heuristic is used in grid-based environments where diagonal moves are allowed. It estimates the distance by considering both horizontal and vertical moves, as well as diagonal moves, using a combination of these distances.
Octile Distance Heuristic: The octile distance heuristic is a variation of the diagonal distance heuristic that assumes diagonal moves are more costly. It uses a weighted combination of horizontal, vertical, and diagonal distances, reflecting the higher cost of diagonal moves.

Custom Heuristics: In many cases, custom heuristics specific to the problem domain can be designed. These heuristics might take into account domain-specific knowledge and features to estimate the cost more accurately.

7. Appendix B

Haversine Distance Heuristic

The Haversine distance heuristic is commonly used in geographic or spatial navigation problems where the Earth's surface is represented as a sphere. It calculates the shortest distance between two points on the Earth's surface, given their latitude and longitude coordinates. This heuristic is particularly suitable for estimating distances when dealing with spherical or ellipsoidal spaces.

Advantages of Haversine Distance Heuristic:

1. **Geographic Accuracy:** Haversine distance accurately estimates distances on a spherical surface, making it ideal for geographic navigation problems, such as GPS route planning.
2. **Real-World Applications:** In logistics, transportation, and geospatial analysis, Haversine distance provides more realistic distance estimates compared to the Manhattan distance heuristic.
3. **Smooth Path Estimation:** Haversine distance yields smooth and continuous distance estimations on a spherical surface, resulting in more natural path planning.
4. **Consistency with Geographic Data:** When working with geographic databases and geospatial data, Haversine distance ensures consistency in distance calculations.
5. **Integration with GIS:** Geographic Information Systems benefit from the accurate distance measurements provided by the Haversine distance heuristic for spatial analysis and decision-making.

In summary, Haversine distance is superior to the Manhattan distance heuristic in scenarios involving geographic navigation or problems on a curved surface, such as Earth. Its accuracy and applicability make it an essential tool for real-world applications and geographic information systems.