M.O.T.A. - Metropolitan Optimization: Two Approaches

Pedro Esteves
83541
pedro.f.esteves@tecnico.ulisboa.pt

Artur Fortunato
86388
artur.fortunato@tecnico.ulisboa.pt

João Coelho
86448
joao.vares.coelho@tecnico.ulisboa.pt

ABSTRACT

In this project, we present the architecture of our implementation for the agent system that aimed to optimize Lisbon's metropolitan system as well as a discussion of results that emerge from reactive and deliberative behaviors, comparing them against a baseline. We achieved a reduction in the number of daily trains of 30.5% using the reactive approach compared to the baseline, which had a very high number of trains (making this a more friendly approach for the service provider). Regarding the deliberative approach, we managed to reduce the number of trains by 32.1%, when compared with the baseline. With this solution, we managed to maintain the average occupancy of the train relatively low (only increased from 15% to 25%), without a substantial increase of the passengers waiting time.

Keywords: Reactive agent, deliberative agents, subway system, AAMAS.

1. INTRODUCTION

This project aims to study whether a fixed schedule, as currently used by Lisbon's subway system, can be improved by an agent system approach. Our implementation is composed of four agents that are responsible for each one of the subway lines, and one orchestrator agent that manages the lines. We used real data from October 2018, provided by Professor Rui Henriques, which means that even with some simplifications in the model we expect the results to be close to real-life scenarios. The objectives are to minimize the waiting time of passengers, bearing in mind that sending too many trains is not good for the service provider (as this would significantly increase his costs), and that trains should be neither very full nor very empty. Hence, our three main metrics for evaluation are the **average waiting time**, **percentage of occupancy of trains** and **number of trains launched during the day.**

In chapter 2, we describe the system modelation. In chapter 3, we present the metrics used for data analysis. Chapter 4 further describes both types of agents (Line and Orchestrator). Chapter 5 briefly explains the usage of the Graphical User Interface (GUI) used to aid visualization of this project. Chapter 6 deeply describes the multiple behaviors implemented (the baseline for comparison and the reactive and deliberative). In chapter 7, we present and discuss the results of this project, comparing the metrics of the multiple approaches. We present the study's conclusions in Chapter 8, concluding with chapter 9, presenting some future work.

2. SUBWAY MODELATION

2.1 Stations

We modeled every station except for Arroios since we had no data for it. The stations are the nodes of a graph. The edges have different lengths, corresponding to the distance between stations, which is not exactly accurate, but an approximated value. The length is measured in *spaces*. The train's velocity is therefore measured in spaces per tick. Train velocity was kept constant for the whole project, for simplicity.

2.2 Trains

Each train is a set of carriages. Carriages were modeled after the ML99 type carriage, each having 185 total capacity (44 seating + 144 standing). There was no distinction made between which carriage a passenger would enter, passengers just keep joining the train until it gets full. For simplicity sake we modeled all the trains with the same capacity.

2.3 Changing Lines

Besides all lines in the subway being connected to each other, making this graph an SCC, (Strongly Connected Component), Lisbon Metropolitan fulfills a harder restriction, all lines are directly connected to each other. We take advantage of this detail to simplify situations where passengers need to exchange lines, as we don't search for the shortest path between two stations from different lines, we take the passenger to the station where both origin and final stations' lines cross, and switch lines there. Despite optimality not being guaranteed by this approach, it is a good heuristic since our goal is broader than line change optimization.

3. DATA ANALYSIS

In our implementation, one system time unit, designated as TIK, corresponds to one real-life minute. The need for data analysis has two main reasons: In each TIK, we need to know how many passengers will enter a given station [3.1], and what will be their final station [3.2].

3.1 Number of Passengers

The first component we extracted from the data were tables in the form: [<*hour*>, <*number_of_people*>], one for each station. This gives us the number of people in the station, at a given hour, in 15-minute intervals from 6h15 to 00h45. Since we have 31 days, we have 31 examples for each 15-minute timestamp. After

normalization, we computed the barycenter for the 31 days using a *SoftDTW* approach. Given that we are querying the model at each minute, instead of returning barycenter[h:m]/15 for each query, our model returns a random integer between: {barycenter[h:m] - stdev[h:m], barycenter[h:m] + stdev[h:m]}, where stdev[h:m] is the standard deviation of the values of a given hour. In each TIK we take a value from this uniform distribution, divide it by 15 and round it to the closest unit. This ensures fluctuation of the results, which implies a more realistic model.

3.2 Final Station

The second extracted component were tables in the form: [<*hour*>, <*final_station*>, <*number_of_people*>], one for each (initial) station. For the same timestamps of the previous model, this gives us the sum of all people that, from the initial station, travelled to <final_station>, at <hour>. From this we can get a probability distribution for each hour and station, by normalizing the <number_of_people> values.

4. AGENT SYSTEM ARCHITECTURE

4.1 Line Agents

There are four line agents. The only difference between them is the set of stations they have access to. These agents have sensors that let them perceive persons joining stations, percept a new day, a sensor to perceive the trains' velocity, as well as their occupancy and a sensor that allows them to receive information from the orchestrator. This information includes whether to create a new train and whether the current train's velocity should be changed. Since we kept the velocities constant, this second information isn't being used on our approaches. For actuators, the agents can move trains, add new trains, and send information to the orchestrator.

4.2 Orchestrator Agent

The orchestrator's sensors and actuators are simply receiving the line agent's information and sending them back information regarding what to do, respectively. The way the orchestrator determines what the line agents will do varies with its behavior. We implemented a reactive [6.2] and deliberative [6.3] behavior. Also, we provided the system with a baseline [6.1] which is a simplification of the real subway schedule.

5. GRAPHICAL INTERFACE

In order to ease the visualization of the execution of the project, we modeled the lines, trains and stations in a GUI. For this, we used PyGame 1.9.6.

In the GUI we have the lines represented, identified by their colors, and every station, identified by their names. Stations also have the number of passengers waiting in each way next to their name. Trains are the rectangles travelling on top of lines, with an id. The id identifies the number of trains that have been launched, not the number of trains that exist. Also, we displayed the day, time, type of behavior and the global average waiting time. The stations present in more than one line, e.g: "Saldanha" have two tuples with the initial letter of each line preceding it.

6. AGENT BEHAVIOUR

As previously stated, we implemented reactive and deliberative behavior. In the proposal, we mentioned that we would try to model the system as a Reinforcement Learning problem. However, due to state space explosion and lack of time, we were unable to do so.

6.1 Baseline

The baseline is a simple schedule that sends new trains every X minutes, from 06h15 to 00h00. This ending at 00h00 is one of the simplifications we made, since some stations close before others, we chose to end at the time the first one closes. The baseline in our experiments sends a train each 8 minutes in each way. All trains have 6 carriages.

6.2 Reactive Behavior

When behaving reactively, in each TIK the orchestrator will analyze the information it received from the Line agents and use a couple of heuristics in its deliberation. For each line, a new train will be sent if the average number of passengers per station (avg_{pps}) is greater than a fixed value a (which we set to 20) and the occupancy ratio(o_{ratio}) of the carriages is greater than a fixed value b (which we set to 0.2). This will ensure that more trains will be sent when it really needs to, while controlling the capacity of the trains. There's a particular scenario that needs to be treated differently: when there are no trains circulating, a new one will be sent if the average number of passengers is greater than the fixed value of c (which we set of 15 because since there are no trains, o_{ratio} can be considered 0 which will fail $o_{ratio} > b$.

This is a slow start strategy since there is not much information in the beginning. Therefore, a large average waiting time is expected in the first hour of the day. We can overcome this issue by lowering the value of c or running a predefined schedule in the first hours of the day, transitioning to the reactive behavior.

Launching train condition: $avg_{ppt} > a$ **and** $o_{ratio} > b$

Launching train condition(particular scenario): $avg_{pps} > c$

Lower values of a and b will result in a more passenger-friendly model.

6.3 Deliberative

When behaving deliberatively, the orchestrator defines a schedule for each day based on the occupancy ratio (o_{ratio}), the average number of passengers per station (avg_{pps}) and the frequency of trains of the previous day ($freq_{prev}$). After running the first day with a base schedule, the data collected for the whole day will be analyzed hourly. For each line, in each way, in hourly time intervals, the train frequency for that hour in the previous day will be either incremented or decremented, based on the metrics above. The implementation is based on a single planner, which input is the data and the output is a schedule. The conditions for updating the frequency are as follows:

$$if\ o_{ratio-prev}[hour] > a\ and\ avg_{pps-prev} > b\ then:$$
$$freq_{new}[hour] = freq_{prev}[hour]++$$
$$else:$$
$$freq_{new}[hour] = freq_{prev}[hour]--$$

The next day, this new hourly schedule will be used, data will be collected, and the process repeats.

There are some things to consider here. First, the lower the value of a and b, the more passenger-friendly the model will be. Second, we could add a condition for decrementation, if the values fall below some threshold. By doing so, we would have a conditional branch that does not change the previous frequency. With this we would give the model an opportunity to converge into a final schedule. However, since no two days are alike, we decided not to do so. Finally, it is easier to see this model behaving if the model starts off with a *bad* schedule. This way, after a couple of days we can clearly see the behaviour changing, trying to balance the benefits for both the passengers and the service providers. We will show this in the next section by starting off this method with a schedule that launches a new train every 16 minutes [7.3].

7. RESULTS

7.1 Baseline

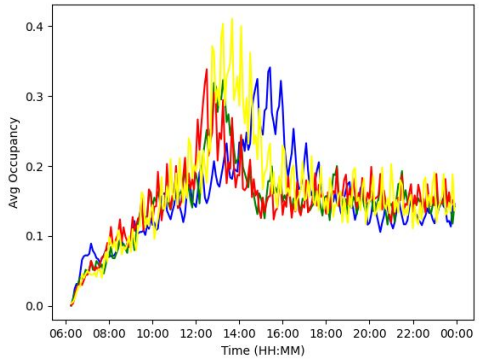

Figure 1: Baseline average occupancy for each line (134 trains daily trains for each line, each way).

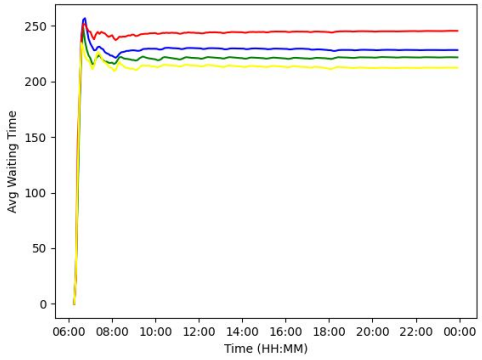

Figure 2: Baseline average waiting time in seconds for each line (134 trains daily trains for each line, each way).

Regarding the occupancy, we can already see that , for most of the day, we have an average close to 15% (except for the morning before rush hour, where there are very few passengers and at lunch time, where we can observe a larger occupancy).

There is a reason why the rush hours aren't completely evident, as it was expected. As both ways of each line are represented together, as well as all trains running at that timeframe - this lowers large values of occupancy on rush hours in specific stations - average occupancy is significantly lowered by trains who are starting their course and have a very low occupancy.

We can see in Figure 2, the waiting time is around 225 seconds (3 minutes and 45 seconds). Despite the fact that the average waiting time is low, the number of trains was around 134 for each line. This together with the low occupancy of the carriages lead to a very poor model for the service provider. We can see that there is a spike at the 30% on the average occupancy, and this is due to the fact that, when a new empty train is launched, the average occupation significantly decreases (as there are very few trains running at that time).

7.2 Reactive

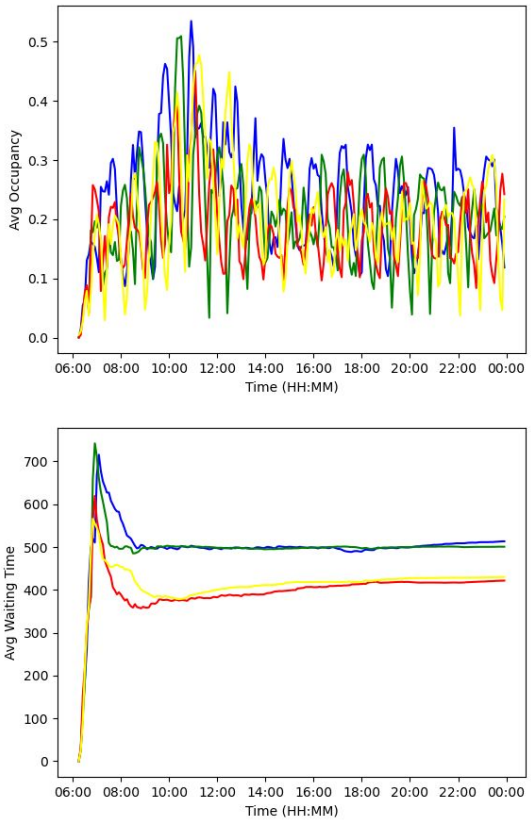

Figure 3: Reactive average occupancy (top) and waiting time in seconds (bottom) for each line (occupancy threshold of 20%, 93 trains each line, each way

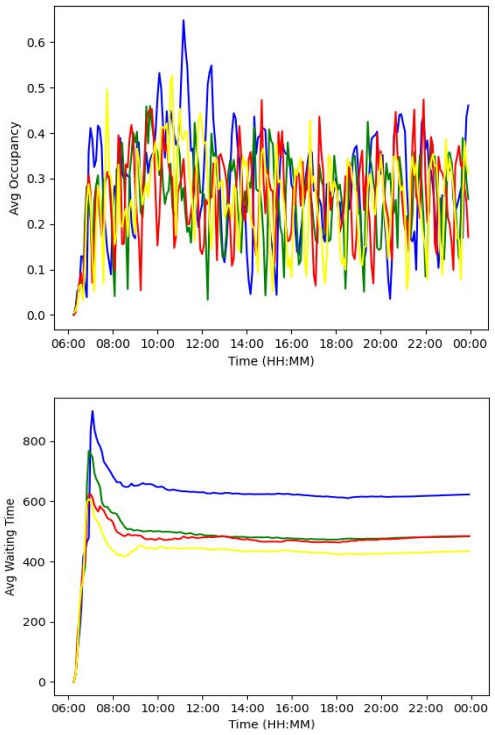

Figure 4: Reactive average occupancy (top) and waiting time in seconds (bottom) for each line (occupancy threshold of 50%, 74 trains each line, each way).

From a direct comparison against the baseline, we can see that the reactive behavior is worse when it comes to controlling the timespan between trains. This is inherent to its behaviour, as it will launch a new train when the average occupancy threshold is met. So, if the occupancy is surpassed very quickly, the next train will be launched very close to the other. There won't be many passengers for this new train to pick, leading to a low occupancy, a higher time for the next one to spawn and an increase in the cost of the service for the service provider.

We also saw that the average passenger threshold is not taking a huge part in deciding the train launch, since as soon as the capacity is met the average number of passengers is already very high. However, we can see that this approach manages the train capacity better than the baseline, since we are only sending new trains when a given average capacity is reached. Nonetheless, as exemplified by Fig.3 and Fig.5, increasing the occupancy threshold won't linearly change the average capacity. This means that, in this case, an average capacity of 50% is achieved only a small number of TIKs later than the 20% one. Of course, the higher the occupancy threshold, the lower the number of trains will be.

This model is now better for the service provider, as the capacity is better used and less trains are being sent. Unfortunately for the passengers, they will have a higher waiting time. We can argue that the waiting time is higher for the lines with more stations (as observed in Fig. 3 and Fig 4) because, as mentioned, the timespan between trains is not handled well, and so the trains have more stations to travel until the next train is launched, resulting in a higher waiting time.

7.3 Deliberative

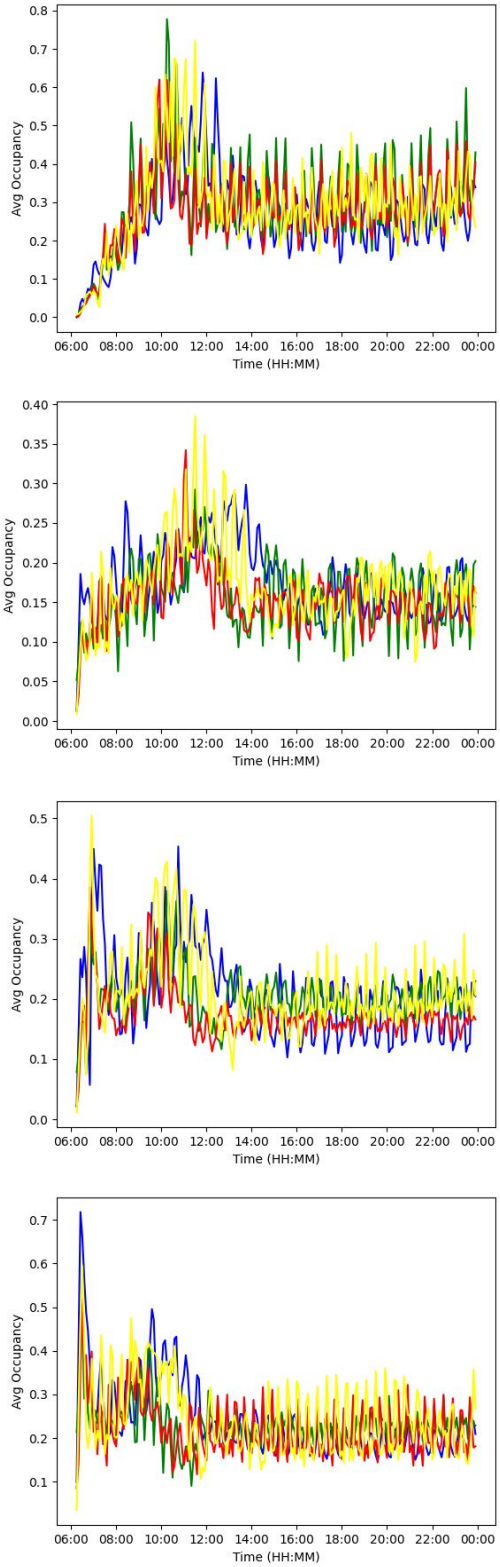

Figure 5: Deliberative average occupancy for each line (from top to bottom - day 1: average 72 trains, day 2: 128 average trains, day 3: 109 average trains, day 4: average 91 trains - average train for each line, each way). Day one started with a 16 min schedule.

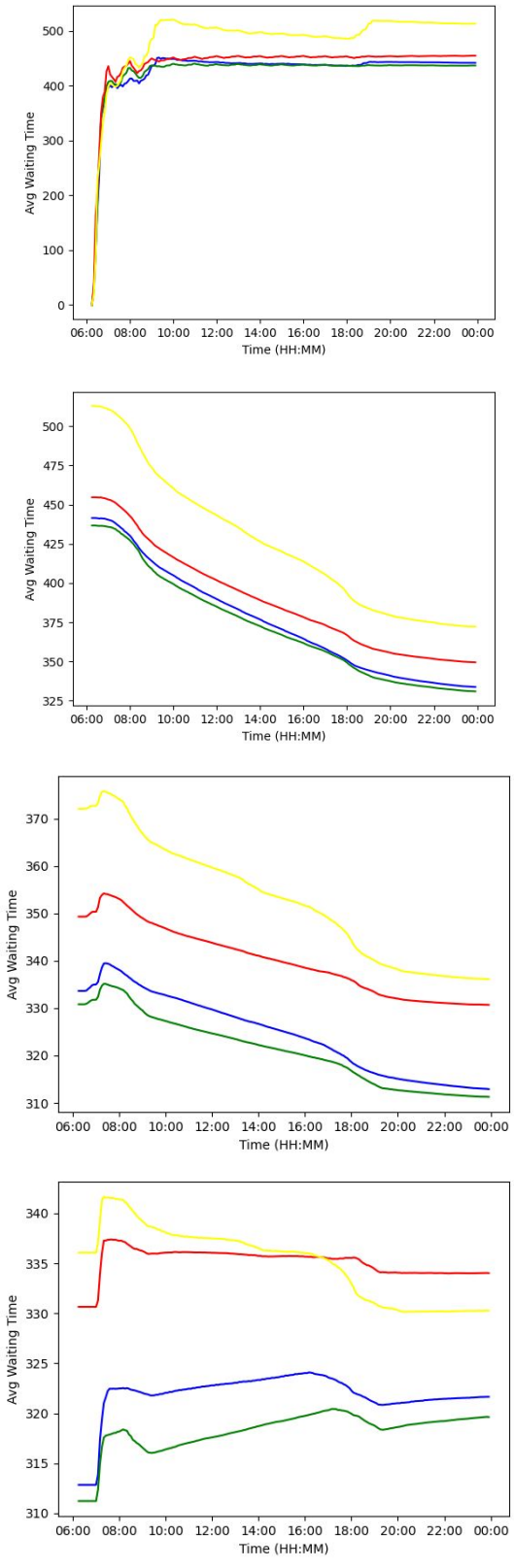

Figure 6: Deliberative average waiting time (in seconds) for each line (from top to bottom - day 1: average 72 trains, day 2: 128 average trains, day 3: 109 average trains, day 4: average 91 trains).

Finally, we have the evolution of the deliberative behaviour after four days (Fig. 5 and Fig. 6). As previously mentioned, we started off day one with a schedule that sends trains every 16 minutes similar to the baseline. We ran the model with an occupancy threshold of 40% and an average person per station threshold of 15 (see [7.3] for clarification on how these parameters are being used). The number of trains is low (72 per line, per way), but the average waiting times are reasonably high. By using the data collected on day one, we can see that the schedule for day two clearly tries to *hurt* the benefits for the service provider, launching more trains (128 per line, per way), which directly reflects on the now lower waiting times. Also, as expected, the average occupancy dropped since there are more trains. From day two to day three, the schedule managed to send less trains while improving the waiting times, which means that it is now making a better judgement on which hours to send more and less trains. On the fourth day, we can see that even less trains were launched, but now some lines have better waiting times while others have worse. Regarding the occupancy, its average is around 25%. 25%*185 is around 46 which is more or less the number of seats of the train, resulting in a comfortable ride for all passengers. Overall, this model was able to achieve a good result starting off with a bad schedule. Had we run this model for more days, maybe we could have more insights, however the planning computation is very memory-heavy, and, as such, we could not do it.

Comparing the final schedule (day 4) against the baseline, which was very beneficial for the passengers, we can conclude that this model is more balanced as it reduces the number of trains quite substantially at a cost of around 1 minute of waiting time for the passenger. Also, on average, the trains are 10% more full in this model.

As this model follows a schedule, it has the same advantages over the reactive approach as the baseline did, which is a better management of when to launch a new train.

8. CONCLUSIONS

The major conclusion we can take, is that there is no “The Schedule”, something that works optimally for both passengers and service providers. By launching more trains, the passengers will wait less time but the service providers will spend more money, and vice-versa. However what can be optimized is the management of “when will I send a new train?”. As we saw, our reactive model was very bad at doing this, while our deliberative model provided schedules that could optimize waiting times using less trains.

Another thing we concluded is, since all lines are different, using the same threshold values for all the lines wasn’t optimal. As such, we could try to improve our results by using different values for different lines.

Comparing the two approaches (reactive and deliberative), we can conclude that, since the baseline we used was a very passenger friendly option, (as it minimizes the waiting time for the passengers), they both worked as expected, trying to balance the resources both ways. However, the reactive approach could use some tuning of the train launch condition to avoid sending two trains very close to one another.

9. FUTURE WORK

This work aimed to test deliberative and reactive agents to help improve the subway system's efficiency using simple constraints. Future work includes implementing multiple machine learning algorithms in order to more accurately detect passengers flow patterns and further increase the subway system's efficiency. As this is a very simplified model, it is also important to improve its similarity to reality, mainly adding occasional problems on the lines and delay. It would also be interesting to extend the orchestrator's reactivity to synchronize trains passing in exchange lines' stations, in order to reduce the average waiting time for passengers when changing lines. The deliberative behavior we have can also be improved. Right now, it is overly-committed to the daily plan. We could add some conditions during the day that would allow the agent to replan. Finally, differences between schedules/plans could be inserted to differentiate weekdays and weekends, as the passengers' flow is very different, which would require a data analysis different from what we proposed here.