AASMA Project

Optimization of Routes for Children's Transportation After School

Group 55

Sara Cruz
79410

Alexandre Guerra
86370

Tiago Mesquita
86520

ABSTRACT

In order to understand the power of an autonomous agent system applied to a real-world problem, we propose a study on children's transportation after school, in particular, the optimization of routes and corresponding fleet. We propose a single-agent system guided by q-learning to deliver suitable solutions for the problem in hand, and present results on instances of different sizes, in order to understand the behaviour of this approach.

1. INTRODUCTION

The optimization of children's transportation after school is a challenging problem, which brings benefits to the whole community. Not only does it help the families, minimizing the time spent commuting, but also the environment and ultimately the overall city traffic, by providing useful and better options for the children's return to their homes.

In literature, this problem is denoted the School Bus Routing Problem, and several studies on it can be found[1][2][3]. We introduce the question of having multiple schools, and allow the same bus to deliver children from different schools, if it is the best option. This problem may be seen as a variation of the Multi-Depot Vehicle Routing Problem (MDVRP), where schools are depots and each children's house is a delivery point. The goal is to find the optimal set of vehicles and corresponding routes, given their capacities, while minimizing the children's waiting time after classes, the overall travel time, the number of vehicles and the number of empty spots. It is a combinatorial optimization problem, with a multi-objective function. Determining its optimal solution is NP-hard[4] and the most common approaches involve metaheuristics such as Genetic Algorithms and Simulated Annealing[5].

We found it interesting to change scenarios and tackle the problem from a single-agent system viewpoint, applying a q-learning approach. With that in mind, our goal was to create a single-agent system where routes are generated aiming at delivering every child to its home, while minimizing the overall travel cost. We also pretend to study the impact of allowing a bus to be shared by different schools on the overall gains. We also studied the possibility of allowing a bus to be shared by different schools and evaluated its impact in comparison to having a separate bus for each school.

In this report, we begin by defining the problem, providing its relevance and requirements. We then propose a single-agent system guided by Q-learning to address the problem and come up with suitable solutions. A description of the system, including its properties such as the agent's definition, the connection to the real-world problem as well as the rewards scheme is also provided. Lastly we will enumerate the tests done, the values found that better suit our Q-learning and all the problems we ran into and solutions found to said problem.

2. PROBLEM

2.1 Definition

Given one or more schools with *m* total children, who need to get to their corresponding homes, the goal is to find the best (time efficiency) combination of routes for a bus (or set of buses) Each child is associated with a house or a block. The bus(es) must then deliver each child to that specific location minimizing the time between the end of classes and the arrival at home. The aim of the problem is to get every child to its place with a minimum amount of resources used by the schools. The same bus can pick up children from a school more than once and can also pick children from multiple schools. Meaning that it has to choose which children should be picked up first and whether or not it should stop at another school to pick up more children.

2.2 Relevance

The need to provide means of transportation for school children is a reality in many contexts. Clearly, not only is it important to satisfy that necessity, but also to do it in the most efficient manner.

Since the large majority of studies on the MDVRP and its variants converge to metaheuristics, we believe this study may contribute with a different way of addressing the problem, and hope to provide interesting solutions. Moreover, given that the proposed method is suitable for generalization, it could also be used to approximate solutions to other variants of the MDVRP.

The provided solutions could have big implications in terms of time savings for both children and service providers, as well as in fuel efficiency, given the optimized circuits. Therefore, saving money for the school or the entity regulating the buses and potentially even reducing overall traffic.

2.3 Requirements

This problem is aimed at a real world situation, so in order achieve a proper representation of it, some factors must be taken into consideration, namely:

I. The representation of the map should be as close as possible to reality, preferably a real map. This ensures proper timing and allows to draw direct parallels with reality;

II. The distribution of the children's addresses should follow a normal distribution and preferably surround the school. Also some clusters may form, similarly to reality. This should ensure the results are as broad as possible, and do not rely on certain anomalies or specificities of the model (this being said, it could also be an interesting feature to investigate);

III. All the buses depart from any school and must return (to any of the schools) when all children have been delivered. All movements performed in order to achieve this goal must be accounted for (this includes the buses returning with no children).

3. SINGLE-AGENT SYSTEM

3.1 Description

As it stands, this is clearly an optimization problem, since the variation of all parameters (capacity of buses, scheduling, routes) must culminate in an optimal result, considering a certain factor (time).

Given this definition, and the underlying theme of the project, we opted for using reinforcement learning to answer our problem. With that in mind we started by representing the bus as a **single-agent,** as is explained in **3.2 III**, that is tasked with the transportation of children, from any school to its respective address.

So as to lower the complexity for the reinforcement learning and reduce the number of possible states, we have decided to create groups of children with the same address, thinking of it more like a bus stop. When doing so we still allow our bus to pick up only a portion of this group if it so decides, meaning that even though the group has the same address each child is still processed individually. For this reason this simplification allows us to partially reduce the complexity and number of states of our problem while still ensuring an optimal or "good enough" solution.

3.2 Requirements Implementation

Throughout our implementation, we were faced with some challenges that lead to some refactoring regarding the original proposition :

I. As proposed, our implementation of the environment is a graph where each vertex corresponds to a child' s address or a school. We've also still considered the complete graph linking every possible node. This detail makes an adjacency matrix the perfect inner representation, since every memory position corresponds to an existing edge, holding the cost (duration) of traversing it. Not only that, in order to better conceptualize the graph, Python's package networkx[8] is used in order to print the obtained routes. Originally the data meant to build this graph was meant to be built using information retrieved through Google's API [6]. This functionality is present, in the sense that the program reads a text file, holding a properly formatted text file holding the data. However, Google has since enforced a stricter policy in the use of it's policy, making it impossible to get access. As such, we were forced to find an alternative. After some research, we've found a new API OpenRouteService [7] that is open source and has all the capabilities we required from google's alternative: distance and durations matrix.

II. The space restrictions we've proposed were successfully implemented. Following a set of school coordinates, and an offset limiting the maximum distance of each student to the schools, we then generate a point for each possible student, uniformly distributed within said radius. From this, we reduce the size of this data through clustering, with Kmeans. The found clusters represent the student groups with close addresses, corresponding the centroid to the address of said group. This not only massively simplifies the complexity of the problem, but also provides results that better relate to reality. This approach can be better visualized on [**Figure 1**] , where the crosses correspond to the centroids (addresses), the white dots to the schools and the scattered black dots each of the students.

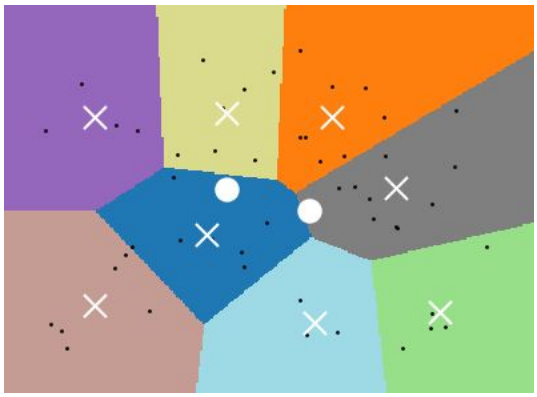


Figure 1 - Generated map from kmeans.

III. Our model has a single-agent, representing the bus. This agent can transport any number of groups of children, within its capacity, so in order to transport all of them, it may take several routes, each responsible for the transport of a different group. It is our understanding that these routes could be seen as either different consecutive passages from the same bus (the actual scenario behind the model), or simultaneous passages from different buses (an extrapolation that should meet the problem's requirements of a varying number of buses). Furthermore, the policy behind the agent, provided enough exploration to optimize every state, means that our agent will hold in it enough information to elaborate a route, given any combination of students inside the bus.

3.3 Sensors and Actuators

Implementation led to some refactors, especially in the agent state. The sensors reflect those changes:

I. The amount of children per address (ordered by id) - previously only the group id was known, so that groups were a direct abstraction of the student's id in the original problem, so that an agent would process a single student or a group interchangeably. This however would limit any group size to the bus capacity. Not only that but in very distributed group sizes, the bus would potentially run at half capacity (worst case scenario)

which would critically harm optimality. This refactor allows the agent to pick only part of the group, always allowing it to fill its capacity. How this new action is processed will be discussed at **3.5.2.**

II. **The location of the bus (vertex in the graph) -** this is maintained as it is already a very accurate representation, intrinsically linked with the world representation.

III. **The number of children left at each school per respective addresses -** also a big refactor of the original proposal. Originally only the number of children was known, mostly to direct the agent to the final state. This however revealed not to be specific enough, leading to considerable loss of information. For instance the agent could pick at two different times, the same number of children, but from different groups and, consequently, taking different routes. Both scenarios would lead to the same state, if the bus contained the same number of students, but would obviously have different results. The agent wouldn't be able to differentiate each scenario, harming optimality. By knowing the distribution of each group in every school, this situation is prevented.

Since each state is determined by the agent's location, the actions correspond to the edges of the complete graph. Thus, given the child groups inside the bus, the most appropriate following states would be such that allow the agent to deliver children to their addresses or pick more children from another school, if there is space available inside the bus. Consequently, even though the set of all actions is the set of edges, we may restrict the set of allowed actions at each state, helping the agent's decision to achieve its goal.

3.4 Rewards Scheme

Given our choice of implementation, the agent will dictate its actions based on rewards. These rewards were carefully thought of so as to make sure the agent reaches the desired solution and has the correct goal.

In our case, this goal is to minimize the time at which all children arrive at home and, in order to do so, we give our agent a reward, time related, for reaching a school when there are 0 children left in all schools and in the bus.

By the time of the proposal, no intermediate penalties or rewards seemed necessary, given our active "filtering" of available actions. In fact, even in the worst case, where the agent delivers a child at a time, the agent would still naturally progress to the end state as there were no possible waiting scenarios. As such, the back propagation of the final reward, through the Q functions should be enough to characterize the states.

Although, in theory, this reward scheme seemed to be the most accurate, in practice, it revealed to perform worse than a scheme with intermediate rewards, reflecting the cumulative travel time. Thus, each time the agent travels to another point of the graph, it receives as a reward the symmetric of the cumulated travel time, i.e., a negative value.. Finally, when the agent reaches the final state, a higher reward should be given, in order to make it value that achievement, and it is reflected in the inverse of the cumulative travel time multiplied by 10000, i.e., a positive value. The factor 10000 was chosen at first to avoid very small numbers, and was kept during the performed tests with apparent fitness.

3.5 Implementation

3.5.1 States

As is stated in before our states have the information of the location of the bus, the number of students at each school discretized by address and the number of students in the bus, also discretized by address. Meaning that for each state we know how many students are left to deliver for each address.

In our program the states are represented by tuples with the position on the first entry, followed by tuples for each school with the number of children by each address, and finally a similar tuple for the children inside the bus.

3.5.2 Actions

In terms of actions during the execution of our code there are three possible actions, drop, pick and travel. The drop action can be done when the agent has reached an address that is not a school, always after an action travel, this action makes the bus "lose" the passengers assigned to the address it is at, effectively delivering the students. The pick action can be done if the position of an agent is a school and if the bus still has available seats in it, if such conditions are met this action will put one chosen child in the bus. Lastly the action travel can be done after any pick or drop actions, in any position. This action will never move the bus to the position it is already at but will allow it to move to any school or to the address of any children that it has inside.

3.5.3 Q-learning

The Q-learning implementation, given the huge number of states in our problem, was initialized iteratively, meaning that our matrix is not initialized in the beginning of the program with zeros for every action in every state, but rather, each position of our matrix is only created when the agent needs to use it, or in other words the Q-value for a specific state and action will only exist and is only initialized if the agent performed that action in that specific state. This will allow us to simplify our initializations and also reduce the space used by our program, since only the Q-values used will exist. This was done because as can be seen in the previous subsections the number of states increases exponentially with the number of students, but we have realised that most of these states are not actually visited during the execution of the program and therefore doing this will greatly reduce the space used by our program.

4. TESTS

4.1 Definition

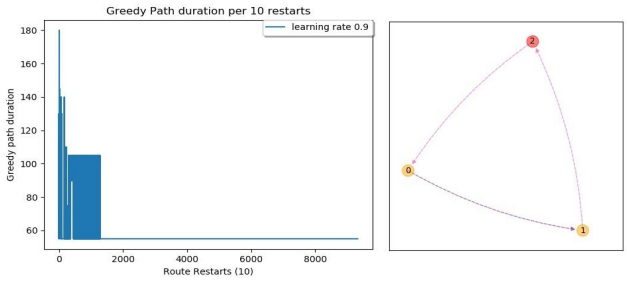
In order to evaluate the performance of our approach, we have defined a set of maps aimed at simulating real conditions, where we vary some factors. But before we can properly evaluate how the map may affect our approach, it is necessary to properly parametrize the Q-learning algorithm, namely: **learning rate** and **discount factor**. Both of these factors are crucial given the sheer size of the problem, namely in the needed propagation of rewards. The first will control the weight of new information, which given how only the last state in a route grants a reward, this information needs to be diffused quickly throughout the other states, so the bigger the **learning rate**, the faster this difusion should occur. However, the bot must be able to retain old information, in order to learn, so this value can't be very high. **Discount factor** will also be crucial given the particular rewards situation. Since value holds the importance of the future states, it is an effective tool for backpropagating the reward, speeding convergence. It requires

however some balancing as well, as higher propagation comes at the cost of less valorization for new rewards.

With this in mind, there must be a metric for evaluating the performance. Given the problem's goal to minimize the total duration for delivering all children, it is clear that this should be the metric used. This was then measured through a greedy traversal of Q, starting at the initial state, also greedily chosen, measuring duration. This is measured every N route iterations (the agent completes a full route), summing all traveled edges.

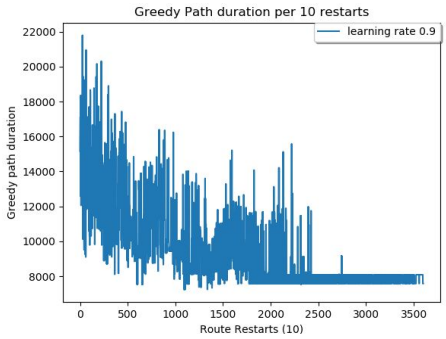
We then started by running our code with a test that could be correlated to reality of around 60 students and 8 addresses, excluding the schools. When doing so we noticed that the tests were not converging, even after the optimizations that had already been done, we increased the number of iterations considerably and there were still no improvements. We then tried some optimizations such as multithreading, which led to no significant improvements. Finally we allowed the agent to choose the first school after each restart, in an increasingly greedy fashion, while beginning doing it randomly, in order for the agent to find the optimal initial point, and consequently, a better route. This, in fact, helped the agent to earlier converge to better routes.

We eventually made some debugging and got some progress and by using a very simple problem, with a bus with 10 seats, 2 schools, 1 student address and 2 students in each school, meaning that all the student need to go to the same address, in this example we could see that it converged, to a route that takes 55 seconds, due to the simplicity of the problem we can also conclude that this is the optimal solution where first the bus picks up all the students, delivers them to their address and then returns to the starting school. The path durations during the execution and the optimal route can be seen in the following images.



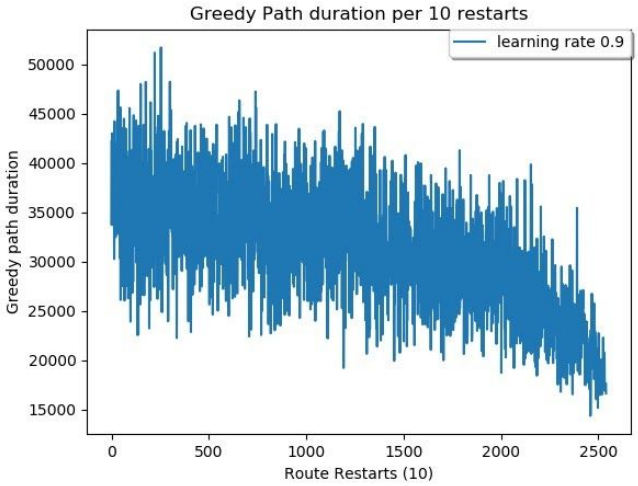
One thing to notice on the graph that shows the duration of the path is the dense rectangle that happens in the from around route restart 500 to 1500 this shows that our agent is converging to two different routes and is switching between them, which will be a problem when doing more complex scenarios.

We understood that this was happening because of the way we give our reward, by giving it a negative reward when it arrives at the final state. As shown in the previous graph and by studying the Q-learning algorithm we know that the Q-function should eventually converge into the optimal solution but, unfortunately this will not happen in a reasonable amount of time when using a more complex problem as can be seen in the following graph where we used a slightly more complex problem maintaining the number of students the bus can take and the number of schools, what changes is the number of students that is now 10 and the number of student addresses, changed to 4.



We will note that the two solutions reached are always sub optimal and also that it changes between them because of how similar the reward for those routes are. We concluded that by using a negative reward, the decrease of the Q-values when rerunning the better route is enough for the Greedy exploration to opt for the second best route. Unfortunately we could not find any solution to fix this problem.

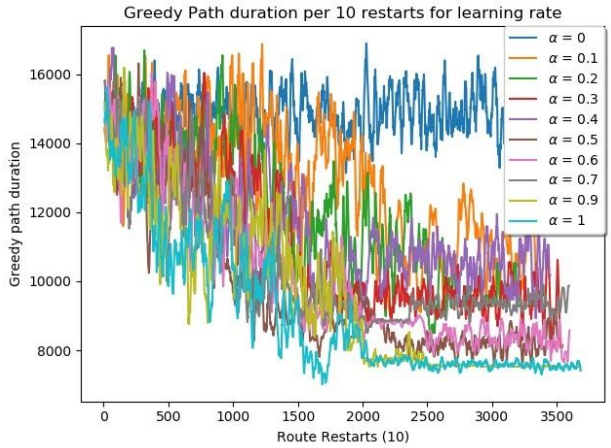
Lastly we attempted to run a harder problem, still maintaining the number of students the bus can take and the number of schools but now with 6 different student addresses and 30 students, due to the extra complexity we also doubled the number of route restarts our agent would make. We obtained the following graph:



In it we notice that, as we expected that during the execution our agent learns with each iteration and keeps diminishing the path duration, even though we doubled the number of interactions the agent never converges to a specific route meaning that due to the complexity of the problem we would still need to increase the number of interactions.

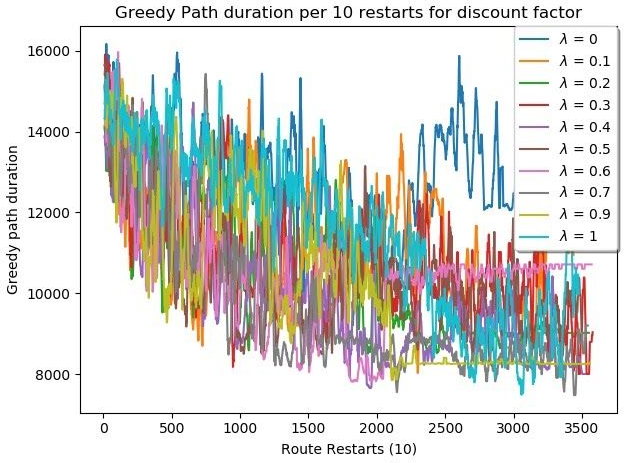
In order to make sure that the comparisons we did previously were as fair as possible we made sure that the values of the Learning Rate, Discount factor and Epsilon were the same in every test. In order to know which values to use for each of these parameters we ran sets of tests where we only changed one of the parameters and evaluated the results. To do so we've created different graphs where we plotted the time series trends of the greedy path duration, for each parameter value, by the number of route restarts.

We started by discovering the Learning Rates and obtained the following plot:



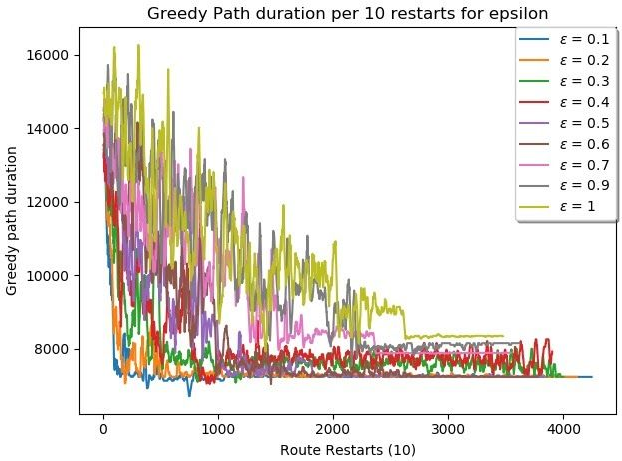
By evaluating it we concluded that the best value of Learning Rate for our project is 0.9, closely followed by 1 as expected meaning that our agent does not value learning that much, prefering to base himself in the information he just got on that iteration and ignoring those he did before. We also note that when the alpha is 0 the agent performs the worst, something that we also expected because for this value our agent won't learn a thing when doing exploration, meaning that its actions are based solely in exploitation and will therefore be always random.

Next we chose to plot the graph changing the Discount Factor, in it we already set the Learning Rate of 0.9 as we knew this was the value that would bring the best results.



In it we can note that the best value for the Discount Factor is 0.9, which leads us to conclude that our agent gives great importance to future rewards. This is also expected, has our agent its trying to find the best possible route and the Q-values are associated from the final and propagated until the initial states it is understandable that our agent will greatly benefit from knowing what are the expected rewards for a specific gain, meaning, in our case, what is the expected time to reach the final state from the state he is evaluating which is something of great importance to our agent and why giving the Discount Factor a value of 0 (meaning that it will only considering current rewards) is the worst optio. We can also conclude that by giving it a valio of one our Q-function does not converge, has can be seen in the image this can be understanded because by giving our formula a Discount Factor smaller than one we are allowing our agent to make a decision and, therefore make an infinite sum finite.

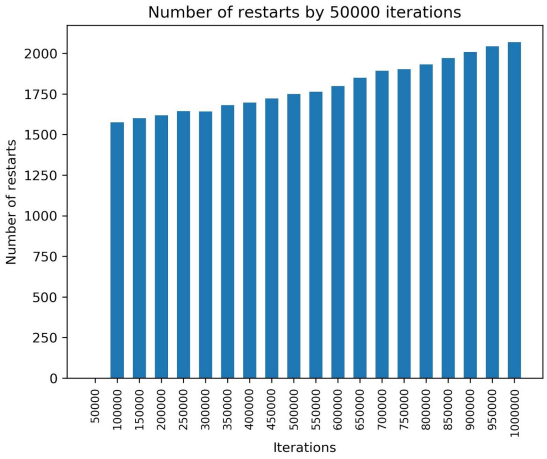
Lastly we chose a value for the Epsilon, using a graph that already had both the Discount Factor and Learning Rate set to 0.9



When evaluating said plot we had some unexpected conclusions. Mainly that by diminishing the epsilon our agent would converge to a faster and to a closer to optimal solution, this goes against our expectations and means that our agent does not benefit from exploring. Having an agent that benefits from exploiting and being greedy, has stated before this came as a surprise to our group and the only explanation we found was that due to the sheer amount of states. We concluded that when our agent has a big epsilon he tends to go into multiple bad solutions and does not have a value of exploitation great enough to try to improve on these routes. In contrast when he is more greedy he tends to find a route that reaches our final state and then through the execution he has the ability to follow this route more in to the end and the make small improvements, eventually leading in to a better position that is considered to be a local minimum and a not our optimal solution.

4.2 Advanced Decision Making

During our project we can note that our agent utilizes reinforcement learning to adapt and learn about its environment, the following bar chart shows us that our agent reaches the final position more often with the elapse iterations.



This will mean that during the execution our agent starts learning what are the best actions to do in each state in order to make it reach the final state. This means that, according to our plans in the final steps of the execution our agent will no longer do exploration because it already knows what are the best actions to do for each position and will now always follow the same trajectory, the one where he knows the reward will be the greatest.

During the execution of our algorithm our agent will learn by doing exploration meaning that even if it already knows what is the best action to do, it may chose another random action, according to its epsilon, and will, because of this, explore new actions and learn whether or not the random action done is or not a decent action.

5. CONCLUSION

After doing all our tests and evaluating their outcomes we can conclude that our implementation of the Q-learning was sub optimal. Through the study of different papers we could see that Q-learning is often used in problems similar to this, so the algorithm is not to blame. We then evaluated our implementation of sead algorithm and concluded that it can be greatly improved, mainly the states. We noticed that our implementation created so many states that by making our agent more greedy he would give us a better result even though many of our states would never be reached. This was explained by the fact that even though less states where explored our agent focused itself on a smaller area of the states and would then, by still making a small amount of explorative decisions make little improvements in the original route. This means that even though there was a very small chance of reaching the optimal solution our agent would reach a local minimum that will normally still be better than the solution found by the explorative agent, due to the massive amount of states in our problem. Even though our agent could not find the best solution for our problem we still found the values for the Q-learning: Learning Rate, Discount factor and Epsilon; that got us the best routes, these values were, respectively 0.9, 0.9, 0.1. Where we can conclude, like was seen before, that our agent does not give much importance to the past and that the rewards from future actions are important, lastly the Discount factor leads us to the conclusion that we will greatly benefit from a more greedy agent due to the same reasons expressed before.

6. REFERENCES

[1] Dimitris Bertsimas, Arthur Delarue, Sebastien Martin, Optimizing schools' start time and bus routes, Proceedings of the National Academy of Sciences Mar 2019, 116 (13) 5943-5948; DOI:10.1073/pnas.1811462116

[2] M. Elgarej, K. Mansouri and M. Youssfi, "Route optimization for school bus scheduling problem based on a distributed ant colony system algorithm," 2017 Intelligent Systems and Computer Vision (ISCV), Fez, 2017, pp. 1-8.

[3] Junhyuk Park, Byung-In Kim, The school bus routing problem: A review, European Journal of Operational Research, Volume 202, Issue 2, 2010, Pages 311-319, ISSN 0377-2217, https://doi.org/10.1016/j.ejor.2009.05.017.

[4] Liong, C.-Y & Wan, I. & Omar, Khairuddin. (2008). Vehicle routing problem: Models and solutions. Journal of Quality Measurement and Analysis. 4. 205-218.

[5] Thibaut Vidal, Teodor Gabriel Crainic, Michel Gendreau, Christian Prins, A unified solution framework for multi-attribute vehicle routing problems, European Journal of Operational Research, Volume 234, Issue 3, 2014, Pages 658-673, ISSN 0377-2217, https://doi.org/10.1016/j.ejor.2013.09.045.

[6] Google Distance Matrix API: https://developers.google.com/maps/documentation/distance-matrix/start

[7] OpenRouteService Distance Matrix API: https://maps.openrouteservice.org/

[8] NetworkX 2.4: https://networkx.github.io/documentation/stable/index.html