

# Optimization of Water Tanking



Jan Vlachý

Department of Mathematics

Utrecht University

Supervisors: KARMA DAJANI, UTRECHT UNIVERSITY

MATTHIEU BOLT, KLM

ADNAN FIAZ, KLM

Co-reader: ROBERTO FERNÁNDEZ, UTRECHT UNIVERSITY

A thesis submitted for the degree of

*Master of Science*

May 31, 2012



### **Abstract**

The drinking water tank of short-haul aircraft is refilled during every stop at the hub airport. This is not necessary and refilling the tank less often can lead to a personnel costs reduction. We implement two Tabu search algorithms to determine a rule specifying during which stops not to refill. The rule minimizes the frequency of refilling while keeping the probability of water shortage sufficiently low. Our algorithms require the probability distribution of water consumption. We suggest a maximum likelihood procedure to estimate this distribution. When used in practice, the refilling rule leads to a decrease in the frequency of water refilling of up to 45%.

# Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
1.1	Overview . . . . .	3
1.2	Problem description . . . . .	4
1.3	Previous work . . . . .	6
1.4	Available data . . . . .	6
1.5	Glossary . . . . .	9
<b>2</b>	<b>Decision Rule</b>	<b>13</b>
2.1	Model description . . . . .	13
2.2	Tabu search to determine two lists . . . . .	23
2.3	Tabu search to determine a list of city pairs . . . . .	29
2.4	Comparison of various methods . . . . .	37
<b>3</b>	<b>Estimation of water consumption</b>	<b>41</b>
3.1	Estimation under multistretching . . . . .	41
3.2	Destinations with few flights . . . . .	59
3.3	Factors determining water consumption . . . . .	62
<b>4</b>	<b>Miscellaneous topics</b>	<b>75</b>
4.1	Data issues . . . . .	75
4.2	Faster evaluation . . . . .	79
4.3	Multistretching more rotations in a row . . . . .	80
4.4	Heaping . . . . .	83
4.5	Deconvolution of empirical distributions . . . . .	87
4.6	Multistretching Boeing 737 . . . . .	90
<b>5</b>	<b>Conclusions</b>	<b>92</b>
5.1	Recommendations . . . . .	92
5.2	Practical consequences . . . . .	92
5.3	Future research . . . . .	93
5.4	Acknowledgments . . . . .	94
<b>A</b>	<b>Useful information</b>	<b>95</b>
A.1	IATA airport codes . . . . .	95

<b>B Selected parts of the R code</b>	<b>96</b>
B.1 Tabu search . . . . .	96
B.2 Tabu search for city pairs . . . . .	104
B.3 Grouping times . . . . .	127
<b>Bibliography</b>	<b>130</b>

# Chapter 1

## Introduction

### 1.1 Overview

The purpose of this project is to determine the feasibility of *multistretching* for KLM City-hopper flights.

Let a *rotation* be a pair of flights of the same airplane – one from Amsterdam to a certain destination and the second subsequently from that place back to Amsterdam. Until recently, drinking water for airplanes has been refilled after every rotation. However, this is often unnecessary since the amount of water consumed during a rotation can be small. To give an example: The water tank of the Fokker 70 airplane has a capacity of 80 liters, but during a typical rotation, frequently no more than 10 or 15 liters is consumed. *Multistretching* means not tanking drinking water between two successive rotations. An example of multistretching between a rotation to B and a rotation to C can be seen in Figure 1.1.

In principle, we would like to multistretch as often as possible. However, if we run out of water during a flight, it is necessary to tank at an outstation (i.e. an airport other than Schiphol), which is usually more expensive for KLM, and it also poses some inconvenience for passengers. To account for this, we impose a restriction that for rotations to any given destination, the chance of encountering water shortage is no more than  $\alpha$ .<sup>1</sup>

#### 1.1.1 Goals

Two main goals are the following:

1. Suggest a decision strategy to determine for which rotations multistretching can be applied. Chapter 2 is mainly dedicated to this topic.
2. The data about water consumption is obtained upon tanking. How to estimate the water consumption on a flight once we – because of multistretching – no longer tank before every rotation? Most of Chapter 3 relates directly or indirectly to this problem.

#### 1.1.2 Outline of the thesis and its original contribution

Most of the work in this thesis was done by the author during his internship at KLM. A lot of ideas were inspired by discussions with the author's supervisors. A part of the problem

---

<sup>1</sup>During this research,  $\alpha$  has been mostly taken equal to 0.05 .

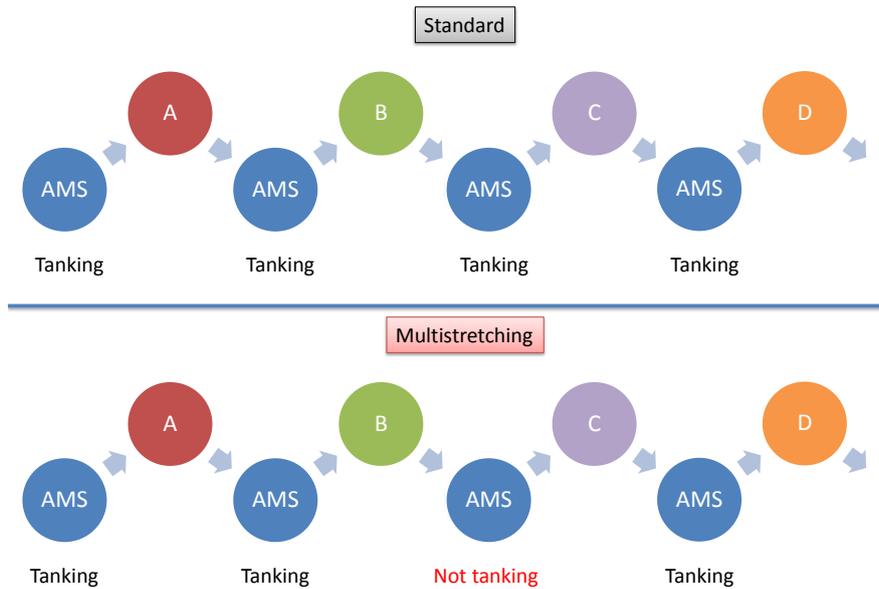


Figure 1.1: Example of multistretching: After returning from B, no water is tanked before flying to C. The water tank is filled again only after returning from C.

formulation originated from the preliminary study done at KLM before the beginning of the internship.

Chapter 2 is dedicated to attaining the first goal from Section 1.1.1. After three classes of strategies are formulated, algorithms are presented to optimize within these classes. Chapter 3 is concerned with various aspects of the water consumption distribution. Of special interest is Section 3.1 which solves the second problem from Section 1.1.1. Chapter 4 provides more in depth information about some issues as well as solutions to problems which are not directly related to the goals outlined in Section 1.1.1. Of particular interest is Section 4.4. Finally, the thesis is concluded by Chapter 5 which summarizes recommendations, lists already reached goals and provides topics for future research.

A considerable part of the internship was dedicated to implementing the ideas in the programming language R. A small (but important) sample of the computer code written is to be found in Appendix B.

## 1.2 Problem description

KLM Cityhopper (KLC) is a subsidiary of KLM focusing on regional flights. All KLM Cityhopper flights are within Europe, to approximately fifty destinations. We are interested in the airplanes Fokker 70 (F70) and Embraer 190 (E90). In comparison to KLM aircraft, these airplanes are smaller and can carry less passengers – see Table 1.1. The rotations are short and in view of this, often three or four rotations per day are made by an individual airplane.

	Fokker 70	Embraer 190
maximum passengers	80	100
tank capacity (liters)	80	90 (or 110)

Table 1.1: Aircraft specifications

One possibility for savings is to minimize the number of tasks to be done with the airplane during a stop in Amsterdam – this saves manpower and in some cases also required resources. By *multistretching*, it is meant skipping a task between two rotations. While this term can be used for any task, in this thesis it will be used almost exclusively for skipping water tanking. We use the term *slipping* interchangeably with the term multistretching.

Normally, water is tanked during every stop in Amsterdam. The purpose of this research is to determine when it is not necessary. For KLC aircraft, the tank is always fully filled. We note that taking less water (and therefore weight) saves per se almost nothing for KLC flights.<sup>2</sup>

The motivation for the research is that slipping has happened in the past in “unstructured” fashion. This means that the decision not to tank has been done ad hoc on the discretion of the employees responsible for tanking when they considered the amount of assigned work too high to handle. This has led to the question if multistretching “structurally” (i.e. using a fixed optimized strategy) could produce better results. Namely more rotations slipped and subsequently less work for operators and ultimately a decrease in the number of operators required.

Drinking water is used for making tea and coffee (other drinks are served from a bottle or a can), washing hands and (for E90) also flushing toilet. The data on the water consumption are inferred from the data on the amount of tanked water. E90 has a tank of either 90 L or 110 L (at the beginning of this research, it was 90, but in the future the airplanes will be modified for 110 L), F70 has a tank of 80 L.

### 1.2.1 Service level

Apart from multistretching as many rotations as possible, care must be taken that water shortages do not occur often. This is embodied in the concept of *service level*. This is a number  $1 - \alpha$  such that for every destination, at most an  $\alpha$  fraction of the rotations to this destination end up with a water shortage if multistretched.

This concept requires some explanation:

1. It is assumed that non-multistretched rotations never end up with too little water.
2. The motivation behind this definition is that a customer regularly flying to one place should encounter water shortage during at most  $\alpha$  fraction of her (return) journeys.

Throughout the work, it has been assumed that  $\alpha = 0.05$ .

---

<sup>2</sup>In general, taking less weight saves fuel; however, at KLC flights, the amount of water taken is very small anyway and furthermore the flights are very short. So the overall effect on fuel is negligible. This is in contrast to larger airplanes flying longer distances – there taking less water can save substantial amount of fuel. The optimal amount of water tanked in these cases was considered both by KLM analysts as well as by [3] and [25].

### 1.2.2 Tanking process

Water tanking is done by operators from the Aqua services department. At the start of his shift, the operator gets a handheld. Using this device, he always communicates what his state is (he communicates when the current state is finished, info about the current state etc.). After entering a tanking vehicle, he first drives to a water pump and there fills the tank of the vehicle. Afterwards, he can start receiving requests for tanking – a request mentions where the airplane is in terms of the pier and a place number. The operator accepts the request, arrives to the airplane, updates his status and starts tanking.

For KLC flights, it is always tanked until the tank is full. For E90, this is made clear by a distinctive sound, for F70, the operator must recognize it himself. Therefore, quite often the operator tanks too much and some water splashes out. Once finished with tanking, the operator reads off the tanked amount from the tanking vehicle. Afterwards, he must reset the water meter, so that it will show again the correct amount the next time. Now, the operator is finished – he enters the amount of water tanked into the handheld and awaits a new request.

The water meter measurements are in liters with accuracy to one decimal point. However, the operators have to enter the tanked amount in whole liters. At the beginning of the research, there was no rule about how the numbers should be rounded. Some operators round upwards, some downwards, as we will see, some operators round even to the nearest multiple of 5 or 10. After being notified, shiftleaders from Aqua said they would address this problem. Hence hopefully this will happen less often or never in the future.

## 1.3 Previous work

There was a preliminary study carried out at KLM, in which the author was not involved – this study was however a trigger for starting the internship. The preliminary study laid the foundations of the model in Section 2.1. It will be further referred to as “the Preliminary study”.

Nothing has been found about the problem in scientific journals. No information is available to us about other airlines. But some work has been related to KLM itself. It seems that “unstructured” slipping (i.e. multistretching at the discretion of individual Aqua operators) has been around for a long time. The internship report [5] investigated the feasibility of slipping “structurally”. The report is almost purely qualitative, its conclusion was that structural multistretching was possible, but concrete implementation remained unaddressed.

The Study Group of Mathematics with Industry in 2005 [3] was concerned with determination of optimal amount of water tanked for large intercontinental aircraft, with the intention to improve on an older KLM model. The assumption was that the water consumption is essentially a (random) function of the number of passengers. Afterwards, an internship project was carried out to work out the results of the Study Group in more detail [25]. In this case, no multistretching is considered; also, taking less water is desirable not because of lower workforce costs but rather because of the decrease of fuel costs because of lower weight.

## 1.4 Available data

The basis for the analysis are the flight and service data from the recent past, this amounts to several tens of thousand data records for F70 and only slightly less for E90. Both analysis

of the data and implementing algorithms has been done in the programming language R [27].

Data is taken from a planning system; at the time of writing, it is available for flights from October 2009 until January 2012. The construction of the final dataset is described in Section 4.1.1. The final dataset consists of a list of pairs of flights (call these “flight-pairs”). The pair consists of one inbound flight (i.e. from an outstation to Amsterdam) and the subsequent outbound flight. Note that between two “subsequent” flights, there may be in principle a gap of several days, but this happens very seldom; by “subsequent” we mean subsequent in time, but also pertaining to the same airplane.

For each flight-pair,<sup>3</sup> we have the following information (among other):

- registration, i.e. the unique identification code of the airplane
- number of passengers on both flights
- tanked amount after the inbound flight and also after the rotation containing the outbound flight; if this tanked amount is not known, it is noted if this is because of (unstructured) multistretching (i.e. it was not tanked at all) or because of badly registered data<sup>4</sup>
- the destination from which the inbound flight comes and the one to which the outbound flight goes
- scheduled and actual arrival times of the inbound flight and departure times of the outbound flight
- distance of both inbound and outbound destinations from Amsterdam

For practical purposes, the dataset of flight-pairs is then filtered to contain only data for one aircraft type and for one period – either summer or winter<sup>5</sup> – for a certain year. For one aircraft type and one period, there is usually something between 5000 and 15000 rotations.

### 1.4.1 Tanking distribution

The important quantities are tanked amounts of water – these are assumed to correspond to consumed water during their respective rotations. Consider Figure 1.2. It corresponds to the histogram of measured tanking data for rotations to Bristol during “the whole history” (of our data) for F70. Records where the previous rotation was multistretched (hence the measured consumption can be assumed to be the sum of consumptions for both rotations) and those where the tanking data were incorrect were filtered out. The number of rotations on which the histogram is based is almost 2400. A more or less similar figure appears for all destinations and both aircraft types. That said, for E90, the consumption is in general higher and more variable. There are a few features to be noticed at this figure:

- Data is apparently often rounded to multiples of five.

---

<sup>3</sup>Note that having a list of flight-pairs is essentially equivalent to having a list of rotations. Flight-pairs are used for convenience, but it is easy to pass, whenever desired, from the flight-pair to rotations representation.

<sup>4</sup>This “metadata” was not completely straightforward to obtain and extract from data, but currently it should be mostly correct.

<sup>5</sup>The summer and winter periods are taken to correspond to summer and winter flight timetables which in turn correspond to summer and winter daylight saving time – i.e. when the summer time is in force, there is a considerable different timetable

## BRS

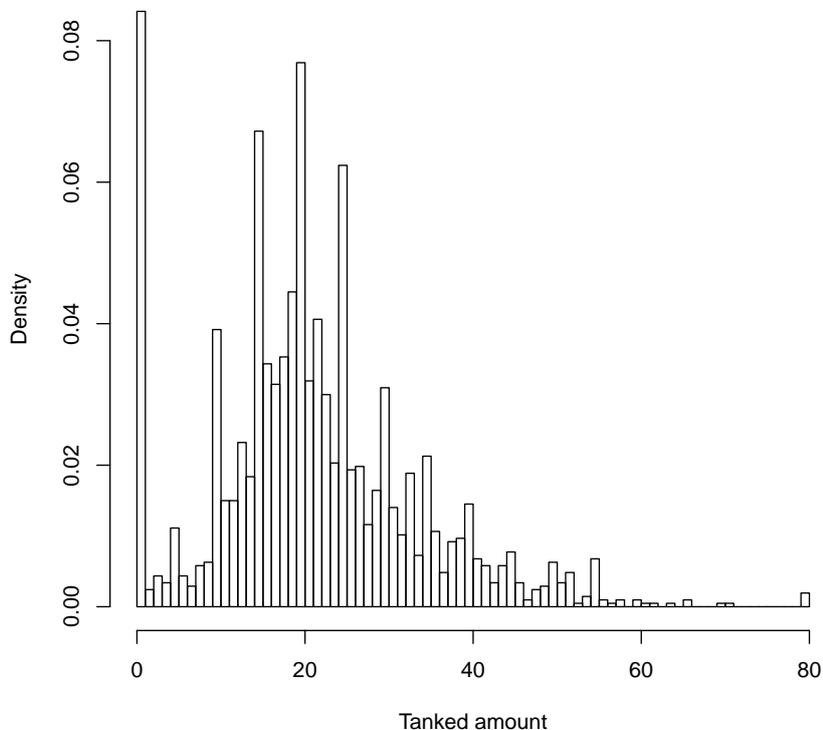


Figure 1.2: Histogram of water tanking data for Bristol, F70

- Ignoring the rounding effect and the peak at zero, the distribution seems unimodal; perhaps slightly skewed to the right.
- There is a high peak at 0 liters. At first sight this could seem as an instance of bad data records. However, in most cases this is not so. The reasons to conclude this are the following:
  - \* The water consumption during a flight is very much dependent on catering – i.e. whether tea and coffee is offered. If not, then the consumption might be much lower – in fact, most zero observations correspond to either early morning or late evening rotations, where it is conceivable that warm drinks are not always offered.
  - \* The aircraft is sometimes tanked at an outstation. In particular, this always happens during a nightstop. Then the tanking data does not correspond to the whole rotation, but only to a half of it. Hence the probability of very low consumption is even higher in this case.
  - \* From the raw tanking data, it can be deduced that tanking tasks which report tanking 0 liters take on average considerably less time than other tanking tasks. This reinforces the belief that if 0 liters tanked is reported, then it is actually true, or at least that the tanked amount was lower than average.

- \* It was confirmed by an Aqua operator that he indeed sometimes tanks almost nothing – and hence enters 0 liters tanked in the handheld. As possible reasons, he mentioned that the flight was short and that it was tanked at the outstation (so that the amount that he tanks corresponds actually only to the consumption during one leg of the rotation).
  - \* It is not well defined what tanking a full tank means. Hence, even if a small amount of water was actually consumed (3 liters say), the operator may still consider the tank as full and tank nothing.
  - \* Since Aqua operators often round the data, it is conceivable that small amounts get often rounded to zero.
  - \* The percentage of the records with 0 liters tanked is very strongly correlated<sup>6</sup> with the distance of the destination from Amsterdam – this further supports the idea that the tanked amount indeed corresponds to water consumption, hence to a true record.
- There is a very small peak at 80 liters. Most of these observations do not correspond to true water consumption. Instead they correspond to filling an empty tank after it was drained during the night.
  - From a quantitative analysis of the data, it seems that the right tail of the distribution decays roughly exponentially: This is deduced by taking the empirical distribution function<sup>7</sup> for a fixed destination (only destinations with more than 1000 rotations were considered in this test, so that it could be assumed that the empirical distribution function is already very close to the cumulative distribution function)  $\hat{F}$  and computing the successive quotients  $\frac{1-\hat{F}(n+1)}{1-\hat{F}(n)}$ , for reasonably large values of  $n$ .<sup>8</sup> These quotients are found to be roughly constant, mostly between 0.85 and 0.95.

Such behavior is consistent with an exponential distribution; and to a certain extent for example with a gamma distribution. On the other hand, for a normal distribution, one would expect that the quotients decay faster.

An average tanked amount for F70 is 21 L and for E90 it is 26 L; however these numbers do not take into account the differences between individual destinations.

These differences are large. Namely, the average consumption during rotations to a given destination ranges between 14 L and 36 L for F70; it ranges between 16 L and 53 L for E90. Also, some destinations are flown by either only F70 or only E90. A direct comparison of F70 and E90 for the destinations common to both is given in Section 3.3.7.

## 1.5 Glossary

- *cdf*: cumulative distribution function

---

<sup>6</sup>In fact, plotting the distance from AMS to the proportion of zero observations, one can fit there a left branch of parabola, i.e. the dependence is very close to quadratic and longer flights have almost no zero observations.

<sup>7</sup>Note that it changes only at integer values.

<sup>8</sup>This means cca between 30 and 60 for F70. For higher values, there are often no observations for a given value, so the quotient turns out to be 1.

- *delta-gamma*: A distribution being a mixture of a gamma distribution (with rounding corrections) and a point mass at zero. The exact definition is given in Section 3.1.4.
- *destination, city*: are used interchangeably to denote an airport. Every airport has a unique three letter code, see Appendix A.1. The set of all considered destinations (which depends on the dataset currently in use) is denoted by DST.
- *draining*: Emptying a water tank. During a nightstop in Amsterdam, this is sometimes done by the Engineering and Maintenance department. Hence not by Aqua Services which is also the reason why we cannot determine whether draining has happened.
- *edf*: empirical distribution function.
- $\hat{F}$ : empirical distribution corresponding to a distribution with cdf  $F$ .

- *multistretch* or *slip*: are used interchangeably to denote skipping water tanking service. When slipping a rotation, it means that it is not tanked before the outbound flight of the rotation (for instance, in Figure 1.1, we multistretch the rotation to C). When slipping a flight, it means that it is not tanked before this flight. Slipping a flight-pair means slipping its outbound flight.

Another convention, which we do not follow in this text, sees multistretching as structured slipping (i.e. determined by a fixed strategy and not the discretion of operators).

- $\mathcal{N}(\mu, \sigma^2)$ : Normal distribution with mean  $\mu$  and variance  $\sigma^2$  (or possibly the multidimensional normal distribution with the vector of means  $\mu$  and the covariance matrix  $\sigma^2$ ).
- *nightstop*: When an airplane stops at an airport for the night. Importantly, after a nightstop in Amsterdam, the airplane must be always tanked before flying the first flight of the day. This is because it should have been drained during the night – even though in practice this does not always happen. It is assumed that airplanes are always tanked during a nightstop at an outstation.
- *outstation*: an airport other than Amsterdam, i.e. other than the Schiphol airport.
- *registration*: the unique code identifying a specific airplane. We will also use this to denote the airplane to which the registration belongs.
- *registration-day*: a series of subsequent rotations for one registration which begins by a flight which cannot be slipped (i.e. one after a nightstop in Amsterdam) and ends with either a flight which cannot be slipped or with the last flight in the dataset (for that given registration).

We note that a “registration-day” may, and often will, span over more than one day. However, it will span over only one day, if at the beginning of the day, the airplane starts from Amsterdam and, at the end of the day, it ends there for the nightstop.

- TC: water tank capacity; this is 80 L for F70 and 90 (or more recently 110 L) for E90 – see also Table 1.1
- $X_d$  for  $d \in \text{DST}$  is a random variable giving the water consumption during a rotation to  $d$ . The random variable  $Y_{d_1, d_2}$  is then defined by  $Y_{d_1, d_2} := X_{d_1} + X_{d_2}$ . The samples generated by these random variables are denoted by the corresponding lower case letters.

**Difference between flight-pairs and rotations.** By a *rotation*, we mean a pair of subsequent flight legs of which one departs from Amsterdam and flies to an outstation and the second one flies from the outstation back to Amsterdam. By a *flight-pair*, we mean a pair of subsequent flight legs such that the first one flies from an outstation to Amsterdam and the second one from Amsterdam to another outstation. See Figure 1.3 for an illustration of the difference between the two concepts.

Even though it would be possible to work with just one of these two related terms, we feel both of them are useful.

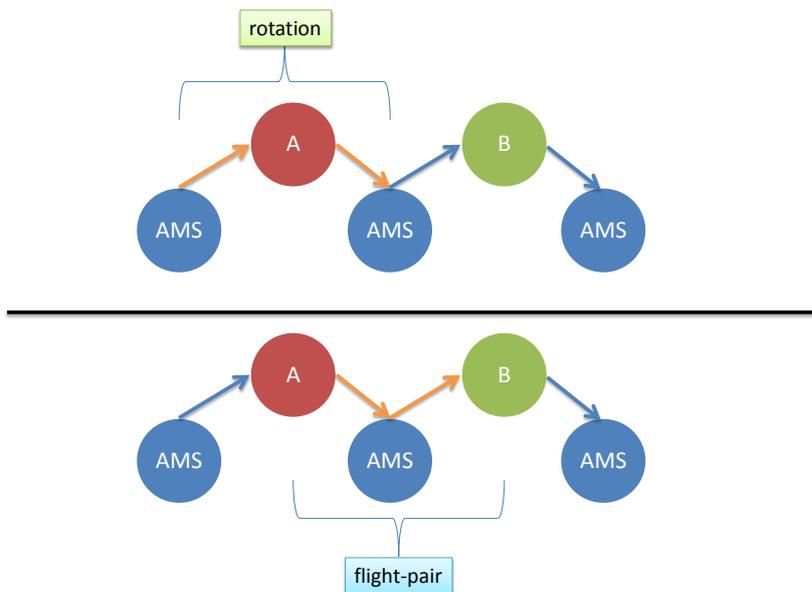


Figure 1.3: The two orange flight legs in the upper part form a rotation to A. The two orange flight legs in the lower part form a flight-pair, with its inbound flight going from A and its outbound flight going to B.

**Notation for rotations.** When discussing rotations, we will use the notation like  $(A - B - C - D)$  to denote a sequence of rotations: the first one goes to  $A$ , the second one to  $B$ , the third to  $C$ ... i.e. we do not mention the stops in Amsterdam in between.

**Convolutions.** By the operator  $*$ , we always mean the convolution operator and not multiplication (except for Appendix B). For functions  $f, g$  supported on  $\mathbb{R}$  or its subinterval, we define it by

$$(f * g)(y) := \int_{-\infty}^{\infty} f(y - x)g(x)dx.$$

If  $f$  and  $g$  have support on (a subset of) integers, then (discrete convolution) is defined by

$$(f * g)(y) := \sum_{x \in \mathbb{Z}} f(y - x)g(x).$$

**Multidimensional derivatives.** For a function  $f : \mathbb{R}^p \rightarrow \mathbb{R}$ , denote by  $\nabla f$  its gradient (i.e. the vector function  $(\frac{\partial f}{\partial x_i})_{i=1}^p$ ) and by  $\nabla^2 f$  its Hessian (i.e. the matrix-valued function  $(\frac{\partial^2 f}{\partial x_i \partial x_j})_{i,j=1}^p$ ).

**Numbering.** Chapters and sections are numbered in the obvious fashion. Furthermore, all definitions, examples and propositions are numbered using the same numbering – so it is possible to encounter Example 2.2 even though Example 2.1 does not exist since there is Definition 2.1 instead.

## Chapter 2

# Decision Rule

### 2.1 Model description

As noted in the Introduction, one of the main goals is to propose a suitable multistretch strategy. A question arises how to define a multistretch strategy and how to evaluate strategies and compare them. A significant piece of work was done already in the Preliminary study, mentioned in Section 1.3. In what follows, we first briefly explain what the model is – i.e. we specify the objective function and constraints. After that a discussion follows about another model considered which was eventually deemed unsuitable.

We begin with an assumption:

**Assumption 1.** Water consumption during a (non-multistretched) rotation equals the subsequently tanked amount of water which we see in the data.

There are two violations of this assumption:

1. The tanked amount of water that is displayed in the data need not exactly correspond to the true amount tanked into the tank (which presumably corresponds to the amount of water consumed since last tanking took place): As is clear from Section 1.2.2, tanking is rather a messy process, so it is for instance possible that some water is spilled on the ground. We neglect such problems. What we do not neglect is rounded data (due to imprecise measurements as well as due to Aqua operators rounding) – this will be tackled in Chapter 3 and also in Section 4.4.
2. If a last rotation of the day goes to an outstation, the subsequent tanking data often accounts only for the second half of the rotation since during a nightstop at an outstation, it is often tanked: This problem is not treated explicitly. However, in Section 3.3.5, the dependence of tanked amount on the departure time is treated. This covers the problem to a large extent since the problematic rotations depart always in the evening.

#### 2.1.1 Multistretch strategy

At the very beginning, it is worth mentioning *restrictions* imposed on our multistretch strategy, no matter how it will look like.

1. The first flight of the day from Amsterdam (if the airplane had a nightstop in Amsterdam and did not come after having a nightstop at an outstation) must always be tanked.

This is because the airplane is drained during the night and so there is no water in the tank in the morning.<sup>1</sup>.

2. It is not allowed to multistretch two or more rotations in a row. This is because presumably multistretching twice would carry too high risk of a water shortage. Another reason is a considerable increase of computational complexity if also strategies allowing more multistretching in a row are allowed. See however Section 4.3 for a discussion of a particular relaxation of this rule.
3. Destinations with too few rotations and, if applicable, city-pairs with too few rotations in the training dataset (see Section 2.1.2) are excluded from multistretching. A standard bound is 50 (or 25) for destinations and 10 for pairs (it was experimented a little bit with these numbers, but there is no real theoretical justification behind them). The bounds are imposed for the following reasons:
  - For estimating the water consumption distribution (as detailed in Section 3.1), we need enough observations, otherwise the results can be worthless.
  - If we want to use actual tanking data (i.e. the boolean shortage vector described in Section 2.1.2), then again, we want enough observations, otherwise individual observations can have a disproportional influence on the total percentage of water shortages.
  - Introducing the bounds reduces the number of destinations and/or pairs, thus leading to a smaller state space (of strategies). Also note that when there are too many “small” moves, the algorithms described in Sections 2.2 and 2.3 spend disproportionately large amount of time shuffling with small moves which considerably slows down the algorithm.

If a strategy suggests multistretching a flight-pair in such a way that the above restrictions would be violated, the multistretch decision is overridden. That said, we are ready to define several classes of strategies now.

The first decision about what a strategy should be was made already during the Preliminary study. The idea at that time was that planning of multistretches would be done manually by planners – it was therefore desirable to produce a rule which would be compact enough to write on a sheet of paper, on the other hand, it should have enough “discriminatory power” to strike the balance between the frequency of multistretching and the service level.

The strategy was therefore defined as *two lists* of cities - inbound and outbound. A flight-pair is multistretched if the inbound flight comes from a destination on the inbound list and the outbound flight goes to a destination on the outbound list. To optimize within such a class of strategies, we developed the Random search and Tabu search algorithms in Section 2.2.

Later on, it turned out that scheduling of multistretches will be after all managed automatically by the planning system. The system can readily accept strategies which consist of only *one list* – this is the special case of the above mentioned two lists when the inbound and outbound lists are the same.<sup>2</sup>

---

<sup>1</sup>It turned out that draining is apparently only performed in winter when it freezes, even though it should be done always. The tanking requirement is respected nevertheless. Unfortunately, the draining is not carried out by Aqua Services, so it is not straightforward to distinguish the drained flights from the not drained ones.

<sup>2</sup>And so, we multistretch a flight-pair if both its incoming and outgoing destinations are on the list.

But this also opens the possibility (after a suitable modification of the planning system) to consider strategies which are intuitively more reasonable and efficient – namely, a strategy consists of a list of city *pairs* and the tanking is multistretched if the flight-pair formed by the inbound and outbound destination is on the list. Since the consumption is presumably the same no matter which city is inbound and which outbound,<sup>3</sup> we assume that the pairs are symmetric. Note that such a list would not be suitable for human planners since it can be rather lengthy, several hundred pairs being a standard. This type of strategies is expected to be used in the future.

To optimize within the strategies consisting of one list, it was enough to extend the Tabu search in Section 2.2. To optimize the lists of city pairs, a different Tabu search algorithm was developed. It is described in Section 2.3.

### 2.1.2 Evaluating strategies

As already mentioned, the goal is to find a strategy, searching within one of the above classes, which maximizes the frequency of multistretches subject to the constraint on the percentage of rotations suffering a water shortage per each destination (see Section 1.2 for the motivation behind this constraint).

The Preliminary study formulates this objective as follows: As a training set, take the timetable of the preceding period – e.g. if we are interested in computing a strategy for winter 2011, take winter 2010.<sup>4</sup> Now try to find the optimal strategy for this training set. The assumption is that the timetable in the following year will stay more or less the same – whether this assumption leads to a useful model will be examined in Section 2.4.1. Note that if we had the timetable for the current period and then true tanking data for it, then this approach would give the “true answer”; i.e. the optimal strategy would multistretch as many rotations as possible while satisfying the shortage constraints.

Yet we do not have the current data. When it comes to tanking, this is obvious since we are not clairvoyant. On the other hand, even though the timetable is known in advance, it is not yet known which registrations will fly which routes – which is necessary to determine the registration-days and hence also to evaluate the strategy in the above mentioned manner. Still, it is hoped that the optimal strategy from the training period will be satisfactory also when used for the current period.

For clarity and future reference, let us denote this symbolically. Denote our state space of strategies  $\mathcal{S}$  (possibly one of the above three state spaces – i.e. either *one list* or *two lists* or *pairs*). Consider the following:

- data  $D$ : corresponding to all rotations during a given time period, contains information about the timetable and water tanking.
- strategy  $s$ : giving a rule how to multistretch.
- function  $\phi(s; D)$ : determining how many multistretched rotations there would be when using the strategy  $s$  for the data  $D$ ; this will be our objective function.

---

<sup>3</sup>Perhaps up to differences due to departure times.

<sup>4</sup>There are different timetables for winter and summer. Summer timetable is in force during the same time that the daylight saving time is observed; winter timetable then throughout the rest of the year.

- function  $\sigma(d, s; D)$ : determining the fraction of rotations to the destination  $d$  suffering water shortage when multistretching according to the strategy  $s$  on the data  $D$ ; this will be our constraint function.
- the maximal admissible shortage level  $\alpha$ ; recall that we usually choose this to be 0.05

We will sometimes suppress the data  $D$  and write only  $\phi(s)$  or  $\sigma(d, s)$ . Now the optimization problem is

$$\begin{aligned} \max_{s \in \mathcal{S}} \phi(s; D) \\ \sigma(d, s; D) \leq \alpha, \quad \forall d \in \text{DST} \end{aligned} \tag{2.1}$$

We would like to solve this problem for the upcoming period, call the data for it  $D_{\text{new}}$ . However,  $D_{\text{new}}$  is not yet known to us; therefore, we determine a (nearly) optimal strategy  $s$  for the data  $D_{\text{old}}$  from the preceding period instead and we hope that it will be a good solution also for the problem with  $D_{\text{new}}$ , implicitly assuming that the timetable and water consumption in the new period will be similar. In Section 2.4.1 it is discussed to which extent this is justified.

How do we compute the objective function value then, given a strategy?

1. Apply the “raw” strategy to flight-pairs to determine what would be multistretched if we disregard the first two restrictions in Section 2.1.1.
2. Apply the restrictions and see which flight-pairs would really get multistretched under the strategy on hand.
3. Compare the really multistretched flight-pairs with the *shortage vector* to come up with the number of water shortage events – this is then divided by the total number of rotations to obtain the total shortage. This must be done for every destination separately.

We note that the most computationally expensive step is the second one – on our computer, it takes several units or tens of seconds, depending on the size of our sample of rotations and other factors. In Section 4.1.2, the technical details of the above steps are explained in detail while in Section 4.2, two considerable improvements upon the original evaluation procedure from the Preliminary study are presented.

### Shortage vector

The *shortage vector* (dependent on the data  $D$ ) is a vector such that each of its components is associated to one flight-pair. The component should capture the probability that, if the given flight-pair is multistretched, the associated rotation<sup>5</sup> will suffer a water shortage. We consider two possible approaches:

1. boolean shortage vector: This approach was used in the Preliminary study. In this case, the tanking data from  $D$  is assumed to be fixed (as opposed to generated by some underlying probabilistic mechanism). A component corresponding to a flight-pair is then either 1 or 0 since the water consumption is assumed to be fully determined by  $D$ .

---

<sup>5</sup>I.e. the one which contains the outbound flight of the flight-pair.

This means that either would the flight-pair, if multistretched, suffer a water shortage or not.

How to assign 0 or 1 to a component? We must assume that  $D$  contains water consumption data about all rotations (i.e. there must not be multistretched flights, otherwise we have no tanking and hence no consumption data for associated rotations). Then it is easy to determine whether a rotation, call it  $r_2$ , would suffer a shortage if we multistretched it. Indeed, consider the rotation preceding  $r_2$ , say  $r_1$ . Let  $T(r_1)$  be the consumption during the first rotation and  $T(r_2)$  the consumption during the second rotation; this data is available from  $D$ . If we multistretch  $r_2$ , this means that no tanking takes place after  $r_1$  and so we do not have enough water at the end of  $r_2$  if  $T(r_1) + T(r_2) > \text{TC}$ .<sup>6</sup>

Note that the above determination of the boolean shortage vector is impossible if  $D$  contains more than few multistretched flights – the only possibility would then be first estimating the missing consumptions.<sup>7</sup>

2. numeric shortage vector: This approach was proposed to address a certain arbitrariness of the boolean shortage vector and also to come up with a method that is useful even when  $D$  contains multistretched flights. The point of view is now that the tanking amounts observed in  $D$  for rotations to a destination  $d$  are samples from a certain random variable, namely from  $X_d$ .

Recall that  $X_i$  for a destination  $i$  was defined as the random variable giving the water consumption for a rotation to  $i$  and  $Y_{i,j} := X_i + X_j$  (with  $i, j \in \text{DST}$ ). Consider the following

**Definition 2.1.** Let a *virtual shortage* (denoted by  $v_{i,j}$ ) for  $i, j \in \text{DST}$  be  $\mathbb{P}\{Y_{i,j} > \text{TC}\}$ .

Typically, the virtual shortages are not known and we have to estimate them using  $D$ . Note that if we have the (possibly estimated) probability mass function  $f_i$  for  $X_i$  and the cumulative distribution functions  $F_j$  for  $X_j$ , then virtual shortage can be computed as<sup>8</sup>

$$v_{i,j} \approx 1 - \sum_{k \leq \text{TC}} f_i(k) F_j(\text{TC} - k). \quad (2.2)$$

Now in the situation as above, we have rotations  $r_1$  (going to the destination  $d_1$ ) and  $r_2$  (going to the destination  $d_2$ ), we want to multistretch  $r_2$ . Then the probability that  $r_2$  will suffer a water shortage is equal to  $v_{d_1, d_2}$ ; if the virtual shortages are not known, an estimate from the right-hand side of (2.2) is used instead.

Now given a shortage vector for data  $D$ , one computes  $\sigma(d, s; D)$  by summing the shortage vector components for the rotations to  $d$  which would be multistretched (taking into account the restrictions) under  $s$  and dividing this number by the total number of rotations to  $d$ .

<sup>6</sup>Here, we implicitly assume that no self-regulatory mechanism for the consumption exists; i.e. it is consumed the same even though it is clear that the water tank is running out of water.

<sup>7</sup>This was actually tried out. However, this solution is problematic for the similar reason that the boolean shortage vector is to be replaced by the numeric shortage vector, as described below; briefly speaking we are using one observation where it seems more justified to take the expectation over all possible observations.

<sup>8</sup>The equality would be exact if  $f_i$  and  $F_j$  were known exactly and Assumption 2 in Section 3.1.1 was valid.

**Justification of the numeric shortage vector.** Apart from the possibility to use the numeric shortage vector even if  $D$  contains multistretched rotations,<sup>9</sup> what is its advantage over the boolean shortage vector? Consider the setting of (2.1). If  $D_{\text{new}}$  is available including the tanking information, then of course the boolean shortage vector is preferable. However, this is not the case, instead, we have only  $D_{\text{old}}$  available. Assume that the timetable data for  $D_{\text{old}}$  and  $D_{\text{new}}$  are the same (i.e. the timetables are the same and the same flights are flown by the same registrations, etc.; this corresponds to the idealization of our implicit assumption that the timetables for both periods are “similar”). Hence  $D_{\text{old}}$  and  $D_{\text{new}}$  differ only in the tanking data. In this way, both of them can be seen as a realization of a random variable  $D(\omega)$  (depending on  $\omega \in \Omega$  for an event space  $\Omega$ ) such that all realizations of  $D$  share the same timetable data and the randomness stems purely from the differences in tanking data. Now for any data  $D$ , we denote the respective optimal strategy  $s^*(D)$ . This can be seen as a vector function – i.e. giving a vector of 0 and 1 with a component for each city-pair, respectively city. Similarly, any realization  $D(\omega)$  can be viewed as a vector giving to every flight-pair<sup>10</sup> 1 if it suffers a water shortage if multistretched and 0 otherwise.

Observe that when using the boolean shortage vector, we approximate  $s^*(D_{\text{new}})$  by  $s^*(D_{\text{old}})$ . Now note that the numeric shortage vector can be viewed as an “average” shortage vector, when averaging over possible tanking data, i.e.  $\mathbf{E}_\Omega D(\omega)$ , where  $\mathbf{E}$  is the expectation operator.<sup>11</sup> This is because virtual shortages are *by definition* the average shortage probabilities under multistretching, for given city pairs. Therefore, using the numeric shortage vector approximates  $s^*(D_{\text{new}})$  by  $s^*(\mathbf{E}_\Omega D(\omega))$ . This seems intuitively appealing in contrast to the boolean shortage vector – indeed, by using  $\mathbf{E}_\Omega D(\omega)$  instead of  $D_{\text{old}}$ , we would expect to filter out the “noise” present in  $D_{\text{old}}$ .

We note that another intuitive estimator of  $s^*(D_{\text{new}})$  would be  $\mathbf{E}_\Omega s^*(D(\omega))$ . This can be in turn approximated by the law of large numbers by  $\frac{1}{n} \sum_{i=1}^n s^*(D^{(i)})$  where  $D^{(i)} \sim D$  are independent and  $n$  is large. This would be a randomized strategy though, i.e. for each city pair (a destination if using a two lists or one list strategy), we would get only a proportion indicating how often should the given city pair be multistretched. Another disadvantage is the necessity to solve  $n$  times a problem equivalent to (2.1) which would be very time consuming.

## Sources of error

Consider finally the sources of error in the approach of solving (2.1) with  $D_{\text{old}}$  instead of  $D_{\text{new}}$ , in comparison with the ideal state when we know the true data  $D_{\text{new}}$  and are able to compute the optimal strategy for it:

1. Timetable changes from one year to another – this might be by introducing new flights or canceling old ones (or old flights being served by different aircraft), shuffling the departure times, different registrations flying different legs...
2. Tanking data – water consumption distribution can be different from year to year (see Section 3.3.6), at least for some destinations; and even if we know it fully, it is still not obvious how to estimate  $s^*(D_{\text{new}})$ , as was discussed above.

---

<sup>9</sup>Multistretched rotations mean that  $D$  does not contain complete tanking – and hence consumption – data. For the numeric shortage vector, the tanking data is needed only to estimate virtual shortages, so the problem with incomplete records can be tackled – this is done in Section 3.1

<sup>10</sup>Since the timetable is fixed, the set of flight-pairs is for every  $D(\omega)$  the same.

<sup>11</sup>Note that the operator  $\mathbf{E}_\Omega$  is itself not known exactly and must be estimated, usually using  $D_{\text{old}}$ .

outstation	F	A	B	B	B	A	C
consumption	44	50	32	25	12	20	63

Table 2.1: Consecutive rotations during a registration-day. The consumption refers to the tanked amount *after* the rotation.

3. Determining the optimal strategy is unfeasible – indeed, if we have for instance the pairs strategy and there are 400 considered pairs, the state space is of order  $2^{400}$  (some states are inadmissible, but the overall space size will not shrink considerably by this), hence too large.

We will return to this topic in Section 2.4.1. There we give an assesment to what extent the results obtained by our approach are useful.

Consider finally an example of computing the objective function and shortages

*Example 2.2* (Evaluation). Since the evaluation of different registration-days can be done independently, we restrict ourselves to the case of just one (short) registration day – the data are in Table 2.1. Assume that the tank capacity is 80 L. Let the (pairs) strategy be  $s := \{(A, B), (B, B), (A, F)\}$ , i.e. we multistretch a pair of rotations if

1. One rotation goes to  $A$  and the other one to  $B$ , or
2. one goes to  $B$  and the other one to  $B$ , or
3. one goes to  $A$  and the other one to  $F$ .

The boolean shortage vector would be  $\beta = (?, 1, 1, 0, 0, 0, 1)$ ; to determine the question mark here, we would need to know the tanked amount *before* the rotation to  $F$ . Assume that the virtual shortages are  $v_{A,B} = 0.06$ ,  $v_{A,F} = 0.08$ ,  $v_{B,B} = 0.02$ ,  $v_{A,C} = 0.15$ ; the numeric shortage vector corresponding to these would then be  $\nu = (?, 0.08, 0.06, 0.02, 0.02, 0.06, 0.15)$ . Here, the question mark is because we do not know the destination of the rotation preceding the one to  $F$ .

In the first step of the evaluation, we apply the strategy naïvely. This means that the multistretch vector reads (with the plus sign denoting a multistretch and the minus sign denoting no multistretch):  $(?, +, +, +, +, +, -)$ . So the strategy  $s$  would multistretch all tanking except the one between the rotations to  $A$  and to  $C$  – this is because the pair  $(A, C)$  does not appear in  $s$ .

In the second step, we must apply the restrictions from the beginning of Section 2.1.1. Hence the first rotation (to F) is not multistretched since it is the first one for a registration-day. Further we do not allow multistretching two rotations in a row; hence the actual multistretch vector:  $(-, +, -, +, -, +, -)$ . We see that the number of multistretches is  $\phi(s) = 3$ . Computing the shortages using the boolean vector yields

$$\sigma_\beta(A, s) = 0.5, \sigma_\beta(B, s) = 0, \sigma_\beta(C, s) = 0, \sigma_\beta(F, s) = 0.$$

Hence in this case, the strategy is inadmissible. Let us explain the computation in more detail. To this end, we compare the actual multistretch vector the boolean shortage vector:

$$\begin{aligned} \text{rotation destination: } & (F, A, B, B, B, A, C) \\ \text{multistretch vector: } & (-, +, -, +, -, +, -) \\ \text{shortage vector: } & (?, 1, 1, 0, 0, 0, 1) \end{aligned}$$

Now, for every destination, we go through the rotations and for those that are multistretched (there is + in the multistretch vector), we add the corresponding component of the shortage vector. Then we divide by the number of rotations. So for instance for  $B$ , only its second rotation is multistretched, but the corresponding contribution from the shortage vector is 0:

$$\sigma_\beta(B, s) = \frac{0 + 1 \cdot 0 + 0}{3} = 0.$$

For  $A$ , both rotations are multistretched, but only the first one encounters a shortage:

$$\sigma_\beta(A, s) = \frac{1 \cdot 1 + 1 \cdot 0}{2} = 0.5.$$

Using the numeric shortage vector yields different shortages:

$$\sigma_\nu(A, s) = 0.043, \quad \sigma_\nu(B, s) \approx 0.007, \quad \sigma_\nu(C, s) = 0, \quad \sigma_\nu(F, s) = 0.$$

Hence in this case the strategy is admissible. The computation is the same as above, but we use the numeric shortage vector now, instead of the boolean one.

### 2.1.3 Other modeling possibilities and counterexamples

The model outlined on the preceding pages is complicated. During the model formulation phase, simpler models were investigated, but were not found satisfying. This section presents one such simpler model. Furthermore, an example is given demonstrating the dependence of the objective function and the constraint functions on the timetable – simpler models are likely to neglect this dependence and hence to lead to incorrect results.

The proposed model is presented below: Assume that our strategies are lists of city pairs. Then an intuitive solution to the above problem could be as follows: Assume that we are given virtual shortages for all pairs of destinations and furthermore “frequencies” of flight-pairs  $N_{i,j}$  for any two destinations  $i, j$  (i.e. this is the number of flight pairs with the first flight coming from  $i$  and the second flight heading to  $j$ ). Then an integer linear program can be formulated (decision variables  $d_{i,j}$  for  $i, j \in \text{DST}$ ; i.e. whether we multistretch a flight-pair from  $i$  to  $j$ ). Apart from the obvious constraints, we must furthermore ensure that the service level for every DST is higher than  $1 - \alpha$ . The optimization program then reads:

$$\begin{aligned} \max \quad & \sum_{i,j} d_{i,j} N_{i,j} \\ & \sum_i d_{i,j} v_{i,j} \frac{N_{i,j}}{\sum_k N_{k,j}} \leq \alpha, \quad \forall j \\ & d_{i,j} \in \{0, 1\} \end{aligned} \tag{2.3}$$

This is an integer linear program (ILP). This means that both the objective function and the constraint functions are linear in the decision variables and the decision variables take only values in a discrete set.

There exists software (for instance SYMPHONY in the Coin-OR package [19]) to get good solutions for ILP within reasonable time; using this software, solving (2.3) should be feasible. However, this approach completely neglects the restrictions from the beginning of Section 2.1.1. Hence the resulting strategies will be way too conservative in the sense that they will

contain less pairs and so lead to less multistretches than it is possible with our approach. Indeed, in (2.3) it is assumed that if we multistretch a city pair  $(i, j)$ , then all flight-pairs containing this city pair will be multistretched. However, because of the restrictions not all the flight-pairs are multistretched and so the true shortage for  $j$  is smaller. Therefore, the assumed shortages in (2.3) are overestimated and consequently the optimal solution contains less pairs than would be possible if the true shortages for strategies were taken into account.

Assuming that the percentages of rotations which indeed get multistretched are available, we could try to incorporate it – but even these percentages are timetable-dependent hence they cannot be readily computed. Consider the following example.

*Example 2.3* (Percentage of multistretches is timetable-dependent). Consider a registration day with rotations  $(A - B - B - A - A - B)$ . Then a strategy of multistretching all pairs comes up, after applying the restriction, with the actual multistretching pattern  $(-, +, -, +, -, +)$ . Hence the city-pair  $(A, B)$  is multistretched three times in the three occurrences, while the pairs  $(A, A)$  and  $(B, B)$  are not multistretched at all.

We have indeed encountered the demonstrated problem in practice: When dealing with one list strategies, we had two strategies which both included Prague and Nuremberg. However, one of these strategies was perfectly admissible while the second exhibited large shortages for both Prague and Nuremberg. One possible explanation is that in the second schedule, flights to Prague and Nuremberg were actually multistretched much more often. Another explanation is (since we used the boolean shortage vector) that the second strategy had differently “spread” multistretches and therefore “picked up” more shortage events in the data.

#### 2.1.4 Justification of using destinations as a basic unit

In the preceding text, destinations are implicitly considered as a main (explainable) factor for the variation of water consumption between different rotations and *therefore* the strategies decide which flight-pairs to multistretch on basis of destinations.<sup>12</sup> This section justifies that a destination of a rotation is indeed a factor influencing water consumption. In Section 3.3, more possible factors are considered as well as strategies deciding also on basis of these other factors.

Because of the format of the data as well as the definition of the service level, which explicitly works with destinations, it is very easy to work with individual destinations. Still, a question is if this is not superfluous – indeed, if destinations could be merged into bigger clusters, this would considerably reduce the state space and subsequently the time needed to search for a good solution. In view of this, two destinations can be considered equivalent and hence merged if they have the same water consumption distribution. We are interested which pairs of destinations are equivalent in this sense.

**Background on edf statistics.** To test for the equality of distributions of the samples of water consumption for different destinations (and in Sections 3.1.5 and 3.2 also for different purposes), we will use edf statistics and some tests based on them. In the following exposition, we draw heavily on [8].

Given a sample  $x_1, \dots, x_n$  of size  $n$ , we define the empirical distribution function (edf) by

$$\hat{F}_n(y) := \frac{|\{x \in \{x_1, \dots, x_n\}; x \leq y\}|}{n}.$$

---

<sup>12</sup>The emphasized *therefore* is however not completely straightforward: see also Section 3.3.2.

The edf-based statistics are often used to test goodness-of-fit, i.e. whether a sample comes from a specified distribution. Frequently used edf statistics for this purpose are for instance the Cramér-von Mises, Anderson-Darling or Kolmogorov-Smirnov statistic.

The Anderson-Darling test statistic  $A^2$  for one sample, i.e. to test goodness-of-fit, is given by

$$A^2 := n \int_{-\infty}^{\infty} \frac{(\hat{F}_n(x) - F(x))^2}{F(x)(1 - F(x))} dF(x), \quad (2.4)$$

where  $F$  is the “suspected” distribution from which the sample  $x_1, \dots, x_n$  is presumably drawn. However, this is not our situation. Instead, we have two samples and the question is whether they are drawn from the same distribution. For this purpose, the Anderson-Darling statistics was extended in [26] to two samples (and in [29] to  $k$  samples). Assume that we have the edf  $\hat{F}_n$  for the first sample (of size  $n$ ) and the edf  $\hat{G}_m$  for the second sample (of size  $m$ ). Let  $N := n + m$  and  $\hat{H}_N$  the edf of the combined sample. Then the test statistic for two samples is defined by

$$A_{\text{two}}^2 := \frac{nm}{N} \int_{-\infty}^{\infty} \frac{(\hat{F}_n(x) - \hat{G}_m(x))^2}{\hat{H}_N(x)(1 - \hat{H}_N(x))} d\hat{H}_N(x), \quad (2.5)$$

where the integrand is defined to be zero for values of  $x$  larger or equal to the largest value of the combined sample. We perform the test using the implementation from the R package `adk`.

We choose to work with the Anderson-Darling statistic since it is, together with the Cramér-von Mises statistic recommended in [31] *in the case of one sample*. We will discuss this recommendation more in depth in Section 3.2.2.

**Implementation.** Without any further information, a reasonable assumption is: if any destinations are equivalent, it will be in the first place those that are geographically similar – hence the time of flight is similar and also to some extent other flight characteristics; see also [3, Section 2]. Therefore, several groups of destinations are created and testing for equality of distributions is done only within these groups. For our dataset, all the available data are taken, testing is then done separately for E90 and F70. To make sure that no two destinations are declared equivalent just because of insufficient power of the test, only the destinations with at least 200 rotations are considered.<sup>13</sup> The groups are formed based on the country<sup>14</sup>:

- Switzerland: Zurich, Geneva
- Germany: Hamburg, Munich, Frankfurt, Stuttgart, Nuremberg, Berlin, Dusseldorf, Hannover, Bremen, Cologne
- Scandinavia: Trondheim, Oslo, Helsinki, Gottingen, Stavanger, Kjevik, Sandefjord, Linkoping, Bergen
- France: Marseille, Toulouse, Nice
- Italy: Bologna, Venice

---

<sup>13</sup>This is probably more than necessary – see Section 3.2.2 for an indication how small samples already provide reasonable power.

<sup>14</sup>Not all destinations were considered for both E90 and F70; destinations presented here are the union of the groups for both aircraft types – this is purely for the sake of brevity.

- Denmark: Billund, Copenhagen, Aalborg
- England and Wales: Birmingham, Newcastle, Manchester, Bristol, Humberside, Liverpool, Leeds, Norwich, Cardiff, Durham, London
- Scotland: Edinburgh, Aberdeen, Glasgow

All the pairs within the groups were tested for the equality of distributions. Note that the multiple testing problem is faced – however, since the tests are not used for explicit decision making, but rather for an insight, we do not make any corrections for it. The main result is: for most pairs, the null hypothesis is rejected (the p-value of the test is very close to zero, e.g.  $10^{-7}$  or similarly small numbers). For interest, we give here the pairs where the null hypothesis is not rejected on the 0.1 level:

- E90: Nuremberg and Stuttgart, Oslo and Trondheim, Manchester and Newcastle, Aberdeen and Glasgow
- F70: Fifteen pairs in England, four pairs in Scandinavia (3 of which involve Stavanger, two Gottingen and two Sandefjord) and four pairs in Germany (two of which involve Berlin).

Take into account that there are 55 pairs for England, 45 for Germany and 36 for Scandinavia. We conclude that rotations for different destination in general exhibit different distributions of water consumption and by merging the above “similar” destinations, the dimension of the state space would be reduced only in order of units.<sup>15</sup> It is therefore decided to leave destinations unmerged.

For a related analysis for large aircraft, see [3, Section 2]. Note that in Section 3.2, a similar coupling will be tried out. However there, the reason will not be reducing dimension of the state space, but instead saving destinations with few flights from being discarded.

## 2.2 Tabu search to determine two lists

This section discusses optimizing within the class of two lists strategies; a modification for one list strategies is also mentioned. Still, some of the concepts presented here will be also used in Section 2.3, which is concerned with optimizing within the class of pairs strategies.

As follows from a discussion in Section 2.1.2, we have complicated objective and constraint functions which we can hardly expect to express and optimize analytically. In such cases, a good solution can be searched for using the Local search.

### Local search

The Local search (see for instance [22, Section 1.1]) works as follows: given a current state  $s$  (in our case a strategy), one defines its *neighborhood*  $N(s)$ . That is a set of the states which are in some sense “similar” to the given state; we allow that this neighborhood contains unfeasible states. Within the neighborhood we search for a (feasible) state having the largest value of the objective function. This state is then chosen as the new current state and this process goes on. In a pseudocode, this is captured by Algorithm 1, where the objective function to be maximized is again denoted by  $\phi$ .

<sup>15</sup>While recalling that there were some 30 destinations with enough rotations per every period.

---

**Algorithm 1** Iterative Improvement

---

**Require:** Initial state  $s_0$

- 1: **repeat**
  - 2:   Find  $s \in N(s_0)$  feasible with maximal  $\phi$
  - 3:    $s_0 \leftarrow s$
  - 4: **until**  $\phi(s_0) \geq \phi(s)$ ,  $\forall s \in N(s_0), s$  feasible
- 

## Neighborhoods

In the current case, a state is given by two lists of destinations, inbound and outbound. We defined the following, rather intuitive neighbors (considering each  $d$  and  $b$  only if applicable given the current state):

- $\text{ADD}(d, b)$ : Add the destination  $d$  to the list  $b \in \{\text{inbound}, \text{outbound}\}$ .
- $\text{DROP}(d, b)$ : Drop the destination  $d$  from the list  $b \in \{\text{inbound}, \text{outbound}\}$ .
- $\text{SWAP}(d_{\text{ADD}}, d_{\text{DROP}}, b_{\text{ADD}}, b_{\text{DROP}})$ : Combination of a drop move and an add move.

We define  $N_{\text{ADD}}(s)$ ,  $N_{\text{DROP}}(s)$ ,  $N_{\text{SWAP}}(s)$  as the sets of all (ADD, respectively DROP, respectively SWAP) neighbors of the state  $s$ . The neighborhood  $N(s)$  is then defined as the union of  $N_{\text{ADD}}(s)$ ,  $N_{\text{DROP}}(s)$  and  $N_{\text{SWAP}}(s)$ . Note that also unadmissible neighbors are included – this must be so because we are able to filter these out only after evaluating them. Furthermore, if  $s$  is unfeasible, define  $N_{\text{DROP}}^{\text{bad}}(s)$  as the set of  $\text{DROP}(d, b)$  neighbors such that  $d$  is causing unfeasibility of  $s$ , i.e.  $\sigma(d, s) > \alpha$ . When talking about a SWAP move, the ADD component is its ADD move and similarly for the DROP component.

Note that searching the whole neighborhood  $N(s)$  is not a realistic option. Indeed, for example when it comes only to ADD neighborhoods, we have one neighbor for any destination which is not in the inbound list and one for every destination which is not in the outbound list – and recall that there are approximately  $m = 30$  destinations with enough flights. Similar reasoning holds for DROP neighborhoods. Finally, the number of SWAP neighbors is in principle the product of the number of ADD neighbors and the number of DROP neighbors. Even if we do not consider SWAP neighborhoods, we have several tens of neighbors. And since one evaluation can take several (tens of) seconds, one iteration could take a few minutes – that is way too many.<sup>16</sup> Hence the full neighborhood must be substantially reduced – this will be achieved mostly by randomly sampling several states from the full neighborhood.

### 2.2.1 Proof of concept – “Random search”

To test the very concept of searching the state space of strategies, we started with a very simple algorithm. Let  $n$  be a sufficiently large number. As an initial solution, we usually take the “multistretch all” strategy, i.e. both the inbound and outbound list contain all admissible destinations. This strategy is almost always unfeasible, but note that for the Random search this does not matter.

---

<sup>16</sup>Consider that a competitive strategy may have for instance 60 cities in case of two lists or 200 city pairs in case of pairs; hence just reaching a first “competitive” strategy could take many hours if starting from a “slip nothing” state.

---

**Algorithm 2** Random search

---

**Require:** Initial state  $s_0$

```
1: for  $i = 1$  to  $n$  do
2:   if  $s_0$  is feasible then
3:     Select a new state  $s$  randomly from  $N_{\text{ADD}}(s_0)$ 
4:      $s_0 \leftarrow s$ 
5:   else
6:     Select a new state  $s$  randomly from  $N_{\text{DROP}}^{\text{bad}}(s_0)$ 
7:      $s_0 \leftarrow s$ 
8:   end if
9: end for
```

---

An advantage of the algorithm is that it always tends to stay relatively close to the “critical level” which we provisionally define as the set of feasible states which become unfeasible if we perform any ADD move on them. Indeed, the farther the state is from the critical level, the higher proportion of ADD moves which lead to a feasible state and hence the algorithm is likely to make another ADD move after the current one. In such way, the algorithm tends to move to the critical level if it is currently far away from it. On the other hand, if the algorithm reaches an unfeasible state, it does its best to regain the feasibility again.

The disadvantage is that the algorithm wanders rather aimlessly. For instance, if there is one “bad” destination which causes shortages for several other destinations, the algorithm may first throw away the other destinations and only then the bad one as shown in the following

*Example 2.4.* Consider “good” (with low water consumption) destinations  $G_1, \dots, G_k$  and a “bad” (with very high water consumption) destination  $B$ . As a result, virtual shortages  $v_{G_i, G_j}$  are very low (say 0.01) while the ones with  $B$  are high, say  $v_{B, B} = v_{B, G_i} = 1.0$ . Assume that a frequency of  $B$  rotations is such that something more than 5% of flight-pairs contain  $B$ .

Then the following is possible<sup>17</sup>: starting with the “multistretch all” strategy, all destinations are “bad” in the above sense, i.e. with excessive shortages. And this will last until  $B$  is dropped from both the inbound and outbound list which may take long since dropping  $B$  is just as likely as dropping any of  $G_i$ .

Another disadvantage of the Random search is that there is no real drive to select solutions with high objective value. Indeed, if there are more possible ADD moves which lead to a feasible state, any of them can be chosen (or even an unfeasible one), not necessarily the one leading to the solution with the highest value of the objective function.

## 2.2.2 Idea behind Tabu search

Recall the idea of the Local search – we go from a state to state, the new state is always determined by searching through the neighborhood of the old state  $s_0$  and choosing the best (feasible) state  $s$  in it – in our case and in terms of functions  $\phi$  and  $\sigma$  from Section 2.1.2, it is the state maximizing  $\phi$  among the states in  $N(s_0)$  for which  $\sigma(d, s) \leq \alpha$  for all destinations  $d$ . There are two problems with this approach:

1. The algorithm ends up in a local maximum, i.e. a state  $s^*$  such that  $\phi(s^*) \geq \phi(s)$  for all feasible  $s \in N(s^*)$  without any other guarantees regarding the objective value of

---

<sup>17</sup>Assuming a suitable configuration of flight-pairs.

$\phi(s^*)$  in comparison with the global maximum. Assuming that we have found a local maximum, we may allow the algorithm to move away in the following iteration, but then it will likely get back to the local maximum in the very next iteration. As already mentioned, the neighborhoods as we have defined them are too large to be searched through completely – hence with some probability, the algorithm will not move back because the local maximum will not be in the searched through part of the neighborhood. However even then, the algorithm will probably “stay around” the local optimum for a while and so there will be a possibility that it moves back.

2. For the current problem, the evaluation of the objective function is by far the most expensive part of a possible algorithm. Hence we want to avoid repeated visits in all already evaluated states, not only the local maxima.

To address the mentioned issues, we augment a Local search algorithm with memory – using its improved version commonly known as the Tabu search. Its most prominent idea is to remember already visited states or some of their features, and forbid the Local search algorithm moving back into them.

The algorithm for two lists in Section 2.2.3 makes use of the memory in a very straightforward fashion. The one to determine list of pairs in Section 2.3 is more sophisticated and adapts some of the ideas presented in the book about the Tabu search [13] to the problem at hand.

### 2.2.3 Algorithm

An outline of the algorithm follows in Algorithm 3. As described there, the algorithm is very

---

**Algorithm 3** Tabu search for two lists

---

**Require:** Initial (feasible) state  $s_0$

- 1: tabu list  $T := \emptyset$
  - 2: **repeat**
  - 3:   Determine a restricted neighborhood  $\tilde{N}(s_0) \subset N(s_0)$
  - 4:    $\tilde{N}^*(s_0) := \tilde{N}(s_0) \setminus T$  {Tabu filtering}
  - 5:   evaluate all  $s \in \tilde{N}^*(s_0)$ ; choose  $s^*$  feasible with  $s^* = \arg \max \phi(s)$
  - 6:    $s_0 \leftarrow s^*$
  - 7:   update  $T$
  - 8: **until** stopping criterion
- 

simple, we now elaborate on the details (the full program code in R can be found in Appendix B.1):

#### Choice of the restricted neighborhood

The algorithm takes initial arguments determining how large the restricted neighborhoods should be – specifically, how many neighbors from  $N_{\text{ADD}}, N_{\text{DROP}}, N_{\text{SWAP}}$  should we generate? After some experiments, it seems that taking ten SWAP neighborhoods, two ADD neighborhoods and four DROP neighborhoods leads to a good performance.

*Example 2.5* (Drawing neighborhoods). Assume that we are working with only 5 destinations:  $A, B, C, D, E$ . Consider the state  $s_0$  with the outbound list  $\lambda_o = \{A, C, E\}$  and the inbound

list  $\lambda_i = \{A, D\}$ . The full neighborhood  $N(s_0)$  need not be determined explicitly. Instead, a list of destinations possibly added to at least one of the list is made (in this case  $B, C, D, E$ ) and a list of destinations possibly dropped from at least one of the list (in this case  $A, C, D, E$ ).

Determining a restricted neighborhood amounts to drawing the desired numbers of SWAP, ADD and DROP neighbors. How to do this? For instance, assume that we want an ADD neighbor. Then we first draw randomly from the list of destinations to add. If the destination  $d$  is only in one of  $\lambda_o$  and  $\lambda_i$ , then the move consists of adding  $d$  also to the other list. If  $d$  is in neither of the lists, we decide randomly to which one it should be added. For drawing a DROP neighbor, the procedure is analogous. And a SWAP neighbor is obtained by combining (simultaneously) one DROP move and one ADD move.

Note that the restricted neighborhood may end up containing one neighbor more than once. Since the full neighborhood tends to be large, this does not happen too often and so it is not considered a serious issue.

At the beginning, we tried ignoring DROP neighbors, however this often led to states for which all their sampled neighbors were unfeasible – in which case no new state is chosen and instead we go on with the current active state. Therefore, now we include the DROP neighbors by default as well. However, the program still offers an opportunity to start considering DROP neighborhoods only after there has been no improvement for a certain number of iterations. This can then make several initial iterations faster because there are not so many neighbors to be evaluated.

## Tabu list

We have tried out two kinds of tabu lists:

1. Including only the states that have ever been active and also the neighbors from the previous iteration.
2. Including all states that have ever been evaluated, i.e. they have been neighbors of active states at least once.

Since an important function of the tabu criterion in our case is reducing the number of evaluations as much as possible (in comparison with a “standard” Tabu search where the most important point is only to avoid cycling and diversify the search) we have usually considered the second option.

This is a very crude method for two reasons:

- We use relatively a lot of memory (to store the tabu lists) and processor power (to filter the restricted neighborhood using the tabu list – see line 4). However, even if we run the program for long, we do not use more than e.g. 2000 evaluations.<sup>18</sup> Even for a tabu list with 2000 entries, the memory required is negligible and even the filtering on line 4 is immediate.<sup>19</sup> On the other hand, one evaluation takes (at least) several seconds; we conclude that the filtering is indeed worthwhile. Should the size of the tabu list ever become an issue, a possibility is to code the states as integers using a hash function (see

---

<sup>18</sup>In contrast, in the algorithm described in Section 2.3, where we use faster evaluations, the number of evaluations can be factor 10 larger.

<sup>19</sup>The filtering corresponds to comparing all states from the restricted neighborhood to all states on the tabu list.

[13, Section 7.4]) – this saves memory as well as makes filtering quicker since comparing two integers is faster than comparing two lists – assuming that the hash function is quick to compute.

- The tabu status is permanent – once a state becomes tabu, it never turns admissible again. This is again in contrast with “usual” Tabu search procedures – there, the tabu status is usually temporary; this has to do with the fact that for the “usual Tabu search”, the tabu status is not based on a list of states, but rather on a list of characteristics of states or moves – these characteristics apply to more states than just the one which triggered them and making them tabu forever can seriously restrict the search. For instance, a characteristic could be a move containing one destination; if this was permanent, after once triggering the tabu status of the move, we could never do anything with the destination anymore.

Even with these drawbacks, the method seems to work well. It avoids cycling and also, importantly, the reevaluation of already evaluated states.

It might in principle happen that a restricted neighborhood becomes empty after the tabu filtering. However, for an aircraft type and a period with a reasonable amount of flights, this has never happened.

### Stopping criteria

Two stopping criteria against which the termination can be tested on the line 8 have been implemented:

- Number of iterations,<sup>20</sup> i.e. the number of active states throughout the run of the algorithm.
- Number of iterations since the last improvement of the objective function: We experimented with different values and taking 20 iterations works fine – this usually corresponds to something between 40 and 100 iterations in total. Requiring only 10 iterations since the last improvement is already too little.

### Other aspects

It is necessary that, whenever we find a new state better than any found so far, we store it in the memory.

Unlike for the Random search algorithm, for the Tabu search algorithm we require the initial state to be feasible. Either we can start from the “multistretch nothing” state or we can start from the “multistretch everything” state using the Random search algorithm and as soon as we find a feasible state, we start the Tabu search algorithm with the feasible state as the initial state.

We note that in contrast to “usual” Tabu search procedures, we filter using the tabu list even before the neighbors are evaluated – this is because of the already mentioned high cost of evaluating the objective function.

---

<sup>20</sup>Alternatively, the total number of evaluations could be considered – that would better reflect the required computational effort.

### Adjustment for computing just one list

After the implementation of Algorithm 3, it turned out to be useful to be able to optimize also within one list strategies. The one list strategy can be seen as a special case of a two lists strategy where both lists are equal.

To this end, we have therefore adjusted the original algorithm. It is possible to require the initial state to have inbound and outbound lists equal. Then special neighborhoods for this case are defined, namely ADD neighbors add to the current state one destination to *both* the inbound and outbound list. Analogously, we define DROP and SWAP neighbors.

## 2.3 Tabu search to determine a list of city pairs

In the future, after a small adjustment to the planning system, more complicated strategies than one or two lists of destinations can be considered. In Section 3.3, other variables than just the inbound and outbound destination are considered. But improvements can be made also when still considering only destinations. A very natural generalization of the one list of inbound and outbound destinations is considering a list of city pairs. Note that in view of the Assumption 2 in Section 3.1.1, the ordering of the cities within the pair does not matter.<sup>21</sup> The comparison of this class of strategies compared to the one list or two lists strategies will be provided in Section 2.4.

In this case, neighborhoods look differently, even though the underlying idea is the same as in the first algorithm). Let  $s$  be a state, i.e. a strategy, i.e. a list of city pairs. The neighbors of  $s$  are the states obtained by applying the following moves to  $s$  for any given city pair  $p$ :

- ADD( $p$ ): Add the pair  $p$  to  $s$ .
- DROP( $p$ ): Drop the pair  $p$  from  $s$ .
- SWAP( $p_1, p_2$ ): Combination of a DROP( $p_1$ ) move and an ADD( $p_2$ ) move.

Again denote the full neighborhood of  $s$  by  $N(s)$  formed by the above moves for all  $p$  that make sense for the current  $s$ . An important point to notice is that (full) neighborhoods are larger compared to the case in Section 2.2. To see this, consider that the number of pairs of destination with sufficient number of flights – say at least 10 per period – is in order of several hundred. The state space is larger as well. To deal with this extra complexity, we have on the one hand adjusted evaluations for this case to make them faster (see Section 4.2); on the other hand, the algorithm now makes much better use of the information available in the problem. Unlike in the algorithm described in Section 2.2.3, this time restricted neighborhoods (also called “candidate lists” in [13]) are not drawn uniformly (given the required constraints) from full neighborhoods, but the drawing is “biased” in order to obtain better states.

The algorithm was heavily influenced by some ideas in the book about the Tabu search [13]. These influences will be acknowledged where appropriate. Since the ideas in the book are mostly supported by heuristic arguments and computational evidence rather than formal theorems, some experimentation was needed to determine which ideas are worthwhile to ultimately adopt and which not – this will be briefly summarized at the end of Section 2.3.1.

---

<sup>21</sup>The pairs strategy is indeed a generalization of the one list strategy. Indeed, given a list with destinations  $A_1, \dots, A_k$ , consider just all (unordered) pairs from this list to get a corresponding pairs strategy. Note that pairs strategies is not a generalization of the two lists strategies since in the latter, the order may matter.

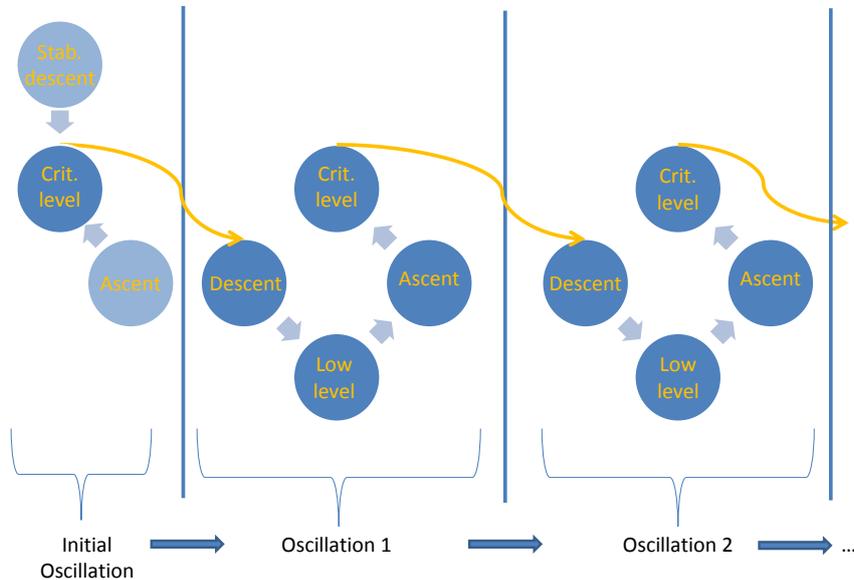


Figure 2.1: Substage

### 2.3.1 Algorithm

We begin with a brief outline of the algorithm, the details are provided afterwards.

#### Outline

On the very lowest level, every iteration looks like an iteration in Algorithm 3. However, the choice of restricted neighborhoods is not the same over time: instead, the algorithm oscillates between different modes and during each mode, different restricted neighborhoods are considered: this is shown in Figure 2.1 – each bubble corresponds to a different mode and it lasts for a certain number of iterations, which itself depends on the mode. Different modes change in a cyclical fashion, one such cycle is called an *oscillation*. After a certain number of oscillations,<sup>22</sup> the algorithm restarts. For each oscillation, the best state found during it is remembered.

We call the segment of the algorithm from one restart until another a *substage*. Different substages are almost independent from each other, the point of separating them is that different substages may end up finding different best states (local maxima), it is then desirable to choose the best one. A sequence of several substages could already make up the whole algorithm, but it is not so. Instead, we call it *Stage 1*. After its end, i.e. after running through a predetermined number of substages  $N_{\text{substages}}$ , we proceed to *Stage 2* which consists of two procedures modifying the existing best states (from individual substages), so that we obtain even better states – these procedures are applying *elite moves* and *path relinking*. The overall bird’s view of the algorithm can be found in Figure 2.2.

<sup>22</sup>This number is denoted by  $N_{\text{substages}}$  and it is 3 by default, not counting the initial one.



- *descent*: For each iteration, a neighborhood consists of one DROP neighbor. Ends after a predetermined number of iterations.<sup>24</sup>
- *low level*: Neighborhoods consist of several (the exact number varies, see below) SWAP neighbors. From these the best one is chosen; and eventually, we update the best one during the whole mode duration. Ends after a number of iterations without finding a better state than the current best.
- *ascent*: Neighborhoods consist of several ADD neighbors;<sup>25</sup> from these the best feasible one is chosen to obtain a new state. Ends at the iteration when no more feasible ADD neighbors are found – so that we are presumably at the critical level.
- *critical level*: Works similarly as the low level mode – Neighborhoods consist of several (the exact number varies, see below) SWAP neighbors. From these we choose the best one; and eventually update the best one during the whole mode duration. Ends after a number of iterations without finding a better state than the current best.

Still, the SWAP neighbors in this case are not sampled uniformly from the full SWAP neighborhood, but rather from a smaller one. The reason for this is that, as already noted, there are likely to be fewer admissible SWAP moves at the critical level. Indeed, consider a swap move, let the ADD component be the pair  $(A, B)$  for some destinations  $A, B$ ; and the DROP component be the pair  $(C, D)$ . Since we are at the critical level, adding the pair  $(A, B)$  on its own should lead to an inadmissible state. However, if both pairs are disjoint, the effects of adding the pair  $(A, B)$  and dropping the pair  $(C, D)$  are almost independent.<sup>26</sup> Therefore one can assume that the SWAP move would lead to an inadmissible state as well since we add the pair  $(A, B)$  and do not compensate for it. The inadmissibility of the resulting state is caused by either the high shortage level at  $A$  or high shortage level at  $B$  (or both).<sup>27</sup> This argument shows that we should require that either  $C$  or  $D$  is equal to either  $A$  or  $B$  – in this way, there is a chance that the shortage caused by adding  $(A, B)$  will be compensated by dropping  $(C, D)$ .

The initial oscillation (the one immediately after (re-)starting) is somewhat special – its exact form depends on the initial state. If the substage starts with an unfeasible state, then we start with a special mode:

- *stabilizing descent*: Neighborhoods consist of at most several<sup>28</sup> neighbors – we end drawing the neighbors if a feasible state is encountered; if this does not happen, we choose the next state from the neighborhood in the way that reduces one of the shortages which cause infeasibility. Stabilizing descent ends when a feasible state is encountered.

Now, if the initial state was feasible or it was unfeasible and we went through the stabilizing descent mode, we proceed to the ascent mode and from this to the critical level – when this finishes, it is the end of the initial oscillation. It might seem unintuitive to run the ascent mode after the stabilizing descent – however note that the stabilizing descent is not very

---

<sup>24</sup>By default ten.

<sup>25</sup>By default five and up to additional five are considered if none of the first five is feasible

<sup>26</sup>Not necessarily; but as a heuristic reasoning this is acceptable.

<sup>27</sup>There can exist some bizarre exceptions, but we disregard them now.

<sup>28</sup>By default three.

“careful” when throwing pairs away; hence after it has ended, we may (and usually will) be *under* the critical level.

After finishing each critical level, the best state achieved at the level is stored. Hence after Stage 1, we have  $N_{\text{substages}} \cdot (N_{\text{oscillations}} + 1)$  of such “elite” solutions. For the future, we also save during every substage several good<sup>29</sup> SWAP moves (during low and critical level modes) and ADD moves (during ascent modes).

### Selecting the number of neighbors at low or critical level

It is too expensive to evaluate the full neighborhood. To face this problem, [13, Section 3.2] considers “candidate list strategies” to reduce the computational effort. Instead of taking the restricted neighborhood (i.e. the candidate list) to be just a subset of the full neighborhood of a constant size, the following method should be able to choose the size more suitably so as to draw enough interesting solutions.

A special case of the “Aspiration Plus” candidate list strategy [13, Section 3.2.1] is used: Consider numbers Plus, Min, Max.<sup>30</sup> Draw (randomly) and examine Min neighbors; we are now always talking about SWAP neighbors, respectively the subset thereof mentioned above in case of the critical level. Based on the objective values and the feasibility of Min neighbors, choose an objective value threshold.<sup>31</sup> Next draw and evaluate neighbors until we either reach the threshold or we have already drawn Max. Then, draw Plus more neighbors; but only if there will have been less than Max in total, otherwise draw what remains until Max. Finally, from all the drawn neighbors, choose the (feasible) one with the largest objective value.

### Short term memory

This paragraph describes the “short term memory” aspects – as opposed to the “long term memory”. As described in [13], the short term memory usually serves to restrict the full neighborhoods of states (in this sense, the memory features from the algorithm in Section 2.2.3 were short term) while the long term memory can also serve to expand neighborhoods in the sense of getting to solutions otherwise unavailable. The long term memory features in the current algorithm are to be described later – restarts, path relinking and applying elite moves.

The most common type of short term memory, the one which we will use as well, is called *recency-based* memory. It consists of remembering recently visited solutions, or more often just some of their features, and forbidding moves leading to solutions with these features. As opposed to the algorithm for two lists, now we do not remember all the solutions ever visited. Every time a move is performed, the pairs forming the move (i.e. either one pair in case of ADD and DROP or two pairs in case of SWAP) are declared tabu – the pair cannot be moved again while its tabu status holds. The tabu status lasts only for several turns though, not forever. Hence a solution feature in our case is a presence or absence of a certain city-pair.

How long exactly does the tabu status last? This duration is also called the *tabu tenure*. This could be always a constant number, however, to achieve more variability, we adopt in-

<sup>29</sup>I.e. feasible and bringing as high change in the objective value as possible.

<sup>30</sup>By default Plus = 10, Max = 30, Min = 5.

<sup>31</sup>In our case, if there were no feasible states within neighbors, then we impose no threshold, otherwise the threshold is the maximum of objective values among the feasible neighbors so far augmented by a number dependent on the preferred weight (to be defined later) of the substage (the higher the weight, the larger the number).

stead the “systematic dynamic tenure” described in [13, Section 2.5]. The systematic dynamic tenure means that the values of the tenure are drawn periodically from a list. Our default list is dependent on a stopping parameter, however under the default settings it is equal to (5, 7, 10). Note that these are rather small numbers – but some experimenting with the admissible lengths revealed that taking the tenures to be longer already restricts the search too much in the sense that the neighborhoods sometimes become too small. The current values seem to work well for the low and critical level modes.

### Restarts and long term memory

Once a substage is finished – i.e. the number  $N_{\text{oscillations}}$  of oscillations has been performed – a new substage begins by a restart. That means that we start again from a scratch, using “slip nothing” as the initial strategy. The motivation behind restarting is that completely different initial conditions can lead to finding a local maximum different from those encountered before.

A somewhat different restarting possibility (we will call it “soft restarting” later on) is to begin with a different initial state which takes into account the “long term memory”. This is inspired by [13, Section 4.3.2] where restarting based on long term memory is illustrated on examples from scheduling and location/allocation problems. To determine how the initial state should look like, we use the so-called *transition frequency* – how many times has a given feature (in our case a city-pair) changed since the last restart?

What we do is to remember during the preceding substage how many times which pairs were added and dropped. For the new initial state, we first consider the final state from the last substage, but then drop from it all pairs contained in it that have been moved at most once and add to it those not contained in it that have been moved at most once. In this way, we create a state that is probably dissimilar to anything that has been seen during the last substage. Indeed, switching the pairs which have never been moved clearly contributes to the dissimilarity; for the pairs moved once: note that if we started before from the “slip nothing” state, a pair added only once is likely to have been added during the very first ascent mode and hence has been also around for too long.

### Using problem specific information

Given rotations and tanking data for the period on which we train, we can do quite a lot of computations with it. Even though we cannot solve the problem just using these computations (this was discussed in Section 2.1.3), we can still use such computations to influence our choice of restricted neighborhoods. Two quantities, each computed for every considered pair of destinations  $(A, B)$ , that we will use are

- coverages: what is the probability that if an airplane flies a pair of rotations – first to  $A$  and then to  $B$  – and we slip it, then we will still have enough water. If we have already computed virtual shortages, the coverages are obtained as 1 minus the respective virtual shortage.
- frequencies: i.e. how many flight-pairs for a pair  $(A, B)$  are there during the period.

The idea is to use these when drawing neighbors. Namely, when selecting a new neighbor, we first check that its tabu status is nonactive. Afterwards, we can still reject it with certain probability on basis of its frequency or coverage. This holds for DROP and ADD neighbors; for SWAP neighbors, we consider the DROP and ADD component separately.

Rejection on basis of coverage is simple: adding a pair is more likely to be rejected if the pair has a low coverage; on the other hand, dropping is more likely to be rejected, if the pair has a high coverage.

Rejection based on frequencies is more complicated. First, divide the pairs in three groups according to frequencies, i.e. the most flown pairs are in the first (“heavy”) group and the least flown pairs are in the third (“light”) group. Now consider a substage – it consists of several oscillation. The first oscillations are set to prefer “heavy” pairs, after that “medium” pairs are preferred and during the last oscillations, the “light” pairs are; i.e. adding or dropping a pair during a “wrong” oscillation has considerable probability of being rejected.<sup>32</sup>

The idea is building up a “skeleton” for good solutions during initial oscillations using heavy pairs – these are most likely to have a considerable impact in the terms of objective function and constraints. During later oscillations, we tune up this skeleton using lighter pairs. This idea is elaborated on in [13, Section 5.6.1].

### Path relinking

The idea of “path relinking” (taken from [13, Section 4.5]) is “mixing” two good solutions, starting out from one of them and gradually incorporating the features of the second one. This is done in hope of creating a solution combining the strengths of both of them.

In our case, one chooses an “initiating solution” to be the best state from one mode and the “guiding solution” to be the best state from another mode. We consider all pairs that are in the guiding solution, but not in the initiating solution and try to, for every such pair,

- i) Add it to the initiating solution.
- ii) If adding does not lead to a feasible state, try to swap the pair with a few other pairs – we accept the swap only if it leads to a state with a higher objective value (and it is still feasible).

Path relinking is used on two occasions:

1. After finishing an oscillation, we try to relink the best solution from the low level of the same oscillation to the best solution from the critical level. In this way, we use the work done during the low level which would otherwise serve only to “remix”.

This is motivated by the “proximate optimality principle” discussed in [13, Section 5.5] – this principle “stipulates that good solutions at one level are likely to be found close to good solutions at an adjacent level.”

2. After finishing Stage 1, we try, in Stage 2, to relink elite solutions from different substages together.<sup>33</sup>

We note that, even though one elite solution from every oscillation is stored, the very best elite solutions are those from the last oscillation of a substage – this is because of the strategy of preferring “larger” building blocks earlier and “tuning” the solution only later. This was also tested empirically. The relinking is therefore used only with these very best solutions.

---

<sup>32</sup>An exception to this rule is dropping pairs with very low coverage – this can be done during any oscillation, otherwise such a pair might ruin admissibility – and dropping pairs when the stabilizing descent mode is active; in that case, rejection of dropping on basis of frequency is also overridden.

<sup>33</sup>Not all of them together, but quite a large subset of such pairings.

## Applying elite moves

For every substage, the algorithm collects “elite moves” – i.e. such that have led to the largest improvement in the objective value, but also to a feasible solution.<sup>34</sup> During the second part of Stage 2, we make use of them.

For each of the most elite solutions (i.e. as in the case of relinking, only those from the ends of substages count; and possibly those relinked during the first part of Stage 2), we try out all the elite moves on it. We perform a move on the solution if it leads to a new feasible state; furthermore, we perform an SWAP move only if it leads to an improvement of the objective value – this because we are in the very last part of the algorithm, so there is no reason to produce worse solutions than we already have.

## Effect of different parameters and features

At the beginning, it is fair to note that it is difficult to exactly quantify effects of all the parameters and features coming into play – there are too many of them, there is a lot of interactions among them and finally, running the full tabu search algorithm takes long.<sup>35</sup>

Still, experimentation yielded several observations. The following results are mostly interesting from the qualitative perspective, the quantitative information should be taken with a grain of salt since the number of experiments was limited.

- Using frequencies and coverages: Useful to reduce the state space; without weights, the algorithm tends to spend a lot of time shuffling with very small destinations all the time.
- Tabu tenures: When these were set longer, they became too restrictive (i.e. the neighborhoods were too small).
- How many substages – the ideal number seems to be 3. Notably, the states from later oscillations are better than those from the earlier ones. Interestingly enough, the best states from different substages tend to be very different, in the sense that there are e.g. 40 city pairs contained in one elite strategy and not in the other one or vice versa.
- Elite moves: Proved to be not really useful. Therefore, they are now turned off by default.
- Relinking: Clearly leads to better solutions. In case of E90, this can for instance amount to 60 more multistretches on top of 1550 after Stage 1 of the algorithm for pairs.
- Aspiration plus: Does not produce better solutions, but the number of oscillations needed can be 25% smaller – when compared to just taking neighborhoods of constant size every turn.<sup>36</sup>
- Oscillations: Lead to somewhat better solutions in comparison with always staying at the critical level. Again taking the case of E90, the gain may be 60 multistretches.

---

<sup>34</sup>By default, we remember 3 SWAP moves and 3 ADD moves per substage.

<sup>35</sup>That is, if we take a sufficiently large subset of data. If we take a small subset, it is a question how reliable the results are since many of the features were introduced specifically to address the problem of a large state space (which corresponds to a large amount of data since destinations are not considered if they have too few flights).

<sup>36</sup>The size of the neighborhoods being similar to the average size for the Aspiration plus.

- When both oscillations and frequencies are disregarded, then this leads to slightly better solutions, gain being e.g. 15 multistretches – even though relinking compensates for this. However, the search costs considerably more iterations – some 40% more.
- Soft restarts: Number of iterations decreases by cca 15%.

### 2.3.2 Heuristic for pairs strategies

As an alternative to the Tabu search for pairs, we present here an intuitively attractive heuristic to find a good pairs strategy:

1. Start with a state slipping all pairs with the virtual shortage less than  $\alpha$ .
2. Afterwards, start adding other pairs in order of their increasing virtual shortage. Stop when adding further would yield an unfeasible state.

The first step always leads to a feasible state, at least when using the numeric shortage vector. The heuristic was tested with winter 2010 and E90.<sup>37</sup> The first challenging fact was that there were actually no pairs with the virtual shortage less than 0.05 (for F70, it was approximately half of all pairs). After applying the heuristic, one ends up with a strategy which multistretches 1505 rotations – this is to be compared with well over 1600 for a standard run of the Tabu search algorithm. On the other hand, one needs only 152 evaluations, as compared to more than 10000 for the Tabu search.

## 2.4 Comparison of various methods

We conclude this chapter with a brief overview of the results achieved with different implemented methods. Some of the methods are explained only in the later sections, so the reader can return to this section as appropriate. Table 2.2 captures the amount of multistretched rotations for the best strategy found using the given method. Data used is those of winter 2010. Note that because of the randomness inherent in the Tabu search algorithms, the results are only indicative and small differences should not be taken too seriously.

A better way to compare would be to run a larger number of tests (for instance a hundred for every method) and then give aggregate results. This was not tried because of prohibitive computational requirements of such an endeavor.

Hereby is a brief explanation for Table 2.2:

- Results in italics do not originate from tests – they are only guesses based on theoretical considerations.
- A question mark means that the result was not computed for winter 2010 and that we are not confident to provide any guess.
- The “pairs”, “two lists” and “one list” classes of strategies are explained in Section 2.1.1.
- “No difference” refers to no difference more precisely no or too small to be considered) when compared to the pure Tabu search algorithm – i.e. either for pairs, two lists or one list.

---

<sup>37</sup>For F70, the results are mentioned in Section 2.4.

- Only E90 was considered with the 110 L tank because for F70 there is no possibility to enlarge the tank to this size.
- “pairs + triples” means finding first a good pairs strategy and enhancing it further by additionally multistretching some triples of rotations – more explanation is to be found in Section 4.3.
- Coupling of small destinations to larger ones is explained in Section 3.2.
- Splitting according to departure times is done following the rules found in Section 3.3.5.
- The heuristic to determine a pairs strategy is discussed in 2.3.2.
- The (basic) pairs strategy fares surprisingly badly for F70. This has two reasons. First, for F70, it is possible to multistretch almost all flights – hence the extra discriminatory power of the pairs class of strategies provides little advantage. Second, the bound of 10 observations for a flight-pair to be considered is very restrictive in this case . Indeed taking the bound equal to 5 instead leads already to 4368 multistretches. Note that further splitting according to departure times again aggravates the problem.

Method \ Aircraft	F70	E90
two lists	4246	912
one list	4180	850
two lists + smaller lower bound (25) for cities	4355	931
two lists + coupling of small destinations	4295	1017
two lists + 110 L tank	NA	2441
pairs	4164	1632
pairs + coupling of small destinations	4229	?
pairs + splitting according to departure times	no difference	?
pairs + splitting according to departure times + smaller lower bound (5) for pairs	no difference	1683
pairs + triples	4544	<i>no difference</i>
pairs using the heuristic	4127	1505

Table 2.2: Differences in the number of multistretched rotations for different methods

### 2.4.1 Validation

As mentioned, to come up with a solution (i.e. a strategy) for the problem (2.1), we approximately solve the same problem using the data from the previous period  $D_{\text{old}}$ . To this end, the algorithms in Sections 2.2 and 2.3 have been developed. Still, the question is how the strategy  $s_0$  found by optimizing using  $D_{\text{old}}$  really fares on  $D_{\text{new}}$ . Indeed, we risk *overfitting* – i.e. the features of  $s_0$  might be too closely tuned to  $D_{\text{old}}$ , so that its performance on new data – in our case  $D_{\text{new}}$  – may be inferior to that of a solution which is simpler and hence perhaps not so good on  $D_{\text{old}}$ . See [16, Section 7] for a relevant discussion which goes deeper. Note that the one list class of strategies is the simplest while the pairs class is the most complicated. Hence, we expect most problems to arise for the pairs strategies.

There are two differences between  $D_{\text{new}}$  and  $D_{\text{old}}$ :

1. Differences in tanking which in principle correspond to the differences in water consumption.
2. Differences in the flight timetable.

The differences in tanking are explored in Section 3.3.6 and are found to be existent but rather small. We therefore expect that if problems arise, it will be due to changes in the timetable. The possible problems are

- the strategy  $s_0$  suffers excessive shortages when used for  $D_{\text{new}}$
- the strategy  $s_0$  multistretches considerably less rotations on  $D_{\text{new}}$  than a strategy  $s_1$  found specifically for  $D_{\text{new}}$  (that said, it is of course to be expected that  $s_0$  performs *somewhat worse* – the issue would be, if it was too much)

We note that the second problem can be fixed only to a certain extent – if  $D_{\text{new}}$  contains new destinations not contained in  $D_{\text{old}}$ , there is nothing we can do with that.

Several tests have been done to assess the different performance on  $D_{\text{new}}$  of solutions  $s_0$  and  $s_1$ .

- Early on during the project, a test for two list strategies was done – this revealed that including small destinations in  $s_0$  often leads to very high shortages for these destinations when used in  $D_{\text{new}}$ . This was the motivation to introduce the restriction 3 in Section 2.1.1.
- For one list strategies, with  $D_{\text{old}}$  corresponding to summer 2010 and  $D_{\text{new}}$  to summer 2011 for E90 with 110 liters tank: There are no problems with shortages. The strategy  $s_0$  for summer 2010 multistretches 76% of the amount of multistretches achieved by a strategy  $s_1$  specifically optimized for summer 2011; however, most of the difference is due to inclusion of two destinations in  $s_1$  which were not flown at all by E90 during summer 2010 and hence  $s_0$  had no chance of picking them up.

This result can be considered very satisfactory – it shows that we do almost as well as we possible can, given the difference of the timetables – which we are not able to predict.

- For two lists strategies, F70 with  $D_{\text{old}}$  corresponding to winter 2009 and  $D_{\text{new}}$  to winter 2010: Then we suffer excessive shortages for Munich (7.6%) and Billund (6.6%), but this strategy multistretches even slightly more than the strategy optimized directly for winter 2010.

Since the two shortages (among 32 destinations with enough rotations to allow multi-stretching) are not too high, we conclude that this test gives support to our approach as well.

- For pairs strategies, E90 with 90 liters tank, with  $D_{\text{old}}$  corresponding to summer 2010 and  $D_{\text{new}}$  to summer 2011: Here, the old strategy achieves 61% of multistretches compared to the new strategy, when tested on the new data. There is a problematic shortage rate for Goteborg (14.2%), the 5% level is also exceeded in Zurich – 7.6% – and Marseilles – 5.4%.

While the underestimation of shortages is not that serious, the amount of multistretches compared to the new strategy seems at first disappointing. The question is whether this

occurs because we are overfitting, i.e. the pairs strategy is already too “fine-grained”. If this is the case, then we would expect a simpler strategy perform better on  $D_{\text{new}}$ . Upon this finding, we have therefore compared the mentioned pairs strategy to the one list and two lists strategies (also trained on  $D_{\text{old}}$ ) – it turns out that the pairs strategy works on  $D_{\text{new}}$  still by far the best when it comes to the number of multistretches.

We conclude that the pairs strategy should still be deemed superior among the strategies that we have. A possible further step would be to find pairs strategies when some pairs are either merged or some pairs are split and to evaluate these. Furthermore, a quantitative method to suggest which shortages are already serious and which are not could be used – for instance one could decide among solutions found in different classes of strategies using the function defined later in (3.15).

## Chapter 3

# Estimation of water consumption

### 3.1 Estimation under multistretching

The goal of the estimation is to find the distribution governing the water consumption in order to use it as an input for the evaluation procedure of Chapter 2. Indeed, using the distribution estimate, we can obtain a numeric shortage vector, in the way described in Section 2.1.2, and run the evaluation procedure with it.

#### 3.1.1 Problem formulation

To compute the numeric shortage vector, we need the quantities  $\mathbb{P}\{Y_{d_1, d_2} > \text{TC}\}$  for  $d_1, d_2 \in \text{DST}$ . Hence it is enough to obtain the distributions of random variables  $Y_{d_1, d_2}$ . However, we have often only few realizations of these<sup>1</sup> and under multistretching we may have for some pairs no realizations at all. In view of this, we do not try to estimate the distribution of  $Y_{d_1, d_2}$  directly, but we instead carry out the following steps:

1. Formulate a parametric model for  $X_d$ ,  $d \in \text{DST}$ : That is, we assume that  $X_d$  has a density  $f(x; \theta_d)$  where  $\theta_d$  is a parameter specific to  $d$ , but the function  $f$  is the same for all  $d$ . The proposed parametric models can be found in Section 3.1.4 – most important are the “delta-gamma” and “normal” models, both will be properly defined there. In Section 4.5, we briefly consider the possibility of nonparametric modeling, but for the reason mentioned there, this is eventually abandoned.
2. Estimate the parameters  $\theta_d$ : To this end, two different estimation procedures will be presented (“MLE” in Section 3.1.2 and “GMM” in Section 3.1.3).
3. Obtain the distribution of  $Y_{d_1, d_2}$ : Since we already know parameters  $\theta_d$  (at least their estimates), the density of  $Y_{d_1, d_2}$  is given by the convolution  $f(\cdot, \theta_{d_1}) * f(\cdot, \theta_{d_2})$ .<sup>2</sup>

Now to begin with, consider what happens when we do not multistretch – then there is for every  $d \in \text{DST}$  a couple of realizations of  $X_d$ :  $x_d^{(1)}, \dots, x_d^{(n_d)}$ . Estimating the distribution of  $X_d$  then does not pose any challenge (apart from correctly specifying the model or choosing

---

<sup>1</sup>As a lower bound for a city-pair, we often take 5 or 10 observations and taking this bound higher actually seriously restricts the percentage of considered rotations – in some cases, 10 is already too many.

<sup>2</sup>This assumes that  $X_{d_1}$  and  $X_{d_2}$  are independent – this will be postulated by Assumption 2.

for a nonparametric procedure – see the beginning Section 4.5 to see how this was done in the Preliminary study).

On the other hand under multistretching, the estimation is more complicated. For example, consider Figure 3.1. If we do not multistretch, the available observations are  $x_A = 25$ ,  $x_B = 13$ ,  $x_C = 16$ . On the other hand, if we multistretch the rotation to  $B$ , we have only observations  $x_C = 16$  and then  $y_{A,B} = 38$ . Assuming additionally that the rotation to  $D$  is multistretched as well, there is suddenly no “direct” data about the variables  $X_d$  at all!

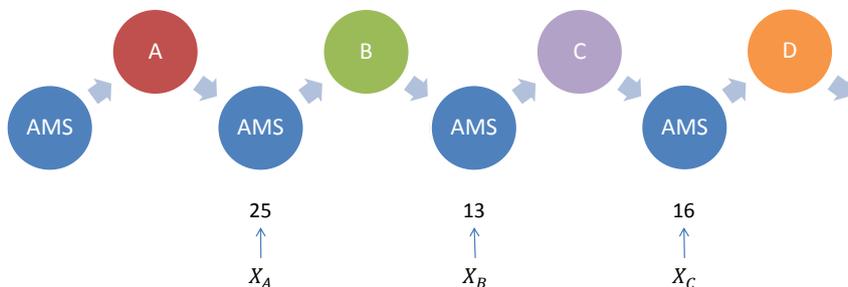


Figure 3.1: Towards estimation under multistretching

Hence under multistretching, we will have for many destinations only handful of direct data; for them, most information about tanking will be contained in the variables  $Y_{d_1, d_2}$ . Assume for the time being that there is actually no direct data and the only information available are the variables  $Y_{d_1, d_2}$ . We will see in Section 3.1.6 how to incorporate also the direct data.

Consider the following:

**Assumption 2.** All water consumptions  $X_d^{(i)}$  are independent.

The justification why this assumption is approximately valid can be found at the end of Section 3.3.

### 3.1.2 Maximum likelihood estimation

Our primary approach to the problem is using the Maximum likelihood estimation (MLE). The reason for this are the favorable asymptotic properties of MLE if certain assumptions are satisfied.

Before explaining the workings of MLE, let us first have a look at an illustrative example.

*Example 3.1.* Taken from [34, Section 4.3]: We are flipping a loaded coin 10 times. Assume that  $p$  is the (unknown) probability that we get a head. Let  $X$  the number of heads out of 10 flips. After 10 flips, we got three times a head. The probability (“likelihood”) that this would happen is

$$\mathbb{P}\{X = 3\} = \binom{10}{3} p^3 (1-p)^7.$$

This probability is the highest for  $p = 0.3$ . The Maximum likelihood estimate of  $p$  is then defined by  $\hat{p} := 0.3$ .

In general, the maximum likelihood estimate is the value of the parameter for which the likelihood of the observed sample is the highest – this will be formalized further on.

### Description of the method

We follow the setting in [11], which is general enough for our purposes.<sup>3</sup> Let  $Z_1, Z_2, \dots$  be a sequence of independent random variables. Let  $Z_k$  have the density  $f_k(\cdot; \theta_0)$  where  $\theta_0$  is the true parameter from the parameter space  $\Theta \subset \mathbb{R}^p$ , where  $p \in \mathbb{Z}_+$ .<sup>4</sup> We wish to estimate  $\theta_0$ . Given a sample  $z_1, \dots, z_n$  from  $Z_1, \dots, Z_n$ , the likelihood function is defined by (for  $\theta \in \Theta$ )

$$\mathcal{L}_n(\theta) := \prod_{i=1}^n f_i(z_i; \theta). \quad (3.1)$$

The maximum likelihood estimator  $\hat{\theta}_n$  is then defined as the maximizer of the likelihood function. For computational reasons, we instead of the likelihood often equivalently maximize the log-likelihood

$$\log \mathcal{L}_n(\theta) = \sum_{i=1}^n \log(f_i(z_i; \theta)). \quad (3.2)$$

Define furthermore the score function (which is actually a random vector):

$$s_n(\theta') := [\nabla_{\theta} \sum_{i=1}^n \log(f_i(Z_i; \theta))]_{\theta=\theta'} \quad (3.3)$$

and the observed information matrix (which is actually a random matrix)

$$\mathcal{J}_n(\theta') := [\nabla_{\theta}^2 \sum_{i=1}^n \log(f_i(Z_i; \theta))]_{\theta=\theta'} \quad (3.4)$$

We will suppress the subscript  $n$  when the size of the respective sample is not relevant to the discussion.

### Theoretical properties and justification

Our choice for the MLE approach was motivated by its desirable asymptotic properties, which hold under suitable assumptions:

1. consistency: i.e. the estimator  $\hat{\theta}_n$  converges to the true parameter value in probability as  $n \rightarrow \infty$ .

---

<sup>3</sup>We simplify it a little bit though, to reflect the fact that our observations are assumed to be independent.

<sup>4</sup>Alternatively,  $f_k$  may represent the probability mass functions if  $Z_k$  are discrete.

2. asymptotic efficiency: within a suitable class of estimators, convergence is asymptotically the “fastest” possible.
3. good behavior under model misspecification: If the true density of  $Z_k$  is instead  $g_k(\cdot) \notin \mathcal{F}_k \equiv \{f_k(\cdot; \theta); \theta \in \Theta\}$ , this is still not a serious problem, assuming that  $g_k$  is “close enough” to  $\mathcal{F}_k$ .

Let us consider the desirable properties in more detail (we follow the exposition in the survey article [11] rather closely).<sup>5</sup> We also give one possible assumption set assuring that the desirable properties hold. However, this is purely for completeness and we do not check the conditions – indeed, some of them are rather formidable even in the case of the normal model, not mentioning the delta-gamma.

**Consistency.** Define the information matrix (with  $s_n$  being considered as a column vector)

$$\mathcal{I}_n(\theta) := \mathbf{E} s_n(\theta) s_n^T(\theta).$$

Loosely following [11], we can formulate the following assumptions:

- (L1) For every  $n$ , the measures associated with the densities  $\{\prod_{i=1}^n f_i(\cdot; \theta); \theta \in \Theta\}$  are mutually absolutely continuous.
- (L2) The log-likelihood  $\log \mathcal{L}(\theta)$  is twice continuously differentiable.
- (L3) The following conditions on the score function hold for any  $\theta \in \Theta$ :

$$\begin{aligned} \mathbf{E} s_n(\theta) &= 0 \\ \mathbf{E} s_n^2(\theta) &< \infty \end{aligned}$$

- (L4) The information matrix  $\mathcal{I}_n(\theta)$  is nonsingular for all  $n$  large enough (independently from  $\theta$ ) and  $\theta \in \Theta \setminus \{\theta_0\}$ .

Now the information matrix is positive definite: indeed, for a  $n \times 1$  vector  $v$ , we have

$$v^T \mathcal{I}_n(\theta) v = \mathbf{E}(v^T \cdot s_n(\theta))(s_n^T(\theta) \cdot v) = \mathbf{E}(s_n^T(\theta) \cdot v)^2 \geq 0, \quad (3.5)$$

so it is positive semidefinite and in view of the condition (L4) it is even positive definite. Hence, it has a symmetric square root, i.e. there exists a symmetric matrix  $(\mathcal{I}_n(\theta))^{1/2}$  with  $(\mathcal{I}_n(\theta))^{1/2}(\mathcal{I}_n(\theta))^{1/2} = \mathcal{I}_n(\theta)$ .<sup>6</sup>

Now define neighborhoods of the true parameter  $\theta_0$  by (for  $\delta > 0$ )

$$N_n(\delta) := \{\theta \in \Theta : \left| \mathcal{I}_n^{1/2}(\theta_0)(\theta - \theta_0) \right| < \delta\}$$

and consider two further assumptions:

<sup>5</sup>We note that a lot of technical complications stem from the fact that  $Z_k$  are not identically distributed.

<sup>6</sup>It is possible to consider nonsymmetric – left and right – pairs of square roots as well, but for simplicity, we do not do this.

(D)  $\lambda_{\min}^{\mathcal{I}_n(\theta_0)} \rightarrow \infty$  as  $n \rightarrow \infty$ , where for a matrix  $A$ ,  $\lambda_{\min}^A$  is its smallest eigenvalue (for a symmetric matrix, this is well defined).

(C) For some  $c > 0$  independent of  $\delta$ , it holds

$$\mathbb{P}\left\{\inf_{\theta \in N_n(\delta)} \lambda_{\min}^{\mathcal{I}_n^{-1/2}(\theta_0)} \mathcal{J}_n(\theta) \mathcal{I}_n^{-1/2}(\theta_0) \geq c\right\} \rightarrow 1$$

Now, under the conditions (L1),..., (L4), (D), (C), the weak consistency holds, that is,  $\hat{\theta}_n$  converge to  $\theta_0$  in probability, i.e. for any  $\epsilon > 0$

$$\lim_{n \rightarrow \infty} \mathbb{P}\left\{\left\|\hat{\theta}_n - \theta_0\right\| \geq \epsilon\right\} = 0.$$

**Asymptotic efficiency.** We consider the class of the so-called *regular estimators*:

**Definition 3.2** (Regular estimator). An estimator  $T_n$  is a *regular estimator* of  $\theta_0$  if: for any  $h \in \mathbb{R}^p$  and associated to it  $\theta_h^{(n)} := \theta_0 + \mathcal{I}_n^{-1/2} h$ , we have, for any continuous bounded function  $b$  (from  $\mathbb{R}^p$  to  $\mathbb{R}$ ):

$$\mathbf{E}_{\mathbb{P}_{\theta_h^{(n)}}} b(\mathcal{I}_n^{1/2}(\theta_0)(T_n - \theta_h^{(n)})) \rightarrow \mathbf{E} b(W(\theta_0)),$$

where  $W(\theta_0)$  is some random vector dependent on  $\theta_0$ , but independent of  $h$ .

These are considered in order to rule out some unintuitive estimators which would pose counterexamples to the following results.

Under additional conditions, it can be shown that the MLE is in some sense ‘‘asymptotically efficient’’ among regular estimators. For such additional conditions, one can take the following ones:

(S) For any  $\delta > 0$ :

$$\sup_{\theta \in N_n(\delta)} \left\|\mathcal{I}_n^{-1/2}(\theta_0) \mathcal{J}_n(\theta) \mathcal{I}_n^{-1/2}(\theta_0)\right\| \rightarrow 0 \quad \text{in probability}$$

(AN)  $\mathcal{I}_n^{-1/2}(\theta_0)s_n(\theta_0) \rightarrow \mathcal{N}(0, I)$  in distribution, where  $I$  is the identity matrix.

Under such conditions (in addition to (L1),..., (L4)), (D), (C)) one gets for instance the following concept of asymptotic efficiency of MLE (see Section 2.4 in [11] for more):

**Proposition 3.3** (Minimal asymptotic variance). *If  $T_n$  is regular and  $W(\theta_0) \sim \mathcal{N}(0, \Sigma)$  for some matrix  $\Sigma$ , then  $\Sigma - I$  is positive semidefinite.*

**Behavior under misspecification.** Assume that the true density for  $Z_k$  is  $g_k(\cdot)$  and it does not lie in  $\mathcal{F}_k \equiv \{f_k(\cdot; \theta); \theta \in \Theta\}$ ; for  $k \in \{1 \dots n\}$ .

Define first the Kullback-Leibler divergence of two distributions  $\mu, \nu$

$$\mathcal{K}(\mu; \nu) := \int \log\left[\frac{d\mu}{d\nu}\right] d\mu$$

where  $\frac{d\mu}{d\nu}$  is the Radon-Nikodym derivative of  $\mu$  w.r.t.  $\nu$  (assuming it exists). The Kullback-Leibler divergence captures a notion of *distance* between two distributions (even though it is not a metric since it is not symmetric). In particular:

- If  $\mathcal{K}(\mu; \nu) = 0$ , then  $\mu$  and  $\nu$  are equivalent in a suitable sense.
- If  $\mathcal{K}(\mu; \nu_k) \rightarrow 0$  for a sequence  $(\nu_k)_{k=1}^{\infty}$ , then the sequence converges to  $\mu$  in the total variation distance.<sup>7</sup>

See for instance Appendix A.2 in [7] for the treatment of the Kullback-Leibler divergence and the mentioned facts in the discrete case.

In case we had  $g_k \in \mathcal{F}_k$ , it would hold

$$\mathcal{K}(\mathbb{P}; \mathbb{P}_{\theta_0}) \equiv \mathbf{E}_{\mathbb{P}} \log \left[ \frac{\prod_{i=1}^n g_i(Z_i)}{\prod_{i=1}^n f_i(Z_i; \theta_0)} \right] = 0 \quad (3.6)$$

where  $\mathbb{P}$  is the probability measure corresponding to the true model. This is unfortunately not the case. However, define  $\theta_0^{(n)}$  to be the minimizer of the left hand side of (3.6). In view of what has been said about the Kullback-Leibler divergence,  $\theta_0^{(n)}$  can be considered the best parameter from  $\Theta$  to approximate the functions  $g_k(\cdot)$ ,  $k = 1, \dots, n$  using just the families  $\mathcal{F}_k$ .

And MLE behaves well, in the sense that the above defined estimator  $\hat{\theta}_n$  approximates  $\theta_0^{(n)}$ . More precisely, we have

$$\hat{\theta}_n - \theta_0^{(n)} \rightarrow 0 \text{ in probability.}$$

## Implementation of MLE

Our problem fits the general framework of MLE outlined above in the following way: Assume that there are destinations  $d_1, \dots, d_m$ . Let  $X_d$  be the random variable giving the water consumption for rotations to  $d$ . Let these have densities  $f(x; \theta_d)$ ,<sup>8</sup> where  $\theta_d$  is the parameter determining the distribution of  $X_d$ . So if for example the assumed model is normal, then  $\theta_d = (\mu_d, \sigma_d)$ . In our case, the role of the random variables  $Z_i$  described in Section 3.1.2 is played by  $Y_{d_i, d_j} = X_{d_i} + X_{d_j}$ , specifically, for every multistretched flight-pair from  $d_i$  to  $d_j$ , we have one independent copy of  $Y_{d_i, d_j}$  – they are indeed independent because of Assumption 2. Recall that we for the time being assume that no observations of non-multistretched rotations are available.

We want to determine the parameter  $\boldsymbol{\theta} := (\theta_{d_1}, \dots, \theta_{d_m})$ . The role of the densities appearing in the likelihood function (3.1) is now played by

$$g_{i,j}(\cdot; \boldsymbol{\theta}) := [f(\cdot; \theta_{d_i}) * f(\cdot; \theta_{d_j})].$$

In the computer program,  $\log \mathcal{L}(\boldsymbol{\theta})$  is determined manually – this allows for a faster code than would result from naively computing (3.2). The log-likelihood is then maximized using the BFGS optimization method. The method requires the gradient of  $\log \mathcal{L}$ ; while this can be computed numerically, for efficiency, it is better to supply an analytical expression – this has been done for the normal model. If more complicated models are to be assumed, it might be necessary to use linearly constrained optimization.

<sup>7</sup>I.e. we have  $\sup_{S \in \mathcal{F}} |\nu_k(A) - \mu(A)| \rightarrow 0$  as  $k \rightarrow \infty$  where  $\mathcal{F}$  is the respective  $\sigma$ -algebra of sets.

<sup>8</sup>The role of this  $f$  does not correspond to the role of  $f$  in the preceding discussion.

**Interlude: BFGS.** The problem on hand is

$$\begin{aligned} \min_x \quad & f(x) \\ & x \in \Theta, \end{aligned} \tag{3.7}$$

with  $f = -\log \mathcal{L}$  in our case.<sup>9</sup> Assume for the time being that  $\Theta = \mathbb{R}^m$  where  $m$  is the dimension of the parameter. In the following exposition, we draw from [9, Chapter 4]. All superscripts in this paragraph denote the iteration number.

The methods considered to solve (3.7) are iterative: one generates a sequence of points  $(x^k)$  and the algorithm stops once  $|f(x^{k+1}) - f(x^k)|$  is small in comparison to to the initial value. The last point in the sequence is then proclaimed as (an approximation of) the maximum. As will be argued later, the function  $\log \mathcal{L}$  is in our case twice differentiable, so we choose a method which can make use of this fact.

A classical method is the Newton's method, in the  $k+1$  iteration, the new point is defined by

$$x^{k+1} := x^k - (\nabla^2 f(x^k))^{-1} \nabla f(x^k)$$

where  $\nabla f$  denotes the gradient and  $\nabla^2 f$  the Hessian. The problems with the Newton's method are:

1. Inverting the Hessian matrix is computationally very expensive.
2. There might be stability problems if the Hessian matrix is not positive definite or it is ill-conditioned. Then additional measures must be taken. Since we will sometimes differentiate  $f$  numerically, this point is of even greater relevance.

Because of these drawbacks, we instead choose from the class of *quasi-Newton methods*. Instead of using the Hessian, these methods approximate it with a matrix  $H_k$ , this is updated at each iteration with a linear update  $D_k$ :

$$H_{k+1} := H_k + D_k.$$

A prototypical quasi-Newton method works as follows:

**Step 0** Let  $x_0$  be given and  $H_0$  the identity matrix

**Step  $k$**

- Calculate the search direction  $s^k := -H_k \nabla f(x^k)$
- Let  $x^{k+1} := \arg \min_{\lambda \geq 0} f(x^k + \lambda s^k)$
- Update  $H_{k+1} := H_k + D_k$

with  $D_k$  having some special properties. Now, the different quasi-Newton methods differ in the choice of the update  $D_k$ . We choose BFGS since it is often recommended (for instance, according to [9]: "(BFGS) is currently the Quasi-Newton method of choice."). Define

$$\sigma^k := x^{k+1} - x^k, \quad y^k := \nabla f(x^{k+1}) - \nabla f(x^k), \quad \tau_k := 1 + \frac{y^{kT} H_k y^k}{\sigma^{kT} y^k}.$$

---

<sup>9</sup>Writing the problem as a minimization problem is just a matter of convention. Indeed,  $\max \log \mathcal{L} = \min(-\log \mathcal{L})$ .

The BFGS update matrix is then defined by

$$D_k := \frac{\tau_k \sigma^k \sigma^{kT} - \sigma^k y^{kT} H_k - H_k y^k \sigma^{kT}}{\sigma^{kT} y^k}.$$

At the end, we mention a little bit about handling constraints. In our case, the constraints are that some parameters must be larger than 0 and some smaller than 1. BFGS is by default a method for unconstrained optimization – it is assumed that the set  $\Theta$  from (3.7) is indeed equal to  $\mathbb{R}^m$  – so we have to make an adjustment if  $\Theta$  is a constrained set.

We use the logarithmic barrier method. Following [18, Section 16.3], we rewrite the constraints into the form  $v_i(x) \geq 0$  for  $i = 1, \dots, q$ . The logarithmic barrier function is then defined by

$$b(x) := \sum_{i=1}^q \log v_i(x)$$

and instead of the function  $f$ , we try to minimize the functions  $h_k := f + \mu_k b$  for a decreasing sequence of “tuning constants”  $\mu_k \searrow 0$ . Denote the corresponding minimizers by  $x_k^*$ . Then under mild assumptions, it holds that any cluster point of  $\{x_k^*\}$  is a minimizer of  $f$  on the closure of  $\Theta$ , i.e. the minimizer that we are looking for.

The barrier method is implemented in the R routine `constrOptim`.

**Maximizing log-likelihood.** A possible problem in maximizing (3.2) is that the maximum found may be only a local maximum – the resulting estimate is then not a true maximum likelihood estimate and we can expect that it will be inferior to it.

To get an insight whether this might be a concern, we ran (for E90, winter 2010, with the normal model as discussed in Section 3.1.4) the estimation procedure twenty times with perturbed initial parameters – in contrast to the standard initial parameters are mean equal to 20 and standard deviation equal to 10 for all destinations which choice is based on the exploratory data analysis on a subset of data.

Ten times, the standard parameters were perturbed very heavily, by a vector of normal distributed random variables with zero mean and large standard deviations. In this case seven times, the same maximum was reached as with standard parameters, three times, a smaller maximum was reached. Afterwards, the standard parameters were perturbed ten times moderately – the perturbing normal vector had lower standard deviations. In this case, all optimization results ended up the same, with the same value as when starting with the standard parameters.

We conclude that it is very likely that the global maximum was found in most cases and hence, for the normal model, there are no problems. A similar test for the delta-gamma distribution has not been carried out because of considerably higher computational requirements.

**Correction for rounding.** As already noted, the tanking data is apparently rounded, i.e. we do not observe the true tanked amount, but only its approximation. Hence we do not observe the random variables  $Y_{i,j}$  directly, but instead random variables  $\rho \circ Y_{i,j}$  where  $\rho$  is a function which represents rounding. The naïve “maximum likelihood estimator” obtained by maximizing the log-likelihood associated to  $Y_{i,j}$  (i.e. the one from (3.1)) will then probably be inferior to the true maximum likelihood estimator (i.e. the one corresponding to  $\rho \circ Y_{i,j}$ ).

There are two sources of rounding in our case

1. The water meter has only finite precision, namely one decimal place, and operators are required to round to integers.
2. Some operators round on their own discretion even more, i.e. to multiples of five or ten.

We have less information about the second source of rounding and neither do we know how many operators round in such a fashion – hence the modeling will be more complicated and will require a stochastic element, i.e. the function  $\rho$  will itself be random. Therefore, we postpone the discussion of this type of rounding until Section 4.4 and ignore its effect for now.

On the other hand, the first effect can be modeled more easily (see also [17], in particular Section 5). Assume that for every integer  $k$ , there is an interval  $I_k$  such that when we tank the amount  $x \in I_k$ , we observe  $k$  instead. The true log-likelihood function corresponding to random variables  $\rho \circ Y_{i,j}$  is then given by

$$\log \mathcal{L}(\theta) = \sum_{t=1}^n \log \left[ \int_{I_{y_t}} f(y, \theta) dy \right].$$

where every observation  $y_t$  is an integer. In case of “standard” rounding to the nearest integer, we would have  $I_k = [k - 0.5, k + 0.5)$  for  $k > 0$ .<sup>10</sup> This kind of correction is implemented for the normal model in Section 3.1.4. That said, given the overall noisiness of our data, such a correction is unlikely to make a large difference.

Furthermore, the tricky part is that the operators do not always round to the nearest integer – for instance the one who the author talked with actually rounds always downwards. So the correction mentioned above may actually have no positive effect at all! In the delta-gamma, the correction is implemented as well. In that case, the reason was not to correct for rounding though – instead, the “correction” served to convert continuous data into discrete data. Two types of rounding corrections are used: to the nearest integer (i.e.  $I_k = [k - 0.5, k + 0.5)$ ) and rounding upwards ( $I_k = (k - 1, k]$ ). Testing estimation under both corrections, differences were negligible; we have therefore chosen for the upward correction since it simplifies computations and hence presumably contributes to a slightly faster estimation procedure.

### 3.1.3 Generalized method of moments

It was suggested [4] that a less complicated (both conceptually and computationally) method than MLE could be used to solve the problem of estimation under multistretching. Perhaps it would be enough to match the sample moments to the moments of the assumed distribution. This would allow more freedom in the choice of distribution – indeed, for MLE, we need that the sum of two independent random variables sampled from the distribution has a tractable distribution; on the other hand moments of a sum of two independent random variables can be expressed as a polynomial in the moments of these random variables. Furthermore, the optimization could be presumably much easier. Perhaps we could even end up with just a (possibly overdetermined) system of linear equations for our estimate and so the use of routines for multidimensional nonlinear optimization, such as BFGS, would be avoided altogether.

These considerations led us to try out a variation on the Generalized method of moments (GMM). In the rest of the thesis, this estimation procedure will be referred to as GMM, but note that it is somewhat different from the GMM known in literature.

---

<sup>10</sup>And  $I_0 = [0, 0.5)$ .

## Description of the method

**Classical GMM.** Consider first the standard generalized method of moments (our exposition combines the descriptions given in [15] and [14]). It is assumed that  $\{Z_k\}_{k=1}^n$  is a stationary time series. In particular we will assume that it is an iid sequence; it could be multidimensional but we will not need this. The generating distribution is dependent on the parameter vector  $\theta$  from a parameter space  $\Theta$  (a subset of the Euclidean space) which is to be estimated. Let  $\theta_0$  be the true parameter value. Now consider a vector function  $f(z, \theta)$  (continuous in the second argument) taking values in  $\mathbb{R}^r$ , each of the  $r$  components of the function corresponds to one “moment condition”  $M$  satisfying

$$M(\theta_0) := \mathbf{E} f(Z_1, \theta_0) = 0. \quad (3.8)$$

Now consider the above quantity where we take expectation w.r.t. the empirical distribution function (given a sample  $S = \{z_1, \dots, z_n\}$ ):

$$M^S(\theta) := \mathbf{E}_S f(Z_1, \theta) \equiv \frac{1}{n} \sum_{k=1}^n f(z_k, \theta).$$

Choose  $A$  to be a positive definite  $s \times s$  matrix – such a matrix induces a norm on  $\mathbb{R}^s$ .<sup>11</sup> Then the GMM estimator  $\hat{\theta}$  of  $\theta$  is obtained as

$$\hat{\theta} := \arg \min_{\theta} M^S(\theta) A (M^S(\theta))^T, \quad (3.9)$$

i.e. we choose the estimate of  $\theta$ , so that the  $A$ -norm of the vector of (empirical) moment conditions is minimized. This idea is intuitively supported by (3.8), which indicates that a good estimator of  $\theta$  should get the components of the vector of moment conditions close to zero.

The role of the matrix  $A$  is then to emphasize the importance of some components and put less weight on other components – the optimal matrix  $A$  can be then taken (or estimated) in such a way that the variance of the estimator is minimized. That said, we will for simplicity only consider  $A$  being an identity matrix. Under the above conditions, the GMM estimator is consistent. Furthermore, there exists an optimal matrix  $A$ , even though it cannot be determined apriori and must be estimated from the data. However, we will only use  $A$  being the identity matrix.

We will not provide the conditions on the optimal matrix  $A$ . However, even with a suboptimal matrix, the estimator may be (strongly) consistent, i.e.  $\hat{\theta}_n \rightarrow \theta_0$  almost surely, if the subscript  $n$  denotes the size of the sample  $S_n$  (assuming that  $S_1 \subset S_2 \subset \dots$ ). The conditions for consistency are (as adapted from the assumptions in [15]):

(GMM.1)  $\mathbf{E} f(Z_1, \theta)$  is finite for all  $\theta \in \Theta$ .

(GMM.2)  $f(\cdot, \theta)$  is Borel measurable for all  $\theta \in \Theta$  and  $f(x, \cdot)$  is continuous on  $\Theta$  for all  $x$ .

(GMM.3)  $A_n \rightarrow A_0$  almost surely for some constant matrix  $A_0$ : Here  $A_n$  plays the role of  $A$  for the sample  $S_n$ ; it is assumed that the matrix  $A$  can be a function of the sample hence each  $A_n$  is itself a random matrix.

---

<sup>11</sup>Namely, for  $v \in \mathbb{R}^s$ , its  $A$ -norm is given by  $\|v\|_A \equiv \sqrt{v A v^T}$ .

(GMM.4) For all  $\theta \in \Theta$ , we have

$$\lim_{\delta \searrow 0} \mathbf{E} \left[ \sup_{\substack{\theta' \in \Theta \\ \|\theta' - \theta\| < \delta}} (|f(Z_1, \theta) - f(Z_1, \theta')|) \right] = 0$$

(GMM.5)  $\Theta$  is compact

(GMM.6)  $M(\theta)A_0(M(\theta))^T = 0$  if and only if  $\theta = \theta_0$

Now, under the stated conditions, the strong consistency of the GMM estimator holds.

**Our approach.** Note that our problem could be recast in the above framework with (a random vector)  $Z_k = (Y_{i,j}^{(k)})_{1 \leq i \leq j \leq |\text{DST}|}$  in case that for all  $Y_{i,j}$ , we have the same number of observations. However, this is by far not the case.

Hence, how to tackle this case, with variables being independent, but not identically distributed? Separate moment conditions are considered for each  $Y_{i,j}$ :

$$M_{i,j}^S(\theta) := \frac{1}{n_{i,j}} \sum_{k=1}^{n_{i,j}} f_{i,j}(y_{i,j}^{(k)}, \theta),$$

assuming samples  $S_{i,j} := \{y_{i,j}^{(1)}, \dots, y_{i,j}^{(n_{i,j})}\}$  for every  $1 \leq i \leq j \leq |\text{DST}|$ . Then taking  $M^S(\theta)$  a vector of all these conditions and  $A$  an identity matrix of the correct dimensions, we again acquire  $\hat{\theta}$  from the equation (3.9).

This was tested for the delta-gamma – where the density of the sum of two random variables is cannot be computed exactly, so this could be an improvement – and normal models. The functions are given under respective distributions in Section 3.1.4.

## Problems

The method has been eventually abandoned. The reasons for this are twofold:

1. Too imprecise estimates found: With the amount of data that we have, the estimate was quite good (in the normal case) for most destinations, but unacceptable for several of them. For example in the normal case, for three destinations with enough data, estimated mean was less than 10 liters, while it should be over 20. The method was also tested for the delta-gamma model, but there the method has not converged to a reasonable solution within a reasonable amount of time.
2. Computational complexity: It turns out that to compute the parameter estimates, one has to resort to nonlinear optimization even in this case (see discussion in Section 3.1.4). Furthermore, proceeding to the optimization, it seemed that in this case, the time requirements were even higher than in the case of MLE.

In view of the above problems, there was almost no reason to prefer this method to the already implemented MLE, perhaps with the exception of more freedom in the choice of distribution.

### 3.1.4 Parametric modeling

Let us consider the requirements that the parametric model for  $X_d$  should satisfy, roughly in order of their importance, taking into account that our primary method is MLE.

1. The distribution should fit the data reasonably. See Section 1.4.1 for a detailed discussion how the distribution looks like.
2. Convolution of two such parametric distributions (corresponding to the distribution of the sum  $X_{d_1} + X_{d_2}$ ) must have a (known) analytical expression, so that it is possible to express the log-likelihood function (3.2) and proceed with MLE.
3. The distribution should work for all destinations. If there were more types of distributions, it would be harder to separate these on the multistretched data.
4. Less parameters is better – with too many parameters, the MLE will be computationally infeasible.
5. Preferably, the density (respectively the probability mass function) for the distribution  $X_{d_1} + X_{d_2}$  has an analytical expression for gradient (w.r.t. the parameters). Then the gradient can be supplied to BFGS thus speeding up the estimation and possibly making it more precise.

Below, the models are listed among which the delta-gamma and the normal model in detail since these are deemed suitable for use in practice.

**Delta-gamma mixture.** By the delta-gamma model, we mean a mixture of the point mass at zero (sometimes denoted  $\delta_0$ ) and the gamma distribution.

For parameters, we denote by  $p$  the mixing parameter denoting the “weight” of  $\delta_0$ , by  $k$  the shape parameter of gamma and by  $\alpha$  the rate parameter of gamma. So the density corresponding (only) to the gamma distribution is

$$x \mapsto \frac{\alpha^k x^{k-1}}{\Gamma(k)} e^{-\alpha x}.$$

We have one set of parameters for every considered destination.

This model was chosen because it is bimodal and positive; the simpler normal model lacks these properties.

The model has also “roughly exponential” upper tails (see Section 1.4.1 for this observation about the data). To make “roughly exponential” more precise, note that the upper tail values correspond to the behavior of the gamma distribution. But for the gamma distribution, we have

**Proposition 3.4.** *For  $Z \sim \Gamma(k, \alpha)$ , it holds*

$$\lim_{z \rightarrow \infty} \frac{\mathbb{P}\{Z > z\}}{(\alpha z)^{k-1} e^{-\alpha z} / \Gamma(k)} = 1. \quad (3.10)$$

*Proof.* By equation (8.11.2) in [1], we have (for fixed  $n$ )

$$\Gamma(a, z) = z^{a-1} e^{-z} \left( \sum_{j=0}^n \frac{u_j}{z^j} + O(z^{-n}) \right), \quad z \rightarrow \infty,$$

where  $u_k$  depends on  $a$ , but not on  $z$ , and  $\Gamma(a, z)$  is the upper incomplete gamma function. Using the gamma density

$$\mathbb{P}\{Z > z\} = \int_z^\infty \frac{\alpha^k x^{k-1}}{\Gamma(k)} e^{-\alpha x} dx = \frac{1}{\Gamma(k)} \int_{\alpha z}^\infty y^{k-1} e^{-y} dy = \frac{\Gamma(k, \alpha z)}{\Gamma(k)}.$$

Combining the above two equations yields (3.10).  $\square$

We note in passing that, contrastingly, a limit analogous to (3.10) is equal to zero if  $Z$  is normally distributed.

Consider the random variable  $Y_{i,j} = X_i + X_j$  under the delta-gamma assumption on  $X_i$  and  $X_j$ . Then the distribution of  $Y_{i,j}$  is a mixture of four distributions: a  $\delta_0$  distribution, two gamma distributions and a distribution of the sum of two gamma random variables. Since we cannot express a likelihood function if we have to do with a mix of a discrete and continuous distribution, to compute the likelihood we instead consider the rounded data, with rounding going upwards, as described at the end of Section 3.1.2. Hence let  $\tilde{Y}_{i,j}$  denote the rounded version of  $Y_{i,j}$ .

Its probability mass function  $g_{i,j} \equiv g(m; p_i, p_j, k_i, k_j, \alpha_i, \alpha_j)$  is then

$$g_{i,j}(m) = \begin{cases} \begin{aligned} & p_i(1-p_j)(F_\Gamma(m, k_j, \alpha_j) - F_\Gamma(m-1, k_j, \alpha_j)) \\ & + p_j(1-p_i)(F_\Gamma(m, k_i, \alpha_i) - F_\Gamma(m-1, k_i, \alpha_i)) \\ & + (1-p_1)(1-p_2)(F_{\Gamma^{(*2)}}(m, k_i, \alpha_i, k_j, \alpha_j) \\ & - F_{\Gamma^{(*2)}}(m-1, k_i, \alpha_i, k_j, \alpha_j)) \end{aligned} & \text{if } m > 0 \\ p_i p_j & \text{if } m = 0 \\ 0 & \text{if } m < 0 \end{cases} \quad (3.11)$$

where  $F_\Gamma$  is the cumulative distribution function for Gamma and  $F_{\Gamma^{(*2)}}$  the cumulative distribution function for the sum of two Gamma random variables.

Unfortunately, no simple closed expression is known for  $F_{\Gamma^{(*2)}}$ . However, in [24, Section 3], known approximations and expansions are reviewed for distribution of sums of gamma random variables<sup>12</sup> – so in particular results can be applied for  $F_{\Gamma^{(*2)}}$ . If we denote by  $f_{\Gamma^{(*2)}}$  the respective density, then the results presented in [24, Section 3] include the following two

$$\begin{aligned} f_{\Gamma^{(*2)}}(z) &= \frac{1_{\{z>0\}}}{\pi} \int_0^\infty \frac{\cos(k_i \arctan(t/\alpha_i) + k_j \arctan(t/\alpha_j) - zt)}{(1 + \frac{t^2}{\alpha_i^2})^{k_i/2} (1 + \frac{t^2}{\alpha_j^2})^{k_j/2}} dt, \\ f_{\Gamma^{(*2)}}(z) &= \frac{\alpha_i^{k_i} \alpha_j^{k_j} z^{k_i+k_j-1}}{\Gamma(k_i + k_j)} \Phi_2^2(k_i, k_j; k_i + k_j; -z\alpha_i, -z\alpha_j), \end{aligned} \quad (3.12)$$

where  $\Phi_2^2$  is the confluent Lauricella hypergeometric function.<sup>13</sup> Nevertheless, we have chosen to use the following result in [24] (originally presented in [23]) because of its relative

<sup>12</sup>The whole article includes approximations for some other random variables as well.

<sup>13</sup>Defined by  $\Phi_2^2(\alpha, \alpha'; \beta; x, y) := \sum_{m,n=1}^\infty \frac{(\alpha)_m (\alpha')_n}{(\beta)_{m+n} m! n!} x^m y^n$ , where  $(z)_k := \prod_{i=0}^{k-1} (z-i)$ .

computational amenability.<sup>14</sup> Assume w.l.o.g. that  $\alpha_i \geq \alpha_j$ . Then define recursively

$$\begin{aligned}\delta_0 &:= 1 \\ \delta_{n+1} &:= \frac{k_j}{n+1} \sum_{i=1}^{n+1} \left(1 - \frac{\alpha_j}{\alpha_i}\right) \delta_{n+1-i}.\end{aligned}$$

Then the cumulative distribution function can be written as an infinite sum

$$F_{\Gamma^{(*2)}}(z) = \left(\frac{\alpha_j}{\alpha_i}\right)^{k_j} \sum_{n=0}^{\infty} \delta_n F_{\Gamma}(z, k_i + k_j + n, \alpha_i). \quad (3.13)$$

To use this in practice, one has to choose a suitable truncation bound  $N$  for  $n$  (and then the sequence of  $\delta_n$  is to be computed only for  $n \leq N$ ). How to choose  $N$ ? There is an upper bound for the truncation error in [23] – if  $F_{\Gamma^{(*2)}}^{(N)}$  is the expression obtained by taking in (3.13) only the first  $N + 1$  terms, we have that the error  $E_N(z)$  is bounded as

$$E_N(z) \leq \frac{\alpha_i^{k_i} \alpha_j^{k_j}}{\Gamma(k_i + k_j)} \int_0^z y^{k_i+k_j-1} e^{-y\alpha_j} dy - F_{\Gamma^{(*2)}}^{(N)}(z).$$

Unfortunately, computing this bound for the values of parameters that we use ( $k_i, k_j$  around 4 and  $\alpha_i, \alpha_j$  around 1.5), we did not get tight enough results; even worse, the bound does not converge to zero as  $N \rightarrow \infty$ . We resorted therefore to numerical experimentation and based on the results, chose  $N = 10$ .<sup>15</sup> Truncating (3.13) at  $N = 10$  then enables us to (approximately) compute (3.11).

Now, the log-likelihood is obtained by summing terms in (3.11) with different fixed values of  $m$ . Since  $F_{\Gamma}$  is a quotient of the lower incomplete gamma function and the standard gamma function, it is infinitely differentiable in parameters as long as they are strictly positive (this follows from the similar property of the lower incomplete gamma function and the standard gamma function – see for instance [2]). In view of (3.13), also  $F_{\Gamma^{(*2)}}$  is infinitely differentiable in parameters whenever they are positive. We conclude from (3.11) that the corresponding log-likelihood is infinitely differentiable for relevant values of parameters and hence the maximization using BFGS is justified.

It is noted that the optimization of the log-likelihood is not done using the standard unconstrained BFGS, but some linear constraints must be imposed. Namely that all parameters are larger than 0 and that  $p_i$  are smaller than 1.

**Normal.** The advantage of the normal model is its simplicity, while it also works good enough for our purposes as will be demonstrated in Section 3.1.5. The normal density (the one pertaining to the  $\mathcal{N}(\mu, \sigma^2)$  distribution) is

$$f(x; \mu, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left[-\frac{(x - \mu)^2}{2\sigma^2}\right].$$

If we assume that  $X_i \sim \mathcal{N}(\mu_i, \sigma_i^2)$ , then  $Y_{i,j} = X_i + X_j$  has  $\mathcal{N}(\mu_i + \mu_j, \sigma_i^2 + \sigma_j^2)$  distribution by the properties of the normal distribution. Also the log-likelihood takes a particularly simple

<sup>14</sup>Contrastingly, one of the results in (3.12) includes a special function which does not seem readily computable without computing a doubly infinite sum; while the other result requires additional integration.

<sup>15</sup>A slightly lower number would likely also work, but this is to be sure.

form: assuming that the observations are indexed by  $t \in T$ ,

$$\log \mathcal{L}(\boldsymbol{\mu}, \boldsymbol{\sigma}) = -\frac{1}{2} \sum_{t \in T} \log(2\pi) - \frac{1}{2} \sum_{t \in T} \log(\sigma_{i_t}^2 + \sigma_{j_t}^2) - \frac{1}{2} \sum_{t \in T} \frac{(y_t - \mu_{i_t} - \mu_{j_t})^2}{(\sigma_{i_t}^2 + \sigma_{j_t}^2)}, \quad (3.14)$$

where  $i_t = i$  and  $j_t = j$  if  $y_t$  is drawn from  $Y_{i,j}$ . We note that this log-likelihood is clearly twice continuously differentiable (in  $\mu_i, \sigma_i$ ) and hence maximizing it using the BFGS method is justified.

The BFGS requires the gradient, to avoid computing it numerically, the log-likelihood can easily be differentiated:

$$\begin{aligned} \frac{\partial}{\partial \mu_k} \log \mathcal{L}(\boldsymbol{\mu}, \boldsymbol{\sigma}) &= \sum_{t \in T_{k,\cdot}} \frac{(x_t - \mu_k - \mu_{j_t})}{(\sigma_k^2 + \sigma_{j_t}^2)} + \sum_{t \in T_{\cdot,k}} \frac{(x_t - \mu_{i_t} - \mu_k)}{(\sigma_{i_t}^2 + \sigma_k^2)} \\ \frac{\partial}{\partial \sigma_k} \log \mathcal{L}(\boldsymbol{\mu}, \boldsymbol{\sigma}) &= -\sigma_k \left( \sum_{t \in T_{k,\cdot}} \frac{1}{\sigma_k^2 + \sigma_{j_t}^2} + \sum_{t \in T_{\cdot,k}} \frac{1}{\sigma_{i_t}^2 + \sigma_k^2} \right) \\ &\quad + \sigma_k \left( \sum_{t \in T_{k,\cdot}} \frac{(x_t - \mu_k - \mu_{j_t})^2}{(\sigma_k^2 + \sigma_{j_t}^2)^2} + \sum_{t \in T_{\cdot,k}} \frac{(x_t - \mu_{i_t} - \mu_k)^2}{(\sigma_{i_t}^2 + \sigma_k^2)^2} \right). \end{aligned}$$

Here,  $T_{k,\cdot}$  is the set of  $t$  such that  $i_t = k$  and  $T_{\cdot,k}$  is the set of  $t$  with  $j_t = k$ . As a matter of interest, if we supply to the BFGS method the gradient using the exact formulas above, the maximization of the log-likelihood takes several minutes. If we instead provide a gradient obtained by the numerical differentiation of (3.14), the maximization takes several hours.

What has been just described is the log-likelihood function that is now by default used in the program. It is also possible to incorporate the correction for rounding (see the last part in Section 3.1.2), even though the simple form of the log-likelihood from (3.14) is then lost. We consider rounding with  $I_k = [k - 0.5, k + 0.5)$ . Then if we observe  $\tilde{Y}_{i,j} = y \in \mathbb{Z}$  where  $\tilde{Y}_{i,j} = \rho \circ Y_{i,j}$ , the correct term in the log-likelihood function to account for it is

$$\begin{aligned} \log \mathbb{P}_{\theta_i, \theta_j} \{\tilde{Y}_{i,j} = y\} &= \log \mathbb{P}_{\theta_i, \theta_j} \{y - 0.5 \leq Y_{i,j} < y + 0.5\} \\ &= \log \left[ \Phi \left( \frac{y + 0.5 - (\mu_i + \mu_j)}{\sqrt{\sigma_i^2 + \sigma_j^2}} \right) - \Phi \left( \frac{y - 0.5 - (\mu_i + \mu_j)}{\sqrt{\sigma_i^2 + \sigma_j^2}} \right) \right], \end{aligned}$$

where in the second equality we recall that  $Y_{i,j}$  is  $\mathcal{N}(\mu_i + \mu_j, \sigma_i^2 + \sigma_j^2)$  distributed. The full log-likelihood function is then obtained by summing over all these observations.

**Other tested distributions** As a proof of concept of the whole MLE approach, we first tried out fitting only gamma distribution. Due to the above problems with a sum of two gamma distributed r.v., the rate parameter was held fixed with a value which lied in the region of interest. In that case we have, for  $X_i \sim \Gamma(k_i, \alpha)$ ,  $X_j \sim \Gamma(k_j, \alpha)$ , that  $X_i + X_j \sim \Gamma(k_i + k_j, \alpha)$ , so the implementation of this model was easy. However, due to neglecting the rate parameter, the model was not really a good fit.

Furthermore, before coming with the delta-gamma, other models to capture the zero observations had been tried. However, this was done from a “continuous context” – mixing exponential and normal distribution or two normal distributions. This was not really a success because from the point of view of the data, there really is a point mass at the zero – so when

optimizing, in case of the mixture of exponential and normal distribution, the rate parameter went to infinity while when working with the mix of normal distributions, with one centered at the origin (or at 0.5), the standard deviation of it went to zero.

### 3.1.5 Comparison of models

#### Suitability as input for the evaluation procedure

To compare the distribution models – in fact, only the normal and the delta-gamma – the evaluation procedure was run using the numeric shortage vector estimated<sup>16</sup> using the models in question. We tested this for a period where we have non-multistretched data available. The computed solutions were then compared to the solutions found using the boolean shortage vector computed from the true tanking data. The solutions using the estimated shortage vector turned out to be comparably good as those using the true shortage vector. We conclude that both models are suitable for our purpose.

*Remark.* One might wish to have a more exact measure of whether a proposed distribution model serves well as input for the evaluation procedure; in particular, this would be handy if there were even more available models. We suggest a following procedure:

The goal of serving well as input could be formulated as minimizing a loss function (over all available distribution models  $m$ ; with a suitable function  $F$ )

$$L(m) = F(\phi(s_m^*; D_{\text{new}}), \sigma(s_m^*; D_{\text{new}})) \quad (3.15)$$

where  $s_m^*$  is the best state found when solving the problem (2.1) with data  $D$  estimated using the distribution model  $m$ . This is also dependent on the algorithm  $a$  which we use; furthermore, since the algorithm is partially randomized, it is actually a random variable – but we ignore these issues.

As already discussed in Section 2.4.1, there are two possibilities how the strategy  $s_m^*$  might fare poorly on  $D_{\text{new}}$ :

- The number of multistretches is too low.
- The shortage rates for some destinations are too high.

We suggest the function  $F$  to reflect these problems:

$$F(\phi(s; D), \sigma(s; D)) := -\phi(s; D) + K \cdot \sum_{d \in \text{DST}} ([\sigma(d, s; D) - \alpha]^+)^2$$

for a suitable constant  $K$ . We count the excessive shortage rates quadratically since a shortage slightly higher than  $\alpha$  is almost negligible while a shortage much higher than  $\alpha$  would be a serious problem.

Now, among several competing models, one can choose the one minimizing the function  $L(m)$  in (3.15). This approach is yet to be tested with the real data.

---

<sup>16</sup>For estimation, the data was presented as if it was multistretched.

## Goodness of fit

While the main requirement for proposed distributions was that they were suitable as input, we have still used also a quantitative measure to compare the fit of the distribution models. The parameters were first estimated using the MLE procedure (on non-multistretched data). We have used a goodness-of-fit distance based on edf statistics – see the beginning of Section 2.1.4 for an introductory discussion. The fitted distributions were compared to the original sample using a goodness-of-fit distance. For the ease of implementation, we have restricted ourselves to the Cramér-von Mises distance, which can be obtained from the R package `distrEx`. Let  $\hat{F}_n$  be the edf of the sample and  $G$  the distribution function of our estimated distribution – for example normal or delta-gamma with the estimated values of parameters. The (square of the) Cramér-von Mises distance is then defined as

$$W^2(\hat{F}_n, G) := n \int_{-\infty}^{\infty} (\hat{F}_n(x) - G(x))^2 dG(x).$$

Delta-gamma then turns out to be better than the normal model for most destinations. However it is hard to say how much better since the procedure returns just one number for the delta-gamma distribution and one for the normal distribution. In comparison with the initial guesses, the differences between the two estimates are rather small.

Another possibility to compare models and judging whether a model fits the data reasonably is to just plot densities corresponding to the model estimates. An example of this is Figure 3.2. There, estimates were done on non-multistretched data. For a comparison, the density estimated by a kernel density estimator<sup>17</sup> is also added (the red line).

The figure indicates that both normal and delta-gamma models fit the data reasonably well. The delta-gamma fit is very close to the estimate found by the nonparametrical method which suggests that the delta-gamma distribution might be very close to the true distribution, at least for some destinations.

### 3.1.6 Combining multistretched and non-multistretched rotations

It is likely that some data for a destination is multistretched and some is non-multistretched. In fact, if we decide not to multistretch a destination – respectively we do not multistretch any city-pair containing it –, then most data will be non-multistretched; but for the multistretched destinations, the percentage of multistretched data may be anything – see also examples in 2.1.3.

A natural way (if we neglect rounded data and other peculiarities and assume that MLE would be optimal for individual problems) to combine estimation on both types of data would be to formulate one log-likelihood comprising both multistretched and non-multistretched data. Let

- $f_i(\cdot; \theta)$ ... the density for  $X_i$
- $g_{i,j}(\cdot; \theta)$ ... the density for  $X_i + X_j$
- observed tanked amounts for nonmultistretched rotations  $x_i^{(1)}, \dots, x_i^{(n_i)}$  for all destinations  $i$

---

<sup>17</sup>Using the function `density` in R, with default parameters except for the bandwidth which was computed using the option `bw = "bcv"` – this corresponds to the gaussian kernel and the bandwidth for it determined by crossvalidation.

## NUE

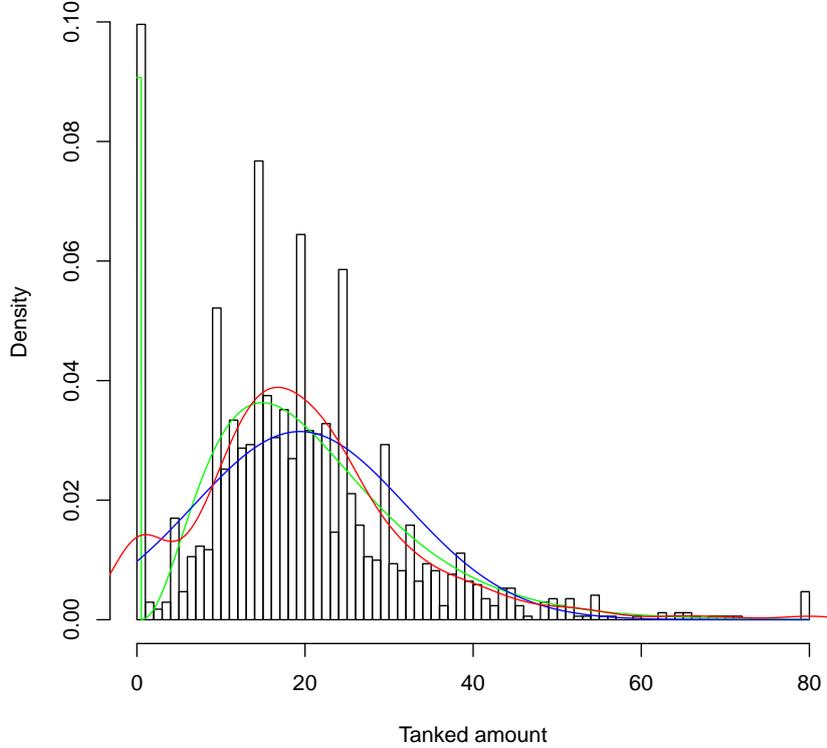


Figure 3.2: Histogram of tanked amounts for Nuremberg, F70, all available data – corresponds to 1707 observations

Green line corresponds to the estimated delta-gamma density, blue line to the normal density, red line to the (nonparametric) kernel density estimate

- observed tanked amounts for multistretched rotation pairs  $y_{i,j}^{(1)}, \dots, y_{i,j}^{(n_{i,j})}$  for all pairs of destinations  $i, j$  (hence we come from  $i$  and go to  $j$  and after  $j$  we observe the sum of consumptions for both rotations)

Assume that destinations are numbered from 1 to  $m$  and identify a destination with its number. Then the log-likelihood from (3.2) would be the sum of the log-likelihoods for multistretched and for nonmultistretched data:

$$\log \mathcal{L}(\theta) = \sum_{1 \leq i \leq j \leq m} \sum_{k=1}^{n_{i,j}} \log g_{i,j}(y_{i,j}^{(k)}; \theta) + \sum_{i=1}^m \sum_{k=1}^{n_i} \log f_i(x_i^{(k)}; \theta)$$

This approach has been tried in some cases – its drawback is that the resulting log-likelihood is much more cumbersome to work with and this has also as a result that the resulting estimation tends to take longer.

Since several different distributions have been tested, a different, less complicated approach was chosen instead: We acquired one estimate separately from each type of data. Recall from Section 3.1.1 that estimating parameters separately from non-multistretched data is very easy

while for multistretched data this has been done in the preceding sections. The final estimate is then the weighted average of these partial estimates, up to some exceptions which reflect our suspicions about multistretched data if there is only little of it.

More precisely, let  $\hat{\theta}^{\text{MS}}$  be the estimate obtained from multistretched data and  $\hat{\theta}^{\text{noMS}}$  the one from nonmultistretched data. Let the subscript  $d$  denote the component for the destination  $d$ . Let  $w_d^{\text{noMS}}$  the proportion of nonmultistretched rotations for  $d$  and  $w_d^{\text{MS}}$  the proportion of rotations within a rotation pair for  $d$ . Note that one rotation pair contains two rotations – only the second one is multistretched, but for the first one, we have no individual tanking observation either. The final estimate  $\hat{\theta}$  is obtained as follows:

- If there are very few multistretched rotations for  $d$  (by default 5), then take  $\hat{\theta}_d := \hat{\theta}_d^{\text{noMS}}$ .
- If there are few (say  $k_d$ ) multistretched rotations for  $d$  (by default between 5 and  $K := 50$ ), then redefine<sup>18</sup>  $w_d^{\text{MS}} := w_d^{\text{MS}} \cdot (\frac{k_d}{K})^2$  and subsequently  $w_d^{\text{noMS}} := 1 - w_d^{\text{MS}}$ . Then take  $\hat{\theta}_d := w_d^{\text{noMS}} \cdot \hat{\theta}_d^{\text{noMS}} + w_d^{\text{MS}} \cdot \hat{\theta}_d^{\text{MS}}$ .
- In other cases, just take the weighted average:  $\hat{\theta}_d := w_d^{\text{noMS}} \cdot \hat{\theta}_d^{\text{noMS}} + w_d^{\text{MS}} \cdot \hat{\theta}_d^{\text{MS}}$ .

## 3.2 Destinations with few flights

### 3.2.1 Goal

It was discussed in Section 2.4.1 that during the validation, it turned out that it was risky to multistretch destinations with few rotations (call them “small destinations”) on their own;<sup>19</sup> furthermore, directly working with such destinations is disadvantageous from the computational point of view since the algorithm then spends a lot of time shuffling with these small destinations. On the other hand, by ignoring (i.e. by default not multistretching) small destinations we neglect a significant percentage of our rotations. For instance, if our bound is 50 rotations per period, we may lose up to 5-10% of all rotations in this way or even up to 20% if we also apply bound of 10 rotations per pair in case of the Tabu search algorithm for pairs.

An idea to avoid this problem is coupling these destinations with other destinations which are “similar” in terms of the water consumption. The second destination is assumed to be large, i.e. with a lot of rotations. The search algorithm then treats the two destinations – the small one and the associated large one – as only one destination. In this way, we do not lose valuable multistretchable rotations, but on the other hand the following desirable properties hold:

1. The algorithm is not hindered by small destinations.
2. The virtual shortage – to be used in determining the numeric water shortage – of the small destination is the same as the virtual shortage of the large destination; hence we do not risk estimating it from insufficient data, a particular risk in case of multistretched data.

Note that after a run of the algorithm, the two destinations should be coupled back and it should be tested that the estimated shortage for each of them separately will not be too large

<sup>18</sup>The motivation is, the less data we have, the more we want to disregard it.

<sup>19</sup>If we use the boolean shortage vector, and to a smaller extent also with the numeric shortage vector, the shortage results are very vulnerable to small perturbations of the schedule and tanking data.

– however, we may decide to allow even for a somewhat excessive shortage since, presumably, this is just a random variation (since the large destination works overall fine; of course this assumes that the exposure to multistretching for both destinations in the timetable is approximately the same, as opposed to the case that for one destination, most rotations can be multistretched while for the other destination not<sup>20</sup>).

### 3.2.2 Implementation

Assume that we have non-multistretched historical data to “learn” suitable pairs of destinations to couple.<sup>21</sup> Now consider a period of interest. We will still discard some (very small) destinations – the justification for the lower bound will be given later. Hence, assume that small (but not too small) destinations are those with between 10 and 50 rotations. We will compare these small destinations to large destinations, namely their water consumption empirical distributions. This will be done using the Anderson-Darling test for two samples (see the beginning of Section 2.1.4 for more background). The approach is similar to the one in Section 2.1.4; however, the method to moderate the possible effects of multiple testing by not testing all small destinations to all large ones, but only to the “similar” ones, is different now. As a measure of “similarity”, we consider the distance from Amsterdam.<sup>22</sup> The justification is that for destinations with the same distance flown, it is feasible that the water consumption patterns coincide. On the other hand if the distance flown is very different, it is more likely that the test acknowledges equality of distributions just by coincidence. If we have a small destination with a distance from Amsterdam  $\delta$ , we consider as “similar” the large destinations with the distance to Amsterdam between  $0.8\delta$  and  $1.2\delta$ . This width of the interval was chosen arbitrary, but it was taken care that there would be enough “similar” destinations in this way.

The upper bound of 50 rotations for small destinations is somewhat arbitrary – it was chosen based on the above mentioned validations. On the other hand, there is a good reason for the lower bound:

#### Motivation for the lower bound for small destinations.

The goal was to choose the bound as low as possible, so that we “save” as many small destinations as possible. On the other hand, the bound must be large enough such that the statistical properties of the Anderson-Darling test are still satisfactory. By satisfactory properties we mean that the power of the test is still sufficient for a sample of a size equal to the lower bound – otherwise, we come always to the conclusion that the small destination and the candidate large distribution have the same water consumption distribution. Such a result would be useless. In the following, we justify why the lower bound was chosen equal to 10.

To assess the power of the Anderson-Darling test in our setting for various sample sizes, results from [31] are used. There, several popular tests to determine whether a given sample comes from a given distribution (i.e. goodness-of-fit tests) are evaluated. The results of the

---

<sup>20</sup>For instance because the flights to it tend to be the first flights of the registration-days.

<sup>21</sup>Of course, these pairs would have to regularly revised and this would have to be done using multistretched data. This issue has not been seriously considered yet.

<sup>22</sup>Alternatively, one could take the similarity in the length of flight. This is even a slightly better indicator however it is not so easy to work with it since there is some variability in it. Note that the length of flight need not correspond exactly to the distance because of different (average) weather circumstances and because of strategic considerations in the KLM’s network planning.

article address the Anderson-Darling test for *one* sample while we use the version for two samples.

We give a heuristic argument to support the use of the results even in our case: Upon considering the definition of the two-sample statistic (2.5) and the statistic for one sample (2.4), one sees that if one sample is considerably larger than the other one, the two samples statistic very much resembles the one-sample statistic for the case of testing the small sample against the empirical distribution given by the larger sample. In our problem, the number of water consumption records for the large destinations is in general indeed considerably larger than the number for small destinations. Furthermore, the empirical distribution of the larger sample is presumably already very close to the true distribution of the larger sample.<sup>23</sup>

Our interest lies in the so-called “Case 0” – this means that the reference distribution is fully specified (which it is in our case by the empirical distribution of the large sample), i.e. it has no unknown parameters. The powers for different test statistics are displayed in [31, Table 3], we summarize the findings which are important to us. First, without the loss of generality only the situation when the reference distribution is uniform is presented. Indeed, if the original reference distribution is fully specified, this is enough: If we are given a sample  $x_1, \dots, x_n$  from the reference distribution with cdf  $G$  then the null hypothesis holds (i.e. the sample comes from the reference distribution) if and only if  $G(x_1), \dots, G(x_n)$  comes from the uniform distribution on  $[0, 1]$ .

Some of the presented results represent power against the changes in variance and some against the changes in mean. We are mostly interested in possible changes in mean and hence concentrate on these results. Here, the true distribution function was given by

$$F(z) = 1 - (1 - z)^k, \quad 0 \leq z \leq 1 \quad (3.16)$$

where  $k = 1.5$  or, for a larger shift,  $k = 2$ . There were 1000 samples drawn from this distribution function and it was tested what percentage of these do the selected methods distinguish, for which percentage the methods reject the null hypothesis that the true distribution is uniform. It is shown that the Anderson-Darling test detects a moderately large (i.e.  $k = 2$  in (3.16)) difference in distribution in cca 60% of cases if the tested sample has size 10 and already in cca 90% of cases with the size 20. We consider the former percentage high enough in order to take the lower bound equal to 10.

### 3.2.3 Results

For the testing purposes, winter 2010 was considered as the target period. As a criterion for coupling destinations, we require the p-value from the Anderson-Darling test larger than 0.1. That is higher than the standard 0.05 because the null hypothesis – that the coupling is possible – is considered a priori not very reliable.

The basic Anderson-Darling test assumes that the samples are drawn from continuous distributions and hence there are no ties among the observations. This is not the case for the considered samples. However, the function `adk.test`, which we used, from the R package `adk`, “adjusts for a moderate number of ties (due to rounding)” [28].

The process is not completely automatized: a question remains, what to do if there are more admissible candidates. It is possible to take one at random or perhaps decide on basis

---

<sup>23</sup>Admittedly, this whole argument is indeed only heuristic. While in (2.5), it is easy to show that  $\hat{F}_n \approx \hat{H}_N$  uniformly, the problem is that from this it seems not immediately obvious that also the associated measures are similar, i.e.  $d\hat{F}_n \approx d\hat{H}_N$  in a suitable sense.

of geographical (or other) considerations. On basis of the computed tests, the decision would be (for winter 2010)

- F70: Couple Aberdeen to Goteborg; Glasgow to Kjevik, Munich or Stavanger; Berlin to Bristol or Cardiff; and leave Newcastle, Oslo and Warsaw uncoupled.
- E90: Couple Edinburgh to Munich; Glasgow to Aberdeen or Stavanger; leave Billund, Bologna, Nice and Newcastle uncoupled.

Plotting the histograms of the recommended couples shows that the above recommendation might be a reasonable one.

When using the above mentioned couplings in the evaluation procedure, there seems to be a small improvement in the objective function, but nothing groundbreaking; see Section 2.4.

### 3.3 Factors determining water consumption

In this section, dependence of the water consumption on several other variables (i.e. other than the destination) is considered. We assume that the data considered are not multistretched.

#### 3.3.1 Background on regression analysis

This section gives some background on the regression analysis, in particular linear regression, which will be used in what follows. Our sources are primarily [12, 30].

The purpose of regression is to explain or model the relationship between a response variable  $Y$  (or possibly a vector, but we will not consider this option) and the explanatory variables  $X_1, \dots, X_k$ . We will assume that water consumption is continuous even though this is not completely correct due to rounding. Hence we will consider only the theory for  $Y$  being a continuous variable.

Linear regression means that the model can be written as

$$y_i = \sum_{j=0}^k \beta_j x_{i,j} + e_i \quad (3.17)$$

where  $y_i$  is the  $i$ -th observation of  $Y$  (for  $i = 1, \dots, n$ ),  $x_{i,j}$  is the value of  $X_j$  during the  $i$ -th observation for  $j > 0$  and  $x_{i,0} \equiv 1$  and  $e_i$  is an error term. Then  $\beta_j$  are the parameters of the model. Note that the regression is called “linear” because of the linearity in  $\beta_j$ : on the other hand, it is possible that  $X_2 = X_1^2$  or that we consider  $\log(X_2)$  instead of  $X_2$  and this would still be considered as a case of “linear regression”. Most of the following is concerned with the linear regression.<sup>24</sup> Even more, as discussed in the paragraph Basic considerations, for the dependence on the number of passengers we will use only the simple linear regression

$$y_i = \beta_0 + \beta_1 x_i + e_i. \quad (3.18)$$

The usual additional assumptions on (3.18) under which many results can be derived are

- errors  $e_i$  are independent

---

<sup>24</sup>With the exception of the proposed model (3.19) where the dependence on the departure time is considered not necessary linear.

- errors  $e_i$  are normally distributed with mean zero and equal variance  $\sigma^2$

We are interested in the values of parameters  $\beta_0, \beta_1$ , so that we can then predict future values of  $Y$  given the values of  $X_1$ . For the values of parameters, we then want to derive confidence intervals, from which one can in turn deduce if the value of parameters  $\beta_1$  is significantly different from 0, i.e. whether there is a significant dependence on the  $X_1$ . Under the above assumptions, coefficient estimates  $\hat{\beta}_0, \hat{\beta}_1$  are usually obtained by the least squares method, the pair  $(\hat{\beta}_0, \hat{\beta}_1)$  being the minimizer of the function

$$(b_0, b_1) \mapsto \sum_{i=1}^n (y_i - b_0 - b_1 x_i)^2.$$

Solving this explicitly yields

$$\begin{aligned} \hat{\beta}_1 &= \frac{\sum_{i=1}^n x_i y_i - n \bar{x} \bar{y}}{\sum_{i=1}^n x_i^2 - n \bar{x}^2} \\ \hat{\beta}_0 &= \bar{y} - \hat{\beta}_1 \bar{x}, \end{aligned}$$

where  $\bar{x}$  is the mean of  $x_i$  and  $\bar{y}$  is the mean of  $y_i$ .

While the assumption of equal variance of errors will not be rejected and their independence is more or less plausible (see the discussion after Assumption 2 in Section 3.1.1, though), the errors are most likely not normally distributed (at the very least because of the zero observations – see Figure 1.2). However importantly, for large samples the confidence intervals, and therefore the associated significance testing as well, obtained under the normality assumption will be valid anyway because of the central limit theorem – indeed, the coefficient estimates  $\hat{\beta}_0, \hat{\beta}_1$  are weighted sums of  $y_i$ .

A special case of (3.17) is when  $X$  takes only values in  $\{0, 1\}$ . Sometimes, the model is referred to as ANOVA.

To judge on the usefulness of the model, the coefficient of determination  $R^2$  is often used, defined by (in case of (3.18))

$$R^2 := 1 - \frac{\sum_i (\hat{y}_i - y_i)^2}{\sum_i (\bar{y} - y_i)^2},$$

where  $\hat{y}_i = \hat{\beta}_0 + \hat{\beta}_1 x_i$  and  $\bar{y}$  is the mean of  $y_i$ .

### 3.3.2 Motivation

If an interesting dependence is found, it is possible to split the relevant destinations into two or more parts. As an example, rotations to Prague might be divided into the rotations to Prague with less than 50 passengers and those with more than 50 passengers. By an “interesting dependence” one means that the subdestinations are as much different as possible. Note that splitting a destination always lead to possibly better solutions, this is because the state space becomes finer. However, the implicit assumption of the current endeavor is that the more different the two “subdestinations” are, the better solutions are enabled. On the other hand, if the subdestinations do not differ at all, then there is no reason to split on this factor – we could equally well just split randomly.

## Splitting data

The above discussion, as well as the one in Section 2.1.4 assumes that when faced with a choice to split data into several groups, one should choose groups which are sufficiently different in terms of water consumption and hence in terms of virtual shortages. And if we already have some groups and want to split them further, the resulting groups should be as different as possible.

To support this, consider the following argument: Assume that for every pair of subsequent rotations  $(s, r)$ , we have a virtual shortage  $v_{s,r}$  defined as the probability that if  $r$  is multistretched, we will suffer a water shortage during  $r$ . Note that this probability depends only on the water consumption during  $s$  and  $r$ . Since the rotation  $s$  is uniquely determined by  $r$ , we will write it as a function of  $r$ , i.e.  $s = s(r)$ . Assume further that all pairs of rotations are independent, i.e. we can decide to multistretch a pair without affecting the possibility to multistretch other pairs. Last, assume that the restrictions from the beginning of Section 2.1.1 do not apply. Then, for a destination  $d$ , if  $\text{ROT}_d$  is the set of rotations going to  $d$  and  $\text{MS}_d$  is the set of the multistretched rotations to  $d$ , the optimal strategy would be the one which for every  $d \in \text{DST}$  solves

$$\max \frac{\sum_{r \in \text{ROT}_d} 1}{\sum_{r \in \text{ROT}_d} v_{s(r),r}} \leq \alpha.$$

Hence as is easily seen, that strategy would multistretch for every destination the pairs of rotations with the smallest virtual shortages, more precisely as many of the smallest ones as the number  $\alpha$  allows. If our groups of rotation-pairs are less fine than individual rotations-pairs, then taking the groups with the smallest average virtual shortages seems the way to go. This is in principle what the heuristic in Section 2.3.2 does. Note that the heuristic does not work perfectly since the computed virtual shortages do not correspond to the true shortages which would occur if the destination is multistretched.

The above insights lead to two observations:

1. The initial groups of rotations should have different water consumption distributions (and note that in Section 2.1.4 it was verified that grouping according to destinations satisfies this requirement) – then it is likely that the groups lead to pairs with different virtual shortages and hence the selection could be made as above.
2. When initial groups are fixed and there is a group that cannot be multistretched, this group can be split into subgroups, one with higher and one with lower water consumption. Then, possibly, at least the subgroup with the lower consumption can be multistretched. And one could go on, splitting the subgroup with the higher consumption further.

The second argument supports the idea that when splitting, one should create two subgroups with water consumption distribution as different as possible. More precisely – two subgroups such that the difference in means<sup>25</sup> is as large as possible. This is because it is then more probable that the subgroup with smaller consumption will become multistretchable.

---

<sup>25</sup>Or another values capturing the distribution's location.

## Basic considerations

In order that splitting leads to a useful result, it must be required that both subdestinations have enough records for the period where we would like to implement the split. Otherwise the algorithm would just a priori reject multistretching of one of the subdestinations.

Note that the problem of dependence of the consumption on different factors was also considered in the Master's thesis [25]. There, large aircraft (Boeing 747, 767, 777 and McDonnell Douglas MD-11) flying intercontinental flights were considered. Here, the ultimate purpose was not multistretching, but taking less water on the board for each flight – this is because for very long flights, less weight<sup>26</sup> means much lower fuel usage. Hence, based on different factors, it was determined how much water was enough to take such that the probability of shortage is less than  $\alpha$ .

We are now working with different airplanes and also considering a slightly different problem:

- Our decision is: “To tank or not to tank”. The decision in [25] was: “How much to tank”. Hence our decision is a priori not so sensitive to the other factors: indeed, there can be a dependence between the consumption and the considered factor, but unless the resulting difference in consumption is large enough that it could possibly change the decision to tank or not, we can discard the result.
- For the smaller airplanes, the range of number of passengers is not so large, most flights are almost full.<sup>27</sup>

In [25], quite a lot of research was done on the dependence on passengers. The dependence on month and time was also considered ([25, Sections 2.2.4, 2.2.5, 2.3.4, 2.3.5]), even though no statistical tests were performed. It is argued that night and day flights can be combined – but this is not interesting for us since KLC flights are always between cca 6:00 and 22:00.

The following explanatory variables for regression modeling are considered:

- number of passengers on the rotation (sum for the outbound and inbound flight)
- month
- time (hour) of departure.

Trying out some further variables was envisioned, namely temperature, data over passengers (e.g. how many business class passengers?) and data about catering on the board. However, this data is not in the planning system and so it could not be used during the online decision process. In [25], flight duration is also considered. However, the scheduled flight duration may be assumed constant for flights to one specific destination while the actual flight duration is only known afterwards.

Note that taking, for a rotation, a total number of passengers on both flights is only justified because we will consider just linear dependence on the number of passengers.

Note that the dependence on month is somewhat tricky to use in practice since it would require to often change the rules in the planning system. Of course there can be an automated rule taking care of all of that, but the planners might find it unpleasant.

---

<sup>26</sup>In contrast to our small aircrafts, the aircrafts considered there had tank capacities between cca 600 and 1600 liters, so it was often possible to take several hundred liters less.

<sup>27</sup>While for the larger airplanes, which fly intercontinental flights, there is quite a wide range of number of passengers which can appear.

All regression models will be considered only after restricting to one aircraft type and destination. Before starting, we should realize the following facts:

- From the exploratory data analysis, it seems that the data is very noisy and the regression will not explain much variance. But even a little bit might help for our purposes. Indeed, in [25], a useful model to determine the amount of water needed based on the number of passengers is derived, even though the coefficients of determination for the regression model are low: in some cases virtually zero and never higher than 0.2; in [25] this is mentioned in passing in Section 3.3.
- The explanatory variables are probably correlated (this is called “multicollinearity”; but a multidimensional model will not be fitted for different reasons, so there is no need to worry about this).
- Note that a feasible state after splitting destinations is still feasible if we evaluate the split destination as a whole. This is because if all subdestinations have shortages less than  $\alpha$ , then this also holds for the whole destination.

Several assumptions are made:

- The number of passengers is considered as a continuous variable, month as a categorical variable. The departure time is more complicated, the method for it will be described below.
- There are no interaction terms between explanatory variables: This seems as a first approximation plausible and furthermore possible interactions could be obscured by the noise anyway.
- Still, the errors are assumed to have the same distribution.
- The dependence on the number of passengers is monotone (i.e. more passengers imply higher water consumption), the dependence on time is not (the consumption can vary during the day).

A model could therefore be (for all  $d \in \text{DST}$ )

$$\text{Consumption}_i^d = \beta_p^d \cdot \text{pass} + \beta_m^d \cdot \text{month} + f^d(\text{time}) + \epsilon_i^d \quad (3.19)$$

where the dependence on the aircraft type is implicit,  $\epsilon_i$  is an error term and  $f$  would be a function. However, recall the goal. We want to split some destination in two or three (and probably not more because the resulting subdestination would be mostly too small then). From this point of view, the model (3.19) will likely lead to rather complicated rules (e.g. split if there are less than 50 passengers, it is June or December and the flights departs at 13:00 or 19:00), the resulting subdestinations will be often too small and there may be more comparable possibilities to split. In view of this, we decided to start considering one-dimensional model separately. The steps could be:

1. Fix an aircraft type and a destination.
2. Regress on passengers. Possibly split.
3. Regress on departure time. Possibly split.

#### 4. Regress on month. Possibly split.

This approach is somewhat less powerful – for instance nothing is found if there is a dependence on passengers and departure time, but the individual dependences are too weak. However this approach is much simpler. In order to split, the “typical” consumption difference between the subdestinations should be “large enough” – we require at least 4 liters “typical” difference. Admittedly, this number is rather arbitrary – it comes from the desire to make the resulting new numeric shortage vector after splitting different enough and also from the realization that with the quality of our data, the difference of 3 liters between two observations could very well be just due to rounding.

### 3.3.3 Dependence on number of passengers

All available data (i.e. from October 2009 until January 2012) are taken and we consider just the destinations with more than 200 rotations – this seems a reasonable number given that afterwards, they will be considered during one half-year period and we will try to split them. Note that our consumption data is per rotation (i.e. not per individual flights). Hence the explanatory variable is the sum of the passengers on the outbound leg and for the inbound leg – in particular, this kind of modeling critically depends on the fact that the assumed dependence on the number of passengers is only linear.

First, some exploratory analysis is performed to see if the modeling makes sense at all and if yes, for which destinations. Note that one would not lose much by directly fitting regression models and then discarding those with nonsignificant linear coefficients, i.e. skipping the exploratory phase.

We computed the Spearman correlation coefficients and their p-values. This is a measure for nonlinear monotone dependence between two samples while the Pearson coefficient represents linear monotone dependence.<sup>28</sup> It is observed that values of both coefficients turned out to be very similar for all destinations. We decided to proceed only with the destinations which had the Spearman coefficient significant at 0.05 level (i.e. the p-value was smaller than 0.05). Those were Aberdeen, Bologna, Bristol, Edinburgh, Humberside, Kjevik, Leeds, London, Linkoping, Liverpool, Luxembourg, Durham, Nice, Prague and Toulouse for F70 and Birmingham, Billund, Bologna, Bordeaux, Glasgow, Goteborg, Geneva, Helsinki, Marseilles, Munich, Nice, Oslo, Stavanger, Toulouse, Venice, Vienna, Warsaw and Zurich for E90. Note that in view of multiple testing, perhaps one or two destinations in each set are incorrectly classified as significant, but since this is just an exploratory analysis, it does not matter. The coefficient values are in all cases around 0.1; in fact, apart from one exception, they are all between 0.05 and 0.2. So they are very low.

To appreciate this, consider the dependence in Figure 3.3 for Bologna, here the correlation coefficient is one of the highest, almost 0.19. The fitted regression line is already shown in the figure as well. It is apparent from Figure 3.3 that even though the trend might be at first sight hardly discernible in view of the noise, the predicted difference in water consumption between 80 and 150 passengers is considerable.

The above lists of destinations for which there is significant correlation between water consumption and number of passengers are surprisingly small. Indeed, both for F70 and E90, the destinations account for less than half of all destinations with enough rotations. Even for

---

<sup>28</sup>The  $R^2$  statistic in a one-dimensional regression model is actually the Pearson correlation coefficient squared.

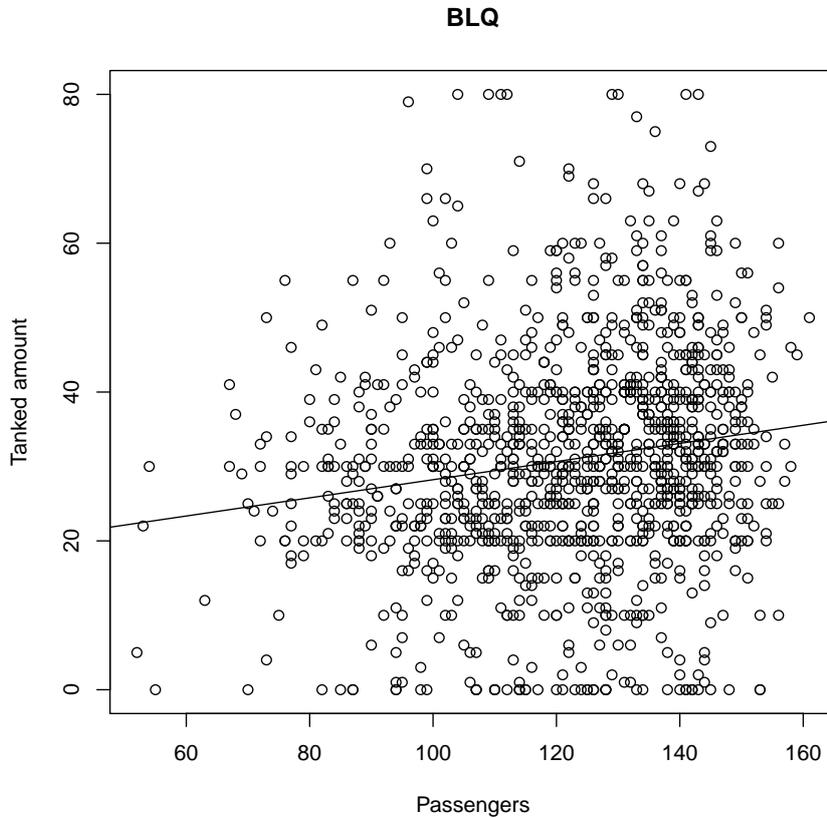


Figure 3.3: Dependence of water consumption on passengers (the sum of passengers on the outbound and inbound flight of a rotation) for Bologna, F70

the rest of destinations where the dependence is significant, the dependence is rather weak in comparison with the underlying noise. Upon consideration, we propose two explanations for this phenomenon:

- For short flights, the toilet, contributing to the consumption by washing hands and in case of E90 by flushing, can be almost permanently occupied – then it does not matter if there are 50 or 75 passengers.
- The drinks consumed are very much dependent on the activity of the cabin crew. In particular, in [25] it is argued that when there are few passengers, the cabin crew has more time to distribute drinks, on the other hand, if there are many passengers, the cabin crew may try not to squander water – even though this will be more of a case for intercontinental flights, for flights within Europe, there is usually enough water.

### Regression

Before starting with regression, we discuss when it is worth to split the destination basing on passengers. A natural splitting criterion is to have a subdestination with “few passengers” and a subdestination with “many passengers”, i.e. choosing a number of passengers serving

as a threshold. This threshold is possibly different for different destinations). Consider two possibilities:

- Split according to a quantile of the number of passengers
- Take the threshold small enough such that the consumption difference is large enough, but on the other hand the two subdestinations have comparable numbers of rotations.<sup>29</sup>

In the second case, it does not seem obvious how to strike the right balance and hence the first method was tried. One could take just the median as the threshold. The problem with such choice is that most flights are close to the full number of passengers. For instance, for Toulouse, the median number of passengers is 138 (out of 160). As a “typical” flight for each of the subdestinations, one can take the one with a median number of passengers. But then the difference in the median of the “few passengers” and “many passengers” subdestinations would be rather small.<sup>30</sup> We therefore use instead for the threshold the  $\frac{1}{3}$ -quantile. Hence a “typical” flight with “few passengers” has the  $\frac{1}{6}$ -quantile of passengers while one with “many passengers” has the  $\frac{2}{3}$ -quantile.

The criterion for splitting is therefore that the difference between consumption for a rotation with  $\frac{2}{3}$ -quantile of passengers and  $\frac{1}{6}$ -quantile of passengers is at least 4 liters (or whatever number we choose).

Now we have tried out regressing for all destinations where significant correlation between the consumption and the number of passengers was found. For F70, the criterion value of 4 liters is exceeded only for Nice (4.2). For E90, it is Glasgow (10.5), Nice (4.8), Stavanger (5.0) and Berlin (4.2) – and among these, Nice and Berlin are forbidden for multistretch. Hence, we see that only for very few destinations the number of passengers makes a large difference. It is therefore probably not worthwhile to include splitting based on the number of passengers in the default algorithm.

Before taking the above conclusions for granted, the fulfillment of the assumptions of the linear regression should be checked. To this end, methods from [30, Chapter 3] were followed. From Figure 3.3, there is no recognizable nonlinear pattern. In fact, there is no easily recognizable linear pattern either – the linear model serves indeed rather to capture a monotone pattern in the simplest possible way. From the residual plot, it is safe to assume that the residuals have constant variance, at least on the range of passengers numbers which are relevant. As expected, the QQ-plot indicates that the residuals are not normally distributed. However, the sample size is large – in fact, only the destinations with at least 200 observations were considered. As mentioned in Section 3.3.1, this means that the non-normality of residuals can be safely ignored when it comes to the estimation of coefficients – and consequently also point predictions.

### 3.3.4 Dependence on month

For a preliminary analysis, we looked at boxplots (dependence of water consumption on month). For both F70 and E90, some differences can be discerned, but nothing special. The overall feeling is that the consumption is higher during the winter. Perhaps people drink more warm drinks?.

---

<sup>29</sup>If the subdestination with the lower consumption is too small, then for an unmultistretchable destination, slipping only the subdestination with lower consumption will have smaller effect than it could have, which is a pity, or even worse, the subdestination has less rotations than deemed acceptable.

<sup>30</sup>This need not be true in general, however in our case it is.

hour	6	9	10	12	13	14	16	17	20	21
flights	523	340	48	1	569	23	243	85	403	217

Table 3.1: Flights according to the departure hour from Amsterdam to Brussels, for F70. Hours not mentioned had no flights.

To consider the result for a certain destination interesting, we require:

- There are significant differences (at 0.05 level) to January for at least 3 destinations (in the same direction), the differences being at least 4 liters.

In this requirement, we consider January since, as noted, the consumption for it might be the lowest. We require the difference for at least 3 months, so that it could be possible to split the data according to these months in such way that both parts of the data would have enough observations. The difference at least 4 liters is required for the same reasons as before.

For F70, the requirement is satisfied only for Bologna (June, July and September show higher consumption). For E90, the requirement is satisfied for Birmingham (June, July, September and October show lower consumption), Stuttgart (May, July, August show lower consumption), Warsaw (here January, February and March seem to have significantly lower consumptions than other destinations).

Since we observe a large enough difference only for few destinations, splitting on month has not been implemented in the algorithm.

### 3.3.5 Dependence on departure time

The formula in (3.19) suggests that the dependence on departure time could be modeled non-linearly. However, it turns out that for each destination, there are several clusters of departure moments during the day. For F70 for Brussels, this clustering (with times rounded to hours) can be seen in the Table 3.1, hours not mentioned have no flights. A similar table is rather typical for all destinations, even though perhaps with less groups – often there is one “morning”, one “afternoon” and one “evening” group.

Hence there is not enough data to model the consumption throughout the whole day since there are no flights during some time periods. On the other hand, the Table 3.1 suggests that one could instead model water consumption differences among these groups – which is what we decided to do. An important advantage of this approach, as opposed to the full fledged modeling based on departure time (i.e. water consumption depends on the departure time as some continuous function) is that it is clear how to then form the subdestinations – some groups of departure times correspond to one subdestination and other groups to the second subdestination. Using the full fledged modeling would probably require splitting into subdestinations in a – to some extent – arbitrary manner. To determine the groups, the scheduled departure times are first grouped according to the hour (e.g. 18:49 is 18). Then, an algorithm which determines the clusters of hours automatically has been implemented; for instance in the previous case, the groups would be (6), (9, 10), (12, 13, 14), (16, 17), (20, 21) and the last group consisting of all other hours. The program can be found in Appendix B.3; the created groups have always enough observations that they can enter as factors into regression. The last group can be an exception – but that one can be ignored in the regression. The group with most rotations is then taken as the reference group in the regression, i.e. other groups are compared to it.

These are scheduled departure times; if we look at the actual departure times (i.e. taking late flights and other peculiarities into account), then the table becomes somewhat more messy. We will not care about the actual times; after all, if we are interested in making predictions, the actual departure times will not be available to us.

For inbound flights, the departure times are not available in our data. We therefore extrapolate them using the arrival times to Amsterdam and using average flight times. It is assumed that a flight from Amsterdam to  $d$  departing at the hour  $H$  is equivalent in water consumption to a flight from  $d$  to Amsterdam departing at the hour  $H$ .<sup>31</sup>

Note that while the departure times might have changed throughout years, Table 3.1 indicates that the changes are not overwhelming. Similar results hold for other destinations as well.

For F70, there are some differences, but none particularly important ones and many are not significant at all. On the contrary for E90, the results are promising – for some destinations, the difference between very early and very late flights and the remaining flights can often make 20 liters.

In this case, splitting was tried out for the winter 2010 data. We do not split according to all time groups. Instead, it is often the case that one or two groups differ in the same “direction” from the rest – then one subdestination is created with departures in one of the two groups and the second subdestination which contains rotations with all other departure times. For winter 2010, we mention the destinations to split in Table 3.2. After the name of the destination follow the hours determining the first subdestination and afterwards the minimal (in absolute value) estimated differences of average water consumption between this subdestination and the second one.<sup>32</sup> The results of the Tabu search using this splitting

Destination	Hours for the first subdestination	consumption difference (L)
Zurich	6, 7, 8, 19, 20, 21, 22	−6.6
Munich	6, 7, 19, 20, 21	−7.2
Vienna	19, 20, 21	−10.9
Marseille	19, 20, 21	−14.9
Bordeaux	19, 20, 21, 22	−17.2
Trondheim	19, 20, 21	−19.1
Stuttgart	19, 20, 21, 22	−8.6
Toulouse	19, 20, 21	−9.8
Helsinki	9, 10, 19, 20, 21	−10.3

Table 3.2: Destinations with large water consumption differences throughout the day

can be found in Table 2.2. As can be seen, there is indeed a noticeable improvement in the objective function.<sup>33</sup>

<sup>31</sup>It is possible that there are on average different numbers of passengers for such flights. However, as has been seen in Section 3.3.3, the dependence of water consumption on the number of passengers is in most cases not too important. Therefore, a special treatment of inbound and outbound flights because of this reason is unwarranted.

<sup>32</sup>The regression gives for each group a difference w.r.t. the reference group. If we then pool several groups together, the minimal difference of consumption averages is the minimum of differences between the groups from the two pools. It would be possible to obtain the difference of averages exactly by refitting the model after pooling the observations. However, to avoid doing it manually, this would require automatizing of the pooling which has not been done.

<sup>33</sup>It might be objected though that the basic tabu search for pairs is not a good reference point. Instead,

A useful rule of thumb seems to be that flights departing at 8 and earlier and 19 (or 20) and later have lower water consumption than other flights.<sup>34,35</sup> Curiously enough, in [25], an almost opposite pattern seems to emerge: night flights have higher consumption than day flights. It is difficult to say this with certainty, though, because only results for too few destinations are shown in [25]. That said, even if it is the case, in view of the large differences between intercontinental flights of large airplanes and short flights of small airplanes, this need not mean any contradiction.

When it comes to checking the regression assumptions, the distribution of residuals is again not normal which was expected (and can be ignored due to the large sample size). The assumption of equality of variances for different groups was tested using the Brown-Forsythe test, which is suitable for non-normal data, and the assumption was not rejected.

**Interlude: Brown-Forsythe test.** The Brown-Forsythe test for the equality of variances was suggested in [6], as a modification of Levene’s test. We are in the situation of (3.17), with  $X_i$  taking only values in  $\{0, 1\}$ . Hence, after reindexing  $y_i$  and  $\epsilon_i$ , we may (assuming that  $X_i = 1$  iff  $X_l = 0$  for  $i \neq l$ ) rewrite the model (3.17) as

$$y_{i,j} = \beta_i + \epsilon_{i,j}$$

where  $j = 1, \dots, n_i$  and there are  $g$  groups (indexed by  $i$ ). The question is if the distributions of groups  $\{y_{i,j}\}_{j=1}^{n_i}$  exhibit the same variance.

Let  $y'_i$  be the median in the  $i$ -th group and define

$$z_{i,j} := |y_{i,j} - y'_i|, \quad \bar{z}_i := \frac{\sum_j z_{i,j}}{n_i}, \quad \bar{z} := \frac{\sum_{i,j} z_{i,j}}{\sum_i n_i}.$$

The test statistic for the Brown-Forsythe test is then defined by

$$W_{50} := \frac{\sum_i n_i (\bar{z}_i - \bar{z})^2 / (g - 1)}{\sum_{i,j} (z_{i,j} - \bar{z})^2 / \sum_i (n_i - 1)}.$$

The advantage of the test is its robustness against departures from normal distribution (which is our case) in the sense that the probability of type I error (i.e. the probability that the null hypothesis, that all variances are equal, is falsely rejected) is not underestimated for non-normal distributions.

### 3.3.6 Differences in water consumption throughout years

It has been implicitly assumed that the water consumption remains the same throughout different years. However, it was stressed already at the beginning of the project that always the most recent data should be used since airplane and passengers characteristics can change unexpectedly and without notice.

---

one should run the algorithm with 9 random destinations split randomly into two halves, to control for the possibility that the improvement is purely due to refining of the state space.

<sup>34</sup>For the later flights, this may partially stem from the fact that during a night stop at an outstation, water is refilled hence our measured data corresponds to only one leg of the rotation. However even in that case, splitting flights at these times and treating them separately is a good option.

<sup>35</sup>For the above flights, this rule of thumb works almost perfectly – the destinations considering just only morning or only evening flights do so because there are no flights of the second category.

To test whether differences exist, the Kruskal-Wallis test was used. The test decides if location parameters of several samples can be considered equal – see for instance [20]. We must assume that different seasons do not differ in the shape of the distribution – this seems a plausible assumption. The advantage of the test is that there is no specific distributional assumption.

The test was performed per destination and aircraft type. Different seasons are compared against each other, with summer and winter periods treated separately. The test shows that for quite some destinations (even when taking into account the multiple testing), the locations for different periods differ. When computing medians, one sees that the difference is probably not too large: The medians differ mostly no more than by 2 or 3 liters, even though a case of a 11 liters difference exists as well.

As has been already mentioned in Section 2.4.1, the differences between different periods are therefore mostly not large enough to endanger our method of using for a new period a strategy optimized on the preceding period.

### 3.3.7 Distinguishing between different aircraft types

We expect that, since in E90 the drinking water is used for flushing toilet and in F70 not (while the other uses are the same), the average water consumption will be larger for E90. It has been already mentioned that for the data aggregated over all destinations, the mean of water consumption for E90 is larger than the one for F70. Still, this is not enough since it is conceivable that E90 flies on average to farther destinations which would also be a reason for higher consumption.

Therefore, we test the water consumption per destination. For this test, the data from the whole history are considered and the destination for which there are at least 100 rotations for both E90 and F70. The non-parametric Mann-Whitney test is used to test whether the location difference of the distributions is zero.

**Interlude: Mann-Whitney test.** Assume that  $V_1, \dots, V_m$  and  $W_1, \dots, W_n$  are two independent i.i.d. samples. In our case, those will be independent copies of  $X_d^{F70}$ , respectively  $X_d^{E90}$ . According to [33, Section 12.2], one defines  $U$  by

$$U := \frac{1}{n \cdot m} \sum_{i=1}^m \sum_{j=1}^n 1_{\{V_i \leq W_j\}}.$$

The random variable  $n \cdot m \cdot U$  is referred to as the Mann-Whitney statistic and “a large value of the statistic indicates that the  $W_j$  are stochastically larger than the  $V_i$ ” (in fact,  $U$  is an unbiased estimator of  $\mathbb{P}\{W_1 > V_1\}$ ). For small samples, the values of  $U$  are tabulated or can be computed exactly, for larger ones ( $m$  and  $n$  larger than say 50), the normal approximation is used instead:

$$\sqrt{\frac{12mn}{m+n}} \left( U - \frac{1}{2} \right) \approx \mathcal{N}(0, 1).$$

We note that for rounded data, where ties may occur, a special correction must be used.

**Results.** It is found that for almost all destinations, the consumption for E90 is higher as expected, even after taking into account that multiple testing takes place. Exceptions are Brussels and Hamburg where the null hypothesis of no difference in consumption is not

rejected. We conclude that treating E90 and F70 separately throughout the project has been justified.

### **3.3.8 Implications for Assumption 2**

It has been shown that there are several variables partially explaining the water consumption. This means that Assumption 2, that water consumptions on individual flights are independent, is not valid. On the other hand, the dependence on those explanatory variables tends to be rather weak – this being measured for example by the small values of  $R^2$  in the regression models.

Disregarding the explanatory variables, there would be no reason not to believe Assumption 2. Since we have included the most prominent explanatory variables in our analysis and their influence tends to be weak, we conclude that the dependence among the water consumption variables is likely also only weak. We conclude that postulating Assumption 2 is likely to be a good approximation of reality.

## Chapter 4

# Miscellaneous topics

### 4.1 Data issues

This section addresses some issues related to our data - how to handle it, quality of it, etc. The section is not strictly essential for understanding of the rest of the report; however, it provides more insight into why some methods during the project were chosen and also might be helpful in order to appreciate some encountered problems which usually do not occur in a purely academic research.

#### 4.1.1 Construction of the data

In 1.4, the available dataset, containing the data about inbound-outbound flight-pairs, is presented. How is the data obtained from raw datasets?

Originally, data was constructed by a method developed during the Preliminary study.<sup>1</sup> It consisted of matching flights in a flight-pair and flight-pairs to the tanking data using the identification numbers of flights.

On the halfway of the project, it was decided to switch to a new method instead<sup>2</sup> – while both methods work equally well under standard circumstances, some special events<sup>3</sup> may lead to phenomena in the data which the original method does not handle correctly, while the new one does. The difference between the methods is in the manner of coupling flights to rotations and the manner of appending tanking data to it. The method described below is the new one.

The raw data from the planning system consists of one file with information about flights and one file with information about Aqua Services subtasks. Refilling or draining water is an instance of such a subtask; a refilling subtask record contains the amount of water tanked, among other fields. The *steps* to obtain the final dataset are as follows:

1. Extract required data fields from the raw data files.
2. Couple corresponding inbound and outbound flights to create a flight-pair: these flight-pairs then make up the records of the final dataset.

---

<sup>1</sup>It is the study mentioned in Section 1.3.

<sup>2</sup>This was shared with us by colleagues from another KLM department.

<sup>3</sup>Such an event is for instance a change of registration assignment for an already scheduled flight.

To pair, one considers different registrations one by one. For each registration, one sorts the flights according to the time, i.e. time of departure from Amsterdam in case of outbound flights and arrival to Amsterdam for inbound flights. Then we pass the flights one by one, pairing always two subsequent flights while checking that the first flight is inbound and the second one is outbound – this piece of information is available in the flights data.

A slightly unintuitive feature of this pairing is that if an airplane stays for several days in a hangar, the inbound and outbound flights can be several days apart from each other. However, this happens very seldom, essentially only in cases of technical inspections or network disruptions. Furthermore, in the way the computer programs work, this does not lead to any problems.

*Example 4.1* (Coupling flights into flight-pairs). For simplicity, assume that we have only one airplane and one day. Furthermore, let the airplane begin and end the day in Amsterdam. Assume that we are given a list of flights in Table 4.1 – only the required fields are mentioned. After sorting, we get the data in Table 4.2. Now, the first flight

Flight ID	00	01	02	03	04	05
time	16:48	17:39	20:21	13:58	12:13	8:06
outbound / inbound	I	O	I	O	I	O

Table 4.1: Flight data

Flight ID	05	04	03	00	01	02
time	8:06	12:13	13:58	16:48	17:39	20:21
outbound / inbound	O	I	O	I	O	I

Table 4.2: Flight data sorted according to the time

of the pair is always inbound and the second one outbound – hence we end up with flight-pairs (04, 03) and (00, 01), the remaining two flights are discarded. Note that if there is also a following day, then 02 would be coupled to the first outbound flight of that day.

3. Couple subtasks to the flight-pairs, in particular assign the amount tanked during the flight-pair<sup>4</sup> and the next tanked amount<sup>5</sup> to each flight-pair:

This is done again by considering registrations one by one. This is possible because every subtask record mentions what the relevant registration is. The flight-pairs for that registration are sorted according to the date. This is then done for the subtasks as well and one passes through subtasks and flight-pairs, assigning a subtask to a flight-pair if the time of the subtask execution falls between the arrival of the inbound flight and the departure of the outbound flight of the flight-pair.

The method reliably detects when there is no subtask for a given flight-pair. This almost always means that the flight-pair was multistretched.

<sup>4</sup>I.e. after the arrival of the inbound flight.

<sup>5</sup>I.e. after the airplane again returns from the destination of the outbound flight.

*Remark.* Interestingly, there are also flight-pairs (actually, quite a lot of them) with two corresponding subtasks. This was checked and it indeed corresponds to tanking twice before a flight. Since for small airplanes, the tank is always fully filled, the second tanking is completely unnecessary. This is confirmed by the fact that the tanked amounts are very small, often zero.

It seems, although this has not yet been confirmed at the time of writing, that this problem occurs when a registration is reassigned to a different flight – the planning system then automatically schedules a refill task for the registration.

*Example 4.2* (Continuation of 4.1). Assume that there were two tanking tasks: tanking 32 L at 17:12 and tanking 12 L at 13:27. We start with the earlier one. We iterate through the flight-pairs in the chronological order. For (04, 03), we see that its inbound flight arrived at 12:13 and its outbound flight departed at 13:58. Hence, we see that the tanking at 13:27 corresponds to the flight-pair (04, 03). It is written to the record of the flight pair (04, 03) that the corresponding tanked amount is 12 L. We move on to the next flight-pair and next task. Since the flight-pair’s inbound flight arrived at 16:48 and departed at 17:39, we see that the tanking at 17:12 pertains to this flight-pair.

Under multistretching, it sometimes happens that a flight-pair does not have a corresponding tanking task, i.e. the currently considered task is performed only after the departure of the considered flight-pair. In that case, it is written to the flight-pair record that no tanking data was available – and we assume that the flight-pair was multistretched.

4. Make possible further adjustments and handle incomplete or incorrect data.

#### 4.1.2 Details of the evaluation procedure

How exactly does the evaluation procedure work? The basic procedure is described here, see Section 4.2 for improvements to make it faster. Assume that we are given a strategy and a dataset to evaluate it on; also assume that we are given a shortage vector. The evaluation should provide the objective function value and shortages for a given strategy. We proceed as follows:

1. Formulate the strategy in terms of the dataset: this is done completely naïvely, for each flight-pair in the dataset, we determine this in terms of the information that we have about the rotation (e.g. inbound and outbound destinations).
2. Not all desired rotations can really be multistretched: It must be checked that the preceding rotation is not multistretched, this is because of the rule that two consecutive rotations may not be both multistretched. Furthermore, the first rotation of a registration-day may not be multistretched. These conditions are checked while iterating through flight-pairs in a chronological order (for each registration separately) – hence to make sure that there are no two subsequent rotations multistretched, it is enough to always check that the preceding rotation is not multistretched.

At this step, it is possible to compute the number of multistretched rotations for the strategy.

The following example illustrates that one strategy might be consistent with multiple “realizations”.

*Example 4.3.* Consider a registration-day beginning as  $A - B - C$ . Assuming a pairs strategy containing both  $(B, C)$  and  $(A, B)$ , two possible realizations of this strategy are  $(-, +, -)$  and  $(-, -, +)$ .

In practice the realization which multistretches the earliest possible rotations is chosen because the rotations are passed in chronological order.

3. Last, one has to compute shortages for the strategy, to see if the strategy is admissible (i.e. for all destination, the shortage is less than  $\alpha$ ). To do this the rotations (with the attached information for which rotations multistretching takes place according to the strategy) are compared with the provided shortage vector. For every multistretched rotation, add the corresponding shortage amount (i.e. 1 for a boolean vector and a number between 0 and 1 for a numeric vector) to the shortage number for the destination of the rotation; afterwards divide the shortage number for each destination by the number of rotations to that destination to obtain the shortage rate. With a little bit of bookkeeping, one can compute the overall shortage rate as well.

In case of the boolean shortage vector, one has to be somewhat more careful because of possibly missing tanking records. This will not be discussed in detail.

### 4.1.3 Data quality

This section describes several problems with data that have been encountered. The flight data usually contains all required information (in particular, if the coupling to flight-pairs is done by the new method, as explained in Section 4.1.1). Canceled flights are ignored. Sometimes, not just one rotation is made, but a multi-leg flight containing three destinations. Such flights do not enter the final dataset either.

With the tanking data, more problems have occurred:

- Missing tanking data: This usually happens when rotations are multistretched. However, particularly in the past (before 2011), the data was missing for other reasons as well – some of it because of problems with the planning system and other information systems.
- Outliers in tanking data: It sometimes happens that 0 liters is tanked – this is mostly correct, as discussed in Section 1.4. It sometimes happens that the full tank is filled – this is mostly because the airplane is refilled after it was drained in the evening. We note that for small airplanes, this is not done often; it must be done, however, when it is freezing during the night. Sometimes, it is shown that more than the full tank is filled. This was investigated and it seems that the reason for this is mostly non-working (or badly working) water meter. Such a problem is usually solved by Aqua Services within a few days.
- Rounded data: This is extensively discussed in Sections 1.4, 3.1.2 (in Correction for rounding) and 4.4.

For the boolean shortage vector, these issues cause a lot of troubles. For the numeric shortage vector – when tanking data are not used directly, but only to create an estimate of water shortage distribution, this is not so bad. There are two cases:

- When it is shown that the full tank is filled, the data is left as it is, even though mostly this corresponds to refilling after draining, i.e. not to actual water usage during the previous rotation. For the normal model, a test was made estimating the parameters both with and without the observations. The result is a slightly overestimated variance in case of including the data – but not to an extent that this would matter. Hence no further action was taken.
- In case that more than the full tank is filled, the tanking record is considered as missing. So we are in the same situation as with missing data. Because of simplicity, the missing data are ignored when it comes to the distribution estimation. Note that in general, this leads to a bias in estimation; but not if the “missingness” is unrelated to other variables in question (see [10, Section 2.4]) – note that this might be plausible since missingness is presumably mostly either a result of a problem with an information system or with a measuring device. In principle, this assumption is testable (see [10, Section 1.9]). We have been boldly assuming that the missing data is not a problem in our setting; but see [10] for possible remedies should this not be a case.

## 4.2 Faster evaluation

In the algorithms to determine an optimal multistretch strategy, described in Chapter 2, by far the most time is spent on evaluation (and in particular the step 2 of it), as described in Section 4.1.2. For a typical period and data for either F70 or E90, one such evaluation may take between 10 seconds and one minute. Since the algorithm may need to perform thousands or even tens of thousands evaluations, any improvement that can reduce the time reduces the duration of the whole algorithm greatly.

We have come up with two such improvements.

### 4.2.1 Using precomputation of registration-days

In the step 2 of evaluation, described in Section 4.1.2, one passes through all rotations. For every rotation, it must be checked whether the previous rotation is multistretched, but furthermore also whether the rotation is starting a new registration-day – in which case it may not be multistretched. Determining a new registration-day consists of checking whether the previous rotation in the dataset belongs to the same registration and also whether the night stop is in Amsterdam – in which case a new registration-day begins; otherwise, when the night stop is elsewhere, we proceed with the same registration-day. In particular this last condition takes considerable amount of time to check: it must be found out whether the inbound arrival and the outbound departure are on the same day or not – this requires comparing two dates. These computations are the same during each evaluation; hence it is easy to determine this (starts of the registration-days) in advance and then it is every time only necessary to check, apart from the multistretching of the previous rotation whether a rotation marks a start of a registration-day or not; this is computationally considerably cheaper.

This improvement can make the evaluation faster by a factor 1.5 or 2.

### 4.2.2 Using similarity of states

This improvement is somewhat more sophisticated; it applies only to pairs strategies. The idea is: the original evaluation method (from Section 4.1.2) considers all flight-pairs in the

dataset. However, when moving from one pairs state to another, only few flight-pairs are actually involved: in step 1 of the evaluation, if considering an  $\text{ADD}(p)$  or  $\text{DROP}(p)$  move, only those for which the inbound and outbound flights correspond to  $p$ ; for a  $\text{SWAP}(p_1, p_2)$  either those corresponding to either  $p_1$  or  $p_2$ ; in step 2, the whole registration-day containing a flight-pair involved in step 1 can actually change. Still, all these registration-days make up generally much less than the whole flight-pairs dataset, which we would consider otherwise.

The modified procedure for step 2 is therefore:

- a) Identify the flight-pairs for which the desired multistretching action (i.e. whether to multistretch or not; from step 1) changes by the considered move.
- b) Compute the registration-days containing those flight-pairs.
- c) Recompute the actual multistretching actions only for flight-pairs contained in those registration-days.

This improvement can make the evaluation faster by factor 2 to 10. In general, the larger the dataset, the larger the improvement; and the shorter an average registration-day, the larger the improvement.

*Example 4.4.* To demonstrate the above procedure, assume that we have just one registration and there are registration days  $\delta_1, \dots, \delta_n$ . Each of the registration days is composed of a series of flight-pairs. So for  $\delta_i$ , we may write the series as

$$(d_1^{(i)}, d_2^{(i)}), (d_2^{(i)}, d_3^{(i)}), (d_{m_i-1}^{(i)}, d_{m_i}^{(i)}) \quad (4.1)$$

where  $m_i$  is the number of rotations during the registration-day. After returning from  $d_{m_i}^{(i)}$ , the registration returns to Amsterdam and it has a nightstop there. Note that the pairs are overlapping, i.e. the outbound destination of one flight-pair is the inbound destination of the following flight-pair.

Now assume that we have a strategy  $s_0$  for which we have computed which flight-pairs would be multistretched and now we are interested in determining the multistretched flight-pairs for the strategy  $s_1$  which differs from  $s_0$  only by addition of the city-pair  $(d^*, d^{**})$ .<sup>6</sup> Note that determining the multistretched flight-pairs corresponds to the step 2 in Section 4.1.2 which is the step of the evaluation procedure that is the most computationally expensive.

With the original method, we would need to recompute all registration-days:  $\delta_1, \dots, \delta_n$ . However, the improvement consists of recomputing only those  $\delta_i$  such that the city-pair  $(d^*, d^{**})$  is contained in the series (4.1). Indeed, if this is not the case, the multistretched flight-pairs in the registration-day cannot change and there is no need to recompute the registration-day.

### 4.3 Multistretching more rotations in a row

In the Preliminary study and during most of the project, it was assumed that multistretching two or more rotations in a row is not possible – presumably, the risk of water shortage would be already too high then. However, a short inspection of the data reveals that multistretching two rotations in a row might quite often work without suffering a shortage.<sup>7</sup> This Section

<sup>6</sup>The case of SWAP and DROP moves is solved similarly.

<sup>7</sup>For completeness, three in a row is apparently already too much.

investigates whether this is really the case and also suggests an algorithm to take advantage of this (even though the algorithm has not been fully implemented in practice). In the whole section, we work with the data from winter 2010.

### 4.3.1 Maximal number of multistretches

At the beginning, let us consider the ideal situation when we can multistretch all rotations without suffering shortages. What is the maximal possible percentage of multistretched rotations, for a given flight-pairs dataset, if we are restricted to at most one or at most two multistretched rotations in a row? We still respect the condition that the first rotation of a registration-day cannot be multistretched. Note that we can determine the number of multistretched rotations for each registration-day separately since multistretching during one registration-day does not influence multistretching on a different registration-day.

Let  $\rho$  be the number of rotations for a given registration-day. Let us ignore the feasibility issues (i.e. shortages) and consider both cases:

- At most one multistretch in a row: We can multistretch  $\lfloor \frac{\rho}{2} \rfloor$  of the rotations belonging to the registration-day. This corresponds to multistretching every second rotation, taking into account that the very first one is not multistretched.
- At most two multistretches in a row: Then we do not multistretch the first one, then multistretch two, then tank one, multistretch two... The function  $h$  corresponding to this is

$$h(\rho) = 2 \cdot \lfloor \frac{(\rho - 1)^+}{3} \rfloor + [(\rho - 1)^+ \pmod 3].$$

Now using the data, we may determine how many registration days there are for each number of rotations  $\rho$ .<sup>8</sup> Using the above functions determining the maximum of multistretched rotations for a registration-day, one finds out that the maximum is almost 50% for both F70 and E90 in case of one multistretch in a row. In case of two multistretches, the maximum is around 64% for both F70 and E90. To put this in a perspective, consider that there are 9462 F70 rotations and 5945 E90 rotations during winter 2010.

### 4.3.2 Implementation ideas

As has become clear in the previous section, there might be a lot to gain by allowing multistretching two rotations in a row. How to test if this is really a case – and if so, how to exploit it? In view of the Tabu search algorithm for pairs from Section 2.3, it could be possible to implement a similar algorithm for pairs *and* triples. An obvious disadvantage is: while there are a lot of city-pairs, there are even more city-triples – if  $n$  is the number of destinations, we may expect  $\binom{n}{2}$  pairs and  $\binom{n}{3}$  triples (afterwards, there is some pruning of the pairs and triples which are observed not frequently enough, but still). Hence the huge combinatorial problem that we have had for just pairs becomes now even more formidable.

A different approach might be reusing what we can already solve satisfactorily: we can find good pairs strategies using the Tabu search algorithm for pairs. A natural idea is then to first find a good pairs strategy and then extend this by allowing multistretching of several triples. In this way, we restrict the state space considerably, possibly losing many good solutions, but

---

<sup>8</sup>For curiosity, a registration-day has on average 12.22 rotations for F70 and 9.38 for E90. The longest registration-day has 61 rotations for F70 and 57 for E90.

on the other hand making the problem much more tractable. Note that now not all triples have to be considered – it is reasonable to take a triple (say  $(A, B, C)$ ) into account only if the pairs strategy contains all three of the pairs  $(A, B), (B, C), (A, C)$ ; indeed if it is “too risky” to multistretch just the pair, it is certainly so if we add one more destination to it.<sup>9</sup>

After this trimming, the good pairs strategy (found by the Tabu search algorithm) for E90, which multistretches originally 84 city pairs, allows 364 city triples on top of it. For the F70 strategy, there are 253 pairs and from this, 2161 triples can be made. Hence a further reduction in the number of triples is certainly desirable.

One possible reduction of this number is to reject triples which are not observed often enough, e.g. more than five times per period. Another possible way to cut on the number of triples is to then consider only the triples such that the mean or some suitable quantile of the sum is less than the tank capacity (for multistretched data, the quantile would be derived from the estimated distribution) – we do not want to multistretch triples such that a too large proportion of their rotations arrives without water. Assume for simplicity that the true distribution is normal. Let the parameters of the water consumption distributions of  $A, B, C$  be  $\mu_1, \mu_2, \mu_3$  for means and  $\sigma_1, \sigma_2, \sigma_3$  for standard deviations. Then the distribution of  $X_A + X_B + X_C$  is  $\mathcal{N}(\mu_1 + \mu_2 + \mu_3, \sigma_1^2 + \sigma_2^2 + \sigma_3^2)$  by the properties of the normal distribution. For  $Z \sim \mathcal{N}(\mu, \sigma^2)$ , we have (for  $\gamma \in \mathbb{R}$ ) that  $\mathbb{P}\{Z < \mu + \gamma\sigma\} = \Phi_N(\gamma)$ , where  $\Phi_N$  is the cdf of the standard normal distribution. Applying this for  $Z := X_A + X_B + X_C$ , a possible rule of thumb is to accept only the triples such that (for some  $\gamma > 0$ )

$$\mu_1 + \mu_2 + \mu_3 + \gamma\sqrt{\sigma_1^2 + \sigma_2^2 + \sigma_3^2} < \text{capacity}. \quad (4.2)$$

Then taking for instance for  $\gamma = 1$  and recalling that  $\Phi_N(1) \approx 0.84$ , we have that more than cca 84% of “rotation-triples” for the triple  $A, B, C$  can make it without shortage if we multistretch it.<sup>10</sup>

Using (4.2) with  $\gamma = 1$  for E90 reduces the 364 triples to zero. So the problem is “solved” in this case. Since F70 has lower water consumption, one expects more admissible triples. Indeed, after applying (4.2) and furthermore considering only triples with at least 10 observations, there still remain 27 triples.

Now one can construct a list of triples in a similar way as in the Tabu search algorithm for pairs. Another option is just to try adding triples one by one, starting from the one with most observations, and see if some improvements can be realized – i.e. it is not strived for the best solution, a modest improvement on top of the original pairs strategy would be considered enough. Note that the original pairs strategy is considered fixed; we only extend it.

*Remark* (Evaluation with pairs and triples). As a final note, the strategy consisting of both pairs and triples will be even more expensive to evaluate than the one with only pairs. A special evaluation procedure has been already implemented, even though the full search algorithm not. We briefly give some main differences of the new evaluation procedure as opposed to the one for pairs described in Section 4.1.2:

0. At the beginning, two shortage vectors must be provided: one gives, for every flight-pair, the chance of shortage if we multistretch that flight-pair. The second one gives,

---

<sup>9</sup>That said, one can of course come up with datasets and strategies such that the triple is multistretchable, but none of the three pairs is. Still, the preceding discussion indicates why we presumably – for an “average” dataset – do not lose much by excluding such strategies.

<sup>10</sup>The author feels that if only less than 84% rotation-triples should make it with enough water, then it is not worth risking it.

for every flight-pair, the chance of shortage if we multistretch that flight-pair as well as the preceding one (i.e. we “multistretch a triple”).

1. To formulate the strategy in terms of the dataset, we first indicate (as before) which flight-pairs we would like to multistretch if the preceding flight-pair is not multistretched, and afterwards, which we would like to multistretch even if the preceding flight-pair is multistretched
2. We apply the restrictions. As before, if a flight-pair is to be multistretched, it must not be the first one in a registration-day. If the preceding flight-pair is not multistretched, then it is possible to multistretch the flight-pair. However, in this case, if the preceding flight-pair *is* multistretched, then it is still possible to multistretch the current flight-pair, if the strategy lets us multistretch the relevant triple. That is, unless the flight-pair preceding flight-pair preceding the current pair is also multistretched, i.e. unless we multistretch already the preceding triple of flight-pairs.
3. When computing shortages for a destination  $d$  under the strategy, we again consider every multistretched rotation to  $d$ . We add the corresponding shortage amount from the standard shortage vector if the preceding rotation *is not* multistretched and we add the amount from the shortage vector for triples if the preceding rotation is also multistretched.<sup>11</sup>

## 4.4 Heaping

In this whole section, only non-multistretched data is considered. *Heaping* is how rounding is called in the literature; a *heap* is then a value to which observations may be rounded; for instance in our case, a multiple of 5.

In Paragraph Correction for rounding in Section 3.1.2, two different rounding patterns that occur in our tanking data were observed. The first one (rounding to integers) was assumed to be deterministic and its effect was built into the proposed probability models – in the delta-gamma distribution, by default, and for the normal distribution optionally.

The question arises how to handle the other type of rounding (i.e. to multiples of five). The inspiration for us was the method of modeling this rounding described in [32]. Their idea is to estimate the probabilities of rounding and using these to “correct the sample”, therefore obtaining a sample presumably similar to the original, unrounded, one. Then all statistical inference is based on this corrected sample which is presumably already free of rounding errors. The whole procedure is done several times and the results averaged, in order to counter the error caused by correcting. To summarize, the steps proposed in [32] are:

1. Rounding estimation phase: Formulate a model with rounding present and estimate parameters of both the rounding model and the underlying distribution model by MLE
2. Imputation phase (this together with the inference phase should be repeated sufficiently many times):
  - (a) For data records in the heaps determine if they have been rounded

---

<sup>11</sup>Without going into details, we note that here a correction must be applied to account for the case that already the preceding rotation suffers shortage – then we tank at the outstation, so the current rotation probably does not suffer shortage.

(b) If yes, input a new value for them using the rounding model

3. Inference phase: Make inference from the imputed samples

4. Average the results

*Example 4.5* (Making inference). What may “make inference” actually mean? We give two examples:

- In [32], data of unemployment durations from the Dutch Labour Force Survey are presented. We are interested in the distribution of unemployment durations, given for instance by a histogram. The data available exhibits distinct peaks for durations which are multiples of six months – in the article, it is argued that these are due to rounding. Using the method above, a new histogram is obtained where no peaks are discernible.

Making inference hence just means producing a histogram; in the end the histograms from different runs are averaged.

- In our case, we assume a parametric model for the tanking data (given an aircraft type and a destination) and we want to estimate the parameter.

Making inference hence means estimating the parameters of the model (and the different estimates are averaged in the end).

After the overview, let us consider the steps in more detail. The theory is inspired by [32], even though it has been slightly modified to our situation.

### Rounding estimation phase

It is assumed that there is a true underlying distribution from which the data records are drawn, corresponding to the random variable  $Z$ . We assume that this distribution is itself subject to imprecise measurements (as discussed in Paragraph Correction for rounding in Section 3.1.2) and hence takes values in a discrete set  $S$ , having a probability mass function  $f(\cdot, \theta)$  from a parametric family indexed by  $\theta \in \Theta$ .<sup>12</sup> Assume that there exists a set  $\{h_1, \dots, h_l\}$  of values (*heaps*) to which observations can be rounded. For each  $h_i$ , and every  $s \in S$ , consider the value  $r_{h_i, s}$  which denotes the probability that the true value was  $s$ , but the reported value is  $h_i$ . Let the random function  $\rho : S \rightarrow S$  represent the rounding process, hence

$$\rho(s) = \begin{cases} h_1, & \text{w.p. } r_{h_1, s} \\ \vdots, & \vdots \\ h_l, & \text{w.p. } r_{h_l, s} \\ s, & \text{w.p. } 1 - \sum_{i=1}^l r_{h_i, s} \end{cases} \quad (4.3)$$

Hence we have observations  $\tilde{z}_1, \dots, \tilde{z}_n$  drawn from the random variable  $\rho \circ Z$  and we are interested in estimating the rounding probabilities  $r_{h_i, s}$  (denote by  $\mathbf{r}$  the vector of rounding probabilities) while necessarily estimating the parameter  $\theta$  as well. This may be done using the MLE approach described in Section 3.1.2. We just note that the density associated with  $\rho \circ Z$  is

$$g(z; \theta, \mathbf{r}) = f(z; \theta) \cdot \left(1 - \sum_{i=1}^m r_{h_i, z}\right) + \sum_{z' \in S} f(z'; \theta) r_{z, z'} \quad (4.4)$$

<sup>12</sup>The theory would, with a few adjustments, also apply if the model assumes a continuous distribution.

where  $r_{h,s}$  is considered to be zero if  $h \notin \{h_1, \dots, h_l\}$ .

### Imputation phase

Now, to recover the random variable  $Z$ , we must counteract the action of the rounding function  $\rho$ . Note that after the estimation in the first phase, we know all parameters specifying  $\rho$ .

To “correct” the data, we must simulate “unheaping”, i.e. for each heap  $h$ , we determine the proportion of its values which are registered as  $h$  only because they were heaped towards it from some other value  $s$ . For each observation of  $h$ , its original (unheaped) value is<sup>13</sup>

$$\begin{cases} h, & \text{w.p. } \frac{f(h;\theta) \cdot (1 - \sum_{1 \leq i \leq m} r_{h_i, h})}{f(h;\theta) \cdot (1 - \sum_{\substack{1 \leq i \leq m \\ h_i \neq h}} r_{h_i, h}) + \sum_{z' \in S} f(z';\theta) r_{h, z'}} \\ z, & \text{w.p. } \frac{f(z;\theta) r_{h, z}}{f(h;\theta) \cdot (1 - \sum_{1 \leq i \leq m} r_{h_i, h}) + \sum_{z' \in S} f(z';\theta) r_{h, z'}}; \quad \text{for all } z \in S. \end{cases}$$

Hence, for all heaps  $h$ , all observations on the heap have a (possibly) new value assigned to them according to the above probabilities. This results in a new sample which presumably corresponds to the drawing from  $Z$ . Note however the uncertainty present in this unheaping process – this is the reason why the imputation and the subsequent inference is repeated several times.

### Implementation

We consider our  $Z$  to be the water consumption for a fixed aircraft type and destination. We assume that the true distribution is the delta-gamma distribution introduced in Section 3.1.4. Based on the data, we assume that the heaps are multiples of 5. We further assume that rounding to multiples of 10 is different (there is more rounding towards them) from rounding to multiples of 5 which are not multiples of 10. Furthermore, a major assumption which we make and which reduces the number of parameters in the model considerably is that all multiples of 5 (and not 10) are equivalent as well as all multiples of 10. A further assumption is that the rounding pattern is symmetric (i.e. rounding from 49 to 50 is as probable as rounding from 51 to 50).

Considering a heap  $h$  being a multiple of 5 (and not 10), the values which can be heaped towards it are  $h - 2, h - 1, h + 1, h + 2$  (this is based on intuition). For a heap  $h$  being a multiple of 10, the values which can be heaped towards it are  $h - 5, h - 4, h - 3, h - 2, h - 1, h + 1, h + 2, h + 3, h + 4, h + 5$  (and if  $h = 0$ , then only the positive values count).

Using the above assumptions, it is enough to consider the heaping probabilities  $r_{5,1}, r_{5,2}, r_{10,1}, r_{10,2}, r_{10,3}, r_{10,4}, r_{10,5}$ . Here for instance  $r_{10,4}$  represents, for all  $h$  multiples of 10 both  $r_{h, h+4}$  and  $r_{h, h-4}$  in the notation of Equation (4.3).<sup>14</sup>

Hence there are in total nine parameters to be estimated for the distribution with the density  $g$  in (4.4). Those are three parameters corresponding to the delta-gamma distribution and 7 heaping probabilities  $r_{k,i}$ . Further conditions on  $r_{k,i}$  which must be imposed, so that  $r_{k,i}$  correspond to probabilities, are

- $0 \leq r_{k,i} \leq 1$ : clear

<sup>13</sup>Letting  $r_{h,h} \equiv 0$ .

<sup>14</sup>The exception being  $h = 0$ .

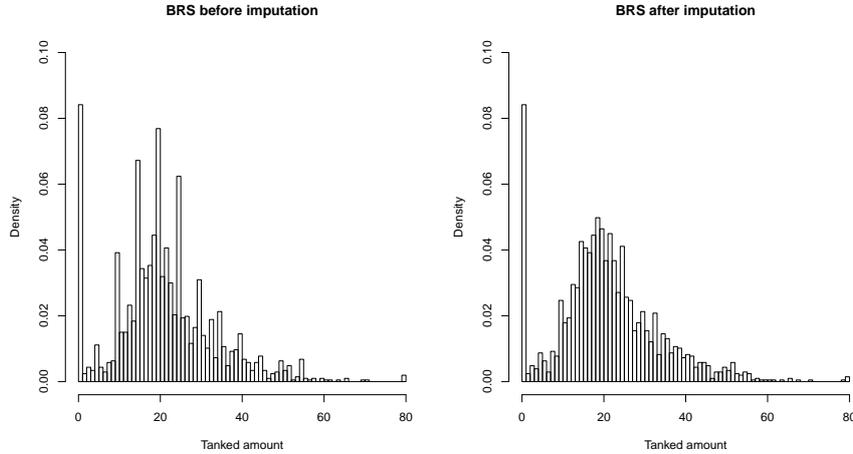


Figure 4.1: Bristol for F70 after “unheaping”

- $r_{10,3} + r_{5,2} \leq 1$  and  $r_{10,4} + r_{5,1} \leq 1$  and  $2 \cdot r_{10,5} \leq 1$ : to bound the proportion of heaped value to 1 if heaping is possible towards more than one heap.

Actually, some upper bounds in the first condition can be then dropped.

In the Rounding estimation phase, the above modification was applied and the corresponding log-likelihood optimized. This leads then to the parameters for the delta-gamma distribution, but also to the rounding parameters. One typical set of estimated rounding parameters is for Cardiff (F70):

$$\begin{aligned}
 r_{5,1} &= 0.2338844, & r_{5,2} &= 0.1243999 \\
 r_{10,1} &= 0.2371811, & r_{10,2} &= 0.1578690, & r_{10,3} &= 0.05728118, & r_{10,4} &= 0.06516210, \\
 r_{10,5} &= 2.876090 \cdot 10^{-9}.
 \end{aligned}$$

That said, not always are the values so intuitively acceptable; often,  $r_{10,5}$  is disproportionately large which is presumably a “compensation” for misspecified rounding model.

The figures after imputations seem very credible for some destinations, particularly some of the largest ones. For an example, consider Figure 4.1 which shows a histogram for Bristol before and after an imputation.

Our main goal was however estimating the parameters for the delta-gamma distribution – to this end, we ran the imputation process ten times, every time estimating a new set of parameters and then averaging them. The resulting parameters obtained are almost the same as the ones obtained when estimating the parameters directly from the heaped data. Therefore, we conclude that in our case the fact that the observations are heaped does not lead to the considerable bias in the estimation of the underlying distribution.

*Remark.*

- As a matter of interest, consider Figure 4.2. As can be seen, the underlying delta-gamma distribution together with the heaping mechanism can “simulate” data very similar to the actually measured one.

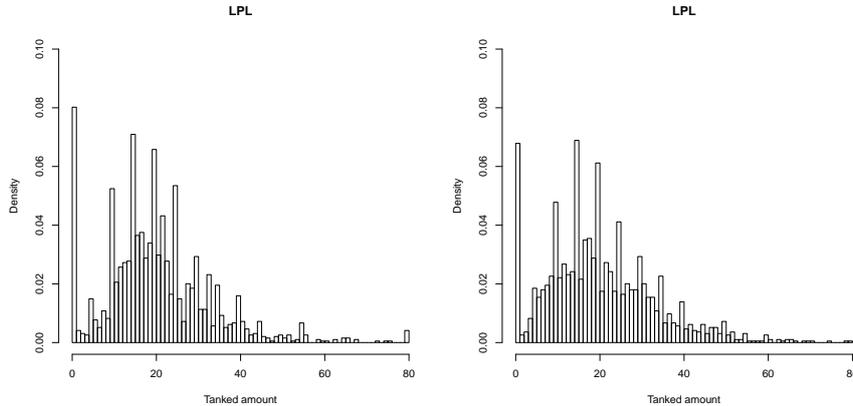


Figure 4.2: Liverpool for F70 – one of the plots is the true tanking distribution, the second one is drawn using delta-gamma with the heaping correction – which one is which?

- As a side note, from having a look at the figures, it almost seems that for some destinations, the non-zero observations are actually formed from a mixture of two distributions – this could actually be the day flights and the early/late flights.
- Note that one obtains yet another estimate of the underlying distribution parameters in the Rounding estimation phase as a byproduct. In [32], a simulation study was run to compare this estimate with the one obtained after the multiple imputation. A notable advantage of the multiple imputation estimate was insensitivity to misspecification of the underlying distribution. Therefore, the multiple imputation approach is recommended.<sup>15</sup>

## 4.5 Deconvolution of empirical distributions

The content of this section is logically connected to Section 3.1, but was deemed not relevant enough to be placed there.

In the Preliminary study, the distribution of water tanking was also estimated nonparametrically using the empirical distribution function. This distribution was used to compute virtual shortages (see Definition 2.1) which were in turn used to (manually) construct strategies, i.e. to multistretch cities with low enough shortages. A question arose if this could not be somehow done in case of multistretching as well instead of the MLE approach. The question therefore is: is it possible to extract approximation to empirical distribution functions for  $X_i$  (which we would have in the case of non-multistretched data) also from the multistretched data?

How could this be done? From the observations  $Y_{i,j}$ , we can get empirical distribution functions  $\hat{G}_{i,j}$  for  $i, j \in \text{DST}$ . From this, we can get a density (estimate)  $\hat{g}_{i,j}$  (either by just taking a histogram; or by using a kernel density estimation method – see the remark below). As will be explained later, for most city-pairs  $(i, j)$  this edf is unsuitable for a direct use.

<sup>15</sup>For some applications – for example if each observation corresponds to a person which has also other variables associated to her and we want to compute regression based on these variables, the multiple imputation approach is actually the only option

However (as was done before by the parametric methods), we hope to use this data in order to extract individual distributions of  $X_i$ . We have that  $g_{i,j}$  is the convolution of  $f_i$  and  $f_j$ , the densities of  $X_i$  and  $X_j$  respectively (assuming independence of  $X_i$  and  $X_j$ ). Hence we expect  $\hat{g}_{i,j} \approx f_i * f_j$ . So our problem is to extract functions  $f_i$  from the system of “equations”

$$f_i * f_j \approx \hat{g}_{i,j}, \quad i, j \in \text{DST}. \quad (4.5)$$

*Remark* (Kernel density estimation). Assume that we are given an i.i.d. sample  $z_1, \dots, z_n$  drawn from a distribution with probability density  $f$ . The kernel density estimation is a method to nonparametrically (i.e. making as few distributional assumptions as possible) estimate  $f$ . As described in [35, Chapters 4 and 6], we define the kernel density estimator of  $f$  as

$$\hat{f}(z) := \frac{1}{n} \sum_{i=1}^n \frac{1}{h} K\left(\frac{z - z_i}{h}\right),$$

where a number  $h > 0$  and a function  $K$  must be given. The function  $K$  is referred to as the *kernel*. The kernel is assumed to satisfy the following conditions:

k.1)  $K$  is sufficiently differentiable

$$\text{k.2) } \int_{-\infty}^{\infty} K(z) dz = 1$$

$$\text{k.3) } \int_{-\infty}^{\infty} zK(z) dz = 0$$

$$\text{k.4) } \int_{-\infty}^{\infty} z^2 K(z) dz > 0.$$

An often used kernel is the *Gaussian kernel*:

$$K_{\text{Gauss}}(z) := \frac{1}{\sqrt{2\pi}} e^{-z^2/2}.$$

As a side note: this kernel is used to produce the red function in Figure 3.2.

To appreciate the complexity of the problem, let us start with a simpler case. Assume that we have a function  $f = g * h$  and we want to estimate  $g$  and  $h$ .<sup>16</sup> However, there can be in general many different solutions.

*Example 4.6.* For a very simple example, consider just discrete convolution and  $f(0) = 1$  and otherwise 0. Then for  $k \in \mathbb{Z}_+$ , one can define the function  $g_k$  by  $g_k(k) := 1$  and otherwise 0 and the function  $h_k$  by  $h_k(-k) := 1$  and otherwise 0. Then for any  $k$ , one has

$$f = g_k * h_k.$$

In [21, Chapter 2], it is assumed that  $g$  corresponds to the unknown “true density” and  $h$  is an error density. It is mostly assumed that  $h$  is known, however in [21, Section 2.6] also the case with only partially specified  $h$  is considered. It is however remarked that the major issue is the possible existence of many different solutions, as demonstrated by our example above. Still, if  $h$  is restricted to a small enough family of functions, the problem can be solved. A

---

<sup>16</sup>Assume all of these are densities – we are interested in the distribution of  $X_i$ , so it does not matter if we obtain it as a cdf or as a density.

case discussed is: assume that  $g$  comes from a suitable function space and  $h$  is known to lie in the function family (for some fixed  $\mu_0$  and  $\sigma_0 > 0$ )

$$\{\phi_{\mathcal{N}}(z; \mu_0, \sigma), 0 < \sigma^2 \leq \sigma_0^2\}$$

where  $\phi_{\mathcal{N}}(z; \mu, \sigma)$  is the normal density with mean  $\mu$  and standard deviation  $\sigma$ .

Unfortunately, the case just described as well as the other situations mentioned in [21, Section 2.6] impose either too restrictive assumptions (indeed – for instance in order to use the above case, we would have to impose the distributional assumption of normality with a fixed mean on the densities of  $X_i$ !) or seem to be outright inapplicable.

The previous negative result should not be completely discouraging since we do not have just one convolution equation, but a system of them. For an analogy, one linear equation with two variables is not uniquely solvable, but if there are two equations, it may be. Assume now for simplicity that we consider only discrete densities (in the same way as our data are measured, i.e. supported on integers between 0 and TC), so we may represent  $f_i$  as a vector  $f_i = (f_i(0), \dots, f_i(\text{TC}))$ . Then the convolutions in (4.5) can be written out in terms of the components and the system looks then like

$$\hat{g}_{i,j}(k) = \sum_{l=0}^{\text{TC}} f_i(l) \cdot f_j(k-l), \quad 0 \leq k \leq \text{TC}, \quad i, j \in \text{DST}.$$

and furthermore a constraint must be posed that all variables are positive; here each component of each function corresponds to one variable. Assuming there are 30 destinations and the tank capacity is 80 liters, this leaves us with some 36000 equations and 2400 variables; with furthermore a linear constraint on variables. While this would could be probably handled for a linear problem, a further trouble is that the (vector) function on the right hand side is quadratic.

To conclude, estimating the density  $f_i$  of  $X_i$  in a nonparametric fashion from (4.5) seems to pose formidable challenges.

#### 4.5.1 Recommendation regarding the use of the empirical distribution function

One may wonder if the use<sup>17</sup> of the empirical distribution function  $\hat{F}$  in the Preliminary study to obtain virtual shortages may be justified.

Among properties of (general) empirical distribution functions, [35, (2.5)] states the Dvoretzky-Kiefer-Wolfowitz (DKW) inequality

$$\mathbb{P}(\sup_x |F(x) - \hat{F}(x)| > \epsilon) \leq 2e^{-2n\epsilon^2} \quad (4.6)$$

if  $\hat{F}$  corresponds to a sample of  $n$  observations and  $F$  is the true distribution function for the distribution of the sample. Note that [35] mentions more powerful bounds, but does not cite them.

Assume that the data is already rounded, hence our observations are integer-valued. Denote by  $F_i$  the true distribution function, by  $f_i$  the true probability mass function and by  $\hat{F}_i$  the

<sup>17</sup>Note that the notation introduced here is at odds with the one used in Section 2.1.4. Namely the subscript here now denotes a destination.

empirical distribution function for  $i \in \text{DST}$  with  $\hat{f}_i$  the corresponding histogram. Note that<sup>18</sup>  $v_{i,j} = 1 - [f_i * F_j](\text{TC})$  and hence it is enough determine the distance between  $[f_i * F_j](\text{TC})$  and its estimate (as it was used during the Preliminary study)  $[\hat{f}_i * \hat{F}_j](\text{TC})$ . Furthermore, under the assumption that the observations are integer-valued,  $f_i(k) = F_i(k) - F_i(k-1)$  and  $\hat{f}_i(k) = \hat{F}_i(k) - \hat{F}_i(k-1)$ . Note also that all the considered functions take values between 0 and 1. Hence it is possible to compute

$$\begin{aligned}
|[f_i * F_j](\text{TC}) - [\hat{f}_i * \hat{F}_j](\text{TC})| &\leq |[f_i * F_j](\text{TC}) - [f_i * \hat{F}_j](\text{TC})| + |[f_i * \hat{F}_j](\text{TC}) - [\hat{f}_i * \hat{F}_j](\text{TC})| \\
&= |[f_i * (F_j - \hat{F}_j)](\text{TC})| + |[(f_i - \hat{f}_i) * \hat{F}_j](\text{TC})| \\
&\stackrel{(*)}{\leq} \sup_k |F_j - \hat{F}_j|(k) + 2(\text{TC} + 1) \cdot \sup_k |f_i - \hat{f}_i|(k) \\
&\stackrel{(DKW)}{\leq} (2\text{TC} + 3) \cdot \epsilon.
\end{aligned} \tag{4.7}$$

Take  $n = \min(n_i, n_j)$  where  $n_i, n_j$  denote the number of observations for  $X_i, X_j$  respectively and consider the equation (4.6) for  $X_i$  and  $X_j$ . Assume that we want the error in the virtual shortage at most 0.01. Then we must take in (4.7) the value of  $\epsilon$  at least  $\frac{1}{2\text{TC}+3} \geq 4 \cdot 10^{-3}$ . Assume that in (4.6) the probability smaller than 0.05 is required. This then requires taking  $n \approx 10000$  which is much more than we have (the largest destinations, if the data for the whole history is taken, have at most 3000 observations). Hence using the available estimates, we are not able to derive useful upper bounds on error when using the empirical distribution function to approximate virtual shortages, as it was done in the Preliminary study. We therefore recommend to use it with caution.

*Remark.*

- It is likely that the estimate (\*) in (4.7) is too conservative.<sup>19</sup> Assume for a while that we can take  $\epsilon = \frac{1}{2}$  (this would correspond to the case that the second term in (\*) can be estimated as sharply as the first one). Then it would be enough to take  $n \geq 15$  which seems on the other hand too good to be true.

As can be seen, usefulness of the above analysis is therefore directly connected to how sharp estimate can we make in the inequality (\*) in (4.7).

- As an afterthought, we would guess that a parametric estimation is more robust than the nonparametric one (presented in this section) with respect to the rounding errors.

## 4.6 Multistretching Boeing 737

After starting multistretching with KLC airplanes, it was desirable to try out multistretching also for Boeing 737. It is a larger airplane which usually flies longer flights. However, one can again obtain a good strategy using one of the algorithms in Chapter 2. Still, there are some novelties; as a matter of interest:

<sup>18</sup>Assuming the independence of  $X_i$  and  $X_j$ .

<sup>19</sup>The author would expect that there exists, under some conditions, a better estimate on  $|f_i - \hat{f}_i|$  than the one coming from  $f_i(k) = F_i(k) - F_i(k-1)$  and the similar equation for  $\hat{f}_i$ .

- There are several different configurations of Boeing 737 – these differ for example in the maximal number of passengers on the board. Even though different configurations often fly to the same destinations, the water consumption patterns are very different. Hence it is necessary to treat these configurations as different aircraft types.
- Flights are regularly drained during a nightstop at Amsterdam (not only in the winter) – and therefore the whole tank is refilled in the morning. Hence, unlike for F70 and E90, it is not acceptable to just ignore this fact. Instead, the relevant records are discarded; it remains a question whether this is acceptable, i.e. whether the introduced estimation bias is not too high, in view of the finding in Section 3.3.5 that morning flights often tend to have lower water consumption.

# Chapter 5

## Conclusions

### 5.1 Recommendations

The following summarizes the most important practical recommendations stemming from the current research:

- Determine multistretch strategy for every new period based on the data for the corresponding period from the previous year.
- As a strategy, use lists of city-pairs (if possible) – a suitable list can be obtained using the Tabu search algorithm for pairs.
- To estimate water consumption under multistretching, use either the delta-gamma or normal distribution.

There are also some more far-reaching recommendations:

- Make sure that the Aqua operators report the true tanked amount value and do not round unnecessarily – this has been already communicated to Aqua Services and they expressed their intent to respect this recommendation.
- For E90, use the water tank with 110 L content, instead of the one with 90 L content – this should happen soon.
- Investigate why some flights are tanked twice before the departure and make sure that this does not occur.

### 5.2 Practical consequences

At the time of writing, multistretching F70 has already started. This is done using the one list strategy, with the lists being provided by the algorithm in 2.2. Multistretching of E90 is expected to start soon. The evaluation is currently under way as is the discussion about possible reduction of the Aqua Services workforce due to multistretching.

In the future, it is also desired to multistretch Boeing 737. Strategies to do this are already computed.

## 5.3 Future research

This section presents several ideas for a future research.

### 5.3.1 Various open problems

- How would the analysis change if the acceptable shortage rate  $\alpha$  might be different for different destinations? In particular, this rate could be based on the additional costs of tanking at the given outstation.

It was not possible to tackle the problem during the current project because the additional costs of tanking at outstations have not been determined yet.

- Flights with less passengers could be counted less heavily when it comes to the service level. Currently, our service level (per destination) is the percentage of rotations which do not suffer water shortage. What about taking this instead to be the percentage of passengers who do not suffer water shortage?
- Finish the algorithm to determine a strategy for slipping more rotations in a row.
- Simulate different changes in the timetable (introducing new flights, canceling old ones, shuffling the departure times. . .) to see how well our strategies cope with it. This could shed light on the assumption that timetables for the same periods in two subsequent years do not differ too much.
- Make more thorough analysis of the “rate of convergence” (i.e. how many iterations are used) of the Tabu search algorithms – either by an analytical argument or by thorough testing.
- Determine how large effect the cabin attendants have on the water consumption. Admittedly, getting data to tackle this question might be very challenging.
- Handle missing tanking data in a better way than by just leaving it out from the analysis.

### 5.3.2 Ab initio modeling of water consumption

The parametric models in Section 3.1.4 were chosen because of favorable properties and not by using deeper insight into the data-generating process. An ambitious project would be to model the distribution starting out from the water consumption process (i.e. requests for warm drinks, washing hands, flushing toilet). There are two major challenges:

1. The resulting distribution might not be compatible with the MLE approach: Most probably, if the distribution can be expressed via density, the distribution of a sum will not be expressible.<sup>1</sup> Hence a different estimation procedure would be necessary.
2. What are the factors behind the water consumption process? A natural idea is to model consumption per passenger (see for instance Sections 5 and 6 in [3]), however, as seen in Section 3.3.3, the dependence on the number of passengers is very weak. Basing the analysis on different data might be complicated because of the need to obtain this data.

---

<sup>1</sup>Note that the distributions used before have been selected very carefully so that it would be possible

### 5.3.3 Toilet service slipping

Toilet service means draining the waste tank after an arrival, cleaning it and filling it with deowater. Unstructured slipping of toilet service occurs, similarly as was the case with water service until recently. Hence a question is whether structural slipping of toilet service can be introduced.

In [5], a qualitative analysis was provided. In contrast to the water service, it was recommended *not to* multistretch toilet service. Several reasons were cited; we believe that the most challenging one is lack of data; namely there is no good measure available of “cleanliness” and the “need to drain” on which the decision which flights to multistretch could be computed.

A possible way to gather data would be by means of an experiment: multistretch a few “promising” rotations (for instance 100). Let cabin attendants decide after both first and the second rotation if the drainage is needed. Then one has a categorical regression model where explanatory variables could be the numbers of passengers, destination and the total time spent on the two rotations. To avoid problems, one would slip only when the total rotation time is short and (perhaps) when the number of passengers is small.

## 5.4 Acknowledgments

Thanks to my advisors Karma Dajani from Universiteit Utrecht and Matthieu Bolt and Adnan Fiaz from KLM Royal Dutch Airlines for supervision, vital suggestions and checking the manuscript.

Thanks to Jeroen Vegter for guiding me during my excursion at Aqua Services and enabling me to present my results at the Aqua Services bosdag. Thanks to the Aqua operator Frankie for letting me shadow him during his typical water tanking shift and for sharing his insights about water tanking with me.

Furthermore thanks to Aqua Services, KLM Cityhopper and K3 analysts for their involvement in the project. Thanks to my colleagues from the Decision Support department of KLM for helping me realize how the work of a Consultant Decision Support looks like and also for the wonderful atmosphere. Thanks to my fellow interns from the Board of Taxibaan (association of interns at KLM) for inspiring cooperation and the possibility to appreciate different aspects of KLM.

# Appendix A

## Useful information

### A.1 IATA airport codes

For convenience, we list here some of the IATA airport codes. All of these were encountered during the research (they are all either KLC or Boeing 737 destinations); in particular, all codes used in this thesis can be found in the list.

AAL Aalborg	ABZ Aberdeen	AMS Amsterdam	ARN Stockholm	BGO Bergen
BHX Birmingham	BLL Billund	BLQ Bologna	BOD Bordeaux	BRE Bremen
BRS Bristol	BRU Brussels	CGN Cologne-Bonn	CPH Copenhagen	CWL Cardiff
DUS Dusseldorf	FRA Frankfurt	GOT Göteborg	GVA Geneve	HAJ Hannover
HAM Hamburg	HEL Helsinki	HUY Humberside	KRS Kjevik	LBA Leeds
LGW London Gatwick	LHR London Heathrow	LIN Milan Linate	LPI Linköping	LPL Liverpool
LUX Luxembourg	MAD Madrid	MME Durham	MLA Milan Malapensa	MRS Marseille
MUC Munich	NCE Nice	NCL Newcastle	NUE Nuremberg	NWI Norwich
OSL Oslo	OTP Bucharest	PIK Glasgow	RTM Rotterdam	STR Stuttgart
SVG Stavanger	SVO Moscow	TLS Toulouse	TRD Trondheim	TRF Sandefjord
TXL Berlin	VCE Venice	VIE Vienna	WAW Warsaw	

## Appendix B

# Selected parts of the R code

### B.1 Tabu search

For a detailed description, see Section 2.2.

```
#tabuSearch
#####
#Purpose: Find a very good state (which in turn corresponds to a slipping policy)
#####
#Returns: The best state found, its evaluation and a tabu list
# (to possibly continue the search later).
#####
#Paramaters
#-outputDir, records: supplied from the main function.
#-waterTekort: generated by the searchStateSpace()
#-state: the initial state
#-tabuList: the initial tabu list
#-size: the (maximal) size of the tabu list. After attaining the size,
# start rewriting the oldest members of the tabu list.
#-alpha: the desired service level.
#-reloadLast: if TRUE, continues searching from the state of the system
# found in tabu.RData
#-saveResults: if TRUE, saves the current system state
# every saveFrequency iterations to tabu.RData,
# thus enabling continuing the search using reloadLast=TRUE later.
#-verbose: level of information given by the function during its run.
# 0 gives nothing, 1 is probably reasonable, 3 is only for debugging.
#-terminationCriterion: either "Iterations" (then stops after iter iterations)
# or "No improvement" (then stops after noImproveTermination iterations
# such that the best state hasn't changed)
# or (works if IncludeWholeHistory==TRUE) Evaluations (i.e. when the total number of
# evaluations exceeds maxEvals).
#-DROP2neighborhoods, EX2neighborhoods, ADD2neighborhoods,
# EXneighborhoods, ADDneighborhoods, DROPneighborhoods:
```

```

# number of neighbors of the given type generated every turn.
#-forceDROP: if FALSE, generated DROPneighborhoods only if there are
#         no other neighbors left after tabu filtering.
#-noChangeTurnsTillforceDROP: to counter the problem on the previous line.
#   Number of turns without improving the best state, after which
#   put automatically forceDROP = TRUE.
#-includeWholeHistory: if TRUE, the tabu list contains (as its size allows)
#   every state that we ever evaluated.
#   If FALSE, only states which were "active" are remembered
#   (and also the neighborhood from the very last iteration)
#immutableDST: Destination where we never change the state.
#   Useful to exclude the destinations with small number of flights
#   (otherwise there is a lot of neighbors which differ very little from the active state)
#####
#What is in the tabu list?
#-If includeWholeHistory=TRUE:
#tabuList contains all states that have ever been evaluated
#   (given that they fit in the tabuList size, otherwise the oldest are being replaced).
#-If includeWholeHistory=FALSE:
#tabuList contains only the states that have been active, at some point in time.
#In this case, there is also a second kind of "tabu list": lastNeighborhood.
#   This contains the checked neighborhood of (only) the last active state.
#   (if includeWholeHistory=TRUE, then this is included in the "standard" tabuList)
tabuSearch <- function(outputDir, records, waterTekort, state = NULL, tabuList = list(),
  size = ifelse(includeWholeHistory,1000,100),alpha=0.05,
  reloadLast = FALSE, saveResults = TRUE, saveFrequency = 10, verbose = 0,
  terminationCriterion="Iterations", noImproveTermination = 10,
  iter=100, maxEvals=1000, DROP2neighborhoods=0, EX2neighborhoods=0,
  ADD2neighborhoods=0, EXneighborhoods=10,
  ADDneighborhoods=2, DROPneighborhoods=4, forceDROP=FALSE,
  noChangeTurnsTillforceDROP= (noImproveTermination %/% 2),
  includeWholeHistory=FALSE, immutableDST=character(0)){

#add is true if adding an false if dropping
#type is either 1 (to switch either slipIn or slipOut) or 2 (to switch both)
draw.candidate <- function(state,add,DSTPossible,type){
  candidateState <- state

  if(type==1){
    if(candidateState[candidateState$DST==DSTPossible,"slipIn"] == add) {
      candidateState[candidateState$DST==DSTPossible,"slipOut"] <- add
    }else if(candidateState[candidateState$DST==DSTPossible,"slipOut"] == add ) {
      candidateState[candidateState$DST==DSTPossible,"slipIn"] <- add
    }else{ #I must toss!
      toss <- sample(c("FALSE", "TRUE"),1)
      if (toss) candidateState[candidateState$DST==DSTPossible,"slipIn"] <- add
      else candidateState[candidateState$DST==DSTPossible,"slipOut"] <- add
    }
  }
}

```

```

    }
  }else if(type==2){
    candidateState[candidateState$DST==DSTPossible,c("slipOut","slipIn")] <- add
  }

  return(candidateState)
}

#Assume that all states have names consistent with records
# (this should be handled in searchStateSpace())
if(verbose >= 1) print("Entering the tabu search procedure.")

#I can read results of the last tabu search from a file (and save it there again).
#I always assume that the state that I am given as the initial one is feasible.
if(reloadLast){
  #Start where the previous search ended. Assume that the file exists.
  load(file=paste(outputDir,"tabu.RData",sep=""))
  if(verbose >= 2) print("Reloading the last search results.")
}

}else{
  if(is.null(state)){
    #Start from the "slip nothing" state.
    state <- data.frame(DST=union(levels(records$DSTOut), levels(records$DSTIn)),
      slipIn = FALSE, slipOut = FALSE)
    lastTabu <- 1; #We have no tabu list yet.
    if(verbose >= 2) print("Starting with the 'slip nothing' state.")
  }else{
    #Take the suggested state, take the suggested tabuList.
    #The given state should be proclaimed as the best state
    #(so I have to evaluate it).
    #lastTabu is the given state found in the tabu list.
    lastTabu <- (length(tabuList) %% size) + 1
    for(i in seq(length.out=length(tabuList))){
      if(identical(state,tabuList[[i]]) ){
        lastTabu <- (i %% size) + 1;
        break
      }
    }
    if(verbose >= 2) print("Starting with the suggested initial state.")
  }
  evaluation <- evaluateState(records,waterTekort,state,"state",alpha)
  bestState <- state;
  bestSlipped <- evaluation$slipped;
  counter <- 1 #How many iterations since beginning?
  counter.NoAdmissible <- 0 #How many iterations have we stayed in the same state
  # because we haven't found any admissible state?
  bestIteration <- 0 #In which iteration did we have the best state?

```

```

lastNeighborhood <- NULL
#If I reload the last search, then these are already defined.
}
if(verbose >= 1) print(paste("Our current maximum is ", bestSlipped, "slipped."))
if(verbose >= 3) print(state)

repeat{
  #Determine a new neighborhood.
  #Take some EX and ADD and DROP neighbors at random.
  #Based on the preferences, choose the actual neighborhood.

  if(verbose >= 2) print("Generating the neighborhood...")
  neighborhood <- list(); #to combine lists, use append
  #In principle, it can happen that one of the following ends up empty.
  # However, this will be the case only if we have very few flights...(as with F100)
  #So I ignore the problem for the time being.
  flippableOn <-setdiff(state[which(state$slipIn == FALSE |
    state$slipOut == FALSE), "DST"],immutableDST) ;
  flippableOff <-setdiff(state[which(state$slipIn == TRUE |
    state$slipOut == TRUE), "DST"], immutableDST);
  flippable2On <-setdiff(state[which(state$slipIn == FALSE &
    state$slipOut == FALSE), "DST"],immutableDST) ;
  flippable2Off <-setdiff(state[which(state$slipIn == TRUE &
    state$slipOut == TRUE), "DST"], immutableDST);

  if(verbose >= 3){
    print("Possible to flip on/off:")
    print(flippableOn)
    print(flippableOff)
    print(flippable2On)
    print(flippable2Off)
  }

  #Generate some neighbors of the current state.
  #It can happen that some of the neighbors are actually identical or even equal
  # to the current state (if I flip one DST on and then immediately again on)
  #Description of the neighbors
  #-EX neighbor: obtained from the current state
  # by flipping one slipping destination
  # to non-slipping and one non-slipping to slipping
  #-ADD neighbor: obtained from the current state
  # by flipping one non-slipping destination
  # to slipping
  #-DROP neighbor: obtained from the current state
  # by flipping one slipping destination
  # to non-slipping

```

```

#-EX2neighbor: flip one slipping destination to non-slipping,
# both for slipIn as well as for slipOut;
# and similarly one non-slipping to slipping
#-ADD2 neighbor: flip one non-slipping destination to slipping,
#both for slipIn as well as for slipOut
for(i in seq(length.out=min(EXneighborhoods,
                          length(flippableOn)*length(flippableOff)))){
  DSTToFlipOn <- sample(flippableOn,1); DSTToFlipOff <- sample(flippableOff,1)
  candidateState <- draw.candidate(state,TRUE,DSTToFlipOn,1)
  candidateState <- draw.candidate(candidateState,FALSE,DSTToFlipOff,1)
  neighborhood <- append(neighborhood,list(candidateState))
}
for(i in seq(length.out=min(ADDneighborhoods,length(flippableOn)))){
  DSTToFlipOn <- sample(flippableOn,1);
  candidateState <- draw.candidate(state,TRUE,DSTToFlipOn,1)
  neighborhood <- append(neighborhood,list(candidateState))
}
for(i in seq(length.out=min(EX2neighborhoods,length(flippable2On)
                          *length(flippable2Off)))){
  DSTToFlipOn <- sample(flippable2On,1); DSTToFlipOff <- sample(flippable2Off,1)
  candidateState <- draw.candidate(state,TRUE,DSTToFlipOn,2)
  candidateState <- draw.candidate(candidateState,FALSE,DSTToFlipOff,2)
  neighborhood <- append(neighborhood,list(candidateState))
}
for(i in seq(length.out=min(ADD2neighborhoods,length(flippable2On)))){
  DSTToFlipOn <- sample(flippable2On,1);
  candidateState <- draw.candidate(state,TRUE,DSTToFlipOn,2)
  neighborhood <- append(neighborhood,list(candidateState))
}
if(forceDROP){
  #Consider then also DROP neighborhoods, to diversify the search.
  for(i in seq(length.out=min(DROPneighborhoods,length(flippableOff)))){
    DSTToFlipOff <- sample(flippableOff,1)
    candidateState <- draw.candidate(state,FALSE,DSTToFlipOff,1)
    neighborhood <- append(neighborhood,list(candidateState))
  }

  for(i in seq(length.out=min(DROP2neighborhoods,length(flippable2Off)))){
    DSTToFlipOff <- sample(flippable2Off,1)
    candidateState <- draw.candidate(state,FALSE,DSTToFlipOff,2)
    neighborhood <- append(neighborhood,list(candidateState))
  }
}

if(verbose >= 1) print(paste("Number of considered neighbors before tabu: ",
                             length(neighborhood)))

```

```

#Check which of the neighborhoods sit in the tabuList and get rid of them.
#If I have includeWholeHistory=FALSE, then tabuList and lastNeighborhood
# are treated separately, otherwise all is in the tabuList
if(verbose >= 2) print("Reducing the neighborhood according to the tabu list and
the last neighborhood...")
#The reason to use seq in for loops is to ensure correct behavior
# in case that some of the lists are empty

matchingLast <- integer(length(neighborhood)); is.na(matchingLast)<-TRUE
#If I use includeWholeHistory, this loop will be automatically empty.
#Recall that lastNeighborhood is also a kind of tabu list
for(i in 1:length(neighborhood)){#Looping over all states in lastNeighborhood
# (if applicable).
  for(j in seq(length.out=length(lastNeighborhood))){
    if(identical(neighborhood[[i]],lastNeighborhood[[j]]) ){
      matchingLast[i]<-j;
      break
    }
  }
}
neighborhood <- neighborhood[is.na(matchingLast)]
matchingTabu <- integer(length(neighborhood)); is.na(matchingTabu)<-TRUE
for(i in seq(length.out=length(neighborhood))){
  for(j in seq(length.out=length(tabuList))){
    if(verbose >= 3) print(all.equal(neighborhood[[i]],tabuList[[j]]))
    if(identical(neighborhood[[i]],tabuList[[j]]) ){
      matchingTabu[i]<-j;
      break
    }
  }
}
neighborhood <- neighborhood[is.na(matchingTabu)]
if(verbose >= 2){
  if(!includeWholeHistory) print(matchingLast)
  print(matchingTabu)
}
if(verbose >= 1) print(paste("Number of considered neighbors after the tabu and
last neighborhood check: ", length(neighborhood)))

if((length(neighborhood) == 0) & !forceDROP){
  #Consider then also DROP neighborhoods, to diversify the search.
  #If I force the DROP neighborhoods, then this step was already done before.
  if(verbose) print("No EX and ADD neighbors. Consider also DROP neighborhoods.")

  for(i in seq(length.out=min(DROPneighborhoods,length(flippableOff)))){
    DSTToFlipOff <- sample(flippableOff,1)
    candidateState <- draw.candidate(state,FALSE,DSTToFlipOff,1)
  }
}

```

```

neighborhood <- append(neighborhood,list(candidateState))
}

for(i in seq(length.out=min(DROP2neighborhoods,length(flippable20ff)))){
  DSTToFlipOff <- sample(flippable20ff,1)
  candidateState <- draw.candidate(state,FALSE,DSTToFlipOff,2)
  neighborhood <- append(neighborhood,list(candidateState))
}
}

#Evaluate the remaining neighborhoods and choose the best among them
#to be the new state.
if(verbose >= 2) print("Evaluating the remaining neighbors...")
bestIndex <- -1; bestSlippedSoFar <- 0;
for(i in 1:length(neighborhood)){
  candidateState<-neighborhood[[i]]
  evaluation<-evaluateState(records,waterTekort,candidateState,"state",alpha)
  if(verbose >= 2) print(paste(i,evaluation$admissible, evaluation$slipped))
  if(verbose >= 3) print(candidateState)
  if(evaluation$admissible){
    if(evaluation$slipped > bestSlippedSoFar){
      bestSlippedSoFar <- evaluation$slipped
      bestIndex <- i
    }
  }
}

#Proclaim the winner as the new state.
if(verbose >= 2) print("Cleaning up...")
if(bestIndex > 0) state<-neighborhood[[bestIndex]] #Otherwise, we have found
# no new admissible state, so we try a new round.
else counter.NoAdmissible <- counter.NoAdmissible + 1;
if(!forceDROP & (counter.NoAdmissible >= noChangeTurnsTillforceDROP)){
  if(verbose >= 1) print(paste("We have not moved from
the current state for last",counter.NoAdmissible,"turns.",
"Starting to consider DROP neighborhoods, to diversify the search.))
  forceDROP <- TRUE
}
counter <- counter+1

#Check if the new state is better than the best state.
if(bestSlippedSoFar > bestSlipped){
  bestState <- state
  bestSlipped <- bestSlippedSoFar
  bestIteration <- counter;
  if(verbose >= 1) print(paste("We have a new best state,
being able to slip", bestSlipped ))
}

```

```

}

#Put the new state on the tabuList. Update the lastNeighborhood.
#If includeWholeHistory == TRUE, put everything in the tabu list.
if(includeWholeHistory){
  for(i in 1:length(neighborhood)){
    tabuList[[lastTabu]] <- neighborhood[[i]]
    lastTabu <- (lastTabu %% size) + 1
  }
}else{
  tabuList[[lastTabu]] <- state
  lastTabu <- (lastTabu %% size) + 1
  lastNeighborhood <- neighborhood
}

if(verbose >= 2) print(paste("Current size of the tabu list is",length(tabuList)))

#stopping condition
if(terminationCriterion=="Iterations"){
  if(counter >= iter) break
}else if(terminationCriterion == "No improvement"){
  if(verbose >= 2) print(paste("Last improvement was before",
    (counter - bestIteration), "iterations"))
  if((counter - bestIteration) > noImproveTermination) break
}else if(terminationCriterion=="Evaluations"){
  if(!includeWholeHistory){
    print("Error: No measure of evaluations available")
    return(NULL)
  }
  if(length(tabuList) > maxEvals) break
}

#Saving the results in order to possibly interrupt computations and
# resume later (e.g. tomorrow or the next week).
if((iter %% saveFrequency == 0) & saveResults) save(bestState,
  bestSlipped, counter, bestIteration, state, tabuList,
  lastTabu,lastNeighborhood,counter.NoAdmissible,
  file=paste(outputDir,"tabu.RData",sep=""))

if(verbose >= 2) print(paste("Here we go again! Entering the iteration", counter))
}

if(verbose >= 2) print("Leaving the tabu search.")

```

```

evaluation <- evaluateState(records,waterTekort,bestState,"state",alpha)

if(saveResults) save(bestState, bestSlipped, counter, bestIteration, state,
  tabuList, lastTabu,lastNeighborhood,
  counter.NoAdmissible,file=paste(outputDir,"tabu.RData",sep=""))
return(list(bestState=bestState, bestState.evaluation=evaluation, tabuList=tabuList))
}

```

## B.2 Tabu search for city pairs

For a detailed description, see Section 2.3.

```

#A version of tabu search to find the optimal set of city-pairs to MS.
#####
#Purpose
#####
#From a given training set of rotations
# (which implicitly provides information about timetable and water usage
#   for the given period),
#computes a list of city-pairs to multistretch in order
# to achieve a very high multistretch rate while keeping the coverage rate
# per destination above alpha.
#The assumption (to be partially validated) is that the list
# will work well also for "similar" periods -
# i.e. those with a similar timetable and water usage:
# this might be for instance the period one year after the training period.
#####
#Parameters
#####
#alpha: the maximum admissible shortage level per DST
#substages, oscillations: The desired number of substages during Stage1,
  respectively number of oscillations within one substage
#AspirationPlus.*: See the comment before the code for low level and critical level mode
#descentDepth: How many moves before ending the descent mode?
#ascent.neighbors: How many neighbors to draw during an ascent iteration.
#ascent.extra: If no feasible neighbors has been found so far during an ascent iteration
  , how many to draw extra (continuing drawing only until a feasible one is found).
#critical.noImprove: For low level and critical level, this determines
  the number of iterations without improvement needed for the mode to be terminated.
#aspiration.default: How many candidate pairs must be rejected because
  of the tabu criterion until we relax the tabu criterion?
#good.moves.*.numberSaved: How many best moves are saved as elite moves, per oscillation?
#penalites$W: The probability of admitting (in the candidate state selection)
  a drop/add of a light/medium/heavy pair if the current oscillation
  has preferred weight W.
#           See draw.candidate() for more insight.

```

```

#reloadLast: Should we start where the last saved search ended?
#saveResults: Should save results every now and then
# (in order to resume the search later)?
#saveFrequency: If we want to save results, once per how many iterations?
#returnedSolutions: How many of the "most elite" solutions the algorithm returns?
#Stage2.allowed = full, relinking, moves or no
#restart.pattern = "all soft", "all hard", "alternating"
#####
#Some additional notes
#####
#-I use, whenever possible, the new, quicker method to compute ActualTanking
#-Subfunctions sometimes use global assignments.
    While this is in general not a good programming practice, I chose to do so
    in order to streamline the code.

tabuSearchPairs<-function(outputDir, records, waterTekort, consideredPairs, frequencies
    , coverages,state = NULL, alpha=0.05,
    substages = 5, oscillations = 3,
    AspirationPlus.Min=5, AspirationPlus.Max=30, AspirationPlus.Plus=10
    , descentDepth = 10, stabDescent.neighbors=3, ascent.neighbors = 5,
    ascent.extra = 5, critical.noImprove = 10,
    aspiration.default = 10, good.moves.swap.numberSaved = 3
    , good.moves.add.numberSaved = 3,
    penalties=list( heavy = c(0.2,0.4,1.0), medium = c(0.5,1.0,0.5)
        , light = c(1.0,0.4,0.2)),
    reloadLast = FALSE, saveResults = TRUE, saveFrequency = 10, verbose = 0
    ,returnedSolutions=min(3,substages), evaluation.startsProvided=FALSE
    , Stage2.allowed = "relinking", restart.pattern = "alternating")
{
#####
#Settings and constants
#####
#Systematic dynamic tenure, see Sec.2.5.2 TS.
#This means that the length of tabu tenure is not constant
# , but instead drawn from the periodical lists.
#According to TS: "Once a good range of tenure values is located
# , first level improvements generally result by selecting different values
# from this range on different iterations."
#Tenures are relatively short, otherwise the search seemed bogged down
# by too many tabu restrictions.
number <- critical.noImprove %/% 2
tenureDrop <- c(max(2,number-5),max(4,number-3),number)#,number+3,number+5)
tenureAdd <- c(max(2,number-5),max(4,number-3),number)#,number+3,number+5)

kThresholdDivisor <- 20

```

```

#####
#Functions
#####
#Draws candidate states, tests the tabu status and probabilistic "discouragements"
# , possibly evaluates the candidate
#candidate.type is ADD, DROP or SWAP
#If removeBadDST, then the candidates (to replace an unfeasible state) are drawn
# such that the pairs inducing the unfeasibility are removed.
#If furthermore badPair is not NULL, I try to take another pair
# with one of the same DST.
draw.candidate <- function(state,candidate.type,evaluate=TRUE,
  removeBadDST=FALSE,badDST=NULL
  ,bad.alreadyDrawn = character(0)){

  candidate.add <- NULL; candidate.drop <- NULL

  #Select the add pair of the move.
  if(candidate.type=="ADD" | candidate.type=="SWAP"){
    addPossible <- setdiff(consideredPairs,state)
    tabu.aspiration <- 0
    #Loop until a pair is found such that it is not rejected by either the tabu list
    # or the weight/coverage discouragements.
    while(is.null(candidate.add)){

      candidate.add <- sample(addPossible,1)

      #test the tabu status.
      if(tabuEnd.add[candidate.add]>iter){
        if(verbose>=3) print(paste("Rejected addition of",candidate.add
          ,"because of its tabu status."))
        tabu.aspiration <- tabu.aspiration + 1
        if(tabu.aspiration > aspiration.default){
          #Override the tabu status of all admissible pairs with
          #the lowest or almost lowest tabu counter.
          #It's possible that m < iter. In that case, little will
          # (significantly) change; but in that case,
          #I should not get to this code anyway.
          m <- min(tabuEnd.add[addPossible])
          tabuEnd.add[addPossible]
            [tabuEnd.add[addPossible]<=(m+(tabu.aspiration %/% 2))]
              <<- iter
          if(verbose>=3) print("Aspiration by default applied.")
        }
        candidate.add <- NULL
        next
      }
    }
  }
}

```

```

}

#Incorporate the weight discouragement.
#Discouragement with probability depending on the preferred weight
# for the current oscillation and on the weight of the pair.
if(frequencies[candidate.add]<=lightBound){
  dismiss <- runif(1) > penalties[[preferred.weight]][1]
}else if(frequencies[candidate.add]<=heavyBound){
  dismiss <- runif(1) > penalties[[preferred.weight]][2]
}else{
  dismiss <- runif(1) > penalties[[preferred.weight]][3]
}
}
if(dismiss){
  if(verbose>=5) print(paste("Rejected addition of",candidate.add
  ,"because of its weight (",frequencies[candidate.add],")."))
  candidate.add <- NULL
  next
}

#Incorporate the coverages discouragement.
#For adding, the probability of adding let be equal to the coverage.
if(runif(1) > coverages[candidate.add]){
  if(verbose>=5) print(paste("Rejected addition of",candidate.add,
  "because of its coverage (",coverages[candidate.add],")."))
  candidate.add <- NULL
  next
}

#If I am at the critical level, then I should always swap two pairs
# which share one of the destinations, otherwise
# I have a meager chance to obtain an admissible state.
#Indeed, at the critical level, presumably most destinations
# have just the coverage around the 1-alpha level.
# Hence if I would to multistretch a pair with them,
#it must come in exchange for another pair with the same destination.
#The above argument is not completely valid since a completely distinct pair
# of DST may block the MS opportunities for the first pair
# because of the "no two MS in the row" rule
#and this would then lead to higher coverages
# for the destinations in the first pair.
if(mode == "critical level"){
  unc <- uncodePair(candidate.add)
  dropPossible<-c(state[sapply(state,isDSTInPair,dst=unc["dst1"])]
  ,state[sapply(state,isDSTInPair,dst=unc["dst2"])])
  if(verbose >= 5){
    print(candidate.add)
    print(dropPossible)
  }
}

```

```

    }
    #No possible pairs with the same destinations as the candidate
    # -> reject the candidate altogether then.
    if(length(dropPossible)==0) candidate.add <- NULL
  }
}

}

#Select the drop pair of the move.
if(candidate.type=="DROP" | candidate.type=="SWAP"){

  if(removeBadDST){
    #Consider only pairs containing DST which cause unadmissible shortages.
    if(is.null(badDST)) badDST <- names(which(currentEvaluation$shortages>alpha))
    dropPossible <- character(0)
    #Test all pairs in state if they contain a badDST.
    # Make admissible only those that do.
    for(dst in badDST){
      dropPossible<-c(dropPossible,state[sapply(state,isDSTInPair,dst=dst)])
    }
    dropPossible <- setdiff(dropPossible,bad.alreadyDrawn)
    if(length(dropPossible)==0){
      if(verbose >= 3) print("No more requested bad pairs to draw.")
      return(list())
    }
    #Are all pairs in dropPossible already drawable now?
    #During the stabilizing descent phase, there is no tabu drop list yet.
    # Furthermore, rejection based on weight is not permitted.
    # And rejection on coverage can be overcome
    # (furthermore, destinations which cause shortage will probably
    # have low coverage anyway).
  }else if(mode=="critical level"){
    #Handled already with candidate.add
  } else dropPossible <- state
  dropPossible<-unique(dropPossible)

  tabu.aspiration <- 0
  while(is.null(candidate.drop)){
    #if(verbose>=5) print(dropPossible)
    candidate.drop <- sample(dropPossible,1)

    #test the tabu status.
    if(tabuEnd.drop[candidate.drop]>iter){
      if(verbose>=3) print(paste("Rejected removal of",candidate.drop,
        "because of its tabu status."))
    }
  }
}

```

```

tabu.aspiration <- tabu.aspiration + 1
if(tabu.aspiration > aspiration.default){
  #Override the tabu status of all admissible pairs
  #with the lowest or almost lowest tabu counter.
  #It's possible that m < iter. In that case, little will
  #(significantly) change; but in that case,
  # aspiration by default is not justified anyway
  m <- min(tabuEnd.drop[dropPossible])
  #What can happen: the moves allowed by the aspiration by default
  # are rejected based on the weight or coverages discouragement.
  #Because of this, enable still more and more moves.
  tabuEnd.drop[dropPossible]
  [tabuEnd.drop[dropPossible]<=(m+(tabu.aspiration %/% 2))]
  <<- iter
  if(verbose>=3) print("Aspiration by default applied.")
}
candidate.drop <- NULL
next
}

#Incorporate the weight discouragement.
#For dropping, don't reject states with low coverage on basis of weight.
#Otherwise it might be tough to get rid of a pair with low coverage
# which ruins admissibility.
if(coverages[candidate.drop] > (1-3*alpha) & mode != "stabilizing descent"){
  if(frequencies[candidate.drop]<=lightBound){
    dismiss <- runif(1) > penalties[[preferred.weight]][1]
  }else if(frequencies[candidate.drop]<=heavyBound){
    dismiss <- runif(1) > penalties[[preferred.weight]][2]
  }else{
    dismiss <- runif(1) > penalties[[preferred.weight]][3]
  }
  if(dismiss){
    if(verbose>=5) print(paste("Rejected removal of",candidate.drop,
      "because of its weight (",frequencies[candidate.drop],")."))
    candidate.drop <- NULL
    next
  }
}

#Incorporate the coverages discouragement.
#For dropping, the probability of dropping let be equal to 1-(1-coverage)*5.
# In other words, anything with coverage lower than 80%
# is dropped without questions.
#This may seem high, but if we strive for the coverage 95%
# (even though it is per destination),
# then even something with the coverage 85% should be "droppable".

```

```

        if(runif(1) > (1-coverages[candidate.drop])*5){
            if(verbose>=5) print(paste("Rejected removal of",candidate.drop,
                "because of its coverage (",coverages[candidate.drop],")."))
            candidate.drop <- NULL
            next
        }
    }
}

#Create the candidate state by combining the above found candidate.drop
# and candidate.add.
candidate <- c(state,candidate.add)
if(!is.null(candidate.drop)){
    #Remove the candidate.drop from the state.
    candidate <- setdiff(candidate,c(candidate.drop))
}

if(evaluate){
    #Works no matter if ADD, DROP or SWAP
    changedPairs<-list(); changedPairs$drop <- candidate.drop;
    changedPairs$add <- candidate.add
    evaluation <- evaluateState(currentEvaluation$rotations,waterTekort,candidate,
        "pair state",alpha,changedPairs=changedPairs,
        startsProvided = evaluation.startsProvided)
    totalNumEvaluations <<- totalNumEvaluations + 1
    if(verbose>=3) print(paste("New candidate drawn:",evaluation$admissible,
        evaluation$slipped, "add:", candidate.add, "drop:", candidate.drop))
}else evaluation <- NULL

return(list(state=candidate,evaluation=evaluation,
    added=candidate.add,dropped=candidate.drop))
}

#Moves to the new state, updates tabu lists.
make.move <- function(candidate){

    #Update tabu lists-if I have just dropped a pair, I don't want to add it
    # anytime soon! And vice-versa.
    #The length of tabu tenure (i.e. how long a pair stays on the list) is drawn
    # according to the tenureDrop/tenureAdd vector (see their definition)
    #Also update the transition vector (the long-term memory...)
    if(!is.null(candidate$added)){
        tabuEnd.drop[candidate$added] <<- iter + tenureDrop[tenureDrop.counter]
        tenureDrop.counter <<- (tenureDrop.counter %% length(tenureDrop)) + 1
        transition[candidate$added] <<- transition[candidate$added] + 1
    }
}

```

```

if(!is.null(candidate$dropped)){
  tabuEnd.add[candidate$dropped] <- iter + tenureAdd[tenureAdd.counter]
  tenureAdd.counter <- (tenureAdd.counter %% length(tenureAdd)) + 1
  transition[candidate$dropped] <- transition[candidate$dropped] + 1
}

#Save the current move.
#Assume all moves to be "good". One filters them at the end of the substage.
if(mode=="critical level" | mode=="low level"){
  move <- list(add=candidate$added,drop=candidate$dropped,
    change=(candidate$evaluation$slipped - currentEvaluation$slipped))
  good.moves.swap[[length(good.moves.swap)+1]] <- move
}
if(mode=="ascent"){
  move <- list(add=candidate$added,
    change=(candidate$evaluation$slipped - currentEvaluation$slipped))
  good.moves.add[[length(good.moves.add)+1]] <- move
}

#Here, I evaluate the candidate again. This might seem wasteful,
# however I need to return slipped rotations for it,
# in order to be able to do quick calcActualTanking
#I don't want to return rotations for all candidates considered
#because rotations are large
# and I can have for instance 20 candidates -> I would run out of memory.
changedPairs<-list(); changedPairs$drop <- candidate$dropped;
changedPairs$add <- candidate$added
currentEvaluation <-evaluateState(currentEvaluation$rotations,waterTekort,
  candidate$state,"pair state",alpha,returnSlipped=TRUE,
  changedPairs=changedPairs,startsProvided = evaluation.startsProvided)
totalNumEvaluations <- totalNumEvaluations + 1
#
#ev <- evaluateState(records,waterTekort,candidate$state,"pair state",alpha)

if(verbose>=3){
  print(paste("New state chosen:",currentEvaluation$admissible,
    currentEvaluation$slipped, "add:", candidate$added, "drop:", candidate$dropped))
}

return(candidate$state)
}
#Changes modeChange.counter and does everything necessary for a mode change.
change.mode <- function(newMode){
  if(verbose>=2) print(paste("Changing the mode from", mode, "to", newMode))
  #Save the best solution from this oscillation.
  if(mode=="critical level"){

```

```

elite.solutions[[length(elite.solutions)+1]] <<- bestState
elite.evaluations[[length(elite.evaluations)+1]] <<- bestEvaluation

#The good state (from the previous low level) was used for path relinking
# at the end of the mode. Now we can discard it.
good.state <<- NULL

#New oscillation starts.
#Adjust the preferred weight.
oscillation.counter <<- oscillation.counter + 1;
if(verbose>=2){
  print(paste("Best solution from this oscillation:",
    bestEvaluation$admissible, bestEvaluation$slipped))
  if(oscillation.counter <= oscillations) print(paste("Entering the
    oscillation", oscillation.counter, "of substage",substage.counter))
}
if(oscillation.counter==oscillation.medium) preferred.weight <<- "medium"
if(oscillation.counter==oscillation.light) preferred.weight <<- "light"
}else if(mode=="low level"){
  if(verbose>=0) print("Saving good.state.")
  good.state <<- bestState;
}

if(mode=="ascent"){
  ascentDuration <- iter-modeChange.counter
}

#The best state is counted only for the current oscillation.
bestIteration <<- iter
bestState <<- state
bestEvaluation <<- currentEvaluation
bestEvaluation$rotations <<- NULL

modeChange.counter <<- iter

mode <<- newMode

}

determine.threshold <- function(observedQuality,amountAdmissible){
  if(preferred.weight=="heavy") number <- heavyBound %/% kThresholdDivisor
  else if(preferred.weight=="medium") number <- lightBound %/% kThresholdDivisor
  else number <- 1

  if(amountAdmissible == 0) return(0)
  else return(observedQuality + number)
}

```

```

#See Sec.4.5 TS.
#Criterion is the value which we would like to reach or exceed, in order
#to return a result.
#Take all pairs which are in state.guiding, but not in state.initiating and
# try to incorporate them in state.initiating
#If swapEquivalent is on, then dropPossible pairs must have
#a destination in common with the pair being included.
#This is particularly handy if the initiating state is at the critical level.
#effort indicates how many pairs to drop we (at most) consider
# together with the desired pair to add.
pathRelinking <- function(state.initiating,state.guiding,criterion,
  effort=20,swapEquivalent=TRUE){
  if(verbose >= 3) print("Trying to path relink the two given states.")
  state<-state.initiating; evaluation<-evaluateState(records,waterTekort,state,
    "pair state",alpha,returnSlipped=TRUE, startsProvided = evaluation.startsProvided)
  totalNumEvaluations.hard <<- totalNumEvaluations.hard + 1
  if(verbose >= 3) print(paste("Initiating state:",evaluation$admissible,
    evaluation$slipped, "; criterion:",criterion))
  success <- FALSE
  #Which pairs do we want to incorporate in state.initiating?
  toInclude <- setdiff(state.guiding,state.initiating);
  toInclude <- sample(toInclude) #shuffle
  if(verbose >= 5){
    print("Pairs to include from the guiding state:")
    print(toInclude)
  }
  for(pair in toInclude){
    #Try to add the pair.
    candidate <- c(state,pair)
    evaluation.cand <- evaluateState(evaluation$rotations,waterTekort,
      candidate,"pair state",alpha,returnSlipped=TRUE,changedPairs=list(pair),
      startsProvided = evaluation.startsProvided)
    if(verbose>=4) print(paste("Trying to add", pair,";",
      evaluation.cand$admissible, evaluation.cand$slipped))
    totalNumEvaluations <<- totalNumEvaluations + 1
    neighborhood <- list(candidate)
    neighborhood.evaluations <- list(evaluation.cand)

    bestSoFar <- -1
    bestIndex <- -1

    #Otherwise try to exchange it with some other pair with the same DST
    if(!evaluation.cand$admissible){
      unc <- uncodePair(pair)
      if(swapEquivalent) dropPossible<-c(state[sapply(state,isDSTInPair,
        dst=unc["dst1"])],state[sapply(state,isDSTInPair,dst=unc["dst2"])])
    }
  }
}

```

```

else dropPossible <- state
print(dropPossible)
#Select (some) pairs from dropPossible to be exchanged with the desired pair.
toDrop <- sample(dropPossible,min(effort,length(dropPossible)))
i <- 1
for(pair2 in toDrop){
  i <- i+1
  #Try out the neighbor obtained by exchanging pair for pair2
  candidate2 <- setdiff(candidate,pair2)
  evaluation.cand <- evaluateState(evaluation$rotations,waterTekort,
    candidate2,"pair state",alpha,returnSlipped=TRUE,
    changedPairs=list(pair,pair2),
    startsProvided = evaluation.startsProvided)
  totalNumEvaluations <<- totalNumEvaluations + 1
  if(verbose>=4) print(paste("Trying to add", pair, "and drop", pair2,";",
    evaluation.cand$admissible, evaluation.cand$slipped))
  neighborhood[[i]] <- candidate2
  neighborhood.evaluations[[i]] <- evaluation.cand
  if(evaluation.cand$admissible & evaluation.cand$slipped > bestSoFar){
    bestSoFar <- evaluation.cand$slipped
    bestIndex <- i
  }
}
}
}else{
  #We have added our desired pair, not needing to drop anything. Well done.
  bestSoFar <- evaluation.cand$slipped
  bestIndex <- 1
}

if(bestIndex > -1 & (bestSoFar >= evaluation$slipped)){
  #There was an admissible candidate
  state <- neighborhood[[bestIndex]]
  evaluation <- neighborhood.evaluations[[bestIndex]]
  if(evaluation$slipped >= criterion){
    #The best of the admissible candidates
    # has even exceeded our desired criterion.
    if(verbose>=3) print(paste("Exceeded the criterion with value:",
      evaluation$slipped))
    bestEvaluation <- evaluation; bestEvaluation$rotations <- NULL
    bestState <- state
    criterion <- evaluation$slipped
    success <- TRUE
  }
}
}
}#end looping over pairs to include
return(list(success=success,state=bestState,evaluation=bestEvaluation))
}

```

```

#####
#Initialization
#####
if(verbose >= 1) print("Entering the tabu search.")
if(reloadLast){
  if(verbose >= 2) print("Resuming a previous search.")
  #Assuming that the data to be loaded indeed exist.
  load(file=paste(outputDir,"tabuPairs.RData",sep=""))
}
}else{
  #Initialize a new tabu search instance.

  elite.solutions <- list(); elite.evaluations <- list()
  #See p.119 TS how to handle the transition memory.
  transition <- numeric(length(consideredPairs));
  names(transition) <- consideredPairs
  #short term memory
  #see p.46 TS on implementation
  tabuEnd.add <-numeric(length(consideredPairs));
  names(tabuEnd.add) <- consideredPairs
  tabuEnd.drop <-numeric(length(consideredPairs));
  names(tabuEnd.drop) <- consideredPairs

  stage.counter <- 1
  substage.counter <- 1
  oscillation.counter <- 0
  iter <- 0
  mode <- "undetermined"
  modeChange.counter <- 0
  bestIteration <- 0
  bestState <- NULL
  bestEvaluation <- NULL
  #To judge the speed of the algorithm:
  #One should count "quick" evaluations and "standard" evaluations separately
  # since the standard ones take much more time.
  #number of quick evaluations
  totalNumEvaluations <- 0
  #number of standard evaluations
  totalNumEvaluations.hard <- 0

  #For evaluation of the Aspiration Plus method
  AspirationPlus.howmany <- 0
  AspirationPlus.evaluations <- 0

  #If the state was not supplied, just take the empty state
  if(is.null(state)){

```

```

#state <- consideredPairs
#It was very difficult for the search procedure to track down
# (starting from "slip everything") the pairs which caused the shortages.
if(verbose>=2) print("Starting with the empty state.")
state <- character(0)

}else if(verbose>=2) print("Starting with the supplied state.")
currentEvaluation<-evaluateState(records,waterTekort,state,"pair state",
  alpha,returnSlipped=TRUE,startsProvided = evaluation.startsProvided)
totalNumEvaluations.hard <- totalNumEvaluations.hard + 1
if(verbose>=4){
  print(paste("First state:",currentEvaluation$admissible
    ,currentEvaluation$slipped))
}

#Save moves that were used for the make.move. After the end of a substage,
# save several of the best ones to elite.moves.
#As a move, save the added and/or dropped pair and also
# the caused change in slipped
#So move = list(add=, improvement=) in case of SWAP
#and move = list(add=, drop=, improvement=) in case of SWAP
elite.moves.add <- list(); elite.moves.swap <- list()
#The following ones are the lists for the current substage.
#After the substage, the best moves from them are added to the global
#lists good.moves.add and good.moves.swap
good.moves.add <- list(); good.moves.swap <- list()
#At the low level, save the best state and then try to use it
# at the critical level (via path relinking).
good.state <- NULL

preferred.weight <- "heavy"
# heavy, medium, light; the heavy ones are () the "foundation components"
# while the light ones are "crack fillers".
#In the current context the "heavy" ones are the pairs with
# high frequencies of flights.
#When to start the "medium" and "light" regime?
# Ideally the number of oscillations is a multiple of 3.
#The idea is to first prefer heavier pairs and then
#gradually lighter ones ("crack fillers"). See also TS 5.6.1.
}

#Which data should be regularly saved?
#Don't remember the precise state within the oscillation
saveList <- c("elite.solutions","elite.evaluations","elite.moves.add",
  "elite.moves.swap","transition", "tabuEnd.add", "tabuEnd.drop",
  "stage.counter","substage.counter","oscillation.counter",

```

```

    "iter","modeChange.counter","state","mode","bestState",
    "bestEvaluation","bestIteration", "good.moves.add", "good.moves.swap",
    "good.state","currentEvaluation","totalNumEvaluations",
    "totalNumEvaluations.hard","preferred.weight",
    "AspirationPlus.howmany","AspirationPlus.evaluations")

#####
neighborhood <- NULL
tenureAdd.counter <- 1
tenureDrop.counter <- 1

oscillation.medium <- (oscillations %/% 3) + 1
oscillation.light <- (oscillations %/% 3)*2 + 1

#What is below the light bound, is light. What is above the heavy bound, is heavy.
lightBound <- quantile(freqs,probs=c(0,1/3,2/3,1))[2]
heavyBound <- quantile(freqs,probs=c(0,1/3,2/3,1))[3]
#Which substages start with a hard restart and which with a soft one?
#For the first substage this is irrelevant
turns <- 1:substages
if(restart.pattern=="alternating"){
  restarts.soft <- turns[turns %% 2 == 0]
  restarts.hard <- turns[turns %% 2 == 1]
}else if(restart.pattern == "all soft"){
  restarts.soft <- turns
}else if(restart.pattern == "all hard"){
  restarts.soft <- c()
}

#####
#First stage
#####
if(verbose>=1) print("Entering stage 1.")
if(substage.counter>substages) break
if(verbose>=2) print(paste("We are in substage",substage.counter,",
  oscillation:",oscillation.counter,", mode:",mode))

#We have just started and try to find the critical level.
#If the current state is not admissible, we must go downwards, otherwise upwards.
if(mode=="undetermined"){
  if(currentEvaluation$admissible){
    mode <- "ascent"
  }else{
    mode <- "stabilizing descent";
  }
}
}

```

```

#substage
repeat{
  #If at the critical level, finish it first and only then end the substage.
  #Otherwise, if the number of oscillation is exceeded, start a new substage.
  if( (mode != "critical level") & (oscillation.counter>oscillations)){
    substage.counter <- substage.counter + 1

    #Save the best of good.moves from the finished substage
    numberSaved.add <- min(good.moves.add.numberSaved,length(good.moves.add))
    numberSaved.swap <- min(good.moves.swap.numberSaved,length(good.moves.swap))
    #print(paste("Number saved:", numberSaved.swap))
    bestIndices.add <- order(sapply(good.moves.add,function(x) x$change),
      decreasing=TRUE)[1:numberSaved.add]
    bestIndices.swap <- order(sapply(good.moves.swap,function(x) x$change),
      decreasing=TRUE)[1:numberSaved.swap]

    #Add the best ones to the global lists
    elite.moves.add <- append(elite.moves.add,good.moves.add[bestIndices.add])
    elite.moves.swap <- append(elite.moves.swap,good.moves.swap[bestIndices.swap])

    good.moves.add <- list(); good.moves.swap <- list();

    if(substage.counter>substages) break
    else{
      if(verbose>=2) print(paste("Restarting and entering substage",
        substage.counter))

      #restart
      if(substage.counter %in% restarts.soft){
        #soft restart
        if(verbose >= 2) print("Performing a soft restart.")
        #Use the long-term memory.
        #print(paste("Length of transition:", length(transition), "Transition:"))
        #print(transition)
        #print(names(which(transition<=1)))
        #Take the last state from the previous substage...
        #Last because that one corresponds truly to the long-term memory
        # stored in transition
        #...include the pairs which were never "in".
        addPossible <- setdiff(consideredPairs,state)
        toAdd <- intersect(addPossible,names(which(transition<=1)))

        #... and drop from it all pairs which were never "out"...
        #If I started from the empty state, then every pair was once "out"
        #But there are those which just moved in and never out again
        # - I should drop those.

```

```

dropPossible <- state
toDrop <- intersect(dropPossible,names(which(transition<=1)))
#This will most likely lead to an unfeasible state
# (because those never "in" will often have low coverages),
#but so be it...
state <- c(state,toAdd)
state <- setdiff(state,toDrop)

#Of course, if my initial state was nonempty,
#then the above description will not fit.
#That's why I include unsharp inequalities for transition.
if(verbose >= 4){
  print("New state to start with:")
  print(state)
}
}else{
  #hard restart
  if(verbose >= 2) print("Performing a hard restart.")
  state <- character(0)
}

#Reset everything.
preferred.weight <- "heavy"
oscillation.counter <- 0
modeChange.counter <- 0
iter <- 0
bestIteration <- 0
bestState <- NULL
bestEvaluation <- NULL
tabuEnd.drop[] <- 0
tabuEnd.add[] <- 0
transition[] <- 0

currentEvaluation<-evaluateState(records,waterTekort,state,"pair state",
  alpha,returnSlipped=TRUE,startsProvided = evaluation.startsProvided)
if(verbose>=4){
  print(paste("First state after restarting:",currentEvaluation$admissible,
    currentEvaluation$slipped))
}
if(currentEvaluation$admissible){
  mode <- "ascent"
}else{
  mode <- "stabilizing descent";
}
}
}

```

```

if(mode=="descent"){
  #Always draw one pair and drop it; do this until enough pairs are dropped
  if(verbose>=3) print(paste("Step downwards",(iter-modeChange.counter)))
  if((iter-modeChange.counter) > descentDepth) change.mode("low level")
  #When dropping, it should not happen that we reach an unfeasible state,
  # but it is in principle possible...
  drawing <- draw.candidate(state,"DROP")
  if(drawing$evaluation$admissible) state <- make.move(drawing)
}
else if(mode=="stabilizing descent"){
  #Always choose one bad DST (i.e. with too many shortages)
  #Draw a neighborhood, if we find an admissible state, hoorah!
  #If not, take the neighbor which has the lowest shortage for the bad DST.
  #The mode changing criterion is achieving an admissible state.
  if(verbose>=3) print(paste("Step downwards",(iter-modeChange.counter)))
  alreadyDrawn <- character(0);
  badDST <- sample(names(which(currentEvaluation$shortages>alpha)),1)
  admissibleIndex <- -1
  if(verbose>=4) print("Creating a new neighborhood.")
  neighborhood <- list()
  for(i in 1:stabDescent.neighbors){
    drawing <- draw.candidate(state,"DROP",removeBadDST=TRUE,badDST = badDST ,
      bad.alreadyDrawn=alreadyDrawn)
    #There was no pair with badDST not already drawn.
    if(length(drawing)==0) break
    alreadyDrawn <- c(alreadyDrawn,drawing$dropped)
    neighborhood[[i]] <- drawing
    if( drawing$evaluation$admissible){
      admissibleIndex <- i
      break
    }
  }
  if(admissibleIndex > (-1)){
    state <- make.move(neighborhood[[admissibleIndex]])
    change.mode("ascent") #indeed, try to go upwards
    #to make sure we have not appeared on a low place
    # by making a big step downwards.
  }
  else{
    index <- which.min(sapply(neighborhood, function(x)
      x$evaluation$shortages [badDST]))
    state <- make.move(neighborhood[[index]])
  }
}

```

```

}else if(mode=="ascent"){
  #Always draw ascent.neighbors pairs, evaluate states resulting
  # from adding and choose the best one.
  #If none of the drawn neighbors contains an admissible state,
  # draw up to ascent.extra more.
  #If still nothing, assume that we have reached the critical level.
  if(verbose>=3) print(paste("Trying to step upwards",(iter-modeChange.counter)))

  bestSoFar <- -1
  bestIndex <- -1

  if(verbose>=4) print("Creating a new neighborhood.")
  neighborhood <- list()
  for(i in 1:ascent.neighbors){
    drawing <- draw.candidate(state,"ADD")
    neighborhood[[i]] <- drawing
    if( (drawing$evaluation$admissible)
        & (drawing$evaluation$slipped > bestSoFar)){
      bestSoFar <- drawing$evaluation$slipped
      bestIndex <- i
    }
  }
  #If no admissible state was found...
  repeat{
    i <- i+1
    if((bestSoFar > -1) | (i > (ascent.neighbors + ascent.extra))) break

    drawing <- draw.candidate(state,"ADD")
    neighborhood[[i]] <- drawing
    if( (drawing$evaluation$admissible)
        & (drawing$evaluation$slipped > bestSoFar)){
      bestSoFar <- drawing$evaluation$slipped
      bestIndex <- i
    }
  }
  if(verbose>=4) print(paste("Total number of neighbors tried:",
    length(neighborhood)))
  if(bestSoFar > -1) state <- make.move(neighborhood[[bestIndex]])
  else{
    if(verbose>=4) print("No admissible neighbors found.
      We have reached the critical level.")
    change.mode("critical level")
  }
}

}else if(mode=="low level" | mode == "critical level"){
  #if on the low level, one should remember good solutions to later apply them
  #on the critical level

```

```

if(iter==modeChange.counter & mode=="critical level" & !is.null(good.state)){
  #Nothing...
  #The path relinking takes place only in the end of this critical level mode.
}else{
  #Use the Aspiration Plus strategy as described in Section 3.2.1 TS:
  #I first draw Min candidates. Based on them, I determine a threshold.
  # I then draw candidates until there is one reaching/exceeding
  #the threshold value
  # (but the total number of candidates might not be larger than Max).
  #Afterwards, I draw Plus candidates more
  #(but again, the total number may not exceed Max).
  # "As an elementary option, the threshold can simply
  # be a function of the quality of
  #the initial Min moves examined on the current iteration."
  bestSoFar <- -1
  bestIndex <- -1
  admissible.counter <- 0
  neighborhood <- list()
  #Draw Min candidates.
  for(i in seq(length.out=AspirationPlus.Min)){
    drawing <- draw.candidate(state,"SWAP")
    neighborhood[[i]] <- drawing
    if( (drawing$evaluation$admissible)
      & (drawing$evaluation$slipped > bestSoFar)){
      bestSoFar <- drawing$evaluation$slipped
      bestIndex <- i
      admissible.counter <- admissible.counter + 1
    }
  }
  #Determine the desired First threshold.
  #This might be dependent on preferred.weight.
  #If admissible.counter==0, then the threshold is set to zero.
  Aspiration.threshold <- determine.threshold(bestSoFar,admissible.counter)

  #Draw candidates until one reaches the threshold.
  while( (bestSoFar < Aspiration.threshold) & (i < AspirationPlus.Max) ){
    i <- i+1
    drawing <- draw.candidate(state,"SWAP")
    neighborhood[[i]] <- drawing
    if( (drawing$evaluation$admissible)
      & (drawing$evaluation$slipped > bestSoFar)){
      bestSoFar <- drawing$evaluation$slipped
      bestIndex <- i
    }
  }
  if(verbose>=3){
    if(bestIndex > AspirationPlus.Min) print("AspirationPlus First reached.")
  }
}

```

```

    else print("AspirationPlus First not reached.")
  }

#Finally draw Plus more.
finish <- min(i+AspirationPlus.Plus, AspirationPlus.Max)
#If I have already reached the AspirationPlus.Plus,
# then the following should do nothing.
for(j in seq(from=(i+1),length.out=(finish-i))){
  drawing <- draw.candidate(state,"SWAP")
  neighborhood[[j]] <- drawing
  if( (drawing$evaluation$admissible)
      & (drawing$evaluation$slipped > bestSoFar)){
    bestSoFar <- drawing$evaluation$slipped
    bestIndex <- j
  }
}
if(verbose>=4) print(paste("Total number of neighbors tried:",
  length(neighborhood)))
AspirationPlus.howmany <- AspirationPlus.howmany + 1
AspirationPlus.evaluations <- AspirationPlus.evaluations
  + length(neighborhood)

#No admissible state was found. Then we either
# start a new oscillation (in the critical level)
# or start an ascent (in the low level).
if(bestSoFar==-1){
  change.mode("descent")
}else{
  state <- make.move(neighborhood[[bestIndex]])
  if(currentEvaluation$slipped > bestEvaluation$slipped){
    bestState <- state
    bestEvaluation <- currentEvaluation
    bestIteration <- iter
    if(verbose >= 3){
      print(paste("We have a new best state, being able to slip"
        ,bestEvaluation$slipped))
    }
  }
}
}

#Stopping criterion for staying in this mode.
#Note that we also stop if during an iteration,
# no admissible state is found.
if( (iter-bestIteration) > critical.noImprove){
  if(mode=="critical level"){
    #If oscillation.counter = 0 and I am at the critical level

```

```

    #, then there was no low level before yet.
    #Hence I have no good.state to relink with.
    if(oscillation.counter > 0){
        #Try path relinking with the best solution from the low level
        # and the best solution from the critical level.
        relinked <- pathRelinking(good.state,bestState,
            bestEvaluation$slipped)
        if(relinked$success){
            bestState <- relinked$state
            bestEvaluation <- relinked$evaluation
        }
    }
    change.mode("descent")
}
else change.mode("ascent")
}
else if(verbose >= 3) print(paste("Last improvement was before",
    (iter-bestIteration),"iterations."))

}#end of else

}#end of repeat

#bookkeeping
iter <- iter+1
if((iter %% saveFrequency == 0) & saveResults)
    save(list=saveList,file=paste(outputDir,"tabuPairs.RData",sep=""))

}#looping through oscillations

#####
#Second stage
#####
if(Stage2.allowed != "no"){
    if(verbose >= 1) print("Entering stage 2.")
    #I want to return returnedSolutions best solutions, so that there is some choice.
    if(returnedSolutions > substages*(substages-1)) returnedSolutions <- substages
    bestIndices <- order(sapply(elite.evaluations,function(x) x$slipped),
        decreasing=TRUE)[1:returnedSolutions]
    #Relink only the very best solutions, i.e. the last ones from the substages.
    IndicesToWorkWith <- seq(from=(oscillations+1),to=length(elite.solutions),
        by=(oscillations+1))
    l <- length(IndicesToWorkWith)
}

```

```

if(Stage2.allowed %in% c("full","relinking")){
  #try relinking elite.solutions - but not everyone with everyone,
  # that would take too long
  #There should be by now (oscillations +1 )*substages elite solutions
  # (note that the 0-th oscillation yields an elite solution as well).
  # I will therefore relink an elite solution for an oscillation
  # with the corresponding solution in the next substage
  # (relinking solutions from the same substage need not have the desired effect
  # since 1) there was already relinking within a substage taking place and
  # 2) the solutions might be simply too similar
  #to yield something new and interesting.
  #Store the find good solutions in the end of the elite.solutions list
  #As the criterion for relinking,
  # use the amount slipped by the third best elite solution.
  for(i in 1:l){
    if(i==1) j <- 1
    else j <- i+1
    sol1 <- elite.solutions[[IndicesToWorkWith[i]]]
    sol2 <- elite.solutions[[IndicesToWorkWith[j]]]
    #If the initiating state belongs to the best three,
    # then the criterion value is always exceeded.
    # This is not a problem per se...

    relinked <- pathRelinking(sol1,sol2,
      elite.evaluations[[bestIndices[3]]]$slipped)
    if(relinked$success){
      IndicesToWorkWith <- c(IndicesToWorkWith,(length(elite.solutions)+1))
      elite.solutions[[length(elite.solutions)+1]] <- relinked$state
      elite.evaluations[[length(elite.evaluations)+1]] <- relinked$evaluation
    }
  }
}
print("Done relinking!")

if(Stage2.allowed %in% c("full","moves")){
  #try applying good moves on all elite.solutions (whenever applicable)
  if(verbose>=5) print(paste("Elite add moves are:",elite.moves.add))
  if(verbose>=5) print(paste("Elite swap moves are:",elite.moves.swap))
  for(i in IndicesToWorkWith){
    state <- elite.solutions[[i]]
    #I need to do this again
    # (and not just take the evaluation from elite.evaluations)
    #because I need returned slipped rotations.
    #...even though I could probably just remember this directly
    evaluation <- evaluateState(records,waterTekort,state,"pair state",
      alpha,returnSlipped=TRUE,

```

```

        startsProvided = evaluation.startsProvided)
totalNumEvaluations.hard <- totalNumEvaluations.hard + 1
if(verbose >= 3) print(paste("Starting out with the ",
        i,"-th elite solution;",
        evaluation$admissible,evaluation$slipped))
for(move in elite.moves.add){
    #This loop could be somewhat vectorized -
    # I could determine the admissible moves for the given state beforehand.
    if(!(move$add %in% state)){
        #Try to perform the move.
        candidate <- c(state,move$add)
        evaluation.cand <- evaluateState(evaluation$rotations,waterTekort,
            candidate,"pair state",alpha,returnSlipped=TRUE,
            changedPairs=list(move$add),
            startsProvided = evaluation.startsProvided)
        if(verbose>=3) print(paste("Trying to perform the move: add:",
            move$add,";",
            evaluation.cand$admissible,evaluation.cand$slipped))
        if(evaluation.cand$admissible){
            if(verbose>=3) print(paste("Good move!",evaluation$slipped))
            state <- candidate
            evaluation <- evaluation.cand
        }
    }
}

for(move in elite.moves.swap){
    if(!(move$add %in% state) & (move$drop %in% state)){
        #Try to perform the move.
        candidate <- c(state,move$add)
        candidate <- setdiff(state,move$drop)
        evaluation.cand <- evaluateState(evaluation$rotations,waterTekort,
            candidate,"pair state",alpha,returnSlipped=TRUE,
            changedPairs=list(move$add,move$drop),
            startsProvided = evaluation.startsProvided)
        if(verbose>=3) print(paste("Trying to perform the move: add:",
            move$add,"",
            drop:",move$drop,";",evaluation.cand$admissible,
            evaluation.cand$slipped))
        #If the move leads to a worse state,
        # then I don't want to incorporate it.
        if(evaluation.cand$admissible &
            (evaluation.cand$slipped > evaluation$slipped)){
            if(verbose>=3) print(paste("Good move!",evaluation$slipped))
            state <- candidate
            evaluation <- evaluation.cand
        }
    }
}

```

```

    }
  }
  #Take the final (improved) state and save it
  #in place of the old elite solution.
  evaluation$rotations <- NULL
  elite.solutions[[i]] <- state
  elite.evaluations[[i]] <- evaluation
}
}
}

#Just for sure.
for(i in 1:length(elite.evaluations)){
  elite.evaluations[[i]]$rotations <- NULL
}
#Finally choose the best ones.
#It can happen that some of them correspond to the same state.
bestIndices <- order(sapply(elite.evaluations,function(x) x$slipped),
  decreasing=TRUE)[1:returnedSolutions]
best.solutions <- elite.solutions[bestIndices]
best.evaluations <- elite.evaluations[bestIndices]

if(verbose >= 1) print("Leaving the tabu search.")
return(list(states=best.solutions,evaluations=best.evaluations,
  states.check = elite.solutions, evaluations.check = elite.evaluations,
  numEvaluations=totalNumEvaluations, numEvaluations.hard=totalNumEvaluations.hard,
  AspirationPlus = c(howmany=AspirationPlus.howmany,
  evaluations=AspirationPlus.evaluations)))
}

```

### B.3 Grouping times

This is the code leading to results described in Section 3.3.5, including fitting the regression models, but excluding most of diagnostics.

```

bigGroupNumber <- 100
smallGroupNumber <- 50
frequencyBound <- 50
listModels <- list(); listGroups <- list()
library("lawstat")
for(dst in topDST){
  obs <- subset(rotations, DSTOut==dst & !is.na(NextTankedAmount) & NoTanking==FALSE)
  ##Make several groups based on hour.
  print(dst)
}

```

```

bigTimes <- as.numeric(names(which(sort(table(obs$hour),
  decreasing=TRUE)>bigGroupNumber)))
#Make the cores of each big group
bigGroups <- list()
for(i in 1:length(bigTimes)){
  group <- letters[i]
  bigGroups[[group]] <- c(bigTimes[i])
}
for(group in names(bigGroups)){
  #The core lies there.
  bigGroups[[group]]
  core <- bigGroups[[group]]
  #Go downwards/upwards
  for(direction in c(-1,1)){
    i <- core
    while(!is.na(table(obs$hour)[as.character(i)]) &
      table(obs$hour)[as.character(i)] > smallGroupNumber){
      i <- i + direction
      bigGroups[[group]] <- c(bigGroups[[group]],i)
    }
  }
}
#Merge the corresponding big groups.
i <- 1
while(i < length(bigGroups)){
  firstGroup <- bigGroups[[i]]
  for(group in names(bigGroups[(i+1):length(bigGroups)])){
    if(identical(sort(firstGroup),sort(bigGroups[[group]])) ) ){
      bigGroups[[group]] <- NULL
      #print(bigGroups)
    }
  }
  i <- i+1
}
#Finally get rid of conflicting "small" hours.
conflicts <- as.numeric(names(table(unlist(bigGroups))
  [table(unlist(bigGroups))>1]))
for(group in names(bigGroups)){
  bigGroups[[group]] <- setdiff(bigGroups[[group]],conflicts)
}
bigGroups[["z"]] <- setdiff(0:23,unlist(bigGroups))

obs$group <- "z"
for(group in names(bigGroups)){
  obs[obs$hour %in% bigGroups[[group]],"group"] <- group
}
obs$group <- as.factor(obs$group)

```

```

zal <- bigGroups
bigGroups <- bigGroups[order(table(obs$group)[-(length(bigGroups)+1)],decreasing=TRUE)]
#Relabel the levels, so that the most "populated" is the first one
#In the way the data was constructed, it should be unlikely
# that the z group would be the largest one.
#obs <- subset(obs, group != "z")
levels(obs$group)
obs$group <-relevel(obs$group,names(which.max(table(obs$group))))

model <- lm(NextTankedAmount ~ group , data=obs)
print(model)
print(table(obs$group))
#Test that all groups have the same variance with a suitable test.
if(min(table(obs$group))>1){
  print(levene.test(obs$NextTankedAmount,obs$group))
}
listModels[[dst]] <- model
listGroups[[dst]] <- bigGroups
}

```

# Bibliography

- [1] Digital Library of Mathematical Functions. Release date 2012-03-23. . <http://dlmf.nist.gov/8.11.E2>.
- [2] Digital Library of Mathematical Functions. Release date 2012-03-23. . <http://dlmf.nist.gov/8.2>.
- [3] M. Bijvank, M. Dobber, M. Soomer, Q. Botton, E. de le Court, J.-C. V. den Schrieck, M. de Viron, M. Cisneros-Molina, K. Schmitz, R. van der Hofstad, E. Jochemsz, T. Mussche, M. Summer, M. Hoekstra, J. Mulder, and M. Paelinck. Planning drinking water for airplanes. Study Group Mathematics with Industry report, February 2005.
- [4] M. Bolt. Personal communication.
- [5] E. Bovenkerk. Gestructureerd slippen. Internship report, July 2002.
- [6] M. B. Brown and A. B. Forsythe. Robust tests for the equality of variances. *Journal of the American Statistical Association*, 69(346):pp. 364–367, 1974.
- [7] N. Cesa-Bianchi and G. Lugosi. *Prediction, Learning, and Games*. Cambridge University Press, 2006.
- [8] R. B. D’Agostino and M. A. Stephens, editors. *Goodness-of-fit Techniques*. Marcel Dekker, 1986.
- [9] E. de Klerk, C. Roos, and T. Terlaky. Nonlinear Optimization Lecture Notes. "<http://www.cas.mcmaster.ca/~cs4te3/book/>"; accessed 5/4/2012, August 2004.
- [10] C. Enders. *Applied Missing Data Analysis*. Guilford Press, April 2010.
- [11] L. Fahrmeir. Asymptotic likelihood inference for nonhomogeneous observations. *Statistical Papers*, 28(1):81–116, 1987.
- [12] J. J. Faraway. Practical Regression and Anova using R. Accessed on 23/1/2012 from "<http://www.maths.bath.ac.uk/~jjf23/book/>", July 2002.
- [13] F. Glover and M. Laguna. *Tabu Search*. Kluwer Academic Publishers, 1998.
- [14] B. Hansen and K. West. Generalized Method of Moments and Macroeconomics. *Journal of Business and Economic Statistics*, 20(4):460–469, October 2002.
- [15] L. P. Hansen. Large Sample Properties of Generalized Method of Moments Estimators. *Econometrica*, 50(4):1029–1054, July 1982.

- [16] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning, Data Mining, Inference, and Prediction*. Springer New York, 2009.
- [17] D. F. Heitjan. Inference from Grouped Continuous Data: A Review. *Statistical Science*, 4(2):164–183, 1989.
- [18] K. Lange. *Numerical Analysis for Statisticians*. Springer New York, 2010.
- [19] R. Lougee-Heimer. The Common Optimization INterface for Operations Research. *IBM Journal of Research and Development*, 47(1):57–6–6, January 2003.
- [20] J. H. McDonald. *Handbook of Biological Statistics*, chapter Kruskal–Wallis test and Mann–Whitney U test, pages 165–172. Sparky House Publishing, 2nd edition, 2009. Accessed from <http://udel.edu/~mcdonald/statkruskalwallis.html>.
- [21] A. Meister. *Deconvolution Problems in Nonparametric Statistics*. Springer, 2009.
- [22] W. Michiels, J. Korst, and E. Aarts. *Theoretical Aspects of Local Search*. Springer, 2007.
- [23] P. Moschopoulos. The distribution of the sum of independent gamma random variables. *Annals of the Institute of Statistical Mathematics*, 37:541–544, 1985. 10.1007/BF02481123.
- [24] S. Nadarajah. A review of results on sums of random variables. *Acta Applicandae Mathematicae*, 103:131–140, 2008. 10.1007/s10440-008-9224-4.
- [25] R. H. E. Pérez. Drinking Water Estimation for Aircraft. Master’s thesis, Technische Universiteit Eindhoven, March 2006.
- [26] A. N. Pettitt. A Two-Sample Anderson–Darling Rank Statistic. *Biometrika*, 63(1):pp. 161–168, 1976.
- [27] R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2011. ISBN 3-900051-07-0.
- [28] F. Scholz. *R Package 'adk'*. Version 1.0-2.
- [29] F. W. Scholz and M. A. Stephens. K-Sample Anderson-Darling Tests. *Journal of the American Statistical Association*, 82(399):918–924, September 1987.
- [30] S. J. Sheather. *A Modern Approach to Regression with R*. Springer New York, 2009.
- [31] M. A. Stephens. EDF Statistics for Goodness of Fit and Some Comparisons. *Journal of the American Statistical Association*, 69(347):730–737, September 1974.
- [32] J. van der Laan and L. Kuijvenhoven. Imputation of rounded data. Discussion paper 201108, Statistics Netherlands, 2011.
- [33] A. van der Vaart. *Asymptotic Statistics*. Cambridge University Press, 1998.
- [34] A. van der Vaart. Algemene statistiek. Lecture notes, January 2008.
- [35] L. Wasserman. *All of Nonparametric Statistics*. Springer, 2006.