Master Industrial and Applied Mathematics

Thesis

Online Matching of Carriers and Shipments

Lucy Verberk

TU/e Supervisor:
Dr.ir. Cor Hurkens

Company Supervisor:
Dr.ir. Bart Post

July 2, 2021

Abstract

UTURN is an online platform for matching the container transport need of shippers to the available capacity of carriers. In the first part of this thesis we identify properties of shipments that have the most influence on whether or not a shipment will receive a quote. With this goal in mind, we use a random forest model to determine the permutation importance of shipment features. From this permutation importance we identify features that are sensible candidates to influence the chance of receiving a quote for a shipment.

In the second part of this thesis we focus on increasing the visibility of shipments. We expect more visibility will result in a higher chance of receiving a quote. We realise this increase of visibility by suggesting combinations of shipments that can be executed after one another. In order to give balanced suggestions, i.e., to suggest all shipments a few times, instead of some shipments a lot, we model the problem as a mathematical b-matching. Each combination of shipments is given a weight, that indicates how attractive it is; a lower weight means more attractive. With the matching we aim to maximize cardinality, minimize the greatest weight used (bottleneck) and minimize the total weight used. We analyse the performance by applying the model to data provided by UTURN. We set the value of b to seven for all shipments, and consequently each shipment is suggested at most seven times. It is favourable that shipments are not suggested a lot, because if that were the case, such shipments might be quoted and accepted quickly. As a result of this carriers get to see meaningless suggestions, which could lead to carriers losing confidence in the suggestions.

Contents

Chapter 1

Introduction

The project described in this thesis was performed at CQM B.V. (Consultants in Quantitative Methods). CQM is a consulting company specialized in applied mathematics. Through CQM we have been working on a project for the company UTURN. UTURN provides an online platform to match the transport need of companies (the `shippers') to the available capacity of carriers. With their platform, UTURN wants to make the world of container transport more e cient, in order to ultimately increase the return for both parties. UTURN was founded in 2018. Since then they have been collecting data from their platform, with the thought that this could someday be used to, among other things, increase their matching percentage.

1.1 The UTURN platform

On the UTURN platform shippers publish their shipments and carriers can place quotes on these shipments. As soon as a shipper accepts a quote for a certain shipment, that shipment is assigned to the corresponding carrier. This is called a *match*.

Shippers have to provide details about the shipment. Some details are mandatory, others are optional. We distinguish between di erent types of details. First, there is some general information about the shipment, e.g., equipment type and grossweight. Secondly, there is information about the locations that have to be visited. A shipment always starts with the pick-up of a container and ends with the drop-o of that container. The other action options are deliver, load and stop. For an action the shipper provides the location (address) and a time frame in which the action needs to be executed. Finally, the shipper has to set a target price; i.e., the price he wants to pay for the shipment. He can choose to set the target price as acceptance price. In that case, if a carrier chooses to quote the target price, that quote is automatically accepted.

Carriers can see all published shipments and relevant details about those shipments. Based on this information carriers decide if they want to quote a shipment. Shippers can see all quotes on their shipments, and can look up additional information about the corresponding carriers, e.g., rating. Based on this information shippers decide which quote to accept.

UTURN also o ers another service: contract shipments. Shippers can make a contract with one or more carriers. The shipments of that shipper will then always be assigned to one of the contracted carriers. We ignore the contract shipments in this research, because there is not much that can be changed about the way such shipments get matched.

1.2 Problem Definition

This thesis consists of two parts. In Part I we analyse the data provided by UTURN with the intention to identify opportunities for improving the matching percentage. In order to come up with innovative and fresh ideas, and to prevent getting trapped in established patterns at UTURN, we aimed to use

a model that could give insight in the relative importance of speci c features by looking purely at the data. In this phase we tried to avoid using speci c domain knowledge of the roles of speci c features.

In Part I we use the following notation. Let S be a set of shipments and let $M(S)$ S be the set of shipments in S that are matched. Then the matching percentage on S is given by

$$MP(S) = jM(S)j=jSj \quad 100\%: \tag{1.1}$$

Let $Q(S)$ S be the set of shipments in S that received at least one quote. Then the quoting percentage on S is given by

$$QP(S) = jQ(S)j=jSj \quad 100\%; \tag{1.2}$$

and the assigned percentage on S is given by

$$AP(S) = jM(S)j=jQ(S)j \quad 100\%: \tag{1.3}$$

In Part II we focus on increasing the visibility of shipments. In particular, we want to increase the visibility of shipments that can be executed one after another, such that it becomes easier for carriers to nd combinations of shipments. We expect that more visibility of these shipments will result in a higher chance of receiving a quote. We realise this increase of visibility by suggesting combinations of shipments that can be executed one after another. The key challenge is to identify a set of attractive suggestions.

1.3 Organisation

This thesis is organised as follows. In Part I we analyse the data of UTURN. In Chapter 2 we give an overview of the data we use. Then in Chapter 3 we take a rst look a the data. Next we explain the random forest model in Chapter 4 and apply it to the data provided by UTURN in Chapter 5. Finally we give our conclusion of this part and directions for further research, in Chapter 6

In Part II we aim to identify an attractive set of shipment combinations, which can be suggested to carriers. In Chapter 7 we give a short introduction to this topic, and introduce some notation. Next we construct a scoring function to score combinations of shipments in Chapter 8. Then in Chapter 9 we formulate the problem as a mathematical matching. In Chapter 10 we apply our model to data provided by UTURN and analyse the results. Finally we give our conclusion in Chapter 11.

Part I

Data Analysis

Chapter 2

Overview of the Data

UTURN has provided the data they collected from their platform for this research. The data spans from the first data they have (2017) until the beginning of November 2020. In Section 2.1 we give an overview of the data features that are relevant for this research. In Section 2.2 we explain the cleaning steps we used on the data, and why cleaning the data was necessary.

2.1 Features in the Data

grossweight The grossweight of a shipment in kilograms.

distancemeters The distance to be travelled by a shipment in meters.

b xedprice This is 1 when the target price of the shipment was set as acceptance price, and 0 otherwise.

dutiespaid This is 1 when the goods of the shipment are cleared, and 0 otherwise. This influences the calculation of the VAT.

generatorset This is 1 when a generator set is required for the shipment, and 0 otherwise. There were some missing values of this feature in the data. If it is not given that a generator set is required, it can be assumed it is not required, hence the missing values are set to 0.

publishfrom This is the date and time of when the shipment was published. In our research we use this as three separate features: publishfrom: month, publishfrom: day of week and publishfrom: hour of day. The day of the week feature is modelled as a categorical feature.

publishto This is the date and time of until when the shipper wants the shipment to be published. In our research we use this as three separate features: publishto: month, publishto: day of week and publishto: hour of day. The day of the week feature is modelled as a categorical feature.

The default value for publishto is the end time of the time frame available for the pick-up/ first action. If a shipment gets assigned before this date, it is not published anymore, but this does not change the value of this feature. However, if a shipment is cancelled when it is still published, the value of this feature is set to the time of cancellation.

publish-/canceltime: hours This feature indicates the difference in hours between publishto and publishfrom.

leadtime: hours This feature indicates the difference in hours between the moment of publishing (publishto) and the latest moment at which the shipment can start (the end time of the time frame available for the pick-up action).

hours indication This feature is an indication of the amount of hours that the shipment will take. There is a feature with the hours indication available in the data, but this is new, and not yet available for all shipments. Therefore we calculated it ourselves, using the following formula, which is also used by UTURN.

action type 1 (load)	:	2 hours
action type 2 (deliver)	:	2 hours
action type 3 (pick-up)	:	0.75 hours
action type 4 (drop)	:	0.75 hours
action type 5 (stop)	:	1 hour
driving per km	:	0.01538 hours

(2.1)

quoted This is 1 when a shipment was quoted at least once, and 0 otherwise.

location: country This categorical feature indicates if the locations that a shipment visits were only in the Netherlands, only in Great Britain, only in Germany, only in Belgium, in one country but not one of those four, or if it was international.

container size (FT) This categorical feature indicates the container size. We chose to model this as a categorical feature, because apart from normal lengths, it contains one length range: 21 29. Also, for some containers no length is speci ed, those are given the category `not given'.

container HC This is 1 when the container is high cube (HC), and 0 otherwise.

container type This categorical feature indicates the type of container.

producttype This categorical feature indicates the product type of the shipment. Missing values are replaced with the category `not given'.

shipmentprice The target price set by the shipper.

target price/hours indication This feature is the shipmentprice divided by the hours indication.

target price/distancemeters This feature is the shipmentprice divided by the distancemeters.

shipmentrequirements This categorical feature indicates the shipment requirements, e.g., general cargo or frozen cargo. Missing values are replaced with the category `not given'.

uncode given This feature is 1 when the UN-code of the shipment is provided, and 0 otherwise.

nractions This feature indicates the number of actions that a shipment has.

type of action This categorical feature indicates the `type of action' of a shipment as a whole, i.e., a shipment consists of multiple actions and the actions together result in the type of action. For example, if a shipment only has a pick-up and drop action, it is called a transhipment.

timespan: hours This feature indicates the total time span available for the shipment. That is the time between the start time of the time frame available for the pick-up action, and the end time of the time frame available for the drop action. Note: the pick-up action should always be the rst action, and the drop action should always be the last action.

This feature contains some missing values, because if a shipment has more than 2 actions it is not mandatory to give the end time for the drop action. From domain knowledge we know that carriers mostly try to drop the container as soon as possible, such that the truck is free again. Hence we decided to replace the missing dates with the end time from the second last action, plus one day.

exibility: hours This feature indicates the shortest time, per shipment, between the start and end time of the time frame available for an action. We added this feature because the timespan: hours does not show how exible a shipment is in planning it. For example, a shipment can give a week for the pick-up action, but then the second action has to be done at a speci c time. This means the shipment is not exible at all, even though it has a large time span.

date of execution This date indicates when a shipment will be executed. From domain knowledge we know that the start time of the time frame available for the second action is most important for determining the date of execution, so that is what we use as date of execution. This is because, as mentioned before, the time frame available for the rst action can cover several days. In our research we use this as three separate features: date of execution: month, date of execution:

day of week and date of execution: hour of day. The day of the week feature is modelled as a categorical feature.

Note that we only consider general information about the actions and locations of a shipment. Each shipment must have (visit) at least two actions (locations), and in the data we see that there are no shipments which have (visit) more than ve actions (locations). In chapter 5 we use the data in a random forest model, and for that model the data must look the same for all shipments. That means that for each feature concerning an action or location, we should add ve new features, one for every possible action/location. But then shipments which have (visit) less than ve actions (locations) contain missing values, and the random forest model cannot deal with missing values. This could for example be solved by replacing the missing values with the mean value of that feature. However, that is not desirable because that shipment does not have ve actions/locations, and hence the entry should be empty. Moreover, a lot of the location features are optional, which already results in many missing values for the locations that do exist. That is why we decided to leave these features out, and focus on general information about the actions and locations.

2.2 Cleaning the Data

There are some parts of the data that are incorrect, irrelevant for our research, or not representative of the situations we are interested in. In order to obtain reliable results we cleanse the data of those parts. We do this using the cleaning steps described in this section.

1. The shipment status is completed or cancelled.

   Shipments eventually all end with having one of the following shipment statuses: released, completed or cancelled. Released is used by UTURN if they want to remove a shipment from the frontend. This happens for shipments that are wrong somehow, e.g., created incorrectly. These shipments are not relevant for this research. So we will only consider shipments that have the status completed or cancelled.

2. The grossweight of the shipment is at most 40000 kilos.

   The legal maximum for grossweight is 40000 kilos in Europe and 50000 kilos in the Netherlands. In Figure 2.1 we see that most shipments on the UTURN platform have a grossweight of at most 35000 kilos. We consider the few shipments that have a grossweight of more than 40000 kilos as outliers; this includes a shipment with a grossweight of 18 million kilos.

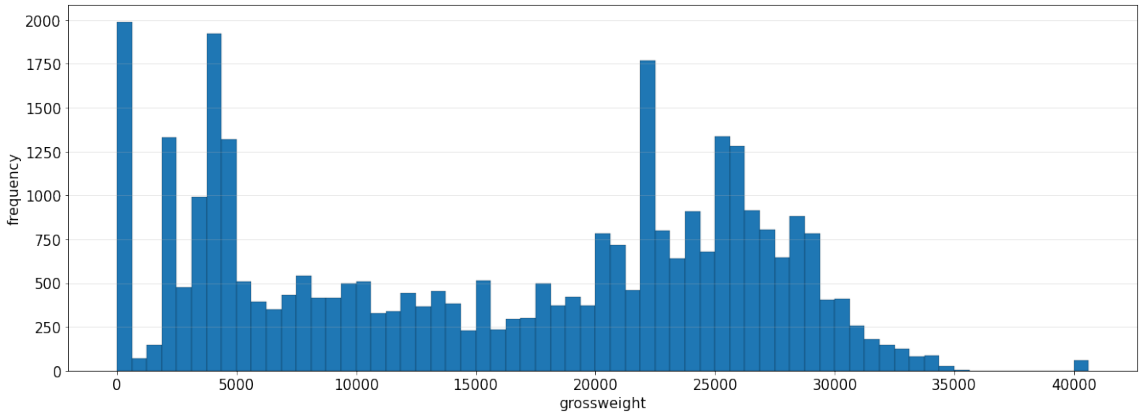


Figure 2.1: The frequency of the grossweight of shipments with status completed or cancelled. All weights strictly larger than 40000, are set to 40001.

3. If the shipment was cancelled, it was cancelled after more than 10 minutes.

   We assume that if a shipment was created incorrectly, the shipper will spot this mistake and immediately change or cancel the shipment. In the case of cancellation, there was no time to

receive quotes and get matched. So such a shipment should not count for the quoting and matching percentages. We assume that if a mistake was made, the shipment was cancelled shortly after publishing, lets say after at most 10 minutes. Also, it is to be expected that all shipments that were cancelled within 10 minutes, were wrong somehow. Hence, we will only consider cancelled shipments that were cancelled after more than 10 minutes.

4. The shipment has a positive lead time.

   The lead time is negative if a shipment is published after it should have already started. We assume that such shipments are invalid and only consider shipments with a positive lead time.

5. The shipment is published in 2019 or 2020.

   As mentioned in Section 3.2, we believe the years 2017 and 2018 are not representative of the situations we are interested in. Therefore we will only focus on shipments published in 2019 and 2020.

6. The shipment is not a contract shipment.

   As mentioned in the introduction, we will ignore contract shipments.

7. The first action of a shipment is a pick-up action and the last action of a shipment is a drop action.

   This should always be the case and currently it is only possible to create shipments that satisfy this structure. However, there are still a few shipments for which this is not the case.

8. For each action, except for the last action in case of at least three actions, the start and end time of the time frame in which the action can be executed, are provided.

   It is mandatory to provide these dates. However, there are a few shipments for which these are not provided. We assume that the data of these shipments is incorrect.

Chapter 3

Preliminary Insights

In this chapter we take a preliminary look at the data. The cleaning steps discussed in Section 2.2 were not yet applied for this exploration. In Section 3.1 we take a look at percentage trees for both the matching and quoting percentage. This gives us insight in what features have an influence on the matching percentage. In Section 3.2 we consider the number of shipments published, the matching percentage, and the quoting percentage per week to investigate if there are any seasonal patterns. In Section 3.3 we investigate if the acceptance price influences the matching percentage. Finally, in Section 3.4 we study the weekly and daily publishing and quoting activity.

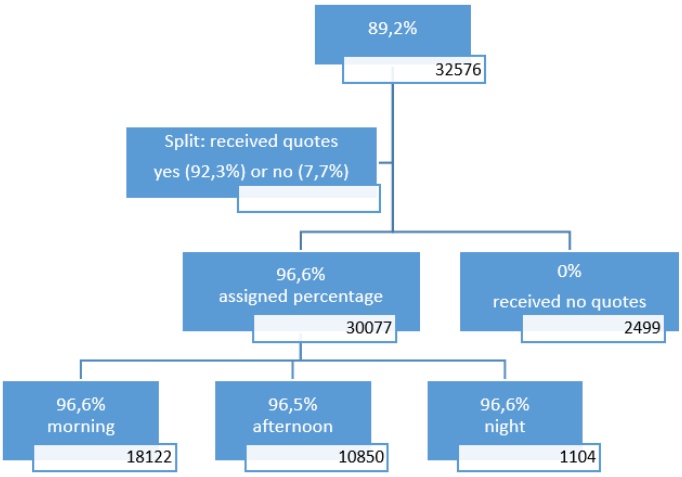
3.1 Percentage Trees

Percentage trees are trees that contain some percentage over some set in each node, for instance the matching percentage over a set of shipments. The root node contains the percentage over the complete set, other nodes contain the percentage over a subset of the set considered in its parent node. Percentage trees can give insights in where improvements can be made by analysing how changes in leaf nodes influence the root node.

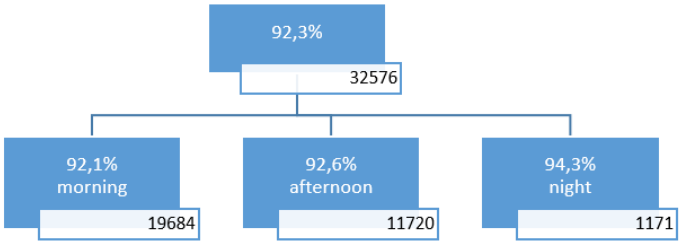
We construct five percentage trees for the matching percentage and five for the quoting percentage, where the root node represents the matching, respectively the quoting percentage over all shipments. Let S be the set of all shipments, then the percentages in the root nodes are $MP(S)$ and $QP(S)$, respectively. In each matching percentage tree we first branch on `quoted' or `not quoted'. Then for the five respective trees we branch the `quoted' node on one of the following categories: cargo requirements, hours indication, grossweight, in which day part a shipment is published, or on which day of the week a shipment is published, respectively. In each quoting percentage tree we immediately branch the root node on one of those five respective categories. A node in a percentage tree corresponds with the matching/quoting percentage of the corresponding category. For example, consider the left most leaf of the tree in Figure 3.1a. Let S' be the set of shipments that are quoted and published in the morning, then the percentage in this leaf equals $MP(S')$.

Note that in the percentage tree for the matching percentage, the matching percentage in the child node of the root node branched on `quoted' equals the assigned percentage. Indeed, let S_Q be the set of quoted shipments. Then we have that $|S_Q| = |Q(S_Q)|$, and hence $MP(S_Q) = AP(S_Q)$. For all nodes that branch from this child node, we also have that their matching percentage equals their assigned percentage. For the child node of the root node branched on `not quoted' the matching percentage will always be zero, because no quotes means no match. However, we can improve $MP(S)$ by making the number of shipments in the `not quoted' branch smaller, so by making sure more shipments get at least one quote. Changing the quoting percentage in a leaf node of the quoting tree influences $MP(S)$ indirectly: changing the quoting percentage in a leaf node changes $QP(S)$, which changes the size of the category `not quoted' in the percentage tree for the matching percentage, which in turn influences $MP(S)$.

The quoting tree branched on day parts, see Figure 3.1b, has the most in uence on the matching percentage. Increasing the quoting percentage of the morning from 92:2% to 100% increases $MP(S)$ by 4:6%. This category is also the one that has the most in uence in the matching tree: increasing the matching percentage of the morning from 96:6% to 100% increases $MP(S)$ by 1:8%. The di erence between these two is immediately clear: changes made in the quoting tree have much more in uence on $MP(S)$ than changes made in the matching tree. For all other percentage trees this is also the case. We also note that this high increase of 4:6% comes from the largest group. We found similar behaviour for the other trees: if there is a large group in comparison to the other groups, increasing the percentage of that group will have the most in uence; in contrast to when all groups are similar in size, then all groups have less in uence.



(a) Matching percentage tree.



(b) Quoting percentage tree.

Figure 3.1: Percentage trees branched on day parts, morning: 6h-12h, afternoon: 12h-17h, night: 17h-6h. The number of shipments in a group are displayed in the white box.

The main observation we get from the percentage trees is that increasing the quoting percentage has more in uence on increasing the matching percentage than increasing the assigned percentage. The assigned percentage is about quoted shipments that are or are not assigned. We can think of two reasons why a quoted shipment does not get assigned: (1) something changed about the shipment, which causes it to be either withdrawn or cancelled, or (2) the shipper is not satis ed with the received quotes and does not assign the shipment. For the rst reason, we see no solution. It just happens sometimes that things change. For the second reason, we think this can be best solved by getting more quotes on the shipments such that there is more choice for the shipper. We think that that is more in line with improving the quoting percentage.

There are some other noteworthy observations.

- In Figure 3.1a we see that there is a massive imbalance between quoted and not quoted shipments; 92:3% of the shipments are quoted.
- In Figure 3.1b the largest category (morning) has the lowest quoting percentage. This might be

caused by too many shipments being published at the same time, rendering many shipments unseen by carriers.

- In the trees branched on weekdays, all leaves containing working days (Monday - Friday) have approximately the same group size, the same percentage, and the same in uence. That suggests there is no di erence between the working days. Also, the weekend days have very high percentages. However, because there are so few shipments published during the weekend, we will most likely not be able to learn from this how to increase the overall matching percentage.
- In the trees branched on cargo requirements, the category `frozen cargo' jumps out. The leaves containing this category have a quoting percentage of 56:1% and an assigned percentage of 92:6%. For the other categories, those percentages lay closer together and the lowest quoting percentage is 82:4%. Probably few carriers are able to transport frozen cargo, which could explain why the `frozen cargo' category has a lower quoting percentage in comparison with the other categories.
- In the trees branched on the category `hours indication', both the percentage, and the group sizes, decrease if the hours indication increases. So there are more shorter shipments available, and they also seem to be more popular.
- In the quoting tree branched on the category `grossweight', the group sizes are evened out. However, the quoting percentage is higher on the ends of the range (0-40000 kilos), and lower in the middle. This is quite remarkable, since we would expect that carriers might have a weight limit of what they can transport, such that heavier shipments get fewer quotes. However, the heavy shipments have a high quoting percentage, so it seems that carriers are able to transport heavy shipments. Then they are also expected to be able to transport less heavy shipments. So there must be another factor in uencing the amount of quotes, for example heavier shipments might pay more.

3.2 Plots Through Time

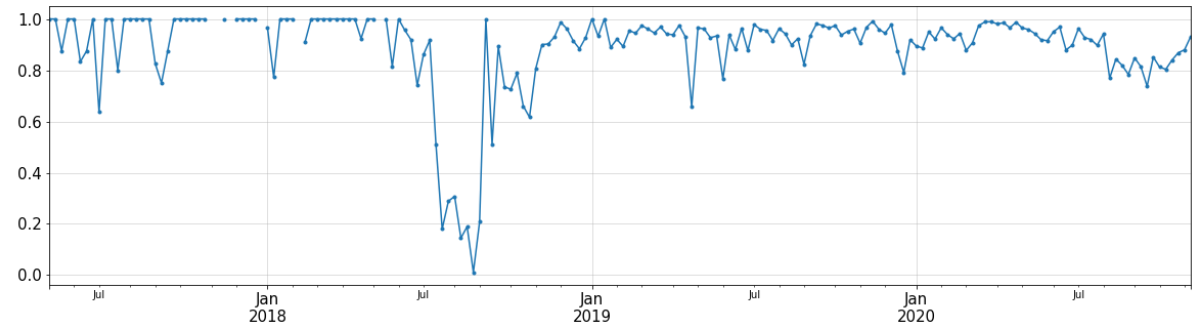
We looked at the number of published shipments, the matching percentage and the quoting percentage through time. We made plots per year, per month, per week, and per day. We think the plots per year and month are too aggregated, and the plot per day is too erratic, hence we only present the plots per week, see Figure 3.2. Note that in Figure 3.2b in 2017 and the rst half of 2018 a few data points are missing. That occurs when there are no shipments published that week.

In Figure 3.2a we see that in 2017 and the rst half of 2018 very few shipments were published. Consequently, only one not matched/quoted shipment can already cause a spike in the matching/quoting percentage. During the second half of 2018, there was a peak in the number of published shipments and a very deep fall in the percentages. During that period UTURN automatically took the shipments from a mailing list and published them on their platform. The plots suggest this did not work out very well; the matching and quoting percentages in this period are very low. Given these observations, we believe that the years 2017 and 2018 are not representative for the situations we are interested in, and therefore we will only focus on the shipments published in 2019 and 2020.

In Figure 3.2a we observe that in 2019 there was a substantial (continuous) rise in the number of published shipments. In 2020 the number of published shipments stays around the level that is reached at the end of 2019. In Figures 3.2b and 3.2c we see that both the matching and quoting percentage are on a good level, and they also seem to stay reasonably stable, with a few outliers. These plots do not seem to suggest any seasonal patterns. The deepest spike in both percentage graphs is around the end of April 2019. That is the period of Easter and Kings day, so maybe carriers took a few extra days o amidst the national holidays.

In Figure 3.2c we see that in 2020 from March until August the quoting percentages are very high, and then from August onward they suddenly go down. In the matching percentage plot it is similar, only from May till August the percentages are already a bit lower. The beginning of the high percentages is actually accompanied by the start of Covid-19 in the Netherlands. After a few months of Covid-19 the percentages go down. This can be explained as follows: during the beginning of Covid-19 there was uncertainty about work, and hence carriers tried to drag in some extra work using the UTURN platform. That is why the quoting percentage is so high. However the supply of shipments did not match the demand from

(a) Number of shipments published per week.



(b) Matching percentage per week.

(c) Quoting percentage per week.

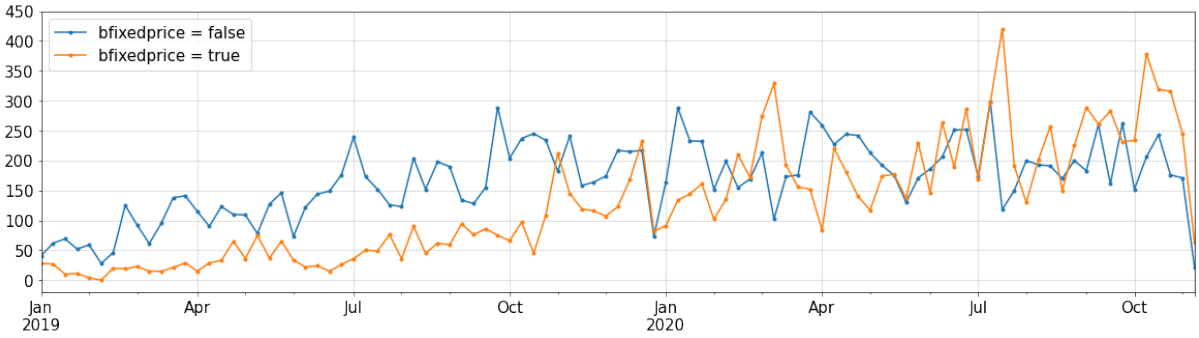
Figure 3.2: Plots through time.

carriers and that may have caused a signi cant amount of carriers to go bankrupt in the meantime. Fewer carriers means fewer quotes, which is re ected in the lower percentages that start in August.

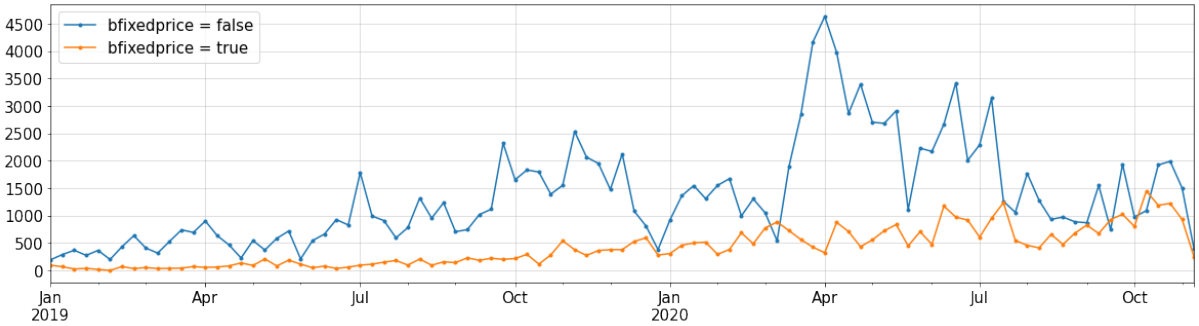
3.3 Acceptance Price

We now investigate if the acceptance price in uences the matching percentage. We expect that it does in uence the matching percentage, because if a carrier quotes the acceptance price, he immediately has assurance that he can execute the shipment.

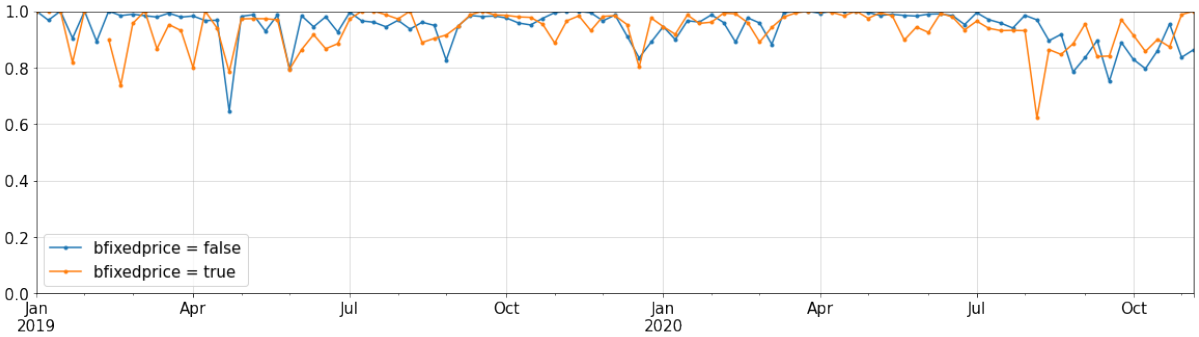
In Figure 3.3a we see that at the start of 2019 most shipments did not have an acceptance price. However, the number of shipments with an acceptance price rises from July 2019 until the beginning of 2020 to the same level as number of shipments without acceptance price. During 2020 the two stay on roughly the same level. In Figure 3.3b we see that shipments without acceptance price receive much more quotes. That is to be expected, since if a shipment with an acceptance price is quoted for the target price it can receive no further quotes. While a shipment without an acceptance price can receive quotes until the shipper manually accepts a quote. It is curious that around July-November in 2020 this changes; shipments with acceptance price now receive about the same number of quotes as shipments without



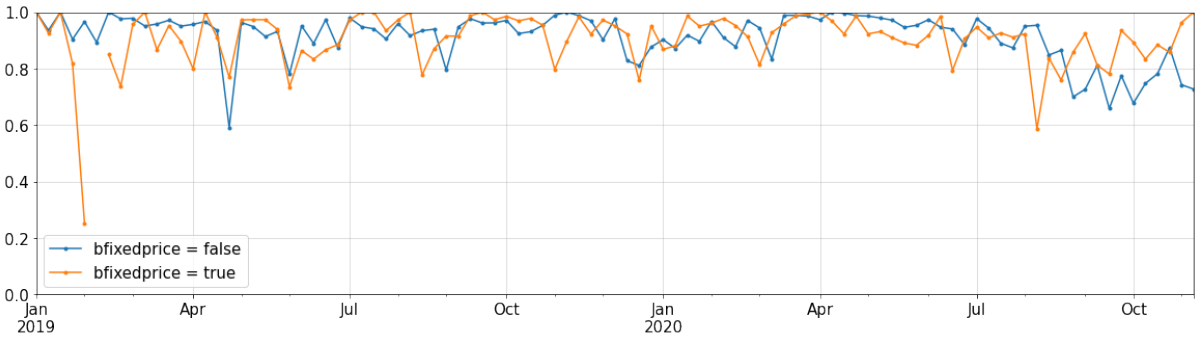
(a) Number of shipments published per week.



(b) Number of quotes per week.



(c) Quoting percentage per week.



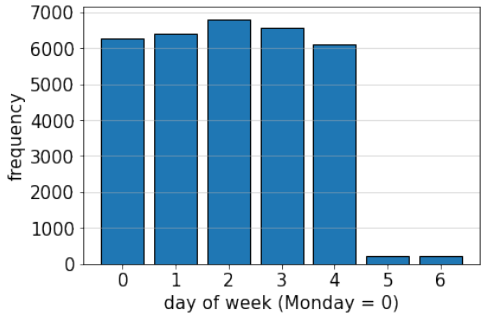
(d) Matching percentage per week.

Figure 3.3: With acceptance price versus without acceptance price.

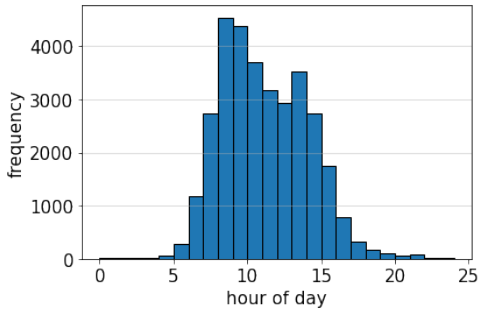
acceptance price, even though the ratio between published shipments with/without acceptance price has not changed. In Figures 3.3c and 3.3d we see that the quoting and matching percentage for shipments with and without acceptance price are roughly equal. This suggests that shipments with an acceptance price are equally popular to shipments without an acceptance price.

3.4 Activity on the Platform

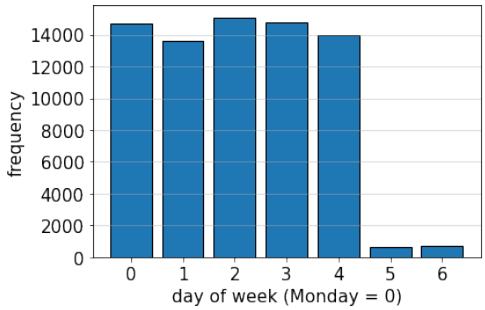
In order to see if we can exploit peaks/falls in the publishing and quoting activity during the week or day, we investigate the weekly and daily publishing and quoting activity. In Figures 3.4a and 3.4c we see that for both the shipment publication and quoting, there does not seem to be a favored day of the week. The activity is evenly spread over the working days, and during the weekend there is not much activity. Figures 3.4b and 3.4d show that there is mainly activity between 6h and 17h. In both gures we see that most activity happens in the morning (8h-10h). Furthermore we see that there is normal activity during lunch breaks, but not really any activity in the evenings. Last we note that, apart from the scale, the gures look quite similar. This suggests that the carriers follow the publications of shipments very closely with giving their quotes. In conclusion, for the weekly activity there does not seem to be any di erence between the working days, and for the daily activity the morning seems to be a favoured period.



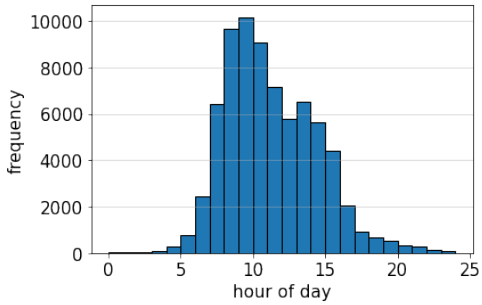
(a) Number of shipments published per day of the week.



(b) Number of shipments published per hour of the day.



(c) Number of quotes received per day of the week.



(d) Number of quotes received per hour of the day.

Figure 3.4: Activity on the UTURN platform.

Chapter 4

Random Forest Model

We want to gain insight in the relative importance and influence of different shipment properties on the quoting percentage. Recall from Section 3.1 that we want to focus on the quoting percentage instead of the matching percentage. With a random forest model it is possible to gain this insight; it can be used to derive a so called feature importance which scores the features on how important they were in making the prediction.

Random forest is an ensemble learning method which is typically applied to classification and regression problems. We train the random forest to classify a shipment as quoted/not quoted. Note that we are interested in the feature importance of the random forest model, not in the classification itself. We assume that when the random forest is sufficiently trained, the resulting predictions are a good representation of the real outcomes. Hence we have faith that the feature importance gives a good indication of the importance of features in obtaining quotes.

The learning in a random forest comes in two flavors: supervised or unsupervised learning. We use it for supervised learning, since we know from the data if a shipment was quoted or not.

The remainder of this chapter is organized as follows. In Section 4.1 we give a short explanation of the random forest model. In Section 4.2 we explain how the nodes are split. Next we explain one-hot encoding in Section 4.3. We explain what a voting threshold is in Section 4.4. Finally, in Section 4.5 we explain feature importance. Then we are ready to apply the random forest model to the data provided by UTURN in Chapter 5.

4.1 Short Explanation

A random forest consists of multiple classification trees. The model is trained on a dataset by constructing the trees based on that dataset. When the random forest model is (sufficiently) trained, i.e., when it has enough trees, a new item can be classified by putting it down each tree. Each tree gives a classification for that item, which counts as a vote for the class that the item is classified into according to the tree. The final classification is the class with the most votes over all trees (majority vote). Note that if you obtain additional data that you want to use to further train your model, you actually have to construct all trees anew, i.e., you have to retrain the model.

Random forests are constructed using an element of randomness. Let N be the number of data points in the training set and let M be the number of features. Randomness is introduced in two ways [2]:

1. The training set for one particular tree is a sample from the training data; N data points are sampled at random with replacement.
2. At each node of a tree, m features are selected at random out of the original M, such that $m << M$. The best split on these m features is used to split the node. The value of m is determined beforehand, and equal for all trees.

Given these two points of randomness, it is very unlikely that any two trees in the random forest are exactly the same. Let B be the number of trees and let n_{min} be the minimum number of data points that have to fall into a node, i.e., the minimum node size. Then we formulate the random forest model as an algorithm, see Algorithm 1.

Algorithm 1: Random Forest for Classi cation.

Construction of the random forest.

1. For $b = 1$ to B:
   (a) Sample N data points of the training data at random, with replacement.
   (b) Construct a tree T_b using the sampled data, by recursively repeating the following steps for each terminal node of the tree.
      i. Select m features at random from the M features.
      ii. Pick the best feature/split-point among the m, which results in two child nodes that both contain at least n_{min} data points.
      iii. If there exists such a feature/split-point combination, use it to split the node into two child nodes.
2. Output the ensemble of trees $\{T_b\}_1^B$, i.e., the random forest.

Prediction using the random forest.

To make a prediction at a new data point x: let $\hat{C}_b(x)$ be the class prediction of the bth random forest tree. Then $\hat{C}_{rf}^B(x) = majority\ vote\{\hat{C}_b(x)\}_1^B$.

This is based on Algorithm 15.1 from Chapter 15 of [8].

4.2 Split at a Node

As mentioned in the previous section at each node a subset of the features is considered, and among them the feature/split-point combination that results in the `best split' is used to split that node. First of all, what is a split? In a node, the values of a feature are split on a split-point x, meaning that if the value of that feature is strictly less than x, then you go to the left child node, otherwise you go the right child node. Second of all, how is determined what is the best split? Each split induced by a feature/split-point combination is given a score, and then the split that gives the best score is chosen.

We determine the score of a split using the Gini impurity. We can think of the Gini impurity as \the probability of incorrectly classifying a randomly chosen data point in the dataset if it were randomly labeled according to the class distribution in the dataset." [12] Let C be the number of classes and let $p(i)$ be the probability of randomly picking a data point that belongs in class i, then the Gini impurity is calculated as

$$G = \sum_{i=1}^{} p(i) \quad (1 \quad p(i)): \tag{4.1}$$

The intuition behind this equation is: we pick an element from class i with probability $p(i)$, and it is incorrectly classi ed with probability $1 \quad p(i)$. In our case we have $C = 2$ with classes quoted and not quoted, which means Equation (4.1) boils down to $2p(1 \quad p)$ for either $p = p(1)$ or $p = p(2)$. Note that $p(1)$ and $p(2)$ correspond to the fraction of shipments that are in the quoted and not quoted class, respectively. We want the Gini impurity to be as low as possible. The value of a split is measured with the Gini gain: the di erence between the Gini impurity before the split and the Gini impurity after the split. We want the Gini gain to be as high as possible. The Gini impurity before the split is simply calculated with Equation (4.1), but after the split it has to be calculated for the two child nodes created

by the split separately. These are then put together as follows:

$$\frac{n_{\text{left}}}{n} G_{\text{left}} + \frac{n_{\text{right}}}{n} G_{\text{right}}; \tag{4.2}$$

where n is the total number of elements, n_{left} and n_{right} indicate the number of elements on their respective sides and G_{left} and G_{right} indicate the Gini impurity on their respective sides.

4.3 One-Hot Encoding

The random forest model is only able to work with numerical features. Categorical features cannot simply be splitted on a numerical value, hence those must be somehow translated to one or more numerical features. One approach is to simply assign numbers to the categories, but this may introduce relations between the categories that are not actually there. A typical alternative approach is to use one-hot encoding: every category gets its own $f0; 1g$-valued dummy feature, see Figure 4.1.

index	color
0	blue
1	red
2	blue
3	green

!

index	red	green	blue
0	0	0	1
1	1	0	0
2	0	0	1
3	0	1	0

Figure 4.1: One-hot encoding of the feature `color'.

4.4 Threshold

When a random forest model is used for binary classi cation, we can introduce a voting threshold. Given a voting threshold 0 T 1, at least a fraction T of the trees must vote for class 1 for the nal classi cation to be class 1. In the original form of the random forest model majority vote is used for the classi cation, which corresponds with a voting threshold of 0.5.

4.5 Feature Importance

As mentioned before, an extremely useful aspect of the random forest model is the feature importance. In this section we discuss two variants of feature importance: Gini importance and permutation importance [1].

The Gini importance of a feature is calculated as the normalized total reduction in Gini impurity induced by that feature, i.e., the sum of reduction in Gini impurity over all nodes that split on that feature, normalized by the number of trees. [9, sec 2.3] The Gini importance has some drawbacks:

- It depends on how the trees are built using the training data, and not on the predictions made with the model. Because of this, the Gini importance cannot be determined for the test data. Also, it cannot take into account a custom threshold, since that is only used for making the predictions.
- Dummy features from one-hot encoded features are treated as separate features. This means you lose track of which dummy features correspond to which original features.
- It favours high cardinality features, i.e., features with many unique values. This can falsely make such features seem more important than they actually are.

The last two items together are especially problematic, because that means that all one-hot encoded features are seen as less important.

The permutation importance of a feature is calculated as the di erence between a baseline metric and the metric after permuting the column of that feature. In our case the metric is the accuracy score of

the prediction. This can be used on both the training set and the test set. We construct an auxiliary function to calculate this metric, such that a potential custom threshold can be taken into account. It is also possible to get the permutation importance of the original features, instead of the dummies created by one-hot encoding.

Remark 1

The permutation importance might give misleading values for correlated features. \When two features are correlated and one of the features is permuted, the model will still have access to the feature through its correlated feature. This will result in a lower importance value for both features, where they might actually be important." [4]

We have several categorical features for which we use one-hot encoding. We want to know the importance of the original features, not the dummies. That is the main reason why we use permutation importance. It is also bene cial that the permutation importance can take our custom threshold into account.

Chapter 5

Random Forest Applied to the UTURN Data

We used Python to apply the random forest model to the data provided by UTURN. In particular we used the RandomForestClassi er from the Scikit-learn package (version 0.23.2) [5,10]. The random forest model has some random steps in it, so to get comparable results between runs we set a xed seed for the random number generator.

In the remainder of this chapter we discuss how we apply the random forest model to the data provided by UTURN. In Section 5.1 we explain how we use the features in the random forest model. We discuss the imbalance of the data in Section 5.2. Next, we discuss how many trees we use in the random forest in Section 5.3. Finally, we discuss our results in Section 5.4.

5.1 Features

We use the features described in Section 2.1. For all categorical features we use one-hot encoding to use them in the random forest model. Recall that this creates separate (unrelated) dummy features. However, we want to remember which dummy features are related, such that we can determine the permutation feature importance per original feature. In order to remember these relations, we use a pipeline from Scikit-learn to combine a pre-processing step with the random forest classi er. In the pre-processing step we one-hot encode the categorical features using the OneHotEncoder function from scikit-learn and we leave the numerical features unchanged.

There are categorical features for which some categories are rare. For such rare categories it may happen they do not appear in the training data. Then if that category does appear in the test data, the model does not know what to do with it. Therefore we set the parameter handle_unknown to `ignore', which means unseen categories will be ignored.

5.2 Imbalance of the Data

As mentioned in Section 3.1 there is a massive imbalance between quoted and not quoted shipments. In this section we look at how to deal with this imbalance. First we discuss the imbalance and what this means. Next we propose a few options on how to deal with the imbalance. One of the options is to set a custom threshold, for which we rst determine to what value we want to set it in Section 5.2.1. Finally, in Section 5.2.2 we compare the proposed options.

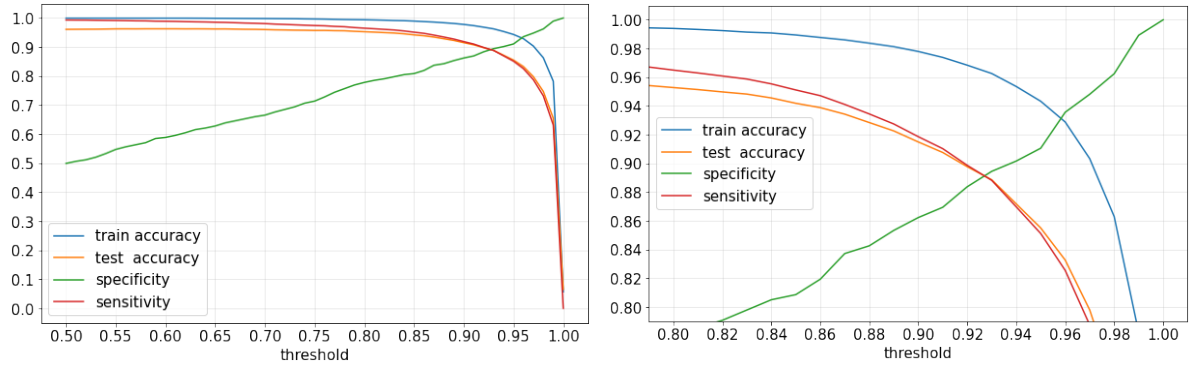
In contrast to the data analysis in Section 3.1, we did use the cleaning steps as discussed in section 2.2 on the data used in the random forest, consequently the imbalance is a bit di erent on this data: 94.1% of the shipments are quoted, instead of 92.3%. This imbalance in uences the speci city and sensitivity of the model. The speci city is the number of not quoted shipments predicted as `not quoted', divided

by the total number of not quoted shipments. The sensitivity is similar, but applies to the quoted shipments. The weighted average of the speci city and sensitivity equals the accuracy. For example, if we have a model that always predicts quoted, it would be 94:1% accurate. However, the speci city is 0 and the sensitivity is 1. A speci city of 0 is unwanted: all shipments that did not receive a quote were predicted as quoted, that is very disappointing for shippers. Therefore, we would like the speci city and sensitivity to be closer together, i.e., to be more in balance. There are a few options to accomplish this.

1. We set class weights. This can be set to `balanced', which means the weights are adjusted inversely proportional to the class frequencies. For example, if the data has 100 entries for class A and 1 entry for class B the weights would be 1 for class A and 100 for class B.
2. We completely remove the imbalance by taking a sub-sample of the quoted shipments, such that the quoted sub-sample has the same size as the not quoted class. Here we use sampling without replacement.
3. We set a custom threshold T .

5.2.1 Custom Threshold

Options 1 and 2 can be immediately implemented, while for option 3 we rst need to determine a suitable value to use as a threshold. By raising the threshold T from its default value of 0:5, we decrease the number of predictions resulting in `quoted', which is precisely what we aim for. We consider the accuracy on the training set and the accuracy, speci city and sensitivity on the test set. The results can be seen in Figure 5.1. The most balanced situation occurs when the test accuracy, speci city and sensitivity are equal. We see that this happens around T 0:925.



(a) $0:5 \leq T \leq 1$

(b) Zoomed in on $0:8 \leq T \leq 1$.

Figure 5.1: Di erent thresholds and their accuracy, speci city and sensitivity.

Recall that we want to use the random forest model for the permutation importance of the features, which is in uenced by the threshold T. We are interested in values of T near the value of balance: 0:925. We determined the permutation importance for T 2 $f0:80; 0:85; 0:90; 0:95g$. For these threshold values the permutation importance is very similar, and for some the complete order of the features is even the same. Moreover, for all of them the top eight in the permutation importance consists of the same eight features. Hence, with regards to the permutation importance, it does not really matter which value of T we take in the interval $[0:80; 0:95]$.

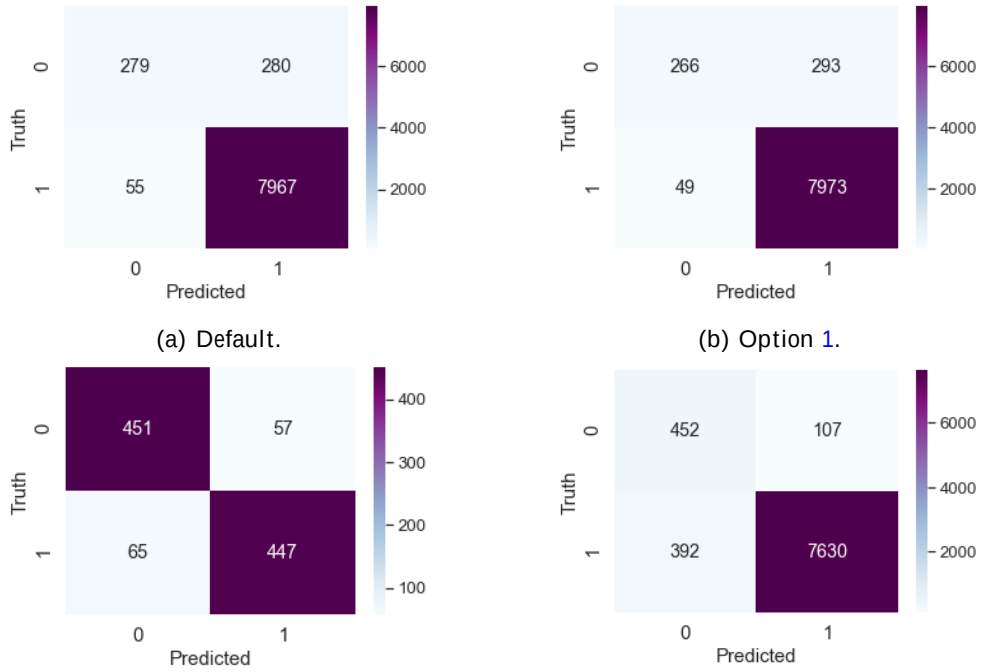
For the threshold $T = 0:925$ everything is completely balanced, but then the accuracy is approximately 0:89, which is quite low compared to the `starting' accuracy of 0:96, which is attained for the default value $T = 0:5$. For $T = 0:85$ we see that the accuracy is still approximately 0:94, and the speci city is already around 0:81. That is a signi cant increase from 0:5 which is the speci city for the default threshold. So the speci city increased by 0:31, while the accuracy went down by only 0:02. Increasing the speci city to the point of balance, gives an additional increase of 0:08, and an additional decrease of the accuracy by 0:05. We think that this gain in speci city is too small compared to the loss in accuracy. Also 0:81 is already a pretty good value for the speci city. Hence, we settle for $T = 0:85$.

5.2.2 Compare Options

Now that we have determined an appropriate threshold, we compare options 1, 2 and 3 listed in section 5.2. See Table 5.1 and Figure 5.2 for the results. In Table 5.1 the negative predictive rate, or NPV, is the number of not quoted shipments predicted as `not quoted', divided by the total number of shipments predicted as `not quoted'. The precision is similar to the NPV, but applies to the class `quoted'.

option	test accuracy	speci city	sensitivity	NPV	precision
default	0.9610	0.4991	0.9931	0.8353	0.9660
1	0.9601	0.4758	0.9939	0.8444	0.9646
2	0.8804	0.8878	0.8730	0.8740	0.8869
3	0.9418	0.8086	0.9511	0.5355	0.9862

Table 5.1: Scores for the di erent options to deal with the imbalance.



(a) Default.

(b) Option 1.

(c) Option 2.

(d) Option 3.

Figure 5.2: The true value of shipments versus their predicted value.

Looking at option 1 in Table 5.1 and Figure 5.2b, we see that there is actually no improvement of the imbalance between speci city and sensitivity, compared to the default option. For option 2 we removed the imbalance, and indeed for option 2 in Table 5.1 and in Figure 5.2c we see a very balanced situation. However, we see that the accuracy has decreased signi cantly. Another downside of this option is that a very large part of the data is not used, which is a shame because you do not know how much information is lost in the unused data. We also need to keep in mind that the ratio between `not quoted' and `quoted' used to be about 1:16 while with option 2 it is 1:1. If we would ever use this model as a prediction model for new data, which will more likely have a ratio close to 1:16, this model may have a disastrous performance. If we look at option 3 in Table 5.1 and Figure 5.2d we see promising results, the imbalance between the speci city and sensitivity has decreased and the accuracy is still quite good. This may not come as a surprise, given that we chose the value of the threshold T for that very purpose. Still, we can compare it with the other options, and it de nitely gives the best results for dealing with the imbalance: the values of the speci city and sensitivity are much closer together, while the accuracy decreased only very little. Hence we apply option 3 to deal with the imbalance, with $T = 0{:}85$.

5.3 Number of Trees

A very important parameter in the random forest model is the number of trees. In terms of the accuracy of the random forest model more trees is actually always better; the model will not over t. However, taking more trees increases the running time, and at a certain number of trees the decrease of the misclassi cation error becomes negligible. In that case more trees is not worth the additional computation time. To nd a suitable value for the number of trees we plot the misclassi cation error of the model versus the number of trees on both the training and test set. It is common practice to pick a value for the number of trees where the misclassi cation error levels o , and then multiply it by three.

We consider the misclassi cation error on the test set with both the default threshold $T = 0.5$ and our custom threshold $T = 0.85$. The results can be seen in Figure 5.3.

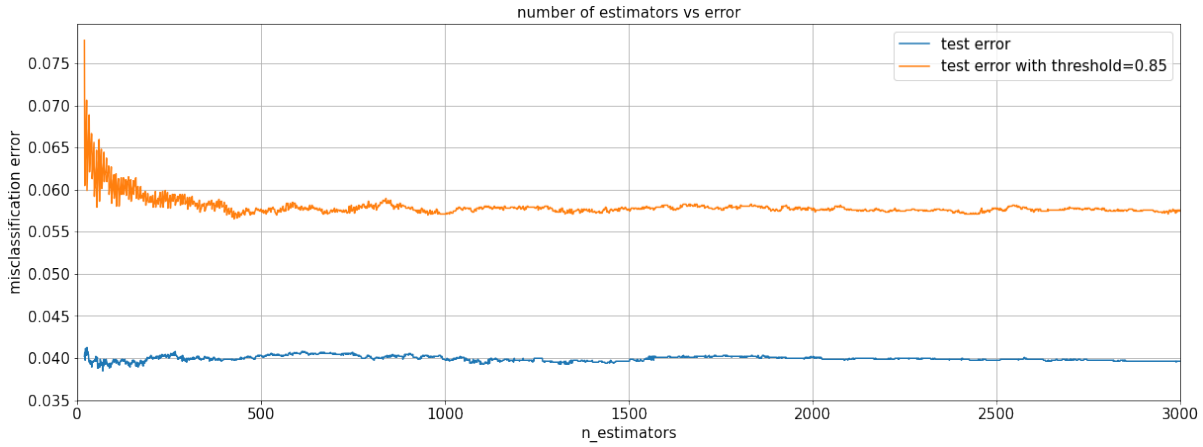


Figure 5.3: Plot of the number of trees versus the misclassi cation error.

Note that the test error with a threshold $T = 0.85$ is worse compared to the other graph, but we settled for this to deal with the imbalance between the speci city and the sensitivity. The test error stays around 0.04 which corresponds with the accuracy of 0.96, and the test error with $T = 0.85$ stays around 0.06 which corresponds with the accuracy of 0.94 for when using that threshold.

For few trees we see that the error with $T = 0.85$ behaves much more erratic than the other error. For both errors we see that they stabilize between 500 and 1000 trees. For more trees there is still some variation, but that occurs very gradually. Hence, we settled on using 3000 trees.

Concerning the execution times: tting the model with 3000 trees to the training data takes between 30 and 60 seconds on a laptop with an *Intel(R) Core(TM) i7-4710MQ CPU @2.50GHz* processor, and 16GB of RAM. Calculating the errors displayed in Figure 5.3 for an increasing amount of trees up to 3000 trees, took almost 7 hours on a computer with an *Intel(R) Xeon(R) Gold 5218 CPU @ 2.30GHz* processor, and 256GB of RAM.

5.4 Results

In this section we take a look at the results obtained by applying the random forest model to the data provided by UTURN. In particular, we determine and discuss the permutation feature importance.

With the data prepared and the parameters set as described previously, we trained the random forest model. Consider the performance indicators listed in Table 5.2, some of which also appear in the bottom row of Table 5.1. Moreover, in Figures 5.5a and 5.5b we list the permutation feature importance for both the training set and test set, respectively.

As mentioned in Remark 1, when using permutation importance it is important to pay attention to correlated features. For the following group of four features there are only three degrees of freedom, i.e., if you determine the value of three of them, the value for the fourth feature is xed.

Performance indicator	Value
accuracy on the training data	0.9895
accuracy on the test data	0.9418
speci city	0.8086
sensitivity	0.9511
NPV	0.5355
precision	0.9862

Table 5.2: Results.

- target price/hours indication,
- target price/distancemeters,
- hours indicaton,
- distancemeters.

In Figures 5.5a and 5.5b, these features still have a pretty high importance score; the last three are all in the top 6 of 30 features in total, only `target price/hours indication' is lower than expected (places 10 and 11). We expected `target price/hours indication' to have a higher importance because the price a carrier gets paid per hour seems like it would be an important factor in deciding to quote a shipment or not. Therefore we additionally decided to look at the Gini importance. However, because of the speci c drawbacks with respect to one-hot encoded features, we only consider the Gini importance of the numerical features, see Figure 5.4. In this gure we see that `target price/hours indication' is indeed considered more important. Apart from this feature and the ` exibility: hours' feature, which is considered as less important with this importance measure, the order of the Gini importance is roughly the same as the order of the permutation importance. Hence we trust the order of importance given by the permutation importance, with the exception that we think it underestimates the importance of `target price/hours indication'.

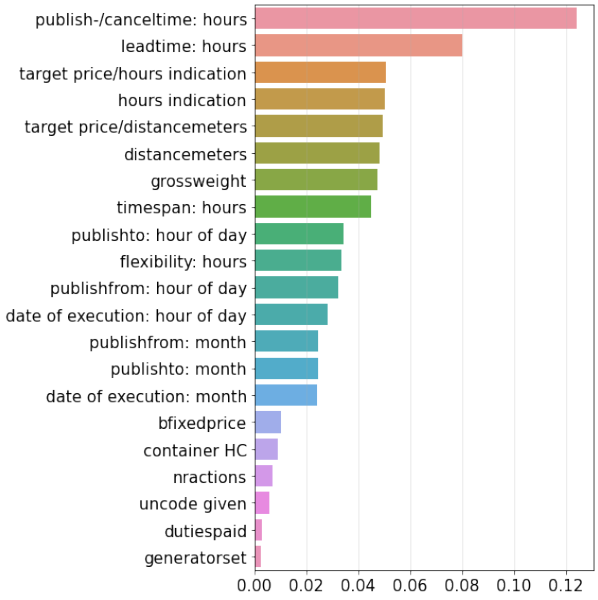
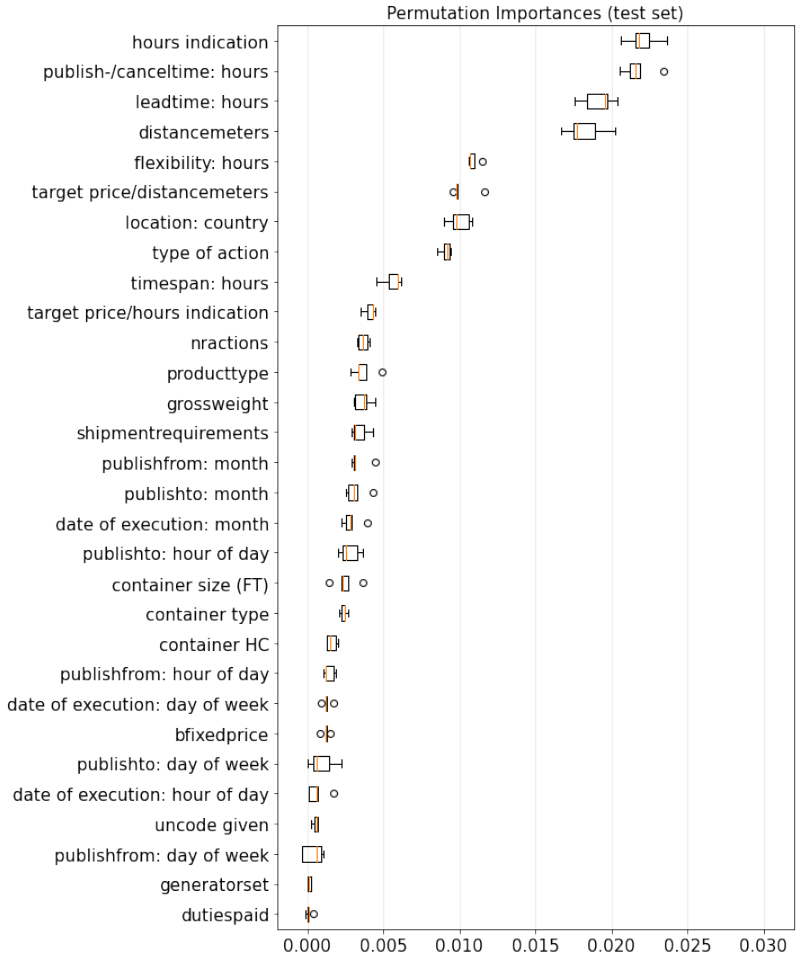


Figure 5.4: The Gini importance of the numerical features.

In both Figures 5.5a and 5.5b the values for `publish-/canceltime: hours' and `leadtime: hours' lay very close together. This is to expected since the values will be the same for many shipments. This happens because the default value for `publishto' is the end time of the rst action. These values will not be the same in the following two situations.

(a) On the training data.



(b) On the test data.

Figure 5.5: Permutation importance of the features.

- The shipment is cancelled, while it is still published. Then the `publishto' time is changed to the time of cancellation.
- The shipper changes `publishto' from its default value. This might happen if the shipper wants to know well before the shipments starts if he will be able to match it via the UTURN platform, such that if needed there is enough time to look for an alternative.

Note that these features are very similar, but they cannot be determined from one another.

The feature importance of `hours indication' and `distancemeters' lay very close together in both Figures 5.5a and 5.5b. Remember from Equation (2.1) that distance has a large influence on the hours indication. Then it is not so strange that these two features score similar. Indirectly, the type of action and the number of actions also influence the hours indication. We see that hours indication is more important than these three features. Note that even though these features depend on each other, none of them can be completely determined from the others.

The features `publish-/canceltime: hours', `leadtime: hours', `hours indication' and `distancemeters' are the top four most important features, according to both figures. In Figure 5.5b these four are separated from the other features. In the same figure, the next four features are also separated from the remaining features. So it is clear that those are also important. This concerns the following features: `flexibility: hours', `target price/distancemeters', `location: country' and `type of action'. These are also the next four most important features in Figure 5.5a, but in a different order. We will focus on these eight features. Additionally, we consider the feature `target price/hours indication', because it appears more important with the Gini importance measure.

publish/cancel time & lead time These can both be influenced, because shippers can change when they publish their shipment.

hours indication & distance The locations that have to be visited cannot be influenced and hence, the hours indication and distance can also not be influenced.

flexibility The actions of a shipment are fixed, but perhaps the available time for an action can be influenced.

target price/hours indication & target price/distance These can be influenced, because a shipper can simply change the target price for his shipment.

location & type of action The locations and actions cannot be influenced.

Based on the feature importance, the features publish/cancel time, lead time, flexibility, target price/hours indication and target price/distance are sensible candidates to help increase the quoting percentage.

Chapter 6

Conclusion

In this chapter we present our conclusion. In Section 6.1 we give directions for further research in the form of services that could be added to the UTURN platform.

In part I we took a look at the data provided by UTURN. The goal was to identify opportunities for improving the matching percentage. In Section 3.1 we studied percentage trees for both the matching and quoting percentage, and we concluded that improving the quoting percentage has the most impact on the overall matching percentage. Hence, we focused on the quoting percentage for the remainder of part I. To gain insight in the relative importance of features based purely on the data, we applied a random forest model to the data provided by UTURN. The model was trained to predict if a shipment is quoted or not, and then we used the permutation importance to determine the importance of features in this prediction. As a result we identi ed the following features as promising candidates to in uence the quoting percentage: publish/cancel time, lead time, exibility, target price/hours indication and target price/distance. We also identi ed the features hours indication, distance, location and type of action as important, however these features cannot be in uenced.

6.1 Directions for Further Research

6.1.1 Advising Shippers

It could be helpful for shippers if a service is added to UTURN that advises them on aspects of shipments, in order to make the shipments more attractive. A prediction model could be used for this; it can give an indication of how attractive a shipment is. Additionally, the prediction model could give feedback on the value of the features of the shipment, and maybe even advise the shipper on which features to change and how, in order to make the shipment more attractive.

Another way to use a prediction model is to send regular updates to shippers, with information about important features: which ones in uence the matching percentage and what are favourable values for those features. Some features might not be adjustable when a shipment is being published. While if a shipper knows beforehand what suitable values are for certain features, he can keep this in mind for all his future shipments. For example, the port in which the container arrives is adjustable, but the moment the shipper is publishing the shipment on the platform, the port is already determined.

For both options, we think the prediction model should pay extra attention to the features that we identi ed as important and that can be in uenced. The publish/cancel time and lead time features can be used to advise shippers on when to publish their shipments. The exibility feature can be used to advise shippers on how much exibility they should try to give their shipments. Both the target price/hours indication and target price/distance feature can be used to advise shippers on what suitable prices are for their shipments. Even though UTURN does not really want to in uence the price (they want to let the market determine the price), it might be useful advice for new shippers that do not yet know what suitable prices are for their shipments.

During our analysis we have additionally identified several interesting questions that may be helpful here.

Shipper personalization What kind of shipments does a shipper publish? Is there a shipper that always publishes popular shipments? Is there a shipper that publishes unpopular shipments? What are the characteristics of the shipments published by a single shipper. This can be used to determine characteristics of a popular shipment, which in turn can be used to advise shippers about their shipments.

Publication activity Consider the number of shipments published at a moment versus the quoting percentage of those shipments, and versus the number of quotes coming in during that time. If a lot of shipments are published at the same time, does that influence if they get quoted or not? For example, can there be too many shipments at a single moment? If so, this can be used to inform shippers about busy periods and advise them to delay the publication of their shipment if possible.

Shipper satisfaction/expectations Consider the quoted shipments that did not get assigned because the shipper was not satisfied with any of the quotes. What characteristics do those shipments have? Why is the shipper not satisfied; are the quoted prices too high, or are the expectations of the shipper not realistic? This can be used to teach shippers what prices to expect for certain shipments.

Shipping lines What are characteristics of a shipping line? How flexible are the action windows at a shipping line? What is the handling time of an action? Are there delays? How often, and how long? Which shipping line gets a lot of quotes? Compare the shipping lines that do well with those that do not. This can be used to advise shipping lines on what they can do better, and to advise shippers on which shipping line they could use to improve the chance of their shipment getting matched.

If this direction is ever explored, the influence of the advice given by a prediction model on the quoting percentage could by investigated by giving indications and improvement suggestions to previous shipments. For shipments that were correctly predicted as `not quoted', do the suggestions for improvements change the prediction to `quoted'? Moreover, we advise to compare different prediction models, instead of only looking at the random forest model, because there might be a prediction model that works better for this application.

6.1.2 Visibility of Shipments

The UTURN platform is growing; more and more shipments are published on the platform. Consequently, it might happen that carriers can no longer see the wood for the trees. In this section we discuss two services that could help increase the visibility of shipments. Both services tackle a different kind of visibility. First we focus on visibility of shipments that could be executed one after another, and secondly we focus on visibility of shipments that fit well with the personal preferences of a carrier.

Combinations of Shipments

To increase visibility of shipments that can be executed one after another, it could be useful to add a service to UTURN that suggests combinations of shipments to carriers. This service can have several forms. For example, we could show several suggestions on the publication page of a shipment of single shipments that can be executed before or after the shipment in consideration. Another option is to give such suggestions directly to a carrier for a shipment that is already assigned to him, or to suggest combinations of more than two shipments, for example as a day-filler.

In Part II we focus on suggesting combinations of two shipments. Both on the publication page of shipments, and to carriers directly.

Presentation Order

We see two ways in which we can increase the visibility of shipments that fit with personal preferences: ordering all published shipments (page rank), or sending a daily top ten, both based on the preferences. For both services there should be a model that learns the preferences of the carriers automatically from the data.

We have identi ed the following questions that may help with carrier personalization. What kind of shipments does a carrier quote, and why? What characteristics do those shipments have? For example, does the carrier have a preferred region? This can be used to in uence which shipments are shown to the carrier. From another perspective: which carrier gets a lot of shipments assigned? What is the rating of such a carrier? Is there a carrier that gives a lot of quotes, but is not that much assigned? This can be used to advise carriers on how to act.

If one wants to implement the daily top ten, we advise to investigate if this can be done in a balanced way. That means investigate if it is possible to distribute the shipments over all top tens such that all shipments appear in approximately the same amount of top tens.

6.1.3 Acceptance Price

UTURN does not want to decide which carrier is assigned to a shipment; that is the choice of the shipper. However, if the acceptance price is enabled by the shipper, he gives up this freedom of choice. In that case we might be able to in uence which carrier gets assigned to the shipment. We see a possibility to transform this into a service as follows: carriers get the option to `tentatively' accept multiple shipments, such that UTURN can determine an overall assignment that maximizes the number of assignments. However, for this to work, carriers must use this option. Therefore, each carrier that `tentatively' accepted multiple shipments, must get a guarantee that at least one shipment is assigned to him. Otherwise, it would be more bene cial for carriers to immediately accept a shipment.

An example of a situation in which this service would be helpful: there are two shipments A and B with acceptance price enabled. There are two carriers x and y. Carrier x can do either A or B, and carrier y can only do A. But x picks a shipment rst, and chooses A. Then carrier y can do nothing and shipment B will not be matched. However, there is clearly a better solution in which both carriers can execute a shipment and in which both shipments are matched.

To know if this will really be helpful, we advise to investigate if situations like in the example described above occur. We expect that this might be helpful when there are not enough carriers, and not enough shipments. Then it might really happen that such a shipment B never gets accepted, and that such a carrier y does not nd a replacement shipment. However, in the case that there are a lot of carriers and shipments, there will most of the time be another carrier/shipment to ll the gap. In other words, as UTURN grows in number of shipments and number of carriers, this service becomes less attractive.

Part II

Combinations of Shipments

Chapter 7

Introduction

In Part II we focus on finding combinations of shipments that can be executed one after another. We want to use this to suggest such combinations to carriers. We envision the suggestions as follows.

- For a published shipment: on the publication page of the shipment we suggest other published shipments that can be executed before or after the shipment in consideration.
- For an assigned shipment we see a few options, all for the carrier corresponding with the shipment:
  1. There is still a page for the assigned shipment, now only available to the corresponding carrier, here we suggest published shipments that can be executed before or after the shipment in consideration.
  2. The suggestions can be displayed on the dashboard of the carrier.
  3. The carrier receives a push notification with suggestions from the mobile UTURN app.
  4. The carrier receives an email with suggestions.

Remark 2

If a published shipment is used in a suggestion and if at some point that shipment is not available/published anymore, it should be removed from all suggestion overviews. This does not influence the way we find combinations and suggestions, but it should be checked after the suggestions are made.

7.1 Notation

In this section we introduce some useful notation that will be used in the remainder of this thesis.

When considering a potential combination of two shipments A and B, we say that A is the shipment to be executed first and B is the shipment to be executed second. For any shipment S let d_S be the distance travelled during S and let t_S be the hours indication of S. Furthermore, for any two shipments A and B, let d_{AB} be the distance between the last location of A and the first location of B and let t_{AB} be the travel time between those locations. Let T denote the time available between the start time of the time frame available for the last action of A and the start time of the time frame available for the first action of B. In addition, we let T_{max} be the maximal time available between A and B, so from the start time of the time frame available for the last action of A to the end time of the time frame available for the first action of B. Finally, let p_S denote the target price of shipment S, set by the shipper.

We also need some constants that are the same for all combinations of shipments. Let t_{pu} and t_d denote the estimated time needed for a pick-up and a drop action, respectively. These are both set to 0.75 hours, which is also how they are used by UTURN. Furthermore, we denote the cost per hour and cost

per kilometer by c_h and c_{km} , respectively. The cost per hour for example includes the hourly wage of the truck driver and the cost per kilometer for example includes the price of gasoline. For these constants, we advise to take the average cost that carriers have per hour and per kilometer.

Chapter 8

Scoring Function for Combinations

In this chapter we discuss examples for two types of scoring functions that are used to score combinations of shipments: a cost function in Section 8.1 and a pro t function in Section 8.2. In Section 8.3 we discuss some constraints that are used to check the feasibility of a combination of two shipments. Finally we discuss the computation time of the scoring function in Section 8.4.

8.1 Cost function

We derive the extra costs (EC) as the costs that occur during the unpaid time between two shipments. This includes the costs for driving between A and B, plus the costs for the waiting time. The waiting time is given by $\max(0; T - t_d)$, since T is the time you have to wait for shipment B to start, but it also includes the drop action of shipment A, which is not waiting time. Then we take the maximum of 0 and $T - t_d$, because the latter can be negative; the time frame available for the rst action of B can start before the time frame available for the last action of A. If that is the case, the waiting time for the carrier is zero. We de ne the extra costs as follows:

$$EC(A; B) = c_{km} \cdot d_{AB} + c_h \cdot \max(0; T - t_d): \tag{8.1}$$

Note that all terms in the function are nonnegative, so the whole function is nonnegative.

8.2 Pro t function

Here we focus on the pro t generated by the shipments minus the extra costs during the unpaid time. We de ne the pro t of the shipments as follows:

$$P(A; B) = p_A + p_B - c_{km} \cdot (d_A + d_B) - c_h \cdot (t_A + t_B): \tag{8.2}$$

Then we de ne the overall pro t function as:

$$P(A; B) - EC(A; B) = p_A + p_B - c_{km} \cdot (d_A + d_{AB} + d_B) - c_h \cdot (t_A + \max(0; T - t_d) + t_B): \tag{8.3}$$

Note that with this pro t function a higher payment results in a better score. This may appear bene cial: if a carrier gets paid more for the shipments, the extra costs might matter less to him. However, we are more interested in nding attractive combinations based on the properties that associate with the combination itself; the payment alone does not say anything about if two shipments can be combined. Hence, we prefer the cost function.

As an alternative idea for the pro t function we could consider the gain in pro t instead of the total pro t: this would cancel out the payment. We de ne the gain in pro t as the di erence in pro t between when a carrier executes shipments A and B separately, and when he executes them combined. However, we have not identi ed a sensible manner to express the gain in pro t based on the available information. Hence, we do not further consider this option.

8.3 Constraints

Not all combinations of shipments are feasible. We filter out infeasible combinations using Constraints 1 to 4. We set the score of combinations that do not satisfy the constraints to -1. Since we use an otherwise nonnegative scoring function, we can filter out the infeasible combinations by only selecting combinations with a nonnegative score.

Constraint 1

The suggested shipment cannot be an assigned shipment.

Constraint 2

Bound on distance:

$$d_{AB} \quad \begin{array}{ll} \max(D;\ {}_s\, d_A;\ {}_r\, d_B) & \text{if } A \text{ is the suggested shipment,} \\ \max(D;\ {}_s\, d_B;\ {}_r\, d_A) & \text{if } B \text{ is the suggested shipment.} \end{array} \tag{8.4}$$

This constraint sets a bound on the distance that can be travelled between shipments A and B. The bound is the maximum of three terms. First we have the parameter D, this ensures that it is always allowed to have a short distance between the shipments even if the shipments themselves are short. Secondly it is bounded by a fraction ($_s$) of the suggested shipment, and thirdly it is bounded by a fraction ($_r$) of the shipment that receives the suggestion. We make this distinction because the distance a carrier is willing to travel for a shipment depends on which shipment is the suggested shipment. For example: let shipment A be a short shipment within Amsterdam and let shipment B be a long shipment from Rotterdam to Spain. If a carrier is looking at shipment A, then shipment B is considered as an attractive suggestion; driving from Amsterdam to Rotterdam for a long shipment seems worthwhile. However, if a carrier is looking at shipment B, then shipment A is not considered as an attractive suggestion; the carrier is planning on going to Rotterdam for a long shipment, so why would he detour for a short shipment in Amsterdam? In the particular case that the carrier is currently close the Amsterdam this is different, but in general that will not be the case.

Constraint 3

Achievability of time:

$$T_{max} \quad t_d + \max(\quad t_{AB}\ ; a) + t_{pu}: \tag{8.5}$$

This constraint is to make sure that the combination is feasible in terms of available time. That means the maximum available time between the shipments (T_{max}) must at least be enough to execute the drop action of A, travel from the end of shipment A to the start of shipment B and to execute the pick-up action of B. For the time needed to travel from the end of A to the start of B we take $\max(\quad t_{AB}\ ; a)$. This includes an achievability time a, such that even if the last location of A and the first location of B are the same, the carrier still has some spare time, e.g. for a break. Furthermore we multiply the travel time by a factor $\quad 1$, introducing some slack to accumulate for e.g. delays.

Constraint 4

Efficiency of time:

$$T \quad t_d + " \quad t_{AB} + e: \tag{8.6}$$

This constraint ensures that the combination is efficient, i.e., to make sure there is not too much

time between the shipments. The time T completely covers the time frame available for the drop action of A, therefore t_d is included in the bound. Furthermore, the factor $\alpha - 1$ ensures there is not too much slack, and the waiting time ϵ bounds the time available for breaks.

8.4 Computation Time

We determine the score of a combination by first checking the constraints, if at least one is violated the score is set to -1, otherwise the score is calculated using the cost function. Checking the constraints and calculating the cost function can be done in constant time for a single combination. However, for the computation time we should also take into account the computation of the properties of the shipments to be combined. For most properties their value can be easily computed in constant time. The `problematic' properties are d_{AB} and t_{AB}; the distance and travel time between the last location of A and the first location of B. We use CQMaps (provided by CQM) to determine both values of these properties at the same time.

We determine d_{AB} and t_{AB} for all possible shipment combinations at the same time. According to UTURN we should count on about 1000 shipments being published/assigned at the same time. We need the start and end locations of all these shipments, so 2000 possible locations. However, it turns out that a lot of these locations are the same. Based on a look at the available data, we can assume there are at most 120 different locations for which we need the distance and travel time information at the same time.

CQMaps is based on Customizable Route Planning (CRP) [3]. CQM believes CQMaps is currently one of the fastest programs to calculate distance matrices, and for the number of different locations in the hundreds, it will be fast. For our instances the matrix computation time was 3:5 seconds for approximately 100 unique locations.

Note that each time we want to determine new suggestions, the distance and travel times between all locations are required. If the UTURN platform grows and the number of locations increases to the point were the matrix computation takes too long, it would be wise to re-use previously determined distances and travel times, to reduce computation time.

Chapter 9

Balanced Suggestions

In this chapter we explain how we generate suggestions in a balanced way. In Section 9.1 we explain the general idea. Then we formulate the idea as a mathematical model in Section 9.2 and we describe how we solve this model in Section 9.3. Finally in Section 9.4 we explain how to interpret the solution.

9.1 General Idea

Using the scoring function of Chapter 8 we could simply give each shipment their top k best scoring combinations as suggestions. However, in that case it might happen that some shipments appear in a lot of top ks and other shipments appear nowhere. We would prefer a more balanced situation where every shipment is suggested a few times. We try to achieve this by using b-matching, which means we try to suggest each shipment (at most) b times. In a general b-matching, b is a function on the vertices, but in our application it will have the same value for each vertex. We want to find as many suggestions as possible, hence we want to find a maximum cardinality b-matching.

We also want to bound the number of suggestions a shipment receives because the overview of suggestions must remain clear; it should not contain too many suggestions as that may result in carriers not being able to see the forest for the trees. Note that the number of suggestions is bounded both when we just suggest the top k, and when we use a b-matching.

The set of published shipments is continuously changing, which we want to reflect in the given suggestions. Therefore we periodically construct a new b-matching, for example after every ten minutes, or after every twenty new published shipments, or after every twenty published shipments that are not available anymore. Note that the time interval should be at least as long as it takes to calculate the matching. Since the matching is used for suggestions, which are not permanent, we do not have to adhere to (parts of) the previously generated matching; we can construct a completely new matching.

We chose a cost function as the weight function on the edges of the graph in which we want to find a b-matching. The cost indicates the attractiveness of combinations and a lower cost means the combination is more attractive. Hence we minimize the weight of the matching, such that the suggestions are as attractive as possible. We consider two options for this: minimize the total weight used or minimize the greatest weight used. The latter is called a bottleneck matching. If we minimize the total weight, a few combinations may turn out very costly, such that all the other combinations can have low costs. This is a problem, because we want all combinations to have low cost. With a bottleneck matching this problem is fixed. However, a downside of this method is that if there is one costly combination that you have to take, then there is no incentive to choose low-cost combinations. A combination of both options may exploit both mechanisms: first minimize the largest weight used in the matching, lets say that weight is W. Then minimize the total weight used in the matching, over all edges that have a weight of at most $(1 + \varepsilon_W) \cdot W$, for some $\varepsilon_W > 0$.

The bottleneck and minimum total weight objective clash with the maximum cardinality objective: an optimal solution to either a bottleneck or minimum weight matching (without any constraint on the

cardinality) is an empty matching. We solve this conflict by adding a lowerbound on the required cardinality of the matching as follows. We first solve the problem of maximizing the cardinality and use the corresponding solution as a constraint. Suppose we obtain a maximum cardinality of C from this step, then the total cardinality has to be at least $(1 - \varepsilon_C) \cdot C$, for some $\varepsilon_C > 0$. We call this the cardinality constraint. Note that this means we need to solve three separate problems. First we maximize the cardinality, which will serve as input for the next problem. Second we minimize the greatest weight used, which, together with the maximal cardinality of the first step, will serve as input for the final step. At last we minimize the total weight used. In Section 9.2 these three problems are reflected in Models 1 to 3, respectively.

9.2 Mathematical Formulation

We construct two undirected, bipartite graphs $G_1 = (A \cup B, E_1)$ and $G_2 = (A \cup B, E_2)$. For each shipment v there is a vertex v_A in A and a vertex v_B in B. The shipments in A are seen as the shipments to execute first, and the shipments in B as the shipments to execute next. In G_1 the shipments in A are the suggested shipments, and the shipments in B receive the suggestions. For G_2 it is the other way around. The edges indicate a combination of the respective shipments. Note that there will be no edges between two vertices corresponding with the same shipment, i.e. $\{v_A, v_B\} \notin E_1, E_2$. We use the previously discussed cost function EC as weight function on the edges. Let EC_1 be the cost function EC where the first input (A) is seen as the suggested shipment, and likewise for EC_2 and the second input (B). Then we define the weight function on E_i as $w_i(u_A, v_B) = EC_i(u, v)$ for any $u_A \in A$ and $v_B \in B$ if $u \neq v$, else $w_i(v_A, v_B) = -1$, for $i = 1, 2$. Then we have that $w_i < 0$ only for combinations that violated the constraints, and for combinations of two vertices corresponding with the same shipment. Hence, we only use edges with $w_i \geq 0$.

The way in which we solve the matching is the same for both G_1 and G_2. Therefore, from now on we talk about one general graph G and in Section 9.4 we translate the results back to G_1 and G_2.

Let $G = (A \cup B, E)$ be the graph in consideration. Let $b : A \cup B \to \mathbb{N}$ be a function on the vertices. Let $w : E \to \mathbb{R}_{\geq 0}$ be a weight function on the edges. Let $x_{ij} \in \{0, 1\}$ indicate if edge $\{i, j\} \in E$ is used in the matching. For $v \in A \cup B$, $N(v)$ is the set of neighbour vertices of v in G. First we look for a maximum cardinality b-matching, see Model 1.

Model 1

$$\text{Maximize} \quad \sum_{\{i,j\} \in E} x_{ij} \tag{9.1a}$$

$$\text{Subject to:} \quad \sum_{j \in N(i)} x_{ij} \leq b(i) \text{ for all } i \in A \cup B \tag{9.1b}$$

$$x \in \{0, 1\}^E \tag{9.1c}$$

Let C be the optimal objective value of Model 1. We add the following bottleneck objective: minimize a value W such that $w(i, j)x_{i,j} \leq W$ for all edges $\{i, j\}$. Furthermore, let $\varepsilon_C > 0$ and add the cardinality constraint, see Model 2.

Model 2

$$\text{Minimize } W \tag{9.2a}$$

$$\text{Subject to: } w(i,j)x_{i,j} \quad W \text{ for all } \{i,j\} \in E \tag{9.2b}$$

$$\sum_{j \in N(i)} x_{ij} \quad b(i) \text{ for all } i \in A \cup B \tag{9.2c}$$

$$\sum_{\{i,j\} \in E} x_{ij} \quad (1 \quad \varepsilon_C) \quad C \tag{9.2d}$$

$$x \in \{0,1\}^E, W \in \mathbb{R}_{\geq 0} \tag{9.2e}$$

Let W be the optimal objective value of Model 2 and let $\varepsilon_W > 0$. De ne a new edge set

$$E^{(1)} := \{\{i,j\} \in E \mid w(i,j) \quad (1+\varepsilon_W) \quad W\}. \tag{9.3}$$

For $v \in A \cup B$, $N^0(v)$ is the set of neighbour vertices of v in $G^0 = (A \cup B, E^{(1)})$. Now we use the minimum weight objective over G^0, to formulate Model 3.

Model 3

$$\text{Minimize } \sum_{\{i,j\} \in E^{(1)}} w(i,j)x_{ij} \tag{9.4a}$$

$$\text{Subject to: } \sum_{j \in N^0(i)} x_{ij} \quad b(i) \text{ for all } i \in A \cup B \tag{9.4b}$$

$$\sum_{\{i,j\} \in E^{(1)}} x_{ij} \quad (1 \quad \varepsilon_C) \quad C \tag{9.4c}$$

$$x \in \{0,1\}^{E^{(1)}} \tag{9.4d}$$

For each model there is at least one feasible solution. For Model 1 the all zero vector is feasible. Let x be an optimal solution for Model 1, then this is a feasible solution for Model 2 in combination with $W = \max_{\{i,j\} \in E} w(i,j)$. Indeed, constraint (9.2b) is satis ed because W is the maximum possible value. Constraints (9.2c) and (9.2e) are satis ed because x is feasible for Model 1, and since x is an optimal solution for Model 1, we have

$$\sum_{\{i,j\} \in E} x_{ij} = C \quad (1 \quad \varepsilon_C) \quad C; \tag{9.5}$$

and hence constraint (9.2d) is satis ed. Now let x be an optimal solution for Model 2. Since x is optimal, we have

$$w(i,j)x_{ij} \quad W \quad \text{for all } \{i,j\} \in E. \tag{9.6}$$

That means that each edge $\{i,j\}$ for which $x_{ij} = 1$ is in $E^{(1)}$, and hence, each edge $\{i,j\}$ that is not in $E^{(1)}$ satis es $x_{ij} = 0$. As a result x can be converted to a feasible solution for Model 3 by leaving out the entries x_{ij} for edges $\{i,j\}$ that are not in $E^{(1)}$.

9.3 Solve the Matchings

In this section we describe how we solve each of the models described in Section 9.2. We solve them by reformulating them as ow problems. The matching we are interested in consists of all edges $\{v_A, v_B\}$ corresponding with arcs (v_A, v_B) in the ow network that have a ow of 1, for $v_A \in A$ and $v_B \in B$.

9.3.1 Model 1

We reformulate Model 1 as a maximum flow problem and solve it by using the Ford-Fulkerson algorithm; in particular, we use the Edmonds-Karp implementation. Note that this is an implementation of the Ford-Fulkerson algorithm with guaranteed termination. Denote the capacity of an arc (i,j) by $c(i,j)$. We define the directed network $N = (A \cup B \cup \{s,t\}, E^{(2)})$, where

- s and t are the source and the sink of the flow network, respectively;
- $E^{(2)}$ contains the edges in G as arcs directed from A to B;
- $(s, v_A) \in E^{(2)}$ for all $v_A \in A$ and $(v_B, t) \in E^{(2)}$ for all $v_B \in B$;
- $c(v_A, v_B) = 1$ for all $v_A \in A$, $v_B \in B$ such that $(v_A, v_B) \in E^{(2)}$;
- $c(s, v_A) = b(v_A)$ for all $v_A \in A$;
- $c(v_B, t) = b(v_B)$ for all $v_B \in B$.

Note that the capacities $c(s, v_A)$ and $c(v_B, t)$ guarantee we satisfy constraint (9.1b).

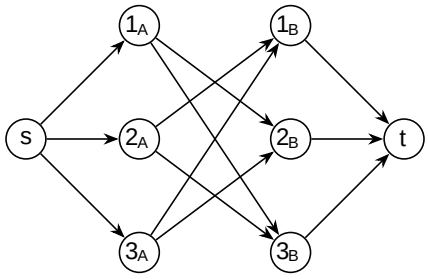


Figure 9.1: Example of the created network with $|A| = |B| = 3$.

The value of the maximum flow in N equals the size of a maximum cardinality matching in G. The complexity of Ford-Fulkerson is bounded by $O(E^{(2)} f_{max})$ [11, p.725] when the capacities are integer, where f_{max} is the maximum flow value and comes from the total number of flow augmentations performed. For the Edmonds-Karp implementation the total number of flow augmentations performed is also bounded by $O(|A \cup B| E^{(2)})$ [11, thm 26.8]. So we have two bounds on the total number of flow augmentations performed: f_{max} and $O(|A \cup B| E^{(2)})$. In our case f_{max} is bounded by the total capacity of the edges between A and B, which is $|E|$. Hence, the f_{max} bound is more tight. Furthermore, in our particular case we have $|A| = |B|$, but for now we assume, w.l.o.g., $|A| \leq |B|$. Then we have

$$E^{(2)} f_{max} = (|E| + |A| + |B|) f_{max} \leq (|E| + 2|A|)|E| = O(|E|(|E| + |A|)): \tag{9.7}$$

9.3.2 Model 2

For Model 2 we also use the reformulation described above. Let w_{max} be the maximum edge weight in G and let C be the output of the algorithm that solves Model 1. Then we use the following algorithm to solve Model 2.

Algorithm 2

Input : a graph $G = (A \cup B, E)$, a function $b : A \cup B \to \mathbb{N}$ on the vertices, a weight function $w : E \to \mathbb{R}_0$ on the edges, w_{max}, C and ε_C.
We start the algorithm with $l = 0$ and $u = w_{max}$.

1: $S \leftarrow \{w(i,j) \mid \{i,j\} \in E, l \leq w(i,j) \leq u\}$ ▷ l not a multiset
2: **if** $|S| = 1$ **then**
3: **return** the single element that is in S.
4: **end if**

5: $m \leftarrow \text{median}(S)$
6: $E' \leftarrow \{\{i,j\} \in E \mid w(i,j) \geq m\}$
7: Construct network N for the graph $(A \cup B, E')$, as described above.
8: Find the maximum flow value F in N using the Ford-Fulkerson algorithm.
9: if $F < (1 - \varepsilon_C) \cdot C$ then
10: $l \leftarrow m$
11: Repeat from step 1.
12: else
13: $u \leftarrow m$
14: Repeat from step 1.
15: end if

Remark 3

We take the median of the set S, and we investigate if the median can be used as bottleneck weight. It is unnecessary to investigate more than once if a single weight can be used as bottleneck weight. Hence, it is unnecessary to have weights appear more than once in S. Therefore, we created S as not a multiset, i.e., S only contains unique weights.

Note that Algorithm 2 finds the maximum flow in each step, which corresponds with a maximum matching when considering the same edge set, but it does not have to correspond with a maximal matching when considering all edges. Since we take the median of S, we half the number of elements we need to check in each step. The set S contains all unique weights, so at the start S contains at most $|E|$ elements. Hence, the algorithm will do at most $\log_2 |E|$ recursions. The complexity of Algorithm 2 comes from executing the Ford-Fulkerson algorithm on line 8. So the total complexity of this algorithm is $O(|E| \cdot (|E| + |A|) \cdot \log|E|)$.

9.3.3 Model 3

Let W be the output of Algorithm 2. Then we define the edge set $E^{(1)}$ for Model 3 as given by Equation (9.3). We reformulate the problem of Model 3 as a minimum weight flow problem. We construct a directed network N similar to the one of Model 1, only now we additionally use arc weights: any arc (u_A, v_B) gets weight $w(u_A, v_B)$ for $u_A \in A$, $v_B \in B$. All other arcs get weight zero. Let $E^{(3)}$ be the new set of arcs.

To solve Model 3 we need an extra property as given in Theorem 4 from [6]:

> If f is extreme and P is a path of minimum weight in N^f from s to t, then a flow f' obtained by augmenting along P is extreme.

We call a flow extreme if it has minimum weight among all flows with the same flow value. We need an auxiliary network N^f to find augmenting paths. This network N^f is constructed as follows: if flow f over arc (i,j) is less than the capacity of that arc, then (i,j) is added to N^f with its corresponding weight $w(i,j)$. If flow f over arc (i,j) is positive, then (j,i) is added to N^f with weight $-w(i,j)$.

We solve Model 3 with Algorithm 3, which is based on the following idea: start with the zero flow, and iteratively augment along s - t paths P of minimum weight. This algorithm can terminate when no augmenting paths can be found, and hence when a maximum flow is found. However, it might happen that to reach this maximum flow, attractive combinations have to be swapped for unattractive combinations, while we prefer to keep the attractive combinations. That is why we want to stop the algorithm sooner. We have the following idea for terminating the algorithm: run the algorithm until the value of the flow reaches the lowerbound on the cardinality of the matching. Note that this is possible, since there is at least one feasible solution to Model 3, as mentioned in Section 9.2. From this moment we consider the ratio between the increase in weight and the increase in flow. When this ratio reaches a certain threshold value ε, the algorithm terminates.

Remark 4

The solution we find with Algorithm 3 differs from the optimal solution for Model 3, because there the main goal is to minimize the total weight. In that case it is optimal to stop increasing the matching as soon as the lower bound on the cardinality is reached.

Let f_i, F_i and W_i be the flow, the value of the flow and the total weight of the flow in step i of the algorithm, respectively. Recall that we are dealing with integer capacities and that the capacities on all arcs between A and B are one. Hence the flow will be increased by one each step.

Algorithm 3

Input : a network $N = (A \cup B \cup \{s, t\}, E^{(3)})$ with arc weights and capacities, C and ε_C.

1: $i \leftarrow 0$
2: Set f_0 equal to the zero flow, so $F_0 = 0$.
3: repeat
4: Construct N^{f_i}.
5: Determine a minimum weight s t path P in N^{f_i}, using the Bellman-Ford algorithm.
6: if there is no path P then
7: return f_i
8: end if
9: Create f_{i+1} by augmenting f_i along this path with value 1.
10: $F_{i+1} \leftarrow F_i + 1$
11: $i \leftarrow i + 1$
12: until $F_{i-1} \geq (1 - \varepsilon_C)\, C$ and $(W_i - W_{i-1})/(F_i - F_{i-1})$
13: return f_{i-1}

To explain line 12 and 13 in words: if the previous flow is large enough and the increment to the next flow weighs too much, we return the previous flow. In Proposition 1 we show that $(W_i - W_{i-1})/(F_i - F_{i-1})$ is nondecreasing, which means we do not miss out on a better solution when the algorithm terminates. We use the Bellman-Ford algorithm to find a minimum weight s t path, since arc weights in N^{f_i} can be negative, but N^{f_i} does not contain negative weight cycles, see Proposition 2. Note that N^{f_i} has the same vertices as N, and can have each arc of N in both directions. Recall our assumption that $|A| \leq |B|$ and recall that $|E^{(3)}| = |E^{(1)}| + |A| + |B|$ due to how we constructed N. Then N^{f_i} has $O(|A| + |B|) = O(|A|)$ vertices and $O(|E^{(3)}|) = O(|E^{(1)}| + |A|)$ arcs. The complexity of the Bellman-Ford algorithm is $O(|A| \cdot |E^{(1)}| + |A|))$ [11, p.651]. Regardless of the stopping criterion, in the worst case scenario Algorithm 3 only stops when the maximum flow is reached. Hence, the algorithm needs at most f_{max} steps, which is bounded by $|E^{(1)}|$. Thus the complexity of Algorithm 3 is $O\left(|E^{(1)}| \cdot |A| \cdot (|E^{(1)}| + |A|)\right)$.

Proposition 1

In Algorithm 3, the additional cost per flow $(W_i - W_{i-1})/(F_i - F_{i-1})$ is nondecreasing.

Proof. We prove this proposition by formulating a suitable parameterized linear program, arguing that the linear program is convex in its parameter, and showing that this implies $(W_i - W_{i-1})/(F_i - F_{i-1})$ is nondecreasing.

We want to formulate a linear program $LP(q)$ such that the optimal objective value is W_i if $q = F_i$. In order to get this, the optimal solution should correspond with an extreme flow that has flow value q.

First of all, to get an extreme flow, we minimize total weight. Let $x_{ij} \geq 0$ indicate the amount of flow over arc (i, j), let x be the vector of x_{ij}'s, and let w be the vector of weights $w(i, j)$. Then the objective is to minimize $w^{\top} x$.

Second of all, the value of the ow should beq. Therefore the net out ow in s should be q and the net out ow in t should be q. Moreover, the net out ow of all other vertices should be zero. The net out ow is calculated by summing the ow over all outgoing arcs, minus the ow over all incoming arcs. We de ne the incidence matrix A as

$$A_{ie} = \begin{cases} 1 & \text{if } e = (i, :); \\ -1 & \text{if } e = (:, i); \\ 0 & \text{if } i \text{ is not an endpoint of } e; \end{cases} \tag{9.8}$$

such that the rst and second row of A correspond with s and t, respectively. Then we can formulate these constraints asAx = q for $q = (q, -q, 0, \ldots, 0)^T$.

Third of all, all capacities should be satis ed: $x_{ij} \le c(i,j)$, or equivalently $x_{ij} + z_{ij} = c(i,j)$ for a slack variable $z_{ij} \ge 0$. Let z be the vector of z_{ij}'s, and let c be the vector of capacities $c(i,j)$. Then we can formulate this constraint as x + z = c.

Finally, we de ne

$$A' = \begin{pmatrix} A & O \\ I & I \end{pmatrix}; \quad b = \begin{pmatrix} q \\ c \end{pmatrix}; \quad x' = \begin{pmatrix} x \\ z \end{pmatrix}; \text{ and } w' = \begin{pmatrix} w \\ 0 \end{pmatrix}; \tag{9.9}$$

where I is an identity matrix, O is an all zero matrix and 0 is an all zero vector, all of appropriate size. We formulate the linear program $LP : [0, f_{max}] \to \mathbb{R}$ as

$$LP(q) = \min \{ w'x' \mid A'x' = b, x' \ge 0 \}. \tag{9.10}$$

By construction we have that the optimal solution to $LP(F_i)$ corresponds with an extreme ow that has value F_i. The weight of such an extreme ow must be W_i, and hence we have $LP(F_i) = W_i$. Then, by strong duality we have

$$LP(q) = \max \{ yb \mid yA' \le w' \}, \tag{9.11}$$

for a row vector y. Let y be an optimal dual vector, such that LP (q) = y b and $y A' \le w'$. The inequality is independent of q, hence y is a feasible solution to LP (q) for any q.

Consider $q = \lambda_1 q_1 + \lambda_2 q_2$, for $\lambda_1, \lambda_2 \ge 0$, $\lambda_1 + \lambda_2 = 1$ and some q_1, q_2. Since b depends linearly on q, we can also write $b = \lambda_1 b_1 + \lambda_2 b_2$, for suitable b_1 and b_2. Then we have

$$LP(\lambda_1 q_1 + \lambda_2 q_2) = y(\lambda_1 b_1 + \lambda_2 b_2) = \lambda_1 y b_1 + \lambda_2 y b_2 \le \lambda_1 LP(q_1) + \lambda_2 LP(q_2); \tag{9.12}$$

which implies that LP (q) is convex in q.

Let $\lambda_1 = (F_{i+1} - F_i)/(F_{i+1} - F_{i-1})$ and $\lambda_2 = (F_i - F_{i-1})/(F_{i+1} - F_{i-1})$. Then $\lambda_1 + \lambda_2 = 1$, and λ_1 and λ_2 are both positive since $F_{i-1} < F_i < F_{i+1}$. Moreover, we have

$$\lambda_1 F_{i-1} + \lambda_2 F_{i+1} = \frac{F_{i+1} - F_i}{F_{i+1} - F_{i-1}} F_{i-1} + \frac{F_i - F_{i-1}}{F_{i+1} - F_{i-1}} F_{i+1} = F_i. \tag{9.13}$$

Then by convexity of LP (q):

$$LP(F_i) = LP(\lambda_1 F_{i-1} + \lambda_2 F_{i+1}) \le \lambda_1 LP(F_{i-1}) + \lambda_2 LP(F_{i+1}). \tag{9.14}$$

Substituting $LP(F_i) = W_i$ and lling in the 's:

$$W_i \le \frac{F_{i+1} - F_i}{F_{i+1} - F_{i-1}} W_{i-1} + \frac{F_i - F_{i-1}}{F_{i+1} - F_{i-1}} W_{i+1}. \tag{9.15}$$

Multiplying everything by $F_{i+1} - F_{i-1} (> 0)$:

$$(F_{i+1} - F_{i-1}) W_i \le (F_{i+1} - F_i) W_{i-1} + (F_i - F_{i-1}) W_{i+1}. \tag{9.16}$$

Moving around some terms and subtracting $F_i W_i$ from both sides:

$$(F_i - F_{i+1}) W_{i-1} + (F_{i+1} - F_i) W_i \le (F_i - F_{i-1}) W_{i+1} + (F_{i-1} - F_i) W_i. \tag{9.17}$$

Finally, dividing both sides by $F_{i+1} - F_i (> 0)$ and by $F_i - F_{i-1} (> 0)$ gives the result:

$$\frac{W_i - W_{i-1}}{F_i - F_{i-1}} \le \frac{W_{i+1} - W_i}{F_{i+1} - F_i}. \tag{9.18}$$

□

Proposition 2

Given an extreme s t ow f , the auxiliary network N^f has no negative weight cycles.

Proof. In N^f there are normal and reversed arcs. The normal arcs have non-negative weight and correspond with an original arc that has a ow value smaller than its capacity. The reversed arcs have non-positive weight and correspond with an original arc that has a positive ow value.

Assume there is a negative cycle C in N^f. Let F_n be the minimum gap between the ow and capacity over the normal arcs on C and let F_r be the minimum ow value over the reversed arcs on C. Note that both F_n and F_r are positive. Then we increase the ow on the normal arcs by $F_{min} := \min(F_n; F_r) > 0$ and lower the ow on the reversed arcs by F_{min}, creating a new ow f^0. By construction of F_{min}, we have that the ow f^0 over any arc is nonnegative and does not transcend its capacity.

If s or t is on C they must be in between a normal and a reversed arc, since in the original graph they only have leaving (s) or incoming (t) arcs. The ow is increased by F_{min} on the normal arc and decreased by F_{min} on the reversed arc. All in all, the out ow of s/in ow of t is unchanged. Clearly, if s/ t is not on C, then the out ow of s/in ow of t is also unchanged.

Consider some vertex v, di erent from s and t. If v is not on C, the ow over v does not change, so the net ow remains zero. Now suppose v is on C. If v is inbetween two reversed (normal) arcs, the ow on both arcs is lowered (increased) by F_{min}. That means that both the in- and out ow of v are lowered (increased) by F_{min}. Hence the net ow of v is unchanged, and remains zero. If v is inbetween a reversed and a normal arc (in either order), the situation is similar as described for s and t. The reversed and normal arc both correspond to either incoming arcs or outgoing arcs. If both arcs correspond with incoming arcs, only the in ow is changed, but the changes cancel out: the in ow is both increased and decreased by F_{min}. Similarly if both arcs correspond with outgoing edges. Hence the net ow of v is unchanged, and remains zero.

Let C_r be the set of reversed arcs on C and let C_n the set of normal arcs on C. Furthermore, let C_r and C_n be the corresponding original arcs. Then we have

$$0 > w(C) = w(C_r) + w(C_n) = -w(C_r) + w(C_n) = -) \quad w(C_r) > w(C_n): \tag{9.19}$$

For the new ow we have

$$\begin{aligned} w(f^0) &= w(f) \quad F_{min} \quad w(C_r) + F_{min} \quad w(C_n); && (9.20a)\\ &< w(f) \quad F_{min} \quad w(C_r) + F_{min} \quad w(C_r); && (9.20b)\\ &= w(f): && (9.20c) \end{aligned}$$

To summarise, we have created a ow f^0 that is nonnegative for all arcs, satis es all capacities, and the net ow of all vertices other than s and t is zero. Thus f^0 is a valid ow. Furthermore, both the out ow of s and the in ow of t are unchanged, indicating that the value of f^0 equals the value of f . Finally, the weight of f^0 is strictly less than f . This is in contradiction with the fact that f is extreme. □

Proposition 3

Suppose we have an extreme ow f with total weight $W > 0$ and ow value F . Then there is no ow with total weight W and ow value strictly larger than F .

Proof by contradiction. Assume there is a ow f^0 with weight W and value $F^0 > F$. Let $d := F^0 - F$. Construct N_{f^0} as follows: $(u; v) \in N_{f^0}$ if $f^0(u; v) > 0$ and $(u; v) \in N_{f^0}$ otherwise. Since $W > 0$, there must be at least one s t path P in N_{f^0} with non-zero weight. Let $\varepsilon := \min\{f^0(e) \mid e \in P\}$. Note that by construction of N_{f^0} we have $\varepsilon > 0$. Lower the ow on P by $\min\{d; \varepsilon\}$. Note that this decreases the total weight of the ow. If we were not able to decrease the ow by d, we still have a ow with value

strictly larger than F. That means there must be others s t paths in $N_{f'}$, such that we can lower the ow on those paths to reach a ow with value F. In either case, we are able to lower the ow value to F, which means we have found a new ow with value F and weight strictly less than W. This contradicts with f being extreme. □

Threshold Value

In Algorithm 3 we use as a threshold to terminate Algorithm 3; the algorithm continues as long as

$$\frac{W_i - W_{i-1}}{F_i - F_{i-1}} < \quad (9.21)$$

This can be interpreted as follows: continue increasing the ow while the average weight of a new ow unit is less than . So we see that the unit of is weight per ow unit. However, if we recall our particular application, one ow unit corresponds with one matched edge: one ow unit crosses the three arcs $(s; v_A)$, $(v_A; v_B)$ and $(v_B; t)$, of which only the middle arc has a nonzero weight and corresponds with the matched edge. Hence, we will look for values of with unit weight per edge in G (so not in the auxiliary network N). In a practical sense we can interpret this as follows: continue increasing the ow while the average cost of a new suggestion is less than

We have identi ed the following options for :

- The median of the weights of all combinations (with a nonnegative cost).
- The median of the weights used in the optimal (matching) solution for Model 1. Note that Model 1 nds any maximum cardinality matching, so not necessarily one with minimum weight.
- The median of the weights used in a minimum weight maximum cardinality matching. Such a matching can be found by using Algorithm 3 to solve Model 1, by continuing as longs as we are able to nd s t paths.
- The 75th percentile of the weights used in a minimum weight maximum cardinality matching.

Note that all of these options depend on the current set of combinations. This is bene cial, since it takes into account the dynamicity of available combinations. On days with unattractive combinations, we also allow more unattractive suggestions, and vice versa. Note that carriers might get to see relatively unattractive suggestions on days where all combinations are more unattractive, but these results will be understandable as they can see for themselves that there are not (m)any attractive combinations available.

9.4 Interpretation of the Matching

If vertices u_A and v_B are matched in G_1, we suggest to shipment v that shipment u is a suitable candidate to execute before v. If they are matched in G_2, we suggest to shipment u that shipment v is a suitable candidate to execute after u. If the shipment that receives the suggestion is published, the suggestion is shown on the publication page of that shipment. If the shipment that receives the suggestion is assigned, the suggestion is given directly to the corresponding carrier.

Note that each shipment appears four times: in both vertex sets A and B of graph G_1, and in both vertex sets A and B of graph G_2. That means a shipment can be used 4 times: it can receive and give b `execute before' suggestions and it can receive and give `execute after' suggestions.

Chapter 10

Analysis of Results

This chapter is organized as follows. In Section 10.1 we discuss the implementation of the algorithms discussed in Chapter 9. In Section 10.2 we describe which datasets we used to test our model. Then we explain which value we used for the threshold value in Section 10.3. Finally, we analyse the performance of our model in Section 10.4.

10.1 Implementation

We use Python for the implementation. We used NetworkX (version 25) [7] to create the graphs G_1 and G_2. Vertices without edges do not add any value to the graph, hence we only added vertices with at least one edge. That means that what we claimed in Section 9.4: `each shipment appears four times', is not practically true. Furthermore, we use the Edmonds Karp algorithm from NetworkX to solve Model 1 and we use it in Algorithm 2 on line 8.

We chose parameter values that are representative of what is customary in container transport, see Table 10.1.

Parameter	Value
b	7
$"_C$	0.1
$"_W$	0.1
D	10 km
$_s$	0.25
$_r$	0
	1.5
a	15 min
"	1.5
e	1 hour
c_{km}	1 e/km
c_h	42.50 e/h

Table 10.1: Values of the parameters.

As we can see in Table 10.1 we use Euro for the cost. For some edges his results in a weight with a lot of decimal places. If we implement it with the edge weights as oats, an error can occur in the Bellman-Ford algorithm, when it evaluates if one oat value is strictly smaller than another. We solved this by converting the edge weights to an integer amount of Euro cents. We can do this, since for the edge weight in Euros, actually only the rst two decimal places are relevant.

10.2 Datasets

UTURN provided additional data for testing; the data we can access now spans until half May 2021. We recreated snapshots of the data, that show how the UTURN platform looked at a certain time. That means we have to know exactly when a shipment was published, assigned and executed. This information is in a relatively new table, and therefore we can only use data from April 2020 and later.

From UTURN we learned that the following times are interesting to consider for recreating snapshots:

- Just before Christmas is a very busy time, we can expect a lot of published shipments.
- In comparison, mid November is mostly very quiet.
- Wednesday is the busiest day of the week, concerning publishing activity.
- The weekends are quiet and not that interesting, so we restrict ourselves to only take one dataset of a weekend day.
- In regards to the time of day, the most interesting times for recreating a snapshot are 10h and 14h. We consider di erent times, all around these two times.
- As a consequence of the Suez Canal obstruction in March of 2021, UTURN experienced a very busy time in May, up to 300 shipments were published at a single time.

Based on this information we selected 10 dates between April 2020 and May 2021 for which we created a dataset. In Table 10.2 we show some general information about the datasets: how many shipments there were in total, how many of those shipments were assigned, how many were published, and how many unique locations are covered by all rst and last locations of the shipments. Since all dates of the datasets are unique, we will from now on refer to datasets by only using their date, for convenience.

Dataset			Shipments	Assigned shipments	Published shipments	Unique locations
Date	Time	Day				
2020-04-21	10:00	Tue	213	203	10	49
2020-08-27	13:30	Thu	320	273	47	72
2020-11-16	09:30	Mon	254	208	46	70
2020-12-21	14:00	Mon	448	307	141	81
2021-01-06	09:00	Wed	297	252	45	71
2021-02-14	14:00	Sun	256	192	64	84
2021-03-10	13:00	Wed	458	288	170	103
2021-04-09	10:30	Fri	275	165	110	88
2021-05-04	15:30	Tue	467	195	272	117
2021-05-13	11:00	Thu	420	124	296	99

Table 10.2: General information about the datasets.

10.3 Threshold Value

For the data we have available now, it does not really matter which of the four proposed options for the stopping threshold we use: the lower and upper bound on the cardinality of the nal matching are very close together for all datasets, see Table 10.3. As it does not really matter which value for we take, we just take the rst option we proposed: the median of the weights of all combinations.

In Table 10.3, we determined the lower bound as the right hand side of Equation (9.4c): $(1 - \varepsilon_C) \cdot C$, which equals $0.9 \cdot C$ for $\varepsilon_C = 0.1$. Since a matching consists of an integer amount of edges, we can round this lower bound up. So, the lower bound becomes $\lceil 0.9 \cdot C \rceil$. Furthermore, we determine the upper bound by setting $= 1$, because then Algorithm 3 continuous until the maximum ow is reached, and hence until the maximum cardinality is reached.

Dataset	G_1		G_2	
	lb	ub	lb	ub
2020-04-21	9	9	8	8
2020-08-27	27	27	26	27
2020-11-16	26	28	23	25
2020-12-21	84	87	55	57
2021-01-06	21	21	23	23
2021-02-14	27	29	33	33
2021-03-10	59	62	74	74
2021-04-09	33	35	41	44
2021-05-04	54	60	86	87
2021-05-13	34	37	81	81

Table 10.3: Lower bound (lb) and upper bound (ub) on the cardinality of the matchings in G_1 and G_2.

These lower and upper bounds are so close together because of our choices for ε_C and ε_W : the lowerbound depends directly on ε_C and the upper bound is in uenced by how many edges are removed between Models 2 and 3, which depends on ε_W . Indeed, the bottleneck weight we nd is such that we can just achieve the lowerbound on the cardinality. With $\varepsilon_W = 0.1$ we give too little room for nding a bigger matching. Increasing ε_W would solve this problem, but that also means we allow more unattractive combinations, which might be undesirable. Increasing ε_C would not help, because with still the same value for ε_W the upperbound on the cardinality would also decrease.

10.4 Analysis of the Results

10.4.1 Comparison with Suggesting Top-b's

We compare the results of our model with the case in which we suggest for each shipment their two top-b's: the b best shipments to execute before the shipment in consideration, and the b best shipments to execute afterwards. We consider the following things.

a) The number of top-b's versus the number suggestions we give.

b) The number of published shipments that are not in any top-b, versus the number of published shipments that are not suggested.

c) The number of vertices that do not appear in any of the `receiver' bipartitions, i.e., the number of shipments that do not receive a top-b. Versus the number of shipments that do not receive any suggestions with our model.

d) The number of shipments that appear in the top-b of at least X % of the other shipments (for $X = 5\%; 10\%; 25\%; 50\%; 75\%$).

e) The percentage of the top-b's that is actually just a top-1, just a top-2 or a full top- b.

The results can be seen in Table 10.4.

As expected, when just suggesting the top-b's, there are shipments that appear in a large part of the top-b's (column 2 from Table 10.4a and columns 5-9 from Table 10.4b). While with our model, all shipments are used at most seven times. This is favourable, because if a shipment is suggested a lot, it might be quoted and accepted quickly, which can result in a lot of carriers seeing meaningless suggestions. If such a scenario happens a lot, carriers might loose con dence in the suggestions, as they receive a lot of empty suggestions. That is undesirable.

We also hoped that with a bound on the number of times a shipment can be suggested, the shipments that never appear in a top-b have more chance to be suggested at least once. However, in columns 4-5

Dataset	Top-b's	Suggestions	Published not in top-b	Published not suggested	Received no top-b	Received no suggestions
2020-04-21	67	17	1	1	149	202
2020-08-27	459	53	1	18	36	285
2020-11-16	355	49	10	21	10	222
2020-12-21	769	139	22	79	5	355
2021-01-06	459	44	7	21	5	273
2021-02-14	426	60	12	28	6	222
2021-03-10	801	133	43	89	5	387
2021-04-09	438	74	48	64	0	233
2021-05-04	797	140	113	180	0	393
2021-05-13	777	115	84	209	0	365

(a)

Dataset	Percentage top-b's that is a			Appear in ... of top-b's				
	top-1	top-2	top-b	5%	10%	25%	50%	75%
2020-04-21	29.85%	53.73%	0.00%	9	9	3	2	0
2020-08-27	23.31%	6.10%	32.46%	26	10	4	1	0
2020-11-16	13.80%	24.51%	20.28%	15	11	3	2	0
2020-12-21	8.45%	4.29%	76.07%	46	14	0	0	0
2021-01-06	0.87%	8.50%	54.90%	18	17	8	1	0
2021-02-14	10.80%	8.92%	42.02%	26	16	6	1	0
2021-03-10	2.00%	7.37%	77.90%	45	26	0	0	0
2021-04-09	13.01%	4.11%	38.36%	24	12	4	1	0
2021-05-04	1.76%	1.00%	75.28%	33	12	3	0	0
2021-05-13	1.54%	1.16%	94.47%	50	11	0	0	0

(b)

Table 10.4: Compare our model with top-b's.

from Table 10.4a we see that even more shipments are never used in our model.

In columns 6-7 from Table 10.4a we see that with our model more shipments do not receive any suggestions. This is to be expected, because there are shipments that appear in more than b top-b's and they can only be suggested b times. This is especially difficult if the top-b of a shipment is actually just a top-1 or top-2, and the shipments that are in that top-1/top-2 are in a lot of other top-b's as well. Even though we expected fewer shipments to receive a suggestion, we thought this was a comprise for an increase in the amount of shipments that is suggested at least once. However, in the previous paragraph we saw that this is not the case.

In columns 2-4 from table 10.4b we see that only a small part of the top-b's is just a top-1 or top-2. For four out of the ten datasets, at least three quarters of the top-b's is even a full top-b. It is favourable that the top-b's are full, such that there is enough leeway to identify an attractive set of suggestions.

10.4.2 Shipments Appearance in Graphs

We look at how many shipments are not in any of the two graphs, and at how many shipments are in at least one graph, but are not used. The results can be seen in Table 10.5.

If shipments are not in any graph, it means that they cannot be combined in any way. We see that for dataset 2020-04-21 there are a lot of shipments for which this happens. However, in Table 10.2 we see that this dataset only has ten published shipments and recall that in each combination of shipments at least one of them must be a published shipment. So it is not strange that a lot of shipments cannot be combined with any of those ten. For the remaining datasets we see that the number of published

shipments grows, and the number of shipments that are not in any graph decreases dramatically. Observe that a lot of shipments that are in at least one graph, are not used at all.

Dataset	Not in any graph	In a graph, not used	Average degree	Average number of received suggestions	Di erence of zero
2020-04-21	148	48	2.4	1.5	100.0%
2020-08-27	27	238	5.5	1.5	48.6%
2020-11-16	9	197	5.3	1.5	59.4%
2020-12-21	4	319	20.3	1.5	35.5%
2021-01-06	3	260	7.8	1.8	41.7%
2021-02-14	6	196	7.7	1.8	55.9%
2021-03-10	5	340	20.1	1.9	69.0%
2021-04-09	0	206	9.0	1.8	21.4%
2021-05-04	0	330	28.6	1.9	43.2%
2021-05-13	0	316	33.7	2.1	41.8%

Table 10.5

10.4.3 Potential versus Received Suggestions

For the shipments that have at least one potential suggestion, we consider the average number of potential suggestions, i.e., the average degree of vertices in the `receiver' bipartition. For the shipments that received at least one suggestion, we look at how many suggestions those shipments received on average. The results can be seen in Table 10.5.

Even though there seem to be enough potential suggestions, the average number of actual received suggestions is low. This might be due to some to-be-suggested-shipments being connected to a lot of other shipments, and this can result in a high average degree. But because those shipments can only be suggestedb times, a large part of the shipments that want to receive a suggestion will not be able to receive any suggestion at all. One way to deal with this is to increase the value ofb for the suggested shipments.

An example of a situation in which one shipment can be suggested to a lot of others is the following. Sometimes there are batches of identical shipments published at once. Suppose there are twenty identical shipments, and they can only be combined with one other shipment. Even though they are all equally suitable for the single shipment, only b will get a suggestion. Maybe this problem can be tackled by grouping such identical shipments, such that they are seen as a single shipment that can be assigned twenty times. Then all those shipments get the same suggestion as a group.

10.4.4 Quality of Suggestions

If carriers can easily nd better combinations than the suggestions we give, then giving suggestions is not useful and carriers might ignore them. Therefore we compare for a shipment the best received suggestion with the best possible combination that was available for that shipment. We look at the percentage of shipments that received at least one suggestion, and for which this di erence is zero. The results can be seen in Table 10.5.

We see that the percentage of shipments for which this di erence is zero varies a lot. Some datasets have a quite low percentage. This percentage could be increased by suggesting to each shipment their top-1 or top-2, and by lling the remainder of the top- b with our suggestions. However, this can increase the number of times a shipment is suggested substantially, if the shipment is in the top-1/top-2 of a lot of other shipments.

10.4.5 Shipment Appearance in Suggestions

We look at the percentage of published shipments that is never suggested, suggested once, or suggested more than once. Also, we consider the percentage of shipments (published and assigned) that received

no suggestions, one suggestion, or more than one. The results can be seen in Table 10.6.

Dataset	Suggested ... times			Received ... suggestions			Average cost zero
	zero	one	> one	zero	one	> one	
2020-04-21	10.0%	70.0%	20.0%	94.8%	4.2%	0.9%	54.55%
2020-08-27	38.3%	48.9%	12.8%	89.0%	9.1%	1.9%	17.14%
2020-11-16	45.6%	43.5%	10.9%	87.4%	10.6%	2.0%	21.88%
2020-12-21	56.0%	34.1%	9.9%	79.2%	18.3%	2.5%	23.66%
2021-01-06	46.7%	44.4%	8.9%	91.9%	6.7%	1.4%	8.33%
2021-02-14	43.8%	50.0%	6.2%	86.7%	10.9%	2.4%	55.88%
2021-03-10	52.3%	41.8%	5.9%	84.5%	12.4%	3.1%	59.15%
2021-04-09	58.2%	35.4%	6.4%	84.7%	13.1%	2.2%	7.14%
2021-05-04	66.2%	29.8%	4.0%	84.2%	12.2%	3.6%	18.92%
2021-05-13	70.6%	27.0%	2.4%	86.9%	9.8%	3.3%	29.09%

Table 10.6

First we note that there are no shipments that are suggested more than eight times, nor are there shipments that received more than eight suggestions. While with $b = 7$ both of these are bounded by 14. Second we note that for most of the datasets, about half of the published shipments are never suggested and only very few published shipments are suggested more than once. Third we note that for all datasets, a very large part of the shipments did not receive any suggestions, and only a very small part received more than one suggestion.

10.4.6 Average Cost

Apart from that we want at least one suggestion to have a cost close to the best possible combination, we also want to make a distinction between the following two cases: one attractive suggestion and $b - 1$ unattractive suggestions, versus one unattractive suggestion and $b - 1$ attractive suggestions. To analyse this, we look at the average cost of the received suggestions of a shipment. Also, we consider the percentage of the shipments that received at least one suggestion and for which the average cost of their suggestions is zero. The results can be seen in Figure 10.1 and in the last column of Table 10.6.

Figure 10.1: Histogram of the average cost (in ϵ) of received suggestions per shipment, over all shipments that received at least one suggestion and taken over all datasets combined.

First we note that, as can be seen in column 7 of Table 10.6, only very few shipments actually receive more than one suggestion. Thus the average cost of the suggestions received by a shipment, mostly just equals the cost of the single received suggestion.

In Figure 10.1 we see that most shipments receive suggestions with a cost between 0 and e 15. These cost seem very reasonable. There are a few outliers, even some shipments that received suggestions that have a cost of arounde 200 on average.

Chapter 11

Conclusion

This chapter is organized as follows. In Section 11.1 we have a discussion about our model and the results. Then in Section 11.2 we discuss our conclusion. Finally, we present directions for further research in Section 11.3.

11.1 Discussion

What is the added value of the suggestions?

When validating the value of the suggestions, the following questions could be relevant: Do the suggestions use the capacity of the market efficiently? Do the suggestions decrease the amount of empty kilometers? Do the suggestions decrease the waiting time inbetween shipments? Note that decreasing the latter two results in time to spare, in which a carrier can execute other shipments. This means extra profit for the carrier, and can also be seen as a better use of the capacity of the market. However, it is hard to answer these questions, since it would rely on the behaviour of carriers, which is extremely hard to capture in a model. Still, such a service is appealing for shippers because it makes shipments more visible. It is also appealing for carriers because it becomes easier to find shipments. Finally, it is appealing for UTURN, because such a service can attract new customers.

For who (carrier, shipper or UTURN) are the suggestions the best?

Carriers We give suggestions of combinations which should help carriers in finding shipments that they can execute. We try to give them b good suggestions, but not necessarily the b best suggestions.

Shippers For shipments that appear in a lot of top-b's, the model is disadvantageous, because those shipments get suggested at most b times, which can be less. But for shipments that appear in very little to none top-b's, it is advantageous, because we try to suggest them b times, which is more. So part of the shippers have to compensate, such that the situation is more balanced for all.

UTURN We hope that giving suggestions will help carriers find shipments that fit with them, and hence that this will increase the number of successful matches, which is profitable for UTURN. Also, we hope this service attracts new customers for UTURN. However, these are just hopes, not guarantees.

All in all, it is not best for any of them, but it is good for all of them. Both the carriers and shippers have to compensate, such that the situation is more even, and good for all. For UTURN there is no guarantee that it will (immediately) be beneficial.

11.2 Conclusion

In Part II we aimed to develop a technique that can be used to suggest combinations of shipments on the UTURN platform. In Section 8.1 we proposed a cost function, that we use to score combinations of

shipments. Next, we developed a theoretical model that identi es a set of attractive combinations based on the proposed cost function. The identi ed combinations can then be used as suggestions. Finally, we translated the theoretical model to a practical implementation. We were not able to suggest shipments as many times as we hoped, nor were we able to give shipments as many suggestions as we hoped. But we did bound the number of times a shipment is suggested by seven. This is favourable, because if a shipment is suggested a lot, it might be quoted and accepted quickly, which can result in a lot of carriers seeing meaningless suggestions. Consequently carriers might loose con dence in the suggestions, as they receive a lot of empty suggestions, and that would be undesirable.

11.3 Directions for Further Research

It might be interesting to combine the random forest model from Part I with the scoring function for combinations. To increase the quoting percentage, it is more important to make shipments predicted as \not quoted" more visible, than to make shipments already predicted as \quoted" even more visible. This could be incorporated in the scoring function, by improving the score a bit if at least one of the shipments in the combination is predicted as \not quoted". This hopefully results in more shipments being quoted at least once.

We use the scoring function to identify attractive combinations, but it can also be used to analyse shipments or shippers. For instance, if we determine the score of one certain shipment with a lot of other shipments, that can tell us if the shipment in consideration is `easy' to combine or not. If this is done for more shipments of the same shipper, it can tell us which shippers generally publish a lot of easy to combine shipments.

With our model we generate a completely new matching each time a new set of suggestions is required. It could be interesting to investigate how two consecutive matchings compare: they could be very similar, or completely di erent. If consecutive matchings di er a lot, carriers might not have a lot of faith in the suggestions; the suggestions might even seem random if they change all the time. In that case it would be wise to investigate if the matching can be done in an online fashion, where the previous matching is used and the deviation from that matching is limited.

For some shipments the time frames available for the actions, all span the same few days. In particular this occurs for shipments with only two actions. Such shipments should be very exible in combining with other shipments; any shipment that starts or ends somewhere in the time span of those few days (and is close in terms of location) should be a candidate for combining. However, due to our e ciency of time constraint (Constraint 4) this is not the case. We want that combinations are e cient, meaning that a carrier does not have to wait too long before he can start the next shipment. But with Constraint 4 in its current form, shipments with actions that span a few days, are disadvantaged. We recommend to investigate how this problem can be tackled, while still guaranteeing e ciency of time.

We assumed that all dates that we need are provided, however that is actually not the case; the dates for the last action of a shipment are not always provided. In that case T and $T_{\max}$ cannot be determined. As a result, they are set to `not a number' in our implementation, and Constraints 3 and 4 are both, maybe wrongfully so, satis ed. If the other constraints are also satis ed, the score of a combination is determined, and the combination appears in the graph. Then the combination can be selected as a suggestion, while it might not be feasible in time. It should be further investigated how to deal with shipments for which not all dates are provided.

As mentioned in Section 10.4, it might be a good idea to suggest to each shipment their top-1 or top-2, and ll the remainder of the top-b with our suggestions. This ensures that each shipment receives at least one or two suggestions, and it ensures that at least one or two of the received suggestions are reasonably attractive. The only unfavourable consequence of this is that the number of times a shipments is suggested can increase substantially. The trade-o between the positive and negative e ects of this idea should be considered.

We think experimenting with the value of b could have a positive impact on the results. We could simply try other values for b, but we could also let it be di erent for the bipartitions of the graph (suggested/receiver), or let it depend on the number of published shipments. If shipments do not have a

full top-b, increasing b for the shipments that receive suggestions is not that useful. However, increasing b for the suggested shipments can be very useful; shipments might not receive a suggestions, because the only shipment that could be suggested to it is already suggested b times. Increasing b here would help. When there are only a few published shipments, a higher value for b will most likely result in more suggestions. On the other hand, if there are a lot of published shipments, it might not make much of a difference if b is increased for the suggested shipments.

We got fewer suggestions than we expected, so it might be profitable to investigate if our scoring function is not too tight. The constraints could be too strict, or maybe the values for the parameters can be changed to improve the results. Or perhaps we got fewer suggestions than we expected, because there are not yet enough shipments available to find a suitable set of suggestions, and the shipments that are available cannot be combined with enough other shipments. It would be interesting to research if there is a number of available shipments, and a degree of cohesion between those shipments, for which our model gives better results.

Bibliography

[1] Andre Altmann, Laura Tolosi, Oliver Sander, and Thomas Lengauer. Permutation importance: a corrected feature importance measure. *Bioinformatics*, 26(10):1340{1347, 04 2010.

[2] Leo Breiman and Adele Cutler. Random forests. website. https://math.usu.edu/~adele/forests/cc_home.htm (Accessed: May 3, 2021).

[3] Daniel Delling, Andrew V. Goldberg, Thomas Pajor, and Renato F. Werneck. Customizable route planning. In Panos M. Pardalos and Ste en Rebennack, editors, *Experimental Algorithms*, pages 376{387, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[4] Scikit Learn development and maintenance team. Permutation feature importance. website. https://scikit-learn.org/stable/modules/permutation_importance.html (Accessed: May 3, 2021).

[5] Scikit Learn development and maintenance team. User guide. website. https://scikit-learn.org/stable/user_guide.html (Accessed: May 3, 2021).

[6] Jack Edmonds and Richard M. Karp. Theoretical improvements in algorithmic e ciency for network ow problems. *J. ACM*, 19(2):248{264, April 1972.

[7] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. Exploring network structure, dynamics, and function using networkx. In *Proceedings of the 7th Python in Science Conference*, pages 11 { 15, 2008.

[8] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer, second edition, 2017.

[9] Stefano Nembrini, Inke R Konig, and Marvin N Wright. The revival of the Gini importance? *Bioinformatics*, 34(21):3711{3718, 05 2018.

[10] Fabian Pedregosa, Gael Varoquaux, Alexandre Gramfort, et al. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825{2830, 2011.

[11] Cormen Thomas H., Leiserson Charles E., Rivest Ronald L., and Stein Cli ord. *Introduction to Algorithms, Third Edition.*, volume 3rd ed. The MIT Press, 2009.

[12] Victor Zhou. A simple explanation of gini impurity. website. https://victorzhou.com/blog/gini-impurity/ (Accessed: May 3, 2021).