Coursework: Robo-Uber

## Initial System

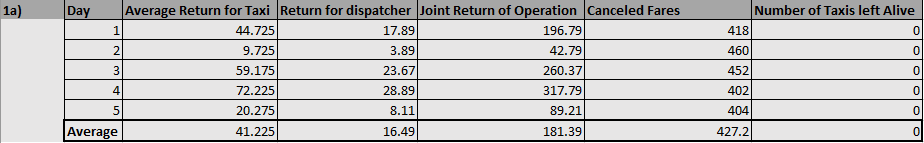
For the initial run I will be considering the impact of path planning. To set this up I ran the test over five days, where each day is 1440 in game minutes (1440 seconds), without any traffic on to determine the statistics below:

Average return to each taxi

Average return to dispatcher

Joint return of operation

Number of taxis left alive



With this table of results we can see that:

Average return to each taxi - £41.23

Average return to dispatcher - £16.49

Joint return of operation - £181.39

Number of taxis remaining - 0

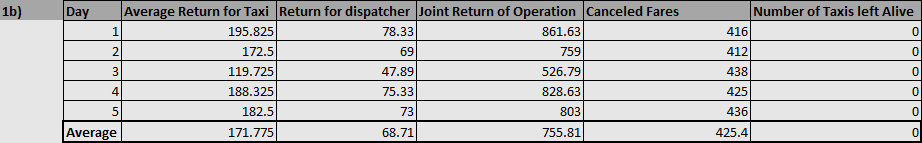
These results can help show that there is a problem with the current path planning algorithm as every minute that passes the taxis lose £1, and the taxis are also struggling to complete many fares at all, thus there being no taxis remaining at the end of the day.

This opens up a lot of opportunities to improve on the depth-first algorithm already built and instead replace it with an A\* search algorithm completed in the next part.

## Modified Path Planning Function

To better improve the path planning algorithm I implemented an A\* search algorithm due to the fact that we can use an informed search as we have all the information about the environment. This will show to be a much more efficient and optimum route plan.

Using the same settings and time span from the tests above I have tested the new algorithm below:



With this table of results we can see that:

Average return to each taxi - £171.78

Average return to dispatcher - £68.71

Joint return of operation - £755.81

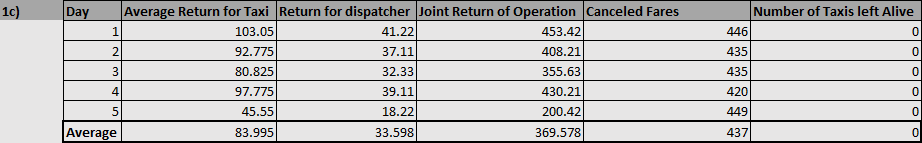
Number of taxis remaining - 0

These results clearly show that this new algorithm has worked perfectly for the taxis with a 316% increase for the average return to each taxi, the average return to dispatcher and the joint return of the operation and that using an A\* algorithm was a good fit for this problem.

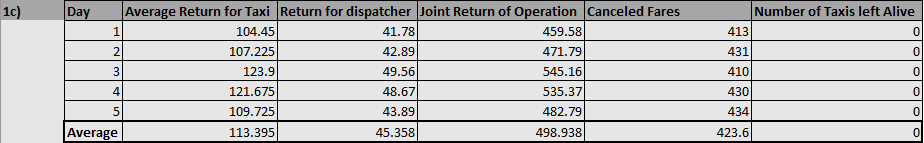
## Probabilistic Path Planner

To further test and develop this algorithm I constructed a probabilistic path planner which considered traffic estimates and could change its path depending on these. In order to ensure that the path planner would consider the traffic I worked out the probability of traffic being at the next node and the higher probability the higher I set a traffic multiplier to the estimated distance. This would in turn make the estimated distance to be longer if there was more traffic allowing the path planner to find the shortest route around the traffic instead if it needed.

I ran tests for this implementation, initially running the system without the traffic probability but with traffic now turned on:

****

I then ran the next set of tests with the traffic probability being used and with the traffic still turned on:



With these two sets of results I can compare the improvement by adding this traffic probability. It is clear that when adding traffic for both tests that the average return drops by about 50%, though when using the traffic probability over not using it, there is a jump of 35% in average return which shows a clear improvement in the path planning. Considering the number of cancelled fares, there was a decrease of 13 on average (about 3% decrease) which could show that the adapted path planner did help solve this problem slightly.

## Modified Allocate Fare Function

It was now the case of improving the dispatcher’s choices as well, which required changing its \_allocateFare function in order to maximise the total returns at the end of the day to both the taxis and the dispatcher.

The important characteristics to check consisted of:

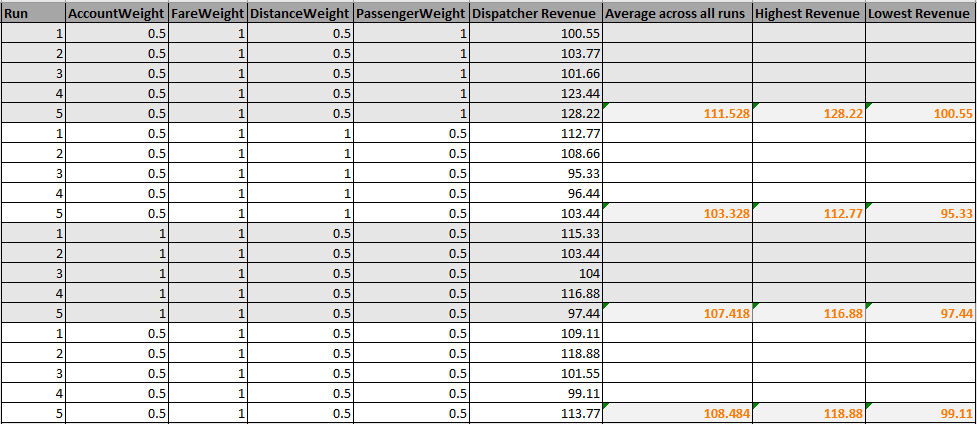
How quickly the taxi can service the fare.

How many bids must be in to allocate.

How to service allocation reasonably fairly.

The way this function was initially set up was to consider all the taxis that are currently bidding on the fare, and whichever taxi had the shortest distance to the fare would be given it. This caused a lot of problems including the fact that taxis could get starved out of fares just due to the fact that they are slightly further away then another taxi, causing the taxi to go bankrupt and have to retire early.

To improve upon this I created a heuristic that would consider each taxi's current account balance, the number of fares a taxi currently has, the distance from the taxi to the fare and whether or not they currently have any passengers. This then includes the consideration about how quickly a taxi can service fares and any number of bids can be going on this fare without change. With all of these variables combined I created a value to determine how severe a taxi needed this fare. I also included a weight system to allow changes to these variables to determine if they should be considered as highly as each other or if the taxis current account needed to be considered higher for example. I ran a range of tests to showcase different weights for each variable to determine what combination would best decide which taxi deserved the fare most reasonably.



\*Further tests and results are shown in the Results Table - Heuristic excel sheet in my GitHug Repository.

After some in depth research about the combination of weights I came to the conclusion that applying all of the variables at equal levels produced the best results.

With these changes it can be seen from the results that the total return at the end of the day has increased. Before implementing these changes the average joint return of operation was £673.30, but with the addition of the heuristic to determine which taxi should get the fare, the average return of operation was up to £1199.08, a 78% increase. From these changes it also improved the taxis’ lifespan before they retired, they would originally retire after 390 minutes due to running out of money, however they now manage to last as long as 520 minutes which is a 33% increase in lifespan.

## Modified Taxi’s Bidding Function

Coming back to maximising the taxis’ returns, modifying the \_bidOnFare function can achieve this. The problem being faced here is that if a taxi wants to bid on every single fare it is given it will start to lose money because it will accepts bids even if they are very far away and would actually make them lose money from the time they spent travelling and on the other hand if the taxi is too strict on its decisions then it won’t pick up as many fares and in doing so starts losing money as it isn’t completing any jobs.

The method I used to improve the taxis’ return here was similar to how the dispatcher improved it’s decision making, by creating a value to determine how likely the taxi will actually want to bid on the fare. To achieve this the CSP factors that I am considering consist of:

Taxis’ Current Financial State

Taxis’ Current Number of Fares

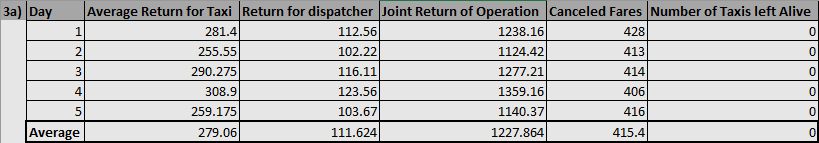
Taxis’ Current Distance to Destination

Taxis’ Time to Destination

Whether the Taxi can afford to reach the destination

With these variables I determine how severe the states are and create a normalised value for each. With these normalised values I add up all of the factors and multiply it by the traffic probability, considering the probabilistic case where there is traffic, to produce a Bidding Score which if it is high enough will allow the taxi to bid for the fare. This then allows the taxi to make better decisions based on these factors and won’t just take whatever fares it comes across but also if it severely needs to make some money for example it will start taking bids even if it is slightly further away, this way it is maximising its expected return.

Below shows the tests run with the same parameters as previous tests without traffic, run across multiple days:



From these results we can see that the average for joint return of the operation produced £1227.86 which is a 2.4% increase from before the previous changes. This method has also shown to slightly reduce the number of cancelled fares down to 415 from an average of 424.

## Commercial Deployment

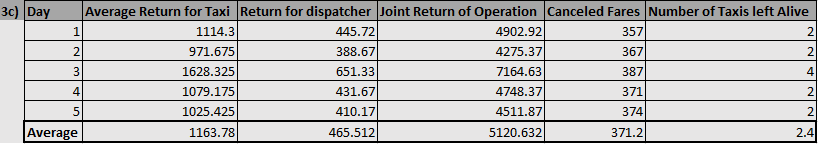
If this system could be implemented into commercial deployment I think that it could be very successful as it heavily considers fares that had been waiting a while which would make people a lot happier than having to wait around until eventually they cancel their fare. It also helps the taxi drivers due to the fact that all the taxis’ would work together in this system to maximise the overall returns which in turn allows each taxi to get more in returns individually. In a commercial deployment there would not need to be many major changes to the path planning as it would still be using an informed search which the A\* algorithm already is.

Though this would require further experimental testing with live trials to more accurately depict traffic probabilities in different areas as well as including traffic probabilities at different times which would help improve the system. In order to optimise the cost of fares further the testing can provide further insight into how much people are willing to spend before they cancel which could show whether the dispatcher needs to charge more or less. Finally further testing would be required with the heuristic weights in the allocate fare function to fully optimise the impact of each variable to the return profit.

## Minimising Cancelled Fares

In order to maximise the probability that each fare will be transported I modified both the \_costFare function for the dispatcher as well as the \_bidOnFare for the taxi. After running all of the tests before I realised that it was very unlikely that a taxi could make it the whole way through a day due to running out of money and needing to retire so the first change I made to \_costFare was to increase the cost of a fare by double the amount. This did not increase the number of cancelled fares but did however allow taxis to survive longer than a day, which meant that they could reach more fares and reduce the number of cancelled ones.

Secondly the changes to taxis \_bidOnFare, I created another CSP factor to take into account, which calculated what the longest wait time there is currently for each fare to create another variable called, WaitTimeSeverity, this would allow the taxi to realise that a fare has been waiting too long and before they cancel it, the taxi should bid on that one even if it may be further out of the way for the taxi. With these changes I ran further tests to prove that these would help reduce the number of cancelled fares.



Before these changes were made the number of cancelled fares averaged at 415, from these tests they dropped as low as 371 on average, a 11% decrease. Which coincides with my proposal and implementation to reduce the number of cancelled fares.

This bidding and dispatch system could be highly used in a commercial environment to help improve efficiency for fare satisfaction, taxis’ lifespan and the overall return from the taxis and dispatcher. This may require further research to optimise though should be heavily considered as the results throughout show that this system massively improves the quality and returns for each party.

## GitHub Repository

https://github.com/JohnnyVH2/RoboUber