Meal Delivery Routing Paris

Group Jasper2

Supervisor Jasper van Doorn

Rens van Haasteren (2738676), Frank Le (2669711), Goos Neefjes (2640149), Roza Hoek Spaans (2783463) & Famke Bergsma (2770279)

February 2, 2024

Introduction

Due to the fast growing demand for meal delivery services, this project focuses on helping a company that delivers meals in the city center of Paris. The company works with an open market of restaurants and uses an online system where delivery requests are posted openly, creating a dynamic and competitive market.

Under the guidance of Supervisor Jasper van Doorn, the objective of this project is to develop a Decision Support System (DSS). This DSS needs to help the company in maximizing profits, with suggesting whether to initiate a new delivery, wait at the current location, or relocate to a more beneficial area.

The tools available for this project include a navigator tool that is based on a network representation of Paris' city center roads and a simulator tool that dynamically generates orders for the online marketplace [1].

This report contains some background information in the first chapter, an exploration of the project data in the second chapter, a scientific explanation of the algorithms used for the DSS in the third chapter, information about the dashboard in the fourth chapter and project verification details in the fifth chapter. The concluding pages of the report contain the conclusion, discussion, references and the appendix.

1. Background information

In this chapter the background information for this project will be documented. In paragraph 1.1 the problem description will be extensively explained. In paragraph 1.2 some literature investigation will be done and in the last paragraph, 1.3, the used tools Navigator and Simulator will be explained.

1.1 Problem description

There are multiple restaurants in Paris where people can order meals online. The restaurants can not handle all the orders that they get, therefore the restaurants can outsource the delivery of the order. If a restaurant decides to outsource delivery, it will be listed on an open online market where external parties are active. This project considers a company which is active on the online market. Orders appear dynamically on the market.

The company can decide whether to take on orders or not. For each order, there is a specific fee that the company earns for handling the delivery. In this project the tip is excluded (the driver receives the tip). There is a chance that an order will disappear from the market because it is accepted by another company. If the company decides to accept an order, the company is obliged to travel to the restaurant immediately, await the pickup time and deliver the meal to the customer. If a vehicle is busy with a delivery, it can not stop or do another delivery at the same time. The company can operate with multiple vehicles on one day. For this project, we will only operate during the day, the start and end locations of the vehicles are not within the scope.

The company aims to maximize the profit. However due to the complex situations where the operator can’t determine the underlying profitability of each action without deeper insights in the current situation, it is often difficult to determine the best course of actions. The operator needs to take into account the vehicle locations, order information, demand and competitors, creating a complicated environment to make justified decisions in.

1.2 Literature

Literature investigation is done for investigating the reward function. Zou, G., Tang, J., Yilmaz, L. et al [2] used the Clipping Rewards technique to reset the reward function. This means that when an order is delivered within the time limit, this order gets a reward of +1. Orders that are not delivered within the time limit will get a reward of -1.

Article 'Courier routing and assignment for food delivery service using reinforcement learning' [3] aims to maximize the reward by decreasing the delivery times and by allowing trading immediate rewards with future rewards. In other words, to find a policy consisting of couriers' optimal actions.

The last analyzed article [4] considered a limited number of available couriers fulfilling customer demands by picking up the meals from requested restaurants and delivering them to the customers as fast as possible. The objective is to maximize the total profit by minimizing the expected delays, postponing or declining orders with low/negative rewards, and assigning the orders to the most appropriate couriers. In addition, they included a reject action in the action space, which corresponds to choosing not to deliver an order that is deemed to be far away from the current courier locations. This model allows assigning multiple orders to the couriers provided that the long run return of that assignment is higher than other options. Here the cumulative collected reward, the number of rejected orders, and the statistics regarding the delivery times will be compared with the extensions. In summary, the objective is to increase the cumulative reward in a given time interval and decrease the overall delivery times.

1.3 Simulator and navigator

For this project, the simulator tool and navigator tool are provided.

The simulator tool dynamically generates orders for the online market for a full day. The simulator tool has the following functionalities:

- The number of considered vehicles can be specified.
- It keeps track of the available vehicles. For the vehicles that are available, new actions can be made. In the simulator class the *current vehicle* is the next vehicle requiring a new action.
- It contains the pickup, relocate and wait functions for the available vehicle.
- It generates the following information about an order: the order number, the pickup time, customer location, restaurant location and the fee.
- Orders become visible for vehicles 30 minutes before the pickup time and if the vehicle is within a radius of 3 kilometers of the restaurant.
- If vehicles are not available (busy with a delivery, relocating or waiting), the simulator can get the orders that vehicle can see at that time. This can be useful for making decisions.

From the navigator tool information about Paris can be collected. The navigator tool has the following functionalities:

- Make a plot of Paris
- Given a certain coordinate, we can find the nearest node from the coordinate. Paris can be divided into 11348 nodes. It is also possible to make this node visible on the plot.
- It is possible to plot a specific region. Paris is divided into 330 areas.
- Between every node the shortest route can be made based on travel time or distance. This can be plotted.

2. Data

In this chapter the data for this project will be documented. The data collection will be explained in paragraph 2.1 and in paragraph 2.2 the data will be explored.

2.1 Data collection

We are assuming that the ordering behaviour will be approximately the same every year, therefore we only run the simulation for the year 2023. This is done with the base algorithm, which will take the order with the highest fee out of the available orders. In order to get a wider spread of the order locations, the number of drivers is chosen to be 10. For the storage of information we keep track of the related restaurant and related day of the week.

2.2 Data exploration

From the collected data, we could observe 39 restaurants that are active in outsourcing their deliveries on the online market. In Figure 1 all these operating restaurants are visible with a black dot. These restaurants have different popularities, so the size of the dot is related to the number of orders these restaurants outsource. The maximum value of orders a restaurant processes is 23877 and the minimum value is equal to 507.

It is also observed that every restaurant has different operating days. The number of restaurants open at a specific day is shown in Figure 2. A restaurant is considered open on a specific day if we observe at least 1 order on that day. Monday exhibits the fewest open restaurants, specifically 17 operating restaurants. Tuesday and Saturday share the same number of operational restaurants, slightly exceeding Monday, with 22 operating restaurants. On Wednesday, Thursday, and Friday, the count rises to 34, signifying a significant contrast with the other days, although Sunday closely follows with 37 operational restaurants.

Every order that a customer puts online is online for a limited amount of time due to competitors that are also operating on this online market. In Figure 3 the cumulative distribution function of the order duration is plotted for each weekday. These durations are in seconds. From this cumulative distribution function we can take the 0.01-quantile. From Monday until Sunday respectively we get the following values: 2457, 2374, 2435, 2394, 2433, 2387 and 2395. These values give us insight about the certainty we have that the order will stay online. Given that 99% of orders have a longer duration than the specified values, you can be highly confident that an order with an online duration less than these values will remain online until this duration has passed the 0.01-quantile values.

To make an informed decision about waiting, it is helpful to know not only the order duration but also the expected number of orders in minutes, as some of the new orders may be more favorable than your current ones. In Figure 4, the order velocity is shown for every weekday. By order velocity, we refer to the quantity of new orders received per hour. If we begin examining Monday, we notice the lowest order velocity for each hour. This order velocity increases as we progress through the week. Differences in order velocity depending on the hour are only observed on Saturdays and Fridays. Not only the day of the week is of great influence, but also the time of the day. It can be observed that there are 2 peak moments, namely one around lunch time (12:30-14:30) and one around dinner (20:00).

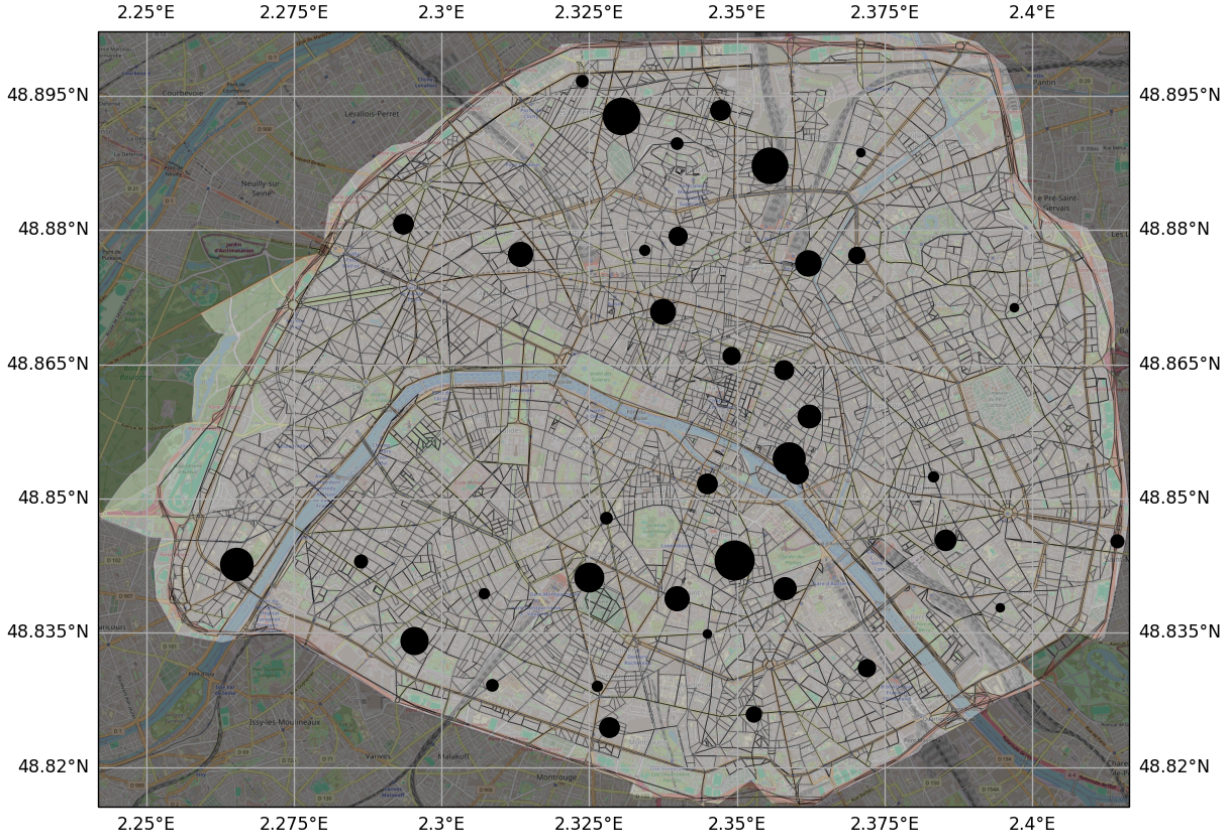

Figure 1. Restaurant locations with order size visualized for Paris.

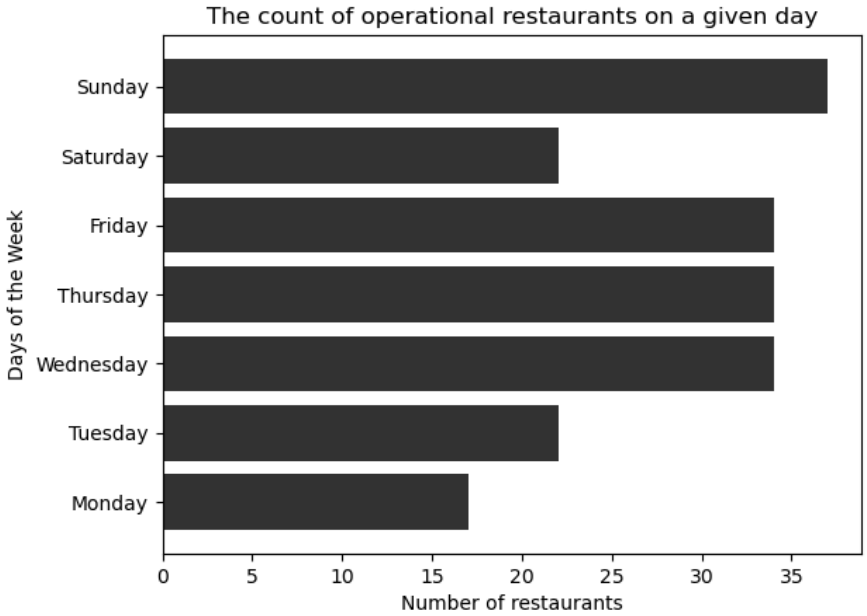

Figure 2. Bar plot of the number of operating restaurants for every weekday.

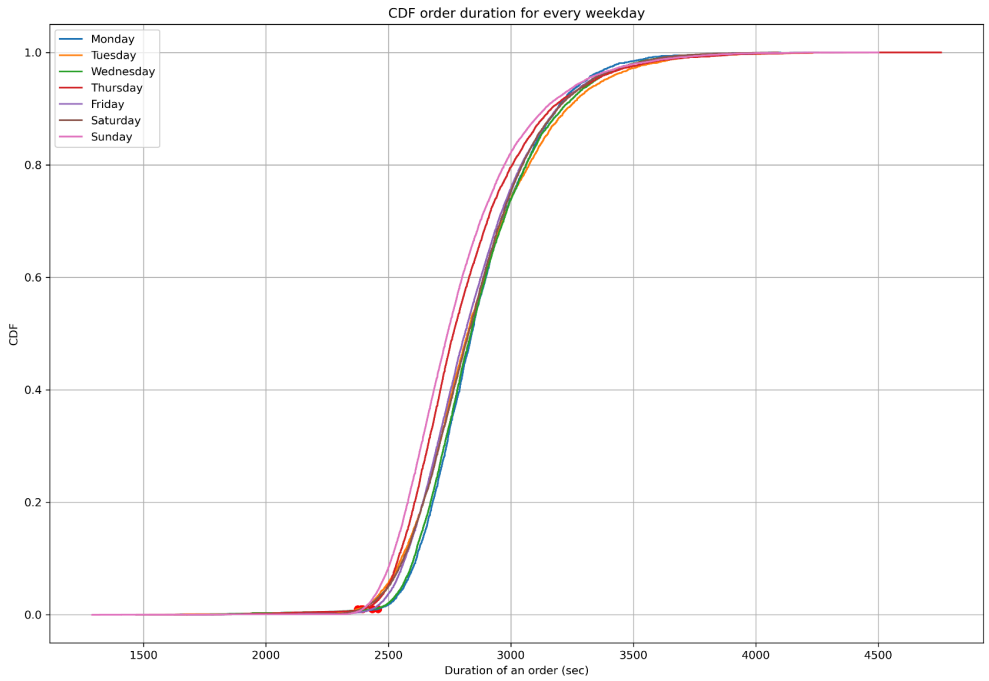

Figure 3. Plot of the CDF of the order duration for every weekday with the 0.01-quantile (the red dots).

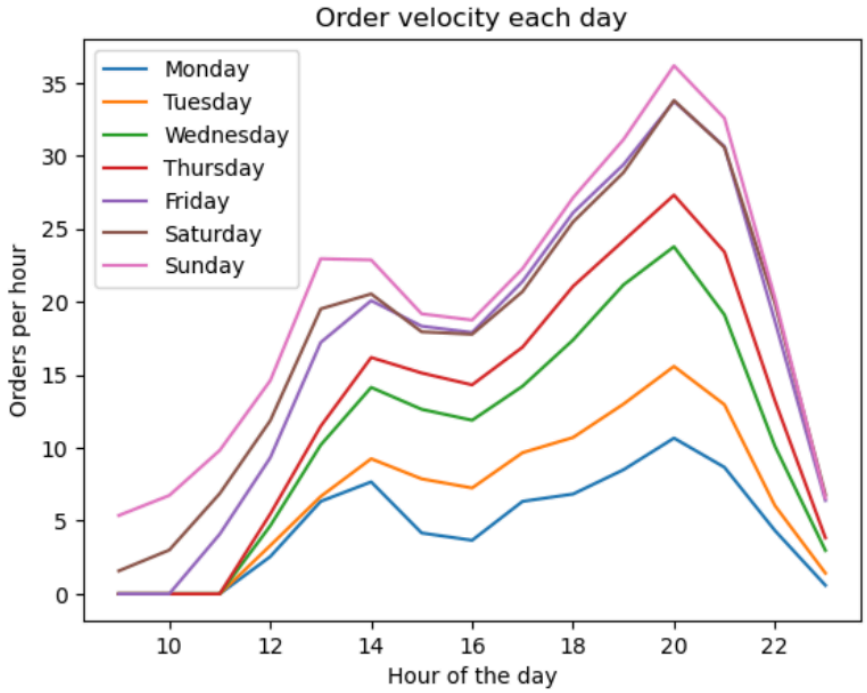

Figure 4. Graph of the order velocity for each weekday.

3. Algorithm

This chapter contains an explanation of the algorithms that are used for the Decision Support System (DSS). The algorithms we use are different approaches to determine the underlying profitability of the available orders. The metric we use for this underlying profitability is the reward, denoted by *R*. The problem to determine this *R* will be tackled step by step, breaking down the environment into different stages, such that the profitability can be researched iteratively. Starting with the most rudimentary solution, and gradually adding features at each stage. This strategy will allow us to build upon a basic solution, ensuring that each added feature is integrated well before moving to the next stage.

Paragraph 3.1 consists of the simplest part of the algorithm, thus stage one of the algorithm. In paragraph 3.2 and 3.3, stage two and three of the algorithm will be discussed, respectively. In paragraph 3.4, the final stage of the algorithm, stage four, will be discussed. In the last paragraph, 3.5, contains the parameter tuning.

3.1 Stage one

The most rudimentary version of determining the reward *R* is by setting it to the order fee. So with the formula:

$$\mathrm{R} = fee.$$

The fee is the financial gain associated with completing a specific order without taking the tip into account, since tip is not a factor the company should consider as explained prior. This form of reward selection does not take the travel time from the current location to the restaurant and then customer into account

3.2 Stage two

As important as it is and yet neglected in stage 1, company time is money. There is only a limited amount of time in a working day and to maximize profit, a time factor is ought to be introduced. In the second version of the algorithm the reward is determined by the formula:

$$\mathrm{R} = \frac{fee}{t},$$

where *t* is the total travel time, i.e. the time to travel from the current location to the restaurant plus the time from the restaurant to the customer. In this way the reward is determined by a certain money per time velocity, in comparison to stage 1 which analogously takes the fee displacement into account.

The computational time of version 2 increased drastically compared to version 1 if we calculated the travel time of each order based on the real restaurant and customer locations. For accurate travel times the nodes corresponding to the locations ought to be determined and then use these nodes to calculate the travel time and distance.

Fast decisions are especially important on an online market, so we sought a way to increase the execution time. For this, we used the areas that divided Paris in 330 regions, as shown in chapter 1. Beforehand we computed the travel time for every possible combination of different areas. With this knowledge the algorithm can simply pull the travel time between two areas out of a dictionary, decreasing the computing time severely as this change led to a significant 86% reduction in overall computing time. It took on average 12:55 to simulate two days in the old version and went down to 1:47 in the new version. The distance calculations are slightly less accurate, but the impact on the overall results is minimal.

3.3 Stage three

In stage 1 and stage 2 of the algorithm the focus is on selecting the most *directly* profitable orders. However, the driver naturally ends up at the customer location since this is the last part of the

delivery process and this end location has influence on selecting the next order. The end location can be beneficial or just detrimental for the driver based on the area he ends up at. Therefore in stage 3 the indirect reward based on the endlocation of the driver is added to the reward of stage 2.

The reward in stage 3 considers both the fee-to-travel-time ratio and the function $F(n_{end})$:

$$R = \beta_1 . \frac{fee}{t} + \beta_2 . F(n_{end}),$$

where *w_1* and *w_2* are weights to be determined by parameter tuning and n_end is the node corresponding to the end location of the vehicle. The function $F(n_{end})$ is determined by:

$$F(n_{end}) = (1 + \frac{\sum_{i=1}^{5} orders_i}{total\ orders})(1 + \frac{\sum_{i=1}^{5} weighted\ average\ fee_i}{total\ weighted\ average\ fee})(1 + \frac{1}{x} \sum_{i=1}^{5} time_i)^{-1}.$$

Where;

- *X* is representing a number that normalizes the t-value, in our case it is equal to 10000;
- The index *i* represents the ranking of the nearest restaurant from the customer location of the corresponding order, with the value 1 indicating the closest;
- The variable *t* signifies the computed travel time from the end location to the restaurant.

In the third stage of the algorithm, the calculation of $F(n_{end})$ required significant computational effort since it also uses the Navigator class. This involved determining distances to all restaurants to identify the five nearest ones during the run. Initially, we used nodes from the Navigator class for distance calculations, but this slowed down the algorithm as explained in stage 2. After switching to perpendicular distance for distance calculations, we observed an improvement in computing speed, but it negatively impacted overall performance. This is expected as perpendicular distance is less accurate. To achieve a balance between computing time and performance, we devised a solution: precalculating all $F(n_{end})$ values for each of the 11.348 nodes in Paris.

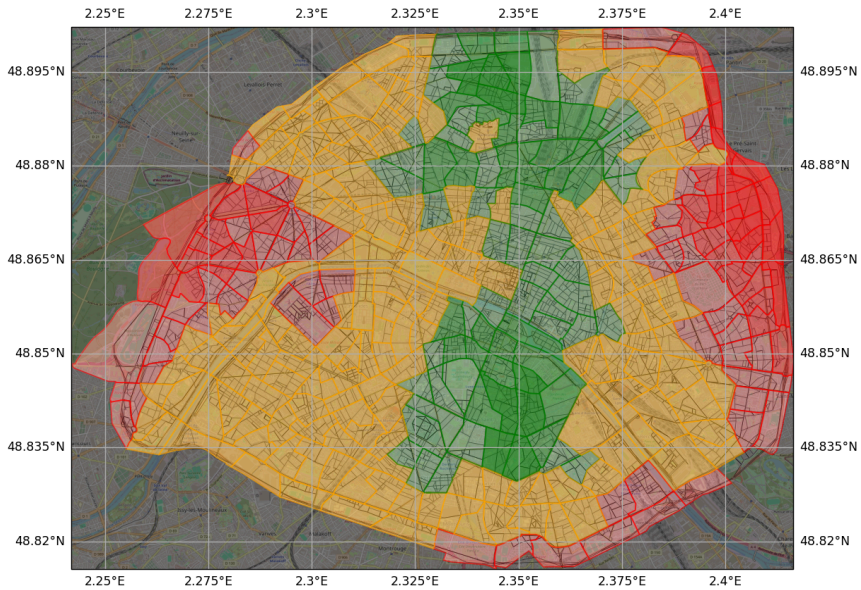

Figure 5. Heatmap of Paris.

For improvement visualization, the value per area is shown in Figure 5. The value in an area is the average of all values of the nodes this area covers. Red indicates a value less than 0,969, orange represents a value between 0,969 and 1,074, and green signifies a value greater than 1,074.

This observation is intuitive. When we compare this map to Figure 1, a pattern can be observed. The areas with higher rewards appear to be in areas located near the restaurant locations.

This is possible since the restaurant locations are fixed, meaning for each node the 5 closest restaurants can be determined. Also the number of orders and fee amount per order can be simulated for a large number of days (> 100 days), yielding an estimate of $F(n_{end})$.

The computation time was examined for a single run, and the results for one vehicle are presented in Table 1.

Method:	**Computation Time:**
Real travel time	+- 30 min
Perpendicular distance	+- 5 min
Heatmap	1:09 min

Table 1. Computation time for a single run for one vehicle.

By using the heatmap, we were able to improve performance and reduce computation time.

3.4 Stage four

Stage 4 is devoted to creating synergy between the drivers in Paris. The reasoning behind this is that spread out drivers will result in a higher variety of orders and less orders that can be snatched by other drivers. To achieve this, the algorithm will take two coherent variables into account: the endlocation of the vehicle and the estimated times of arrival (ETA) at the corresponding end locations.

If the endlocations of the vehicles are in close proximity to each other, it is in itself not an issue since the vehicles will most likely arrive at different times and see and take different orders. Similarly, if the vehicles arrive at their destination at about the same time, this is naturally also a nonissue. However, if the vehicles end close to each other at the same ETA, this is an issue, since now vehicles have overlapping order and therefore the reward needs to be smaller. The formula we contrived to take this circumstance into account is:

$$R = \beta_1 . \frac{fee}{t} + \beta_2 . F(n_{end}) + \beta_3 . G(n_{end}, t_{end}).$$

The reward function for stage 4 is the reward function of stage 3 plus G(n_{end}, t_{end}), where

$$G(n_{end}, t_{end}) = (1 - \frac{1}{1+min(loc\ diffs\ .\ ETA\ diffs\ )}),$$

and 'loc diffs' is a list containing all the travel times from the node n_{end}, that corresponds to the end location, to every other vehicle location. 'ETA diffs' is a list containing $|ETA(current\ order) - ETA(vehicle\ i)|$, that computes the difference between the ETA of the specific order with the ETAs of the other vehicles. We have chosen to take the minimum as an aggregate after the multiplication of the lists because now the locations and ETAs are coupled for each vehicle, meaning that the minimum of these coupled values will give the worst situation. If the value rolling out the minimum is small, the vehicles can snatch orders from each other, from the function G(n_{end}, t_{end}) will also be small, yielding a low reward.

3.5 Parameter tuning

In stage 3 and stage 4 the reward function consists of two components which can be differently weighted. Therefore, in the last part of the algorithm design we want to find the optimal weights. For this, we want to do a gridsearch for every day and calculate for each combination of weights the

profit for that specific day. If the gridsearch is done, we want to order all profits and take the weight combination that results in the highest profits. If we compare this for multiple days we require the optimal weights that perform the best on average.

4. Dashboard

This chapter explains the methodology behind creating the dashboard.

Since the simulator, that is in line with rudimentary real world examples, works by going forward in time until the next vehicle is available for another action, i.e. pickup, wait, or relocate. The dashboard contains input, decision variables, three main tables, the vehicle information, order information and a table for the vehicle-order combination information, and finally the Key Performance Indicators (KPIs).

The input is a selected date that the user wants to simulate and the number of vehicles to employ that day. The decision variables are the current time, the id of the vehicle that currently needs to take an action (actionable vehicle), the order velocity in orders per hour and the minimum order age with a 1% error in minutes.

The vehicle table contains the availability of each vehicle. If the vehicle is available its current location is displayed and if the vehicle is unavailable its destination and ETA are displayed. The order information table contains the order fee, restaurant, order age and customer location and travel time. Due to the complexity to determine the best order for each available vehicle, the order - vehicle combination table is displayed. Every vehicle (regardless of availability) is coupled to every order that its corresponding vehicle can see. The focal point of this table is to showcase the estimated reward that is determined by the algorithm, this information will aid the DSS user in his decision making process. For example when he sees that a vehicle that is available in 30 seconds has the highest reward, he can choose to wait instead of making an *uninformed* decision.

All locations given in the tables are given by a longitude (North) and latitude (East) decimal degree. Only in cases of very experienced users these coordinates have meaning, therefore we include a map of Paris that shows the start and end locations of vehicles and the corresponding routes. In addition we show the restaurant and customer locations and the routes of the all available orders of the vehicle needing an action. Due to this feature the user has a geographical understanding of the current situation and can pick a substantiated action based on all information at hand.

We are maximizing for profit, hence we want to keep track of the profit made on that day. The other interesting KPIs are the number of orders taken, distance travelled in kilometers and the profit per vehicle per hour.

The dashboard will be made in Python with web application *Dash*, since the framework is secure, user friendly and easily scalable. Besides this, *Bootstrap's grid system* will be used to organize the content in the dashboard contents.

The user manual of the DSS is provided in Appendix A.

5. Verification

In this chapter the results of the four different algorithms will be documented and compared with each other. In 5.1 a quick recap is given on the used reward functions and the factors considered by the corresponding algorithm stage. Paragraph 5.2 contains the results from the parameter tuning. The profit results are compared in paragraph 5.3, the distance results in paragraph 5.4 and paragraph 5.5 contain the results of the runtime. After that, in section 5.6 the statistical tests and results that are used are explained. The chapter is ended with section 5.7 with the results for the optimal number of vehicles.

5.1 Algorithm choices

Table 2 shows a recap of the used reward functions and the factors considered by the corresponding algorithm stage.

Stage	Reward function	Factors
1	$R = fee$	Fee amount
2	$R = \frac{fee}{t}$	Fee amount, total travel time
3	$R = \beta_1 . \frac{fee}{t} + \beta_2 . F(n_{end})$	Fee amount, total travel time, end location
4	$R = \beta_1 . \frac{fee}{t} + \beta_2 . F(n_{end}) + \beta_3 . G(n_{end}, t_{end})$	Fee amount, total travel time, end location, vehicle configuration

Table 2. Reward functions and their considered factors per algorithm stage.

where,

$$F(n_{end}) = (1 + \frac{\sum_{i=1}^{5} orders_i}{total\ orders})(1 + \frac{\sum_{i=1}^{5} weighted\ average\ fee_i}{total\ weighted\ average\ fee})(1 + \frac{1}{x} \sum_{i=1}^{5} time_i)^{-1},$$

and

$$G(n_{end}, t_{end}) = (1 - \frac{1}{1+min(loc\ diffs\ .\ ETA\ diffs)}).$$

5.2 Results parameter tuning

Algorithms 3 and 4 are dependent on the weights that can be freely chosen. A combination of weights indicates how much impact one part of the function has.

Algorithm 3:

The reward function for algorithm 3 is given by: $R = \beta_1 . \frac{fee}{time} + \beta_2 . F(n_{end})$. We performed a grid search, considering some possible combinations for β_1 (β_2 is implicitly tuned by varying β_1). The simulation for these results are based on 5 vehicles that are deployed one random week each month for a month.

β_1	β_2	**Profit**
8	1	€65.090,49
11	1	€65.098,61
9	1	€64.985,86
7	1	€64.944,04
6	1	€64.935,34

10	1	€64.905,93
12	1	€64.896,76
13	1	€64.847,15
14	1	€64.815,70

Table 3. Top 10 results for different configurations of β_1 and β_2 for algorithm 3

Based on the results from Table 3, we have chosen $\beta_1 = 8$ and $\beta_2 = 1$ for the optimal parameters for algorithm 3.

Algorithm 4:

Algorithm stage 4's reward is determined by the following formula:

$$R = \beta_1 . \frac{fee}{time} + \beta_2 . F(n_{end}) + \beta_3 . G(n_{end}, t_{end}),$$

which we want to optimize for β_1, β_2 and β_3. This simulation is done based on a single Saturday (due to time constraint) with 5 vehicles.

β_1	β_2	β_3	**Profit**
8	2	0	€636,09
8	2	2	€636,09
8	2	4	€636,09
8	2	6	€636,09
8	2	8	€636,09
8	8	0	€635,04
8	8	2	€635,04
8	8	4	€635,04
8	8	6	€635,04
8	8	8	€635,04

Table 4. Top 10 results for different configurations of β_1, β_2 and β_3 for algorithm 4

Table 4 shows that the profit does not change when β_3 is varied, therefore we have a degree of freedom to choose it. Since the algorithm runs faster if $\beta_3 = 0$, we choose this option.

From Table 3 and 4 the result is that we will use the weights: $\beta_1 = 8$ and $\beta_2 = 1$ for algorithm 3 and for algorithm 4: $\beta_1 = 8$, $\beta_1 = 2$ and $\beta_3 = 0$ for the rest of the rapport.

5.3 Results profit

Results are collected to investigate and compare the performance of the algorithms. The profit is observed for different days throughout one year, selecting the order with the highest reward for the

actionable vehicle (if no orders are available, wait 5 minutes).

In Figure 6 is, for each algorithm separately, a histogram is plotted for the profit of the company per day where a single vehicle is used.

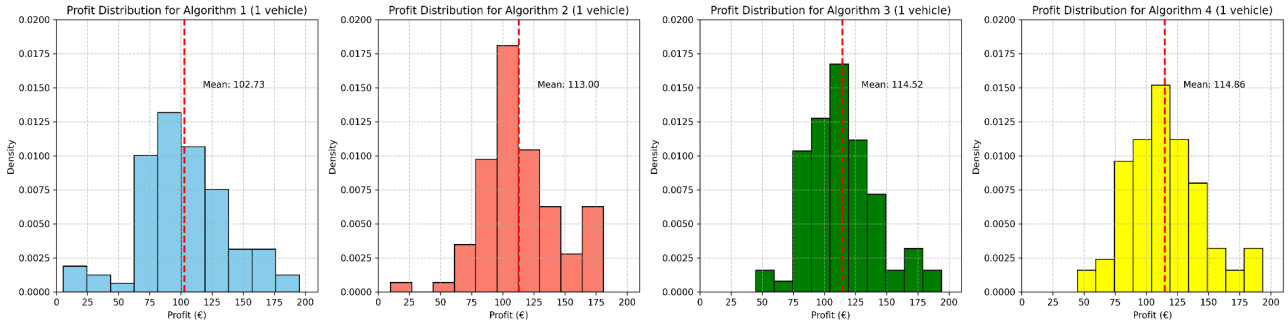

Figure 6. Histogram of the profit per day for the four stages of the algorithm for 1 vehicle.

Figure 6 shows that algorithm 2, 3 and 4 are improvements compared to algorithm 1. There are no big gains made going from algorithm 2 to 3 or 4, since the means are strikingly similar.

The same histograms are plotted in Figure 7, for the algorithms, but now 10 vehicles are deployed for the simulations.

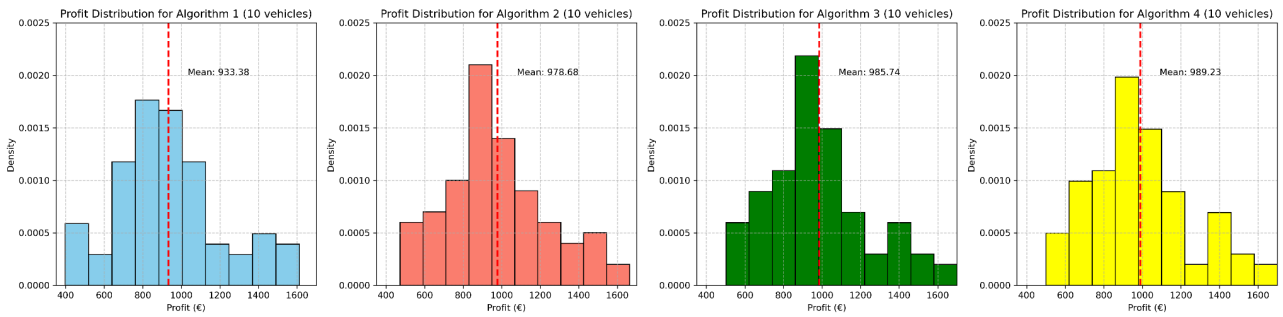

Figure 7. Histogram of the profit per day for the four different algorithms for 10 vehicles.

Figure 7 shows about the same pattern as in the case with a single vehicle. This is most likely due to the limited interaction between the vehicles resulting from the relatively large region the online market operates on and the order velocity.

Interestingly for both the 1 and 10 vehicle case the distribution for algorithm 2, 3 and 4 is bimodal.

The second peak is at the upper echelon of the histogram and quickly cuts off at the maximum profit.

5.4 Results distance

Similar to the profit results, the travel distances of the vehicles are collected. In Figure 8 are the distances per day plotted for each algorithm separately with 1 vehicle.

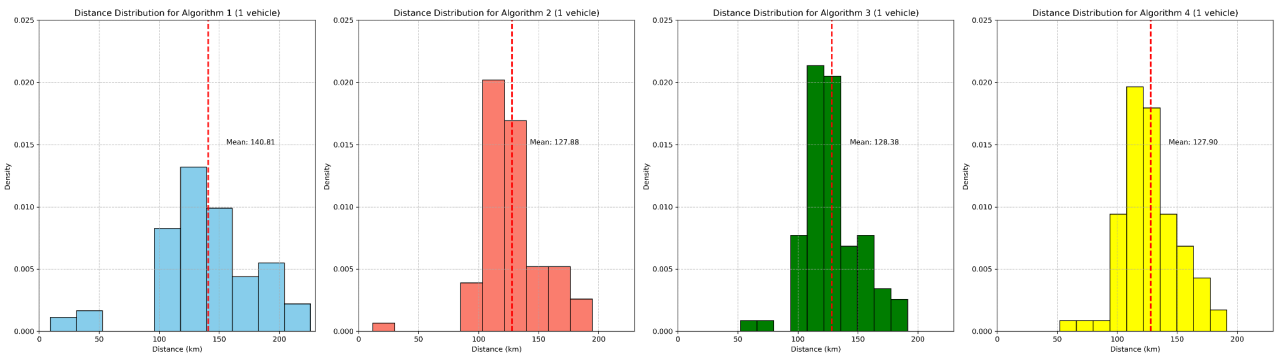

Figure 8. Histogram of the distance per day for the four different algorithms for 1 vehicle.

Figure 8 shows that the mean of the distance is pretty decreasing by using algorithm 2, 3 and 4 compared to algorithm 1. This may be due to the fact that algorithm 1 has some high outliers, while algorithm 2, 3 and 4 are having high peaks for approximately 100 - 150 km. The distance distribution for algorithm 2, 3 and 4 are quite similar.

Similar histograms are plotted for 10 vehicles in Figure 9.

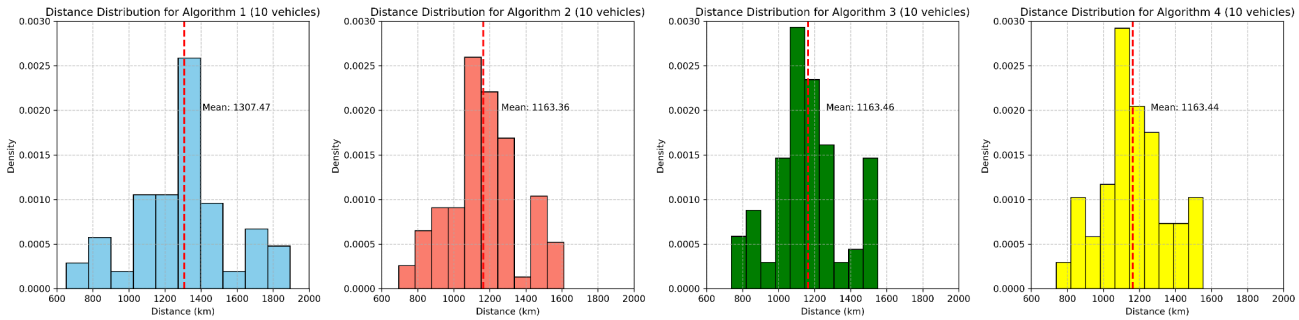

Figure 9. Histogram of the distance per day for the four different algorithms for 10 vehicles.

Figure 9 shows again that the mean distance is decreasing by using algorithm 2, 3 and 4 compared to algorithm 1. This may again be due to the fact that algorithm 1 has some high outliers. However, the distance distribution differs a bit compared to Figure 8. Namely, Figure 9 is showing a higher peak for a larger distance, while Figure 8 is only having a peak around the mean value. Besides this, again the distance distribution for algorithm 2, 3 and 4 are quite similar.

5.5 Results running time

The runtime of the simulation is observed for every algorithm. In Figure 10 the runtime results are visualized on the basis of boxplots where 1 vehicle is used during the simulation.

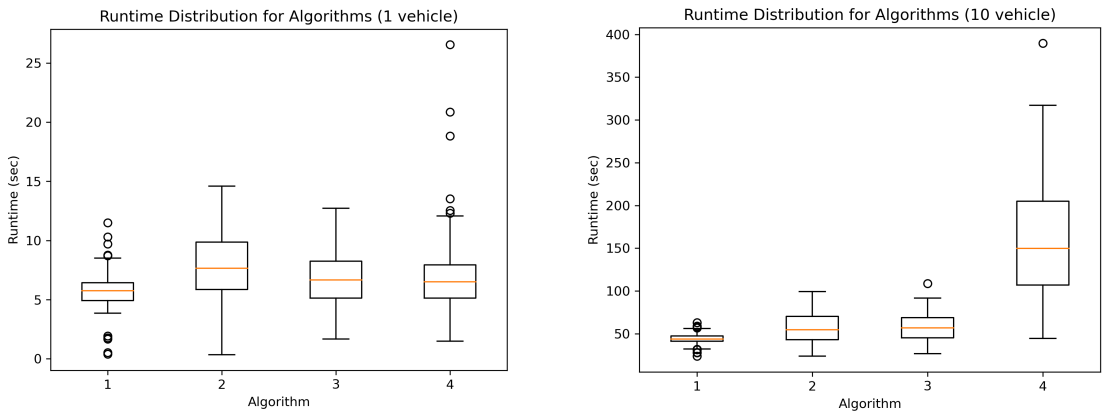

Figure 10. Boxplots of the running time for the four different algorithms for 1 vehicle and 10 vehicles.

Figure 10 indicates a similarity in the running times of each algorithm. Algorithm 2 has the most spread, and algorithm 4 has a few outliers. In the simulation where 10 vehicles are used, algorithm 4 has a high running time in comparison to the other algorithms. This makes a lot of sense, since algorithm 4 is way complexer than the other algorithms.

5.6 Statistical testing

To check if the results of various algorithms significantly differ, we conducted a one-way ANOVA test. We use Welch's ANOVA that compares two means to see if they are equal.

To perform a Welch's one-way ANOVA, two conditions must be satisfied:

- Normality
- Homogeneous variance

We will use a confidence level $\alpha = 0.05$. To test the homogeneous variance we use the Bartlett test and we define the null and alternative hypotheses by:

H0: the groups have an homogeneous variance
H1: the groups have an inhomogeneous variance

For the one-way ANOVA test we define the hypotheses:

H0: all means are the same
H1: at least one of the means is different

Statistical tests for the profit of 1 vehicle
For the profit for a single vehicle (showcased in Figure 6), the Bartlett test statistic is 3128.19 and has a p-value of 0.0. Therefore we reject the null hypothesis, and state that the profit does not have a homogenous variable. Due to this we can't perform an ANOVA test.

Statistical tests for the profit of 10 vehicles
For 10 vehicles (showcased in Figure 7) the Bartlett's test statistic is 0.096 and a p-value of 0.99, so the variance is not inhomogeneous. Therefore we will use a Welch ANOVA which has the following results: one-way ANOVA test statistic: 0.80 and a p-value: 0.49. Therefore we don't reject the null hypothesis that the means are all the same.

5.7 Results optimal number of vehicles

To determine the optimal number of vehicles per day, we conducted simulations for each day of the week, on a monthly basis. For each month, we selected a specific day and iterated through varying quantities of vehicles to identify the point where profit converges. Once this convergence is reached, it indicates that adding more vehicles is unnecessary for optimal profitability. As shown generally in Figure 11, the profit per vehicle decreases when more vehicles are deployed. This is a result from the fact that with increasingly more vehicles, the number of conflicts that orders get snatched also increases. The plot also shows that Sunday and Friday are by far the most profitable days regardless of number of vehicles, suggesting that optimizing for these two days of the week is most rewarding.

The optimal number of vehicles depends on the costs of deploying a vehicle. For example if the cost to deploy a vehicle is €100/day, the company should never accept orders on monday. To obtain the optimal number of vehicles one can multiply the profit per vehicle with the number of vehicles and determine the maximum value. Even with 40 vehicles the profit is still increasing, suggesting that the optimal number of vehicles is more than 40.

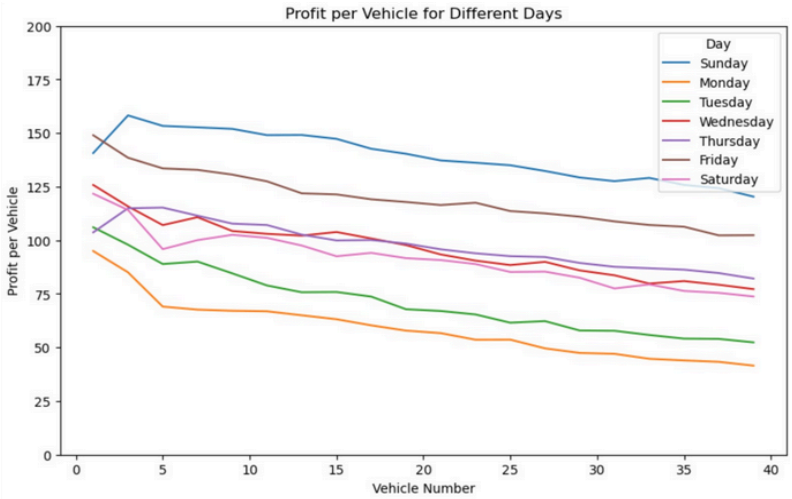

Figure 11. Graph of the profit per number of vehicles, separated by weekday.

Conclusion

In this rapport a meal delivery routing problem has been tackled. The problem is a dynamical problem with no prior data. A DSS is developed with an algorithm as base which estimates the profitability of each order.

Since there was no initial data available, the data was generated by running a simulation over the year 2023 in several instances. We pulled the restaurant locations and a heatmap indicating the hotspots from the generated orders.

A CDF was used to examine the duration an order stays on the market and a 0.01-quantile (based on weekday) is determined from it to give a safe minimum expected age for an order. In line with this, the number of orders that occur every hour is computed based on the weekday and hour. Shown in Figure 4, the order velocity differs throughout the day according to practice, i.e. high order velocity at noon and the evening. This order velocity combined with the quantile and order age allows the DSS user to still make an justified action in a complex situation.

Due to the insights generated by the data exploration, we iteratively expanded on the existing algorithm. These changes are summarized in Table 2 in section *5.1 algorithm choices*, showing that in the stage fee, travel time, end location, and finally vehicle configuration was added to the algorithm.

For algorithm stage 4, most profit was obtained with $\beta_1 = 8$ and $\beta_2 = 2$, since we have a degree of freedom to choose the weight multiplying $G(n_{end}, t_{end})$, we set $\beta_3 = 0$. With this configuration of β, the direct - indirect reward ratio was about 48% - 52%, suggesting that the end location plays a significant role in determining the strategy to maximize profit. This is due to the inhomogeneous restaurant locations and the vicious circle of sparse orders when ending in a *bad* place (for example the red areas indicated in Figure 5.

As priorly suggested, it was *discovered*[1] that differentiating weight β_3 coupled to $G(n_{end}, t_{end})$ at the highest paying parameter configuration does not yield a different result, indicating that taking the configuration of vehicles into account does not have an effect. The size of the region that the online market operates on, together with the order velocity displayed in Figure 4, could be an explanation for this phenomenon. We have simulated a meal delivery day with no more than 10 vehicles (due to the exponentially increasing computing time) which is a relatively low number of entities to deliver

[1] Results are based on a single simulation day, due to time constraints.

over Paris. The instances that vehicles will snatch orders from each other is limited and even when it happens a new order will most likely occur the next action.

That end location should play a significant role is contradicted with the unexpectedly high profit returns for stage 2. Due to this discrepancy, we can only conclude that taking the total travel time into account (from stage 1 to 2) yields a significantly higher profit. Shown in the statistical tests, we see that for 10 vehicles the profit means are not all the same however deeper statistics is required to determine a justified conclusion on the performance of the algorithms.

Discussion

During this project, there are a lot of opportunities and uncovered constraints through the development of the algorithm. These possible improvements are covered for each subject separately below.

Constraints based on simulation:

- The actual locations of the unavailable vehicles at each time could not be determined since the *Simulator.vehicle_locations* array contained the destination of each vehicle. Due to this we could not give the DSS user insight into the actual situation in Paris. This means that at the moment the user has to guess the real configuration of vehicles based on the travel speed of vehicles and ETA.

- The map in the dashboard needed to be embedded into html code since *Dash* does not support *matplotlib.pyplot* figures. Due to this, the *networkx* graph lost some functionalities.

Computational constraints and opportunities:

- The *Simulator* class is designed such that it updates each time a vehicle is done with their action. Due to this feature, we can't determine the exact time an order is taken off the market by competitors and if that differs based on the location, day and month of the year and restaurant. Based on this information one can change the routing of vehicles, which might yield substantial returns.

- The reward function in algorithm 3 and 4 consists of two components which can be differently weighted. These weights are now computed by doing a gridsearch for every day. Since the results are not as were expected, it will be recommended to investigate in a deeper parameter tuning.

- Due to the small number of vehicles that are deployed compared to the size of Paris and order velocity, there haven't been measurable instances when conflicts occur. Conflicts are for example when a vehicle snatches an order from another vehicle. Adding more vehicles increases the computing time exponentially, which makes it difficult to examine these situations. This is interesting, especially with a non-zero β_3 in algorithm 4.

Some general opportunities:

- Since the unexpected result, that algorithm 4 is not having the highest profit, it will be recommended to do a better investigation in the algorithms. This unexpected result may be due to the fact that the F and G functions (which are used in algorithm 4) are not based on literature.

- Restaurants operate in varying numbers each day of the week, leading us to create distinct heatmaps for each day. These heatmaps revealed diverse patterns. To assess their performance, we compared them to the original heatmap depicted in Figure 5. Surprisingly, the performance was found to be similar, contrary to our expectations. Further investigation and a more robust performance evaluation are necessary to draw more accurate

conclusions.

- The distance travelled KPI opens a door into the realm of sustainability. The carbon emissions of the daily operations can easily be determined by the type of vehicle and distance travelled, therefore in further iterations of the algorithm a sustainability aspect can easily be added if the company wished to steer on reducing emissions.
- For the simulator the locations of the vehicles are not fixed, the shifts of the workers, lunches, gas station trips etc. are not taken into account. An interesting further study could be done to get more realistic results.
- At the moment the amount of vehicles depends on the weekday, costs per vehicle and capacity. For further research, it may be interesting to do some deeper investigation in the optimal number of vehicles per day.
- The profit data is not totally normal distributed. However, we assumed that a higher sampled database will definitely be normal distributed. For further research, investigation in a larger dataset will be recommended.
- In order to analyze the profits a better test statistic than ANOVA needs to be found since the data is not normally distributed and the variance of the distributions are not always homogeneous.

References

1. GitHub Meal Delivery Routing 2024, Jasper van Doorn, https://github.com/jmhvandoorn/meal-delivery-routing-2024.git.

2. Zou, G., Tang, J., Yilmaz, L. *et al.* Online food ordering delivery strategies based on deep reinforcement learning. *Appl Intell* **52**, 6853–6865 (2022).

3. Aysun Bozanta, Mucahit Cevik, Can Kavaklioglu, Eray M. Kavuk, Ayse Tosun, Sibel B. Sonuc, Alper Duranel, Ayse Basar, Courier routing and assignment for food delivery service using reinforcement learning, Computers & Industrial Engineering, https://www.sciencedirect.com/science/article/pii/S0360835221007750 (2022).

4. Hadi Jahanshahi, Aysun Bozanta, Mucahit Cevik, Eray Mert Kavuk, Ayşe Tosun, Sibel B. Sonuc, Bilgin Kosucu, Ayşe Başar, A deep reinforcement learning approach for the meal delivery problem, Knowledge-Based Systems, https://www.sciencedirect.com/science/article/pii/S0950705122002088 (2022).

5. GitHub Meal Delivery Routing 2024, Frank Le, https://github.com/ikbenlong/meal-delivery-routing-2024.

Appendix

A. DSS User manual

A.1 Github

The full DSS is available on Github [5] and one can run the Git repository on their local machine to access the DSS. In the *environment.yml* file the dependencies are indicated. The *generator.pyc* is a compiled python file that determines the behaviour of the Simulator class in *simulator.py* such as the occurrence of orders, vehicle starting locations and the locations of restaurants in the Navigator class in *navigator.py*. The *support_functions.py* contains functions that are not structurally used by *algorithm.py* and *dashboard.py* but only called several times in order to provide some help with creating data or for importing/exporting files.

Creating a virtual environment that is connected to the Github repository on your local machine is the easiest way to organize the environment and manage the dependencies. Running the *dashboard.py* in this virtual environment will give a https address that routes to a local server on your browser. In this server you can view and use the dashboard.

A.2 Dashboard

The dashboard is designed to first require input of the user. The input consists of the date of the day that the user wants to operate in and the number of vehicles that the company wants to deploy that day in Paris. After selecting these inputs, the user clicks on the initialize button to initialize the Navigator and Simulator to obtain the vehicle starting locations and order information.

With this, the user now sees the current time and for what vehicle ID the user needs to take an action. The actions consist of picking up a chosen order, waiting for a certain time and relocating to a different place.

Pick up
With the pickup dropdown the user can select an order that he wants the vehicle, that is awaiting action, to pick up. The DSS is designed in such a way that the user has the freedom to pick any order he wants, but simultaneously the user is given the best options calculated by the algorithm through the vehicle - order combination table.

Wait
The second action the user can take is to let the vehicle wait for a specified number of seconds and clicking on the *ok* button. With this, the vehicle does not take an order and can therefore wait for a better one. The age of each order is displayed in the order table and gives, together with the order velocity, the user an indication of the optimal waiting period.

Relocate
The user can choose to relocate the vehicle to a different position by giving a longitude and latitude coordinate and clicking the *ok* button. The three tables in combination with the previously mentioned map, the user can make an informed decision about the location that the vehicle should relocate to.

Relocate to nearest restaurant
Lastly the user can choose to relocate the vehicle to the nearest restaurant based on the area that the actionable vehicle and restaurants are in.

The final button that requires explanation is the map toggle. Due to the heavy computations that are required for displaying the map of Paris and vehicle and order locations, and routes, the map is a toggle that can be switched on and off at any time.

B. Time registration

	Name				
Date, 2024	**Frank**	**Goos**	**Rens**	**Roza**	**Famke**
January, 9	7	7	6,5	4	5
January, 10	5	6	0	5	4
January, 11	8	7	0	6	6
January, 12	8	7	0	2	4
January, 13	7	0	0	4	4
January, 14	5	0	1,5	0	0
January, 15	2	6	1	3	4
January, 16	0	7	2	6	5
January, 17	0	7	8,5	6	7
January, 18	4	6	7,5	4	5
January, 19	4	7	6,5	6	6
January, 20	5	0	0	0	0
January, 21	3	0	0	0	0
January, 22	2	6	2	2	4
January, 23	8	4	2	6	5
January, 24	6	6	7	6	7
January, 25	3	4	6	5	7
January, 26	3	5	5	2	0
January, 27	0	0	3	0	0
January, 28	5	0	6	4	4
January, 29	3	5	3	6	5
January, 30	0	6	3.5	5	6
January, 31	8	5	10	8,5	8
February, 1	5	6	11	8	8
February, 2	3	4	5	4	5
Sum	**104**	**111**	**99**	**102,5**	**109**