# **AC21007 – Algorithms and Artificial Intelligence**

# **Team members:**

Gerda Ugne Pupelyte **190013631**

Laura Naslenaite **190013642**

Rael Watt **190011819**

Sem Jasaitytė **190014736**Matthew Bain **190012264**

# **Word count:** <1649 words – 1500 words +/- 10%>

**AI Project Report:** Searching for the closest restaurant of a specific cuisine within the chosen area

Introduction

# The project focuses on identifying an eatery that specialises in a particular cuisine in the vicinity of the chosen location. It can be done by specifying the preferred type of cuisine (goal). Each eatery has a place on a map with a specific longitude and latitude. The map is used to judge the distance between the restaurants and is used to determine the available travelling directions.

Problem Definition

The problem is to determine the closest restaurant with a preferred cuisine in the chosen location.

* **States:** The state is determined by the cuisine the restaurant provides, its name, unique ID and location, specified by longitude and latitude. There are *n* restaurants, each with *x* cuisines, where *x* > 0. The location is processed using a coordinate system.
* **Initial State:** The agent starts in the restaurant *Hauz Khas Social*. The provided cuisines are *Continental, American, Asian, North Indian*; unique ID is 308322, and the location: 28.5542851 (Longitude), 77.1944706 (Latitude).
* **Actions:**
  + **Scan nearby area:** The eligibility for travelling to a restaurant is determined by the agent’s location. By using a coordinate system, the map of restaurants is examined, and restaurants within the range of the agent for a set distance are shown as available paths to travel to.
  + **Restaurant inspection:** The agent can inspect a restaurant’s cuisine within a set distance (150KM) from their current location by travelling to the chosen restaurant. After the travel, the agent’s location is updated to match the visited restaurant’s location.
* **Transition model:** the state is changed by the agent going from the current restaurant (i.e., node) to another one that is in the searching distance, and inspecting its cuisine. The state is then updated with the latter restaurant’s state. The agent can then continue travelling to another restaurant if the goal is not met.
* **Goal test:** Checks whether the current restaurant’s cuisine matches the preferred cuisine of the Agent (Mexican).
* **Path cost:** Each step costs the distance between the restaurants, so the path cost is the distance between the starting location and the final restaurant.

# 

Why it Matters

# The problem offers an efficient way to find a close restaurant with preferable cuisine. This is based on your original location, saving time by not having to explore every single option there is. An optimal solution is provided by saving the agent’s time - considering the path cost (distance) to the destination.

The Data

A proposed data set is Zomato Restaurants data [1]. To fit the scope of the project, only the first file of the JSON data was used, containing 76 restaurants.

The data was then filtered to fit the following criteria:

* Restaurant name
* Cuisines provided
* Unique ID
* Location, specified by Longitude and Latitude

Solutions and Expected Results

Before initializing searches, we proposed that uniform-cost search would be more effective than depth-limited due to the consideration of the path cost, whereas A\* would be the most efficient search of all. This is because it not only considers path cost but also an additional heuristic to estimate the closest goal.

# Uninformed searches

## Depth-limited search

The depth-limited search was carried out in 5 depth levels, and it found a goal cuisine at a restaurant *Qubitos - The Terrace Cafe*, with a path cost of **12.76 kilometres**, and runtime of **12.9 milliseconds**. Although the depth-limited search is incomplete, it did find a solution in our case. This was not an optimal search, as we discovered later there are closer restaurants with the goal states

For the complexity analysis, the time complexity is exponential, O(B^L), where B is the branching factor, and L is the proposed depth search. In this case, the resulting complexity is O(21^5). The space complexity is O(BL), resulting in O(21\*5), O(105), which is much better news.

Time complexity is a bigger issue while using this search strategy.

## Uniform-cost search

The uniform cost search found a goal state in a different restaurant, called *FLYP@MTV.* In comparison to a depth-limited search, the path cost dropped from **12.76 kilometres to 0.13 kilometres**, marking a vast improvement.

Uniform-cost search is both optimal and complete. We can see how this holds compared to the previous results of the depth-limited search. The runtime was cut to **11.9 milliseconds**.

The time complexity of the algorithm depends on the search parameters. In the worst case, it’s O(B^[1+C\*/e]), where B is the branching factor, C\* is the cost of the optimal solution, and e is the action cost. If all step costs are equal, it’s O(B^d+1). The space complexity is the same as time complexity.

The depth-limited search has a better space complexity in comparison.

# Informed searches

## Heuristics explained

For the informed search strategies, a heuristic function has to be considered first.

As the heuristic function utilizes the knowledge of a specific goal node that is outside the scope of the problem definition, an additional method is used to determine the goal node first which makes use of the restaurants’ list and applies a path cost function [2]

The heuristic function returns a value given by the Euclidean distance. Since the movement from one node to another could be preceded by taking any direction rather than restricting to 8 or 4 directions (which are supported by Diagonal and Manhattan distances respectively), specifically Euclidean distance was chosen for this reason to solve the problem.

## Greedy best-first search

The greedy best-first search reaches the same goal state as the Uniform-cost search, at restaurant *FLYP@MTV*. However, this time the path cost increases dramatically from **0.13 kilometres to 8.84 kilometres**. The running time, however, is almost halved at **5.9 milliseconds.**

The greedy best-first search is incomplete and non-optimal, as it uses only the heuristics to find the goal state. The path cost is still greater than depth-limited, and it’s apparent that greedy-best first search has the lowest running time.

The time complexity of Greedy best-first search is O(B^M), where B is the branching factor, and M is the maximum depth. In this case, it is O(21^M).

## Extra: A\* search

The A\* search provides identical results as does the uniform cost search, with the goal state ending up in the restaurant *FLYP@MTV*, and the path cost is **0.13 km**. However, the running time is slightly shorter - **10.9 milliseconds**, one millisecond less.

Since data is finite, A\* search is complete. As our heuristics function is admissible (i.e. it is a direct line distance), and consistent (*Fig.1*). The A\* search is therefore optimal.

|  |
| --- |
| h(n) = 0.0805  h(n’) = 0.0798  step cost = 1  h(n) ≤ h(n’) + step cost  0.0805 < 1.0798 |

*Fig.1. The proof of consistency*

The time complexity depends on the heuristic function. The Euclidean distance is calculated in constant time. However, A\* search still holds exponential time complexity. A much bigger problem is the space complexity, as the algorithm runs out of space before it runs out of time.

Custom interface

To better showcase and manipulate the data, we have developed an interface for this project. It shows each restaurant node and cuisine type and allows the user to expand a node to see its ID, location, and cuisines. Once the user selects the initial state and goal cuisine, the interface prints out the results found using all four search algorithms. The user is also allowed to close the result window and change the initial and goal states to recalculate the results.

How can we solve it in a better way (faster, more robust and efficient)?

Due to limitations that traditional search algorithms propose including extensive time and space complexities, incompleteness and non-optimality for finding the solution, other techniques could be considered to combat the constraints of algorithms used in a more efficient way capable of handling larger datasets.

One of them would be an advanced machine learning approach - reinforcement, or more specifically, *Q-learning*.

Q-learning is based on assigning reward values to goal and non-goal states across several repetitions of actions. With each repetition, an agent is gaining more knowledge about the states and their reward values. For example, a closer restaurant would grant a much higher reward than a distant restaurant. Then, from this data, the trained agent can find the most efficient way to the goal in the shortest time possible.

Besides, Deep Learning could be used to group similar users and calculate which restaurants they are more likely to choose, and then suggest these restaurants as alternatives to the ones closest to the user.

Conclusion

The depth limited search is the least optimal solution. It had the overall highest path cost and resulted in the longest-running time. Uniform search was just 1ms faster than the depth-limited search, yet the path cost was the lowest of the project. The greedy best-first search gave the quickest running time however the second-worst path cost of the searches. A\* resulted in the second-best running time overall and gave the same path cost as the uniform search.

The two most optimal search solutions were uniform cost and A\* as both resulted in the shortest path cost. Due to a constant heuristic, A\* search became a normal breadth-first search - uniform cost search precisely. This is why the results were identical to the uniform-cost search.

For the running time, A\* was 1ms quicker to gather a result. It may not be the quickest of the algorithms used, with a 5ms increase on greedy best-first search, but it gave the absolute closest restaurant in the fastest time.

To conclude, A\* is the best overall search methodology of the ones we used with a little trade-off for space and time complexities. It is the most optimal solution as no other algorithm can expand fewer nodes using the same heuristic function.

References

[1] Metha, S. 2019. *Zomato Restaurants Data*. [Accessed on 9 March 2021] [Online]. Available from: <https://www.kaggle.com/shrutimehta/zomato-restaurants-data>

[2] Patel, A. 2011. *Heuristics from Amit’s Thoughts on Pathfinding.* [Accessed on 20 March 2021] [Online]. Available from: <http://theory.stanford.edu/~amitp/GameProgramming/Heuristics.html>