Amir Azzam
Sònia Leal
Knowledge-Based Systems

Search Algorithms

Theoretical explanation of algorithms and heuristics

In this practice, we used two algorithms which are A* and CSP. These two algorithms differ in the way they look for the solution and the type of solution they look for.

A*

This algorithm searches for the best solution. It does that by adding an additional cost to traversing the tree, so the program doesn't keep on going down one path with disregard to the other existing paths.

The A* algorithm chooses a node depending on the value of f(x) = g(x) + h(x) where g is the cost to move from the origin city to a specific city, the one currently being explored, and h estimates the cost to the destination.

In our case, g(x) is equal to 0.4*distance and 0.6*duration. For this algorithm, we decided to give more weight to the duration as usually, individuals tend to prefer the fastest route rather than the one with less distance to cover. We calculate h(x) by adding 1000 times the number of successors without taking into consideration the first one.

We developed A* as an iterative algorithm. The other algorithm was done recursively.

In order to implement this algorithm, we created two lists, one that is called "future" and the other is called "visited." The future list holds the values or nodes that are explorable. On the other hand, the visited list stores the nodes that we have already visited.

We start by adding the origin node to the future list, and then we start looking for the solution. We have created a loop that iterates while the future list isn't empty; we sort the values in that list based on their weighted distance (0.4*distance + 0.6*duration). We then pop the node at the bottom of the list (the one with the lowest weighted distance). If this node is the destination then

we have found our solution. However, if it isn't then we need to get the successors of this node and calculate their weighted distance. If one of those successors was already in the future list then we only update its distance if it's less than the distance it originally had. Moreover, if this successor was already in the visited node, then we need to check if the distance of getting to this successor from this node is less than the cost of how we got there previously; if it's the case then we need to remove it from the visited list, update its cost and add it to the future list.

Finally, we add this successor to the future list and we continue the loop until we find the final answer.

CSP

This algorithm looks for a solution that satisfies a set of constraints. There could be multiple solutions that satisfy the given criteria but this algorithm only finds one possible path, if any. Therefore, the path found in this algorithm might not be the most optimal yet it certainly meets the criteria established.

Due to the behavior of this algorithm, once it finds a path that matches the constraints it returns that path as the final solution. It will not go back to explore other nodes to find a path with a lower distance.

These are the constraints that we established for this algorithm:

- Do not go through a city more than once.
- Make decisions based solely on the distance.

The algorithm works as follows. The first thing the algorithm does is to order the children (cities) by distance ascendingly. As previously mentioned, the algorithm only takes into consideration the distance one step at a time.

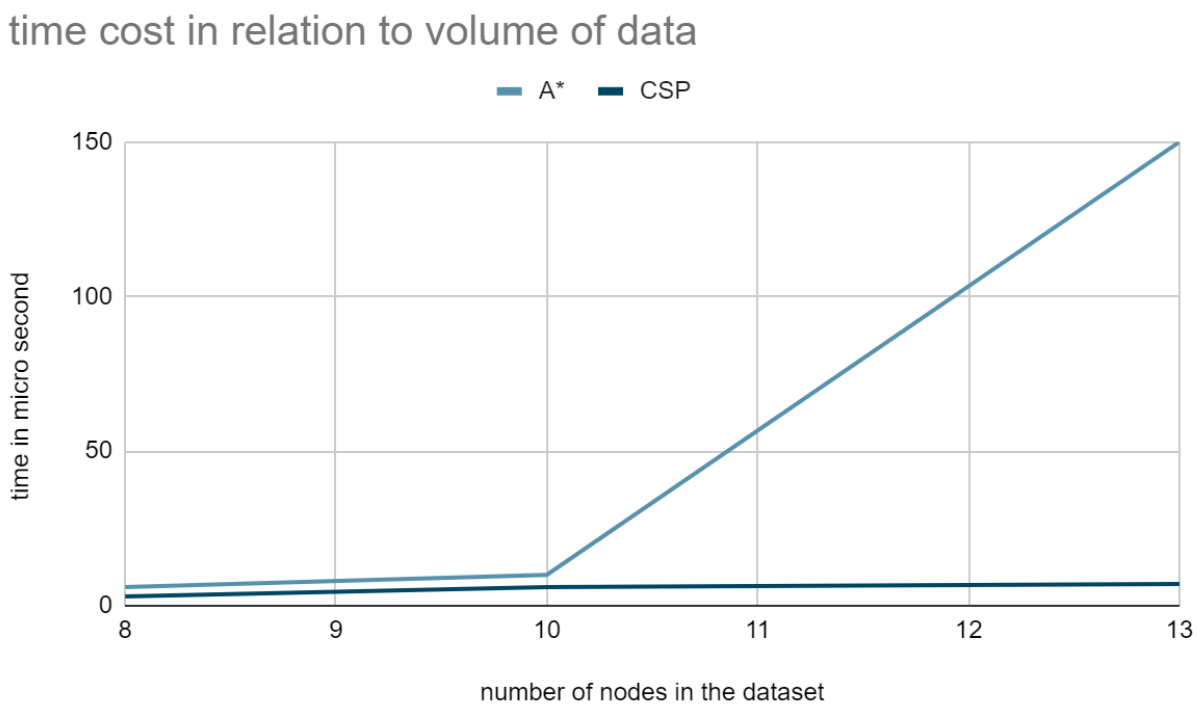
Taking the first child, as it has the lowest distance, checks if that city is the destination. If that is the case then the distance is accumulated, and the city is returned as the solution.

In case it has not found the destination yet, it checks if the city has been visited. If it has not been visited, it marks the city as visited and recursively calls the function again to explore its successors. When going upwards it will accumulate the distance and append the city to the path of cities so it would be possible to print the final solution.

Algorithm Comparison

Time costs in relation to the volume of data

Here is a graph to illustrate the time evolution in relation to the density of the data:



As shown in the graph above A*'s time cost increases exponentially with the increase of the data volume. However, in CSP there was little to no change that took the form of a linear increment in time in relation to the volume of data. These results make sense as in CSP you are only looking for a solution, while in A* you are looking for the best solution.

Justification of the results.

Spain_routes.json

Starting city: Salamanca
Destination: A Coruña

A*:

```
-------- time consumed in seconds is :  0.0001222000000211665  --------
total distance:  1403059 m
Salamanca
Madrid
Zaragoza
Pamplona
Bilbao
A Coruña
```

CSP:

```
-------- time consumed in seconds is :  6.680000001324515e-05  --------
total distance:  1403059 m
Salamanca
Madrid
Zaragoza
Pamplona
Bilbao
A Coruña
```

This example shows how both algorithms were able to find the same exact path. However, CSP was faster to obtain the results.

Starting city: Barcelona
Destination: Valencia

A*:

```
-------- time consumed in seconds is :  6.51999999945474e-05  --------
total distance:  349571 m
Barcelona
Valencia
```

CSP:

```
-------- time consumed in seconds is :  7.420000000024629e-05  --------
total distance:  349571 m
Barcelona
Valencia
```

This example shows how both algorithms found the same path. A* was a bit faster to obtain the solution in comparison to CSP.

2021_GI004_GIA04_Exercici_3.6_dades_routes_1.json
Starting city: Tremp
Destination: Mataró

A*:

```
-------- time consumed in seconds is :  0.00037990000001286717  --------
total distance:  211838 m
Tremp
Granollers
Mataró
```

CSP:

```
-------- time consumed in seconds is :  6.759999999417232e-05  --------
total distance:  302215 m
Tremp
Sort
Vic
Granollers
Mataró
```

This output illustrates how A* finds the most optimal path, considering time and distance. CSP took less time to execute than A*, but the path found has a higher distance.

Starting city: Palamós
Destination: Sort

A*:

```
-------- time consumed in seconds is :  0.0001841000000126769  --------
total distance:  289436 m
Palamós
Vic
Sort
```

CSP:

```
-------- time consumed in seconds is :  0.00010560000001191838  --------
total distance:  617096 m
Palamós
Blanes
Mataró
Granollers
Barcelona
L'Hospitalet de Llobregat
Vic
Reus
Tremp
Sort
```

Both algorithms took more or less the same time to calculate the route, but the fastest one to execute the algorithm was CSP. As we can see, A* provides the path with the fewest cities in between, considering distance and time. On the other hand, CSP is not as optimal but does provide a path with the parameters specified.

Justification of the language

We decided to use Python for this project because we are both more comfortable in using this language, and it is very easy to use.

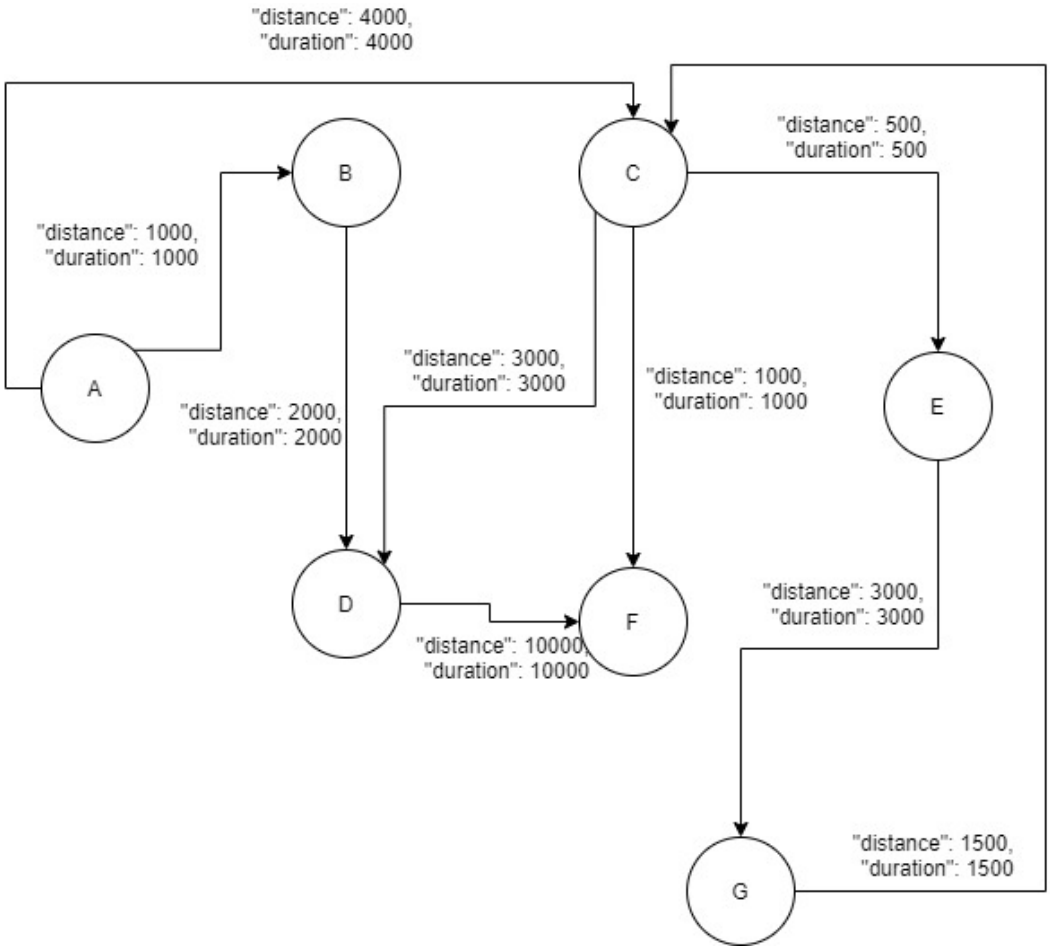
Results obtained and tests carried out

Some examples of the results obtained can be found in the section “Algorithm Comparison”. In this section other examples will be illustrated, indicating how we tested the results.

To verify that we were obtaining the expected path for both algorithms we carried several tests. With the debugging tool offered by PyCharm we were able to see how the path was updated in the algorithm. This was a very useful tool to make sure the algorithms worked properly.

Also, by looking at the datasets we were able to construct our own paths and figure out the most optimal path. With that we were able to compare it with the results obtained from both algorithms.

In addition to testing and verifying the results with the two given datasets, we decided to create our own dataset with a few edge cases to make sure everything worked smoothly. The dataset created follows the structure of the graph depicted below:



This is one of the routes we tried to verify the functioning of the algorithms:
A - F
A*:

```
-------- time consumed in seconds is :  8.449999999982083e-05  --------
total distance:  5000 m
A
C
F
```

CSP:

```
-------- time consumed in seconds is :  3.7300000002460365e-05  --------
total distance:  13000 m
A
B
D
F
```

Another method to verify the results obtained was to check if the path and the distance from the origin to the destination matched those provided to us:

- Barcelona - Seville
  - Total distance: 1,190,908m
  - Route: Barcelona, Valencia, Murcia, Málaga, Seville
- Zaragoza - Málaga
  - Total distance: 1,210,201m
  - Route: Zaragoza, Madrid, Salamanca, Seville, Málaga
- Valladolid – Zaragoza
  - Total distance: 650,671m
  - Route: Valladolid, Salamanca, Madrid, Zaragoza
- A Coruña – Madrid
  - Total distance: 795,685m
  - Route: A Coruña, León, Valladolid, Salamanca, Madrid

The routes shown above illustrate the minimum path. Thus, we were only able to be completely sure of the results with the A* algorithm. These are the routes and the results we obtained. As you can see it matches the outputs of the image above:

Barcelona - Seville

```
-------- time consumed in seconds is :  0.0002557000000251719  --------
total distance:  1190908 m
Barcelona
Valencia
Murcia
Málaga
Seville
```

Zaragoza - Málaga

```
-------- time consumed in seconds is :  0.00014469999996435945  --------
total distance:  1209148 m
Zaragoza
Madrid
Salamanca
Seville
Málaga
```

Valladolid - Zaragoza

```
-------- time consumed in seconds is :  0.00010259999999107094  --------
total distance:  650668 m
Valladolid
Salamanca
Madrid
Zaragoza
```

A Coruña - Madrid

```
-------- time consumed in seconds is :  0.0001173999999082298  --------
total distance:  795685 m
A Coruña
León
Valladolid
Salamanca
Madrid
```

Bibliography / References

U. (n.d.). A* Algorithm pseudocode. Retrieved from https://mat.uab.cat/~alseda/MasterOpt/AStar-Algorithm.pdf

A* Search Algorithm. (2021, May 27). Retrieved from https://www.geeksforgeeks.org/a-search-algorithm/

Python tutorial. (n.d.). Retrieved from https://bogotobogo.com/python/python_graph_data_structures.php