Delft University of Technology
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft Institute of Applied Mathematics

Noise minimization on houses around airports

A thesis submitted to the
Delft Institute of Applied Mathematics
in partial fulfillment of the requirements

for the degree

MASTER OF SCIENCE
in
APPLIED MATHEMATICS

by

Teun Michiel Louis Janssen

Delft, the Netherlands
October 2013

MSc THESIS APPLIED MATHEMATICS

"Noise minimization on houses around airports"

Teun Michiel Louis Janssen

Delft University of Technology

Daily supervisor

Prof. dr. G. Schäfer

Responsible professor

Prof. dr. K.I. Aardal

Other thesis committee members

Dr. M.J.A. van Eenige

Dr. R.J. Fokkink

October, 2013

Delft

Abstract

This thesis was a combined project of the CWI (the national research institute for mathematics and computer science in the Netherlands) and the NLR (National Aerospace Laboratory of the Netherlands). In this thesis we examine noise pollution around airports. Given an airport we will consider the problem maximizing the number of flights, while minimizing the number of houses suffering more than the threshold amount of noise pollution. We will formulate this problem as a multi-objective optimization problem. We will look at the computational complexity of the problem and its approximability. Using the concept of Pareto optimality we will consider methods to rewrite our problem to a single objective optimization problem. Using the methods and the computational complexity and approximability results obtained, we will construct algorithms to solve them problem. Finally we will use these algorithm on problem instances to give insights in their performance.

Preface

The master thesis marks the end of my study and my time as a student at the TU Delft. Seven years ago, I started my study mathematics. I want to thank the people who supported me during these years. I want to thank Chris, Elwin, Jacob, Jarno and Paolo, who I live with most of these years at the Markt in Delft. I would like to thank Rens en Mark with whom I could share my love for mathematics, games and other things.

I want to thank my brother Reinier and my little sister Marieke, who always showed interest in what I was doing. I want to thank my girlfriend Stefanie, who was a great support and kept my going during my years as a Master student. But I special word of thanks goes to my mother, who was always there when needed. No matter what I was doing, she was interested and more than happy to offer a helping hand. Even if she would have to drive all the way to Osnabrück.

I did my master thesis as a combined project of the CWI (the national research institute for mathematics and computer science in the Netherlands) and the NLR (National Aerospace Laboratory of the Netherlands). I would like to thank them for the educational environment and the interesting conversations we had during the last nine months. A special thanks goes out to my roommates Anarchyros and Bart at the CWI, who always were ready to help me when needed and enjoy the little things in life, while desired. I also want to thank Sander at the NLR for helping me with obtaining the different problem instances and making them error free. I would also want to thank Christiaan, Jacob, Jarno and Rens for commenting on me and my writing.

Lastly, I would like to thank my three supervisor Guido, Michel and Theo. I would like to thank Michel for his interest in my work, always ready to help me or proof read some part of my thesis, as “optimally as possible”. I would like to thank Theo. Although Markenesse is quite far, it never stopped you from helping and keeping a watchful eye on my thesis and its progress, either by email, phone or if needed in person. My last thanks goes out to Guido, who during these nine months showed my the beauty of discrete optimization. His enthusiasm for the problem we worked on inspired me during the past nine months and I always looked forward to our meetings. I felt privileged working with all three of them.

Teun Michiel Louis Janssen
Delft, the Netherlands
September 27, 2013

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Air transport has seen an enormous increase in demand over the last decades and forecast suggest an increase of 2.3% for the next 7 years [10]. This growth of air transport is likely to be accompanied with an increase of enviromental pollution. This thesis focuses on the noise pollution. To fulfill the increasing demand of air transport and taking into account its effects on noise pollution, it is important to schedule flight movements efficiently, i.e., scheduling enough flights to cope with the demand while limiting the noise pollution suffered.

Currently the Dutch government attempts to limit the noise pollution by enforcing legislations on air traffic. One of these legislations prescribes a maximum level of yearly noise pollution outside a predefined zone around an airport. Inside this zone there is no legal limit to the noise pollution. The general idea behind this legislation is that, if one is living outside the zone, the noise pollution will not affect ones quality of life. However even outside the zone one can experience nuisance due to noise produced by aircrafts [9].

Thus from an environmental point of view one wants to keep noise pollution at a minimum, while from an economic and operational point of view one wants to schedule an amount of flight movements in accordance with demand. These are two conflicting objectives and a decision maker has to strike a golden mean between the two. In the thesis by Jonker [16] this is modeled as a multi-objective optimization problem. In this problem one tries to maximize the number of flights, while also maximizing the gap between the legislated maximum level and the total noise pollution of the flight movements on the zone around the airport. A more formal definition of the problem can be found in Appendix A.1.

However, since the goal of the legislator is to keep the noise pollution suffered by people living in the neighborhood of an airport acceptable, we chose to focus on the houses in the vicinity of the airport. So opposed to Jonker, for noise pollution, our objective will be to minimize the number of houses suffering more than an acceptable amount of noise pollution. This objective does more justice to the goal of the legislator. The multi-objective optimization problem this thesis considers becomes then: maximizing the number of flights, while minimizing the number of houses suffering more than an acceptable amount of noise pollution given a certain airport. We will refer to this problem as the *route scheduling problem*. In this multi-objective optimization problem we will take into account the effects of the legislation that defines a zone around an airport.

The goals of this thesis are to find the different features of the route scheduling problem, to find the connection it has with other known problems and to find efficient and effective algorithms to solve the problem.

In line with our goals we will begin by identifying the features of the problem and connections it has to know problems. Then based on these features and connections we will devise algorithms to solve the problem. After obtaining these, we will use them to solve problem instances and look at their performance.

The thesis is organized as follows. In Chapter 2 we will review the mathematical theory needed for the remainder of the thesis. In Chapter 3 we will describe the problem in more detail and describe it mathematically. In Chapter 4 we will look at the features of the problem in terms of and establish connection to known problems. In Chapter 5 we will look at Pareto optimality, which is concept used to describing optimality in multi-objective optimization problems. Then in Chapter 6 we will describe the algorithm we use to solve the problem based on the results of Chapter 4. In Chapter 7 we will consider three different problem instances. We will solve these problems using our algorithms and analyze their performance. Finally in chapter 8 we will present the conclusion and our recommendations.

Chapter 2

Preliminaries

In this chapter we will consider some basics definitions and concepts, which we will use in subsequent chapters. First we will shortly review the basic concepts of *linear programming* and sensitivity analysis in linear programming. This theory is based on the book by Bertsimas and Tsitsiklis [5]. We will use this theory devising our algorithm and solving our problem. Then, we will consider the basics of complexity theory and approximation algorithms. The theory found for these two topics is based on the books by Papadimitriou and Steiglitz [22] and Vazirani [26]. The theory found in these sections we be used in Chapter 4 to analyze our problem.

2.1 Linear programming

Linear programs are optimization problems, where one tries to minimize (or maximize) a linear *objective function* (sometimes called cost/profit function) given certain requirements to which it has to comply. These requirements are linear equalities or inequalities. A linear program in standard form is defined as

$$\begin{array}{ll} \text{minimize} & c^T x \\ \text{subject to} & Ax \geq b \ . \\ & x \geq 0 \end{array} \tag{2.1}$$

Here $c = (c_1, \dots, c_n)^T \in \mathbb{R}^n$ is the cost vector, A the coefficient matrix of size $m \times n$, i.e., $A \in \mathbb{R}^{m \times n}$ and $b = (b_1, \dots, b_m)^T \in \mathbb{R}^m$ the requirement vector. We assume that the columns of A are linearly independent. The vector x contains the *variables* $x_1, \dots, x_n$. A vector x that satisfies all the linear constraints is called a *feasible solution.* A feasible solution x^* is called an *optimal solution* if it is feasible and it minimizes the objective function, that is $c^T x^* \leq c^T x$ for all feasible solutions x. A linear program for which such a x^* exist is called *bounded.* A linear program is called *unbounded* if for every constant $\omega \in \mathbb{R}$ there exists a feasible solution x such that $c^T x \leq \omega$.
The set $S_{LP} := \{x : Ax \leq b, x \geq 0\}$ is called the *feasible set* and is a *polyhedron*, i.e., the intersection of a finite set of half spaces. It has the property that it is *convex*, i.e., $\forall x, y \in S_{LP}, \forall \lambda \in [0, 1]$ it holds that $\lambda x + (1 - \lambda) y \in S_{LP}$. If a polyhedron is bounded it is called a *polytope.* A point $x \in S_{LP}$ is an *extreme point* of S_{LP} if x cannot be written as a convex combination of two other points in S_{LP}. If $S_{LP} = \emptyset$ the linear program is called *infeasible.*
Any extreme point of a polyhedron in $\mathbb{R}^n$ can be written as an intersection of n linearly independent hyperplanes, that are bounding the polyhedron. In the case of a linear program, these hyperplanes are defined by the linear constraints. A constraint is called *tight* or *active* in

a point $x \in S_{LP}$ if the constraint is satisfied with equality. In an extreme point of the LP n constraints are satisfied with equality. Of these n constraints m are from $Ax \leq b$. The other $n - m$ constraints are from non-negativity constraints. Such an extreme point is called a *basic feasible solution.*
In a basic feasible solution we distinguish between *basic variables* and *non-basic variables.* In our basic feasible solution we have $n-m$ of the non-negativity constraints satisfied with equality. The $n - m$ variables associated with these constraints are the non-basic variables. The other m variables are the basic variables. Let $B(1), \ldots, B(m)$ denote their indices and let A_i denote the ith column of the matrix A. Then the *basis matrix* B is defined as

$$B := \begin{bmatrix} | & | & & | \\ A_{B(1)} & A_{B(2)} & \cdots & A_{B(m)} \\ | & | & & | \end{bmatrix}.$$

Since the columns of A are linearly independent, it holds that $Bx = b$ and that the values of x on the indices $B(1), \ldots, B(m)$ are determined uniquely by $x_B = B^{-1}b$.
In many optimization algorithms one starts with a feasible solution and after finding one, one searches in the neighborhood of this feasible solution for a new one with an improved cost. If the algorithm cannot find a new feasible solution it has obtained a local optimum and terminates. In linear programming we are minimizing a convex function over a convex space, therefore, a local optimum is a global optimum. Thus, using an algorithm as described above will give a global minimum. In order to find a new solution with improved cost, most algorithms use the concept of *reduced cost.*

Definition 2.1.1. Let x be a basic solution, let B be the associated basis matrix and let c_B be the vector of costs of the basic variables, i.e., $c_B = (c_{B(1)}, \ldots, c_{B(m)})^T$. Then, we define, for each $j \in \{1, \ldots, n\}$, the *reduced cost* of variable x_j as

$$\bar{c_j} = c_j - c_B^T B^{-1} A_j.$$

Given a feasible solution the objective function can be improved by adding a basis variable for which $\bar{c_j} < 0$ and removing another basis variable accordingly. To assert that a basic solution is optimal, we need it to be feasible and to have nonnegative reduced costs.

Definition 2.1.2. A basic solution is optimal if its basis matrix B satisfies the following *optimality conditions*:

1. It is feasible, i.e., $B^{-1}b \geq 0$, and
2. It has nonnegative reduced cost, i.e., $\bar{c}^T = c^T - c_B^T B^{-1} A \geq 0$.

2.1.1 Duality theory

One topic often considered in combination with linear programming is *duality.* In duality theory one considers the original optimization problem (known as the *primal* problem) and constructs an associated *dual* optimization problem. This dual problem gives a lower bound to the solution of the primal problem. The linear program (2.1) has the following dual problem (or dual program)

$$\begin{array}{ll} \text{maximize} & p^T b \\ \text{subject to} & p^T A \leq c \\ & p \geq 0 \end{array} \tag{2.2}$$

Here $p = (p_1, \dots, p_m)^T \in \mathbb{R}^m$ is the *dual* vector. There are two import concepts in duality theory for linear programming. The first one is that he dual of the dual problem is the primal problem. Thus the primal and the dual problem form a pair. The second one is that the dual problem gives a lower bound to the solution of the primal problem. This relation will be expressed in the following two theorems.

Theorem 2.1.1 (Weak duality)**.** If x is a feasible solution to the primal problem and p is a feasible solution to the dual problem, then

$$p^T b \leq c^T x.$$

Proof. Suppose that x and p are a primal and a dual feasible solution. Then because of feasibility and because $x \geq 0$ and $p \geq 0$, it holds that

$$(c^T - p^T A)x \geq 0$$

and

$$p^T (Ax - b) \geq 0$$

Thus

$$0 \leq (c^T - p^T A)x + p^T (Ax - b) = c^T x - p^T Ax + p^T Ax - p^T b = c^T x - p^T b.$$

□

Theorem 2.1.2 (Strong duality)**.** If a linear program has an optimal solution x^*, the dual also has an optimal solution p^* and the objective value of the two problems is equal, $(p^*)^T b = c^T x^*$.

Proof. We consider the dual pair (2.1) and (2.2). Let x^* be an optimal basic feasible solution to the primal problem and let B be the corresponding basis matrix. Because x^* is optimal, it holds that $c^T - c_B^T B^{-1} A \geq 0$. Now, take $p^T = c_B^T B^{-1}$. Then, it follows that

$$0 \leq c^T - c_B^T B^{-1} A = c^T - p^T A.$$

Hence p is feasible. Furthermore $c^T x = c_B^T x_B = c_B^T B^{-1} b = p^T b$ and thus, the two objective values are equal. This, together with weak duality, proves the theorem. □

Note that it is a consequence of weak duality that if the primal problem is infeasible, then the dual is unbounded, and vice versa. Furthermore, by the proof of strong duality, we know that if p^* is an optimal solution to the dual, it holds that $p^* = B^{-1} b$, where B^{-1} is the basis matrix of the optimal primal solution x^*.

2.1.2 Sensitivity analysis

It is interesting to look at the dependence of the optimal solution x^* on A, b and c. This is an important issue in practice because often one does not know the exact values of certain elements in A, b and c. We will look thus at some methodologies used in sensitivity analysis for linear programming problems.
After a change of these elements one considers whether or not the original optimal solution is still feasible and optimal for the new linear program. In other words, one determines whether or not the optimality conditions are still satisfied. In our case we want to understand what happens if we change the requirement vector b. Suppose that we only change one entry in the vector b. The vector b is changed to $b + \delta e_i$, where e_i is the ith unit vector. Since b does not

influence the optimality conditoin, we only need to check the feasibility condition for the new linear program is still satisfied

$$B^{-1}(b + \delta e_i) \geq 0. \tag{2.3}$$

We see that δ is only multiplied with the ith column of B^{-1}. Let $\beta = \{\beta_1, \ldots, \beta_m\}$ be the ith column of B^{-1}, then we can reformulate (2.3) as

$$x_{B(j)} + \delta\beta_j \geq 0, \quad j = 1, \ldots, m \ .$$

From this it follows that in order for B to also be the optimal basis for the new linear program, we need that

$$\max_{\{j:\beta_j>0\}} \frac{-x_{B(j)}}{\beta_j} \leq \delta \leq \min_{\{j:\beta_j<0\}} \frac{-x_{B(j)}}{\beta_j}.$$

If δ falls in this range, the solution remains feasible and the basis does not change. If for δ the above equation does not hold, the basis is no longer feasible and thus, no longer the optimal solution. Given that the above equation does hold, the optimal cost is given by

$$c_B^T B^{-1}(b + \delta e_i) = p^T b + \delta p_i, \tag{2.4}$$

where p is the optimal dual solution associated with the basis B.

2.1.3 Integer linear programming

Integer linear programming is closely related to linear programming. In integer linear programming one does not only require the variables to be positive, one also requires them to be integer, sometimes even binary. Let $\mathbb{Z}_+ = \{z \in \mathbb{Z} : z \geq 0\}$. Then we can define an integer program as the follows

$$\begin{array}{ll} \text{minimize} & c^T x \\ \text{subject to} & Ax \geq b \\ & x \in \mathbb{Z}_+^n \end{array} \tag{2.5}$$

Where again $c = (c_1, \ldots, c_n)^T \in \mathbb{R}^n$ is the cost vector, A the coefficient matrix of size $m \times n$ and $b = (b_1, \ldots, b_m)^T \in \mathbb{R}^m$ the requirement vector.

2.2 Complexity theory

In computational complexity theory one studies the resources, i.e., the memory and time, needed to solve a computational problem. One considers the relations between different problems and at the relations between different classes of problems. These are called *complexity classes*. These classes consist of decision problems. A *decision problem* is an arbitrary yes-or-no question on an infinite set of inputs. More formaly we have the following definition

Definition 2.2.1. A *decision problem* Π is given by a set of of instances $\mathcal{I}$. Each instance $I \in \mathcal{I}$ specifies the following

- a set $\mathcal{F}$ of feasible solutions for I;
- a cost function $c : \mathcal{F} \to \mathbb{R}$;
- a constant k.

Given an instance I the objective is to determine whether or not there exist a feasible solution $F \in \mathcal{F}$, such that $c(F) \leq k$. If such a solution exist, we will call I a *yes-instance*, if such a solution does not exist, we will call I a *no-instance*.

Note that in our case, we may want to write our multi-objective optimization problems as decision problems. If we do this we will have multiple cost functions $c_1, \ldots, c_s$ and multiple $k_1, \ldots k_s$. We then only have a *yes-instance* if $c_i(F) \leq k_i, \forall i \in \{1, \ldots, s\}$.

The two main classes we will consider are the classes $\mathcal{P}$ and $\mathcal{NP}$.

Definition 2.2.2. The complexity class $\mathcal{P}$ consists of all decision problems for which there exist an algorithm that for every instance $I \in \mathcal{I}$ can determine whether I is a yes- or a no-instance in polynomial-time, i.e., the time it takes to compute a solution to the decision problem, is polynomial in the input size.

The complexity class $\mathcal{P}$ contains, for example, the *shortest path* problem and linear programming. Now suppose we have yes-instance $I \in \mathcal{I}$ for a given decision problem Π, then we say that F is a *certificate* for I if $F \in \mathcal{F}$ and $c(F) \leq k$. Every yes-instance I must have at least one certificate, since otherwise it would not be a yes-instance.

Definition 2.2.3. The complexity class $\mathcal{NP}$ consist of all decision problems with the property, that every yes-instance $I \in \mathcal{I}$ admits a certificate that can be verified to be a yes-instance in polynomial time.

The complexity class $\mathcal{NP}$ contains, for example, the *set cover* problem, the *traveling salesman* problem and integer linear programming. Although no proof exists, it is considered unlikely that for all problems in $\mathcal{NP}$ there is a polynomial time algorithm solving it, i.e., $\mathcal{P} \neq \mathcal{NP}$. But it is true that $\mathcal{P} \subset \mathcal{NP}$. Another important subclass of $\mathcal{NP}$ is the class $\mathcal{NP}$-complete. This is the subclass of the most difficult problems in $\mathcal{NP}$. To define it, we first need to define a *polynomial-time reduction.*

Definition 2.2.4. A *polynomial-time reduction* from a decision problem Π_1 to another decision problem Π_2 is a function $\phi : \mathcal{I}_1 \to \mathcal{I}_2$ that maps every instance $I_1 \in \mathcal{I}_1$ to an instance $I_2 = \phi(I_1) \in \mathcal{I}_2$ of Π_2 such that

1. the transformation is done in time that is polynomially bounded in the input size of I_2;
2. I_1 is a yes-instance of Π_1 if and only if $I_2 = \phi(I_1)$ is a yes-instance of Π_2.

Often we will state that if there is a polynomial time reduction from Π_1 to Π_2 that Π_1 can be *reduced to* Π_2. Note that if a polynomial-time algorithm for Π_2 is known and Π_1 can be reduced to Π_2, then we can also solve Π_1 in polynomial-time. Furthermore if Π_1 can be reduced to Π_2 and Π_2 can be reduced to Π_3, then also Π_1 can be reduced to Π_3. In other words polynomial-time reductions are transitive.

Definition 2.2.5. The class $\mathcal{NP}$-complete consist of all problems Π for which it holds that

1. $\Pi \in \mathcal{NP}$, and
2. for every problem $\bar{\Pi} \in \mathcal{NP}$ there is a polynomial-time reduction form $\bar{\Pi}$ to Π.

Problems who possesses the second property are called $\mathcal{NP}$-hard decision problems. At first sight, the second property seems very hard to prove, since one has to find a polynomial-time reduction for *all* problems in $\mathcal{NP}$. However, because of the transitivity of polynomial-time reduction, one can prove that a decision problem Π is $\mathcal{NP}$-hard by reducing a $\mathcal{NP}$-complete problem $\bar{\Pi}$ to Π. Thus, in order to prove that a decision problem Π is in $\mathcal{NP}$-complete one needs to show that

1. It is in $\mathcal{NP}$, i.e., every yes-instance $I \in \mathcal{I}$ admits a certificate that can be verified to be a yes-instance in polynomial time.
2. It is $\mathcal{NP}$-hard, i.e., there is a problem $\Pi_0 \in \mathcal{NP}$-complete, which can be transformed in polynomial time to Π.

Furthermore, we will consider two subclasses of $\mathcal{NP}$-complete. *Weakly $\mathcal{NP}$-complete* is a subclass of $\mathcal{NP}$-complete, where problems can be solved in pseudo-polynomial time. This means that the problem can be solved in time bounded by a polynomial function in the input size and the inputs numerical values. The length of storage needed for the numerical values do not have to be polynomial[1]. A problem is *strongly $\mathcal{NP}$-complete*, if it remains $\mathcal{NP}$-complete even if all numerical values in the input are bounded by some polynomial.

2.3 Approximation algorithms

Because, in general, it is believed that $\mathcal{P} \neq \mathcal{NP}$, there is very little hope that we can find efficient algorithms to solve decision problems, which are $\mathcal{NP}$-complete. Therefore we look at different kind of algorithms to solve these $\mathcal{NP}$-complete problems.

- *Exponential algorithms*: These algorithms solve the problem, but in the worst case can take exponential running time.
- *Approximation algorithms*: These are algorithms that compute a solution efficiently and with a certain performance guarantee. In our case we consider algorithms that run in polynomial time and for which we can prove that they are 'close' to the optimal solution.
- *Heuristics*: In this category are all the algorithms that solve a problem, but cannot give any formal guarantee on the quality of the solution. Heuristics are interesting when a problem cannot be approximated well and known exact algorithms take to much computation time.

For all three types of algorithms, it holds that in practice some may perform very well. The most notorious is the simplex algorithm for solving linear programming. In theory the simplex algorithm can run for exponential time in the input size, but in most practical instances it solves a linear program quite fast.
For an exponential algorithm it is required that it computes an optimal feasible solution, while the time needed to compute this solution is not an issue. A heuristic algorithm only requires a feasible solution, while one does not consider optimality or computation time. A heuristic that is neither fast nor performs well will not be considered. For an approximation algorithm one needs a time guarantee and a performance guarantee. We will consider approximation algorithms that run in polynomial time and are not more than a certain factor from the optimal solution. More formally, we have the following definition:

Definition 2.3.1. An algorithm for a problem Π is an α-approximation algorithm ($\alpha \geq 1$) if for every instance I it computes, in polynomial time, a solution such that for the objective value of the problem $\texttt{ALG}(I)$, it holds that

$$\texttt{ALG}(I) \leq \alpha \texttt{OPT}(I) \qquad \text{if } \Pi \text{ is a minimization problem,}$$

[1]For example, take the Knapsack problem which can only be solved in pseudo-polynomial time $O(nW)$, where n is the number of items and W the size of the knapsack. For example if W is $2^{(2^n)}$ then it takes up 2^n bits to represent this number, while the running time is $O(n2^{(2^n)})$ which is exponential in n.

$$\texttt{ALG}(I) \geq \frac{1}{\alpha}\texttt{OPT}(I) \qquad \text{if } \Pi \text{ is a maximization problem,}$$

where $\texttt{OPT}(I)$ is the optimal solution for instance I. α is called the *approximation ratio* of the algorithm.

We see from the definition that we desire an approximation ratio as close to 1 as possible.

Inapproximability

For some problems we know an upper bound on how well they can be approximated within polynomial time. For example, it was proved by Håstad [14], that the *maximum independent set* problem cannot be approximated better than within a factor of $n^{1-\epsilon}$, for any $\epsilon > 0$, unless $\mathcal{NP}$-hard problems have randomized polynomial algorithms. Such a result is called an *inapproximability* result. Thus, given an $\mathcal{NP}$-complete problem Π_1 an inapproximability result states that there is a function $f(n)$, dependent on the instance size n, such that for the approximation ratio α_1 of Π_1, it must hold that $\alpha_1 > f(n)$. Inapproximability results are often proven using *approximability preserving reductions.* One of the best known approximability preserving reductions is the *L-reduction.*

Definition 2.3.2. Suppose we are given two $\mathcal{NP}$-complete problems, Π_1 and Π_2, and a polynomial-time transformation $\phi : \mathcal{I}_1 \to \mathcal{I}_2$ that maps every instance I_1 of Π_1 to an instance I_2 of Π_2. Here $\mathcal{I}_1$ and $\mathcal{I}_2$ are the collections of all instances of the given problems. Then, ϕ is an *L-reduction* from Π_1 to Π_2 if there are constants $\alpha > 0$ and $\beta > 0$ such that, for every instance I_1, it holds that

1. $\texttt{OPT}_2(\phi(I_1)) \leq \alpha\texttt{OPT}_1(I_1)$
2. For every feasible solution F_2 of $\phi(I_1)$, with objective value $c_2(\phi(I_1), F_2) = \gamma_2$, we can find a feasible solution F_1 of I_1, with $c_1(I_1, F_1) = \gamma_1$, such that $|\texttt{OPT}_1(I_1) - \gamma_1| \leq \beta|\texttt{OPT}_2(\phi(I_1)) - \gamma_2|$ in polynomial-time.

Where $\texttt{OPT}_1(I)$ and $\texttt{OPT}_2(I)$ are the optimal values of the two problems given an instance I and where $c_1(I, F)$ and $c_2(I, F)$ are the objective function values given instances I and yes-instance F for the two problems.

Now suppose we are given a L-reduction from problem Π_1 to problem Π_2 and suppose that for the problem Π_2 we have a polynomial-time algorithm that approximates every instance I_2 of Π_2 to within a factor θ. Then, from the L-reduction, we know that Π_1 can also be approximated by a polynomial time algorithm within a factor $\alpha\beta\theta$. On the other hand, suppose that we know that problem Π_2 cannot be approximated within a factor τ. Then from the L-reduction we know that Π_1 cannot be approximated within a factor $\alpha\beta\tau$ [23]. If for an L-reduction, we have $\alpha = \beta = 1$ then, we have a *cost preserving transformation.*

Most inapproximability results are obtained in this way by doing a long sequence of reductions. Often the first reduction in this sequence is the famous *PCP-theorem* [3, 4, 12]. Without going in too much detail, the PCP-theorem states that $\mathcal{NP} = \text{PCP}(O(\log n), O(1))$. Here $\text{PCP}(O(\log n), O(1))$ is the class containing all decision problems that have probabilistically checkable proofs that can be verified in polynomial time ,using at most $O(\log n)$ random bits and by reading at most $O(1)$ bits of the proof.
One of its implications is on *maximum 3 satisfiability* problem (MAX-3SAT). In MAX-3SAT one is given a set of clauses with at most three variables. One tries to find a truth assignment,

such that the maximum amount of clauses is satisfied. The PCP-theorem implies that for some $\epsilon > 0$ it is $\mathcal{NP}$-hard to approximate MAX-3SAT within a factor of $1 - \epsilon$. The result by Håstad [14], that the *maximum independent set* problem cannot be approximated better then within a factor of $n^{1-\epsilon}$, for any $\epsilon > 0$, unless NP-hard problems have randomized polynomial algorithms, is also due to the PCP theorem.

Chapter 3

The model

In this chapter we will begin by describing the problem mentioned in Chapter 1 in more detail. After this we will describe it formally as a multi-objective optimization problem. We will find two formulations, which we will use in subsequent chapters to analyze and solve the problem.

3.1 Problem description

We consider an airport. For this airport we are given the location of its runways and the locations of houses in its neighborhood. Furthermore we are given a collection of flight routines. Each flight routine is a unique combination of the following:

- The aircraft type associated with the flight movement.
- The runway used and in what direction.
- The flight procedure the aircraft is following (which incorporates its ascend or descend, its speed profile and the accompanying thrust settings.)
- The flight trajectory it follows, i.e., the coordinates a flight is traversing in the plane.

Now for every single flight routine and every housing location we are given an amount of *noise pollution* that this flight routine confers to that housing location. For the noise pollution on the houses we are given a threshold. This threshold represent the maximum amount of noise pollution, that is acceptable.
Around the airport we are also given a zone imposed by government legislations. Outside this zone the noise pollutions may not exceed a maximum legislated level. The government enforces this legislation by set set of measure points in the neighborhood of this zone. Since this measure points will not always be on the zone, maximum levels in the measure points may defer from the government legislated level. Given these, we will consider the problem of maximizing the number of fights, while minimizing the number of houses suffering more than the threshold amount of noise pollution.

We will now formulate this problem as a multi-objective optimization problem. As mentioned before we will call it the *route scheduling problem* (RS). For an instance I_{RS} of the route scheduling problem, we have the following sets given

Furthermore we have the following given constants

Flight routines	$\{r_1,\dots,r_m\} = R$
Measure points	$\{q_1,\dots,q_l\} = Q$
Housing locations	$\{h_1,\dots,h_n\} = H$

$\alpha_{qr} \in \mathbb{R}_+$	$\forall q \in Q, \forall r \in R$	Representing the noise pollution of flight routine r on measure point q
$\beta_{hr} \in \mathbb{R}_+$	$\forall h \in H, \forall r \in R$	Representing the noise pollution of flight routine r on house h
$\omega_h \in \mathbb{R}_+$	$\forall h \in H$	Representing the number of houses on every housing location h
$\theta \in \mathbb{R}_+$		The maximum noise allowed by the government
$\lambda \in \mathbb{R}_+$		The maximum noise allowed on every house

An instance I_{RS} will contain the above constants and sets. Now we introduce the set of variables $x = \{x_{r_1},\dots,x_{r_m}\}$ which represent the amount of times a routine r is used with $x_r \in \mathbb{Z}_+, \forall r \in R$. Note that sometimes we assume without loss of generality that $x_r \in \{0,1\}, \forall r \in R$. In this case we assume that we have enough duplicates of every flight movement in our set R such that we can take any reasonable amount of every flight movement, i.e., we consider R as a multiset. For modeling the noise pollution around airports we will consider two different problem formulations; the Big-M formulation and the maximum feasible subsystem formulation.

3.2 Big-M formulation

We can formulate our problem as an integer linear program using the 'big-M trick'. In order to do this we introduce the following set of variables $z = \{z_{h_1},\dots,z_{h_n}\}$ with $z_h \in \{0,1\}, \forall h \in H$. A variable z_h is assigned to the constraint corresponding to the housing location h. It is set to one, if the constraint for house h is violated and to zero otherwise. We want to construct our problem in such a way that if $z_h = 1$ the constraint for h is always satisfied. Therefore we introduce the large constant $M_h \in \mathbb{R}_+, \forall h \in H, M_h \gg 0$, which whenever a housing location h suffers to much noise pollution makes sure the constraint associated with h is still satisfied. Thus for every feasible x we need that $\sum_{r\in R} x_r - M_h \leq \lambda, \forall h \in H$. We have the following multi-objective formulation $\Pi_{\mathrm{Big}M}$ of our problem

$$\begin{array}{llll} \max & \sum_{r\in R} x_r & \text{and} & \min \quad \sum_{h\in H} \omega_h z_h \\ \\ \text{s.t.} & \sum_{r\in R} \alpha_{qr} x_r \leq \theta, & \forall q \in Q & \\ & \sum_{r\in R} \beta_{hr} x_r \leq \lambda + M_h z_h, & \forall h \in H & \\ & x_r \in \mathbb{Z}_+, & \forall r \in R & \\ & z_h \in \{0,1\}, & \forall h \in H & \end{array} \tag{3.1}$$

Note that if we would impose a ordering on the sets R and H, we can also write our lesser-or-equal constraints in matrix vector notation.

3.3 Maximum feasible subsystem formulation

It would be better to have a formulation which does not use the big-M method. The problem with using this method is the dependent on the choice of the M_h's. If one does not take $M_h, \forall h \in H$ large enough one cannot find a feasible solution, while if one take them to large this may lead to numerically instability ([8, 18]). Therefore in this subsection we will formulate the problem as a *maximum feasible subsystem* problem (MFS). In MFS we are given a set of linear relations and we want to find a solution that satisfies the most of these relations. A relation can be of the type $=$, $\geq$, $>$ or $\neq$. More formally one is given a linear system $Ax\Diamond b$, with $A \in \mathbb{R}^{n\times m}$, $b \in \mathbb{R}^n$ and with $\Diamond = \{=, \geq, >, \neq\}$, where one wants to find an $x \in \mathbb{R}^m$ that satisfies the most linear equations.
In addition to the set of optional relations $Ax\Diamond b$, one sometimes also has a set of binding relations $Cx\Diamond d$ that have to be satisfied, with $C \in \mathbb{R}^{l\times m}$, $d \in \mathbb{R}^l$. This version of MFS is called the constrained maximum feasible subsystem problem. Furthermore sometimes one requires x to be binary or integer instead of real valued. Lastly one can also consider a weighted version of MFS, where all optional constraints have a certain weight w_i assigned to them for all $i \in \{1, \ldots, n\}$. One then maximizes the weight of the satisfied constraints instead of the number of satisfied constraints.
Since MFS only has one objective function, we need to get rid of one of the objective functions. In Chapter 5 we will look at different ways of doing this. For now we will stick to setting a lower bound on the number of flights.

$$\sum_{r\in R} x_r \geq \kappa$$

We will proceed by writing our problem as a weighted constrained integer maximum feasible subsystem problem. We consider the following system of constraints

$$\begin{array}{lll} \sum_{r\in R}\beta_{hr}x_r \leq \lambda & \forall h \in H & \text{(optional constraints)} \\ \sum_{r\in R}\alpha_{qr}x_r \leq \theta & \forall q \in Q & \text{(binding constraints)} \\ \sum_{r\in R} x_r \geq \kappa & & \text{(binding constraint)} \\ x_r \in \mathbb{Z}_+, & \forall r \in R & \text{(binding constraints)} \end{array} \tag{3.2}$$

Thus we have a system of n optional constraints and $l + n + 1$ binding constraints. Lastly we assign weights to our optional constraints, i.e., $w_h = \omega_h, \forall h \in H$. We will refer to this problem as Π_{MFS}. If we would relax $x_r \in \mathbb{Z}_+, \forall r \in R$ to $x_r \geq 0, \forall r \in R$ in (3.2), we could rewrite our problem in the general form of a constrained maximum weighted feasible subsystem problem.

Chapter 4

Complexity Analysis

In this chapter we will analyze the problem in terms of complexity and approximability. We will show the results found in chronological order. We will begin by proving that that the route scheduling problem is weakly $\mathcal{NP}$-complete and afterwards that it is strongly $\mathcal{NP}$-complete. In Section 4.2 we will consider the approximability of our problem and will obtain an inapproximability result. Because of this inapproximability result, we will consider more restricted version of the route scheduling problem. We will see that given certain restrictions on our input variables the route scheduling problem is equal to the *maximum edge cover problem* and to the *maximum multi-coverage problem*, two problems which we define ourselves, and to the *maximum coverage problem*, which is know in literature. In Chapter 4.4 we will summarize the results found.

4.1 Complexity

We want to investigate the complexity of our route scheduling problem (RS). Therefore we first need to reformulate it as a decision problem. We will consider for our complexity results Π_{MFS} with $Q = \emptyset$ and binary variables. This is a special case of RS, where we do not have any measure points. Thus we want to maximize the number of optional constraints that can be satisfied given the following set of inequalities

$$\begin{array}{lll} \sum_{r\in R} \beta_{hr} x_r \leq \lambda & \forall h \in H & \text{(optional constraints)} \\ \sum_{r\in R} x_r \geq \kappa & & \text{(binding constraint)} \\ x_r \in \{0,1\}, & \forall r \in R & \text{(binding constraints)} \end{array} \tag{4.1}$$

Let $S(x)$ be the set of optional constraints satisfied given a specific $x \in \{x_r \in \mathbb{Z}_+, \forall r \in R\}$, i.e.,

$$S(x) := \{h \in H : \sum_{r\in R} \beta_{hr} x_r \leq \lambda\}.$$

Let $|S(x)|$ denote the cardinality of $S(x)$, i.e., the number of constraints satisfied. The corresponding decision problem for RS is defined as follows

$$\text{RS-DECISION}(R, H, \beta, \lambda, \kappa, c) = \left\{ \begin{array}{c} \exists x \text{ with} \\ \sum_{r\in R} x_r \geq \kappa \\ x_r \in \{0,1\}, \forall r \in R \\ \text{such that} \\ |S(x)| \geq c \end{array} \right\}.$$

Now we will prove that the decision variant of RS is weakly $\mathcal{NP}$-complete for $c \geq 2$.

Theorem 4.1.1. For $c \geq 2$ the route scheduling decision problem RS-DECISION is weakly $\mathcal{NP}$-complete.

Proof. We first show that a yes-instance can be verified in polynomial time. Suppose that R, H, β, λ, κ and c are given. Furthermore suppose that we are given a yes-instance, i.e., $x_r \in \{0,1\}, \forall r \in R$ such that $|S(x)| \geq c$.
Then we can verify that this is a yes-instance by checking which of the $|H|+1$ constraints of (4.1) are satisfied, i.e., a matrix vector multiplication, a vector subtraction and a check whether their signs are positive or negative. If the binding constraint is satisfied and we satisfy at least c optional constraints, then we have a yes-instance. Since all this can be done in polynomial time RS-DECISION is in $\mathcal{NP}$.
For the reduction from an $\mathcal{NP}$-complete problem to our route scheduling problem we will consider the *equipartition problem* (EP). Suppose we have a set of items $X = \{1, \ldots, n\}$ with integer weights w_j for item $j \in 1, \ldots, n$ and $n = |X|$ even. In the equipartition problem EP one searches for a subset of the items $S_{\text{EP}} \subseteq X$ such that S_{EP} consist of exactly half of the items and the total weight of the items in S_{EP} is equal to the total weight of the items not in S_{EP}.
We can state the equipartition problem more formally as the following decision problem.

$$\text{EP-DECISION}(X, w) = \left\{ \begin{array}{c} \exists S_{\text{EP}} \subseteq X \text{ with} \\ \sum_{j \in S_{\text{EP}}} w_j = \sum_{j \notin S_{\text{EP}}} w_j \\ |S_{\text{EP}}| = \frac{n}{2} \end{array} \right\}.$$

From Garey and Johnson [13] we know that equipartition is weakly $\mathcal{NP}$-complete. Now we take for RS-DECISION the following values for its input variables $R = X$, $|H| \geq 2$ and $\lambda = 1$. Let $W := \sum_{j=1}^{n} w_j$ and $w_{\max} := \max_{j \in \{1,\ldots,n\}} w_j$. We take $\kappa = n/2$, $c \geq 2$ and set

$$\beta_{hr} = \begin{cases} \frac{2w_r}{W} & \text{if } h = 1 \\ \frac{w_{\max} - w_r}{\frac{n}{2} w_{\max} - \frac{W}{2}} & \text{if } h = 2 \\ \frac{1}{|R|} & \text{if } 2 < h \leq c \\ 2 & \text{otherwise} \end{cases} \tag{4.2}$$

Thus we have the following binding and optional constraints

$$\sum_{r \in R} x_r \geq \frac{n}{2} \qquad \text{(binding constraint)}$$

$$\sum_{r \in R} \frac{2w_r}{W} x_r \leq 1 \qquad \forall h \in \{c+1, \ldots, |H|\} \qquad \text{(optional constraint)} \tag{4.3}$$

$$\sum_{r \in R} \frac{w_{\max} - w_r}{\frac{n}{2} w_{\max} - \frac{W}{2}} x_r \leq 1 \qquad \text{(optional constraint)} \tag{4.4}$$

$$\sum_{r \in R} \frac{1}{|R|} x_r \leq 1, \qquad \forall h \in \{3, \ldots, c\} \qquad \text{(optional constraint)}$$

$$\sum_{r \in R} 2x_r \leq 1, \qquad \forall h \in \{c+1, \ldots, |H|\} \qquad \text{(optional constraint)}$$

with $x_r \in \{0,1\}$, $\forall r \in R$. Note that the constraints for $h \in \{3, \ldots, c\}$ are always satisfied, since $\sum_{r \in R} x_r \leq |R|$, and that the constraints for $h \in \{c+1, \ldots, |H|\}$ cannot be satisfied, since

$\sum_{r\in R} x_r \geq \frac{n}{2}$, we will therefore not list them explicitly in the discussion below. We can rewrite the above system as

$$\sum_{r\in R} x_r \quad \geq \quad \frac{n}{2} \tag{4.5}$$

$$\sum_{r\in R} w_r x_r \quad \leq \quad \frac{W}{2} \tag{4.6}$$

$$\sum_{r\in R} (w_{\max} - w_r) x_r \quad \leq \quad \frac{n}{2} w_{\max} - \frac{W}{2} \tag{4.7}$$

Suppose that we have a yes-instance of EP-DECISION. Then by setting $x_r = 1$ if $r \in S_{\text{EP}}$ and $x_r = 0$ if $r \notin S_{\text{EP}}$, we see that (4.5) is satisfied with equality, i.e $\sum_{r\in R} x_r = \frac{n}{2}$. Since S is a yes-instance EP-DECISION we know by our choice of the x_r that $\sum_{r\in R} w_r x_r = \frac{W}{2}$ and thus (4.6) is also satisfied with equality. Now since (4.5) and (4.6) are satisfied with equality also (4.7) is satisfied with equality and we have a yes-instance of our RS-DECISION.
Now suppose we have a yes-instance of RS-DECISION with the values as above. Then by adding (4.6) and (4.7) we obtain that

$$w_{\max} \sum_{r\in R} x_r \leq \frac{n}{2} w_{\max} \;\Leftrightarrow\; \sum_{r\in R} x_r \leq \frac{n}{2}$$

This together with (4.5) yields $\sum_{r\in R} x_r = n/2$. But this implies that (4.7) is equivalent to

$$-\sum_{r\in R} w_r x_r \leq -\frac{W}{2},$$

which together with (4.6) implies that

$$\sum_{r\in R} w_r x_r = \frac{W}{2}.$$

Hence by letting $r \in S_{EP}$ if $x_r = 1$ and $r \notin S_{EP}$ if $x_r = 0$, we obtain a yes-instance of our EP-DECISION. Thus RS-DECISION is $\mathcal{NP}$-hard and thus $\mathcal{NP}$-complete. □

Note that in the proof of Theorem 4.1.1 we choose the β_{hr} in (4.2) such that in order to have a yes-instance of RS one must satisfy (4.3) and (4.4). One could say, that we make them binding instead of optional. We will use the same method in the following proof where we show that for c large, RS is strongly $\mathcal{NP}$-complete. We will use a polynomial-time reduction from *maximum independent set* to prove this.

Theorem 4.1.2. For $c = |H|$ large the route scheduling problem RS is strongly $\mathcal{NP}$-complete.

Proof. From Theorem 4.1.1is in $\mathcal{NP}$. Therefore we will continue by showing a polynomial-time reduction from *maximum independent set* to our route scheduling problem. Let $G(V, E)$ be a graph with vertices V and edges E. In the *maximum independent set problem* (MIS) one wants to find a subset of vertices V' such that no vertices in E' share an edge. We will call such a subset E' an *independent set.* We can formulate the maximum independent set problem as the following decision problem.

$$\text{MIS-DECISION}(V, E, K) = \left\{ \begin{array}{c} \exists x \text{ with} \\ \sum_{v\in V} x_v \geq K \\ x_u + x_v \leq 1, \ \forall \{u, v\} \in E \\ x_u \in \{0, 1\}, \ \forall u \in V \end{array} \right\}. \tag{4.8}$$

Thus this decision problem tells us whether or not a graph has an independent set with cardinally K or more.
For the route scheduling problem take $R = V$, $H = E$, $\lambda = 1$ and

$$\beta_{hr} = \begin{cases} 1, & \text{if } r \in \{u, v\} \text{ and } h \in \{u, v\} \\ 0, & \text{if otherwise} \end{cases}$$

Furthermore take $\kappa = K > 0$ and $c = |E|$. We will show the following: there is a independent set with cardinality K or bigger if and only if there is a feasible solution to RS with $\kappa = K$ or more flights scheduled and satisfying $c = |E|$ constraints.
Suppose we have a yes-instance of MIS-DECISION. We take R, H, λ and β as above and we set $x_r = x_u$. Then we have an instance of RS-DECISION that satisfies $c = |E|$ optional constraints and for which $\sum_{r\in R} x_r \geq \kappa = K$, since $\sum_{v\in V} x_v \geq K = \kappa$. Thus we have a yes-instance of RS-DECISION.
Now suppose we have a yes-instance of RS-DECISION with input parameters as above. Then it holds that $\sum_{r\in R} x_r = \sum_{v\in V} x_v \geq K$ and since $c = |E|$ it also holds that $\forall\{u, v\} \in E, x_u + x_v \leq 1$ by our construction of β. Thus we have a yes-instance of MIS-DECISION. Hence a yes-instance of MIS-DECISION corresponds to a yes-instances of RS-DECISION and thus the route scheduling problem RS is strongly $\mathcal{NP}$-hard for large c (i.e., $c = |E|$ the number of edges of the maximum independent set problem). □

Now that we know that RS is $\mathcal{NP}$-complete for $c \geq 2$ we are interested in whether or not the problem is also $\mathcal{NP}$-complete for $c = 1$ the following theorem shows it is not.

Theorem 4.1.3. For $c = 1$ the route scheduling problem RS is polynomial-time solvable.

Proof. If $c = 1$ we want to find (at least) one optional constraint that we satisfy. Suppose that we are given a certain constraint υ. Then we can formulate a decision problem RS(υ)-DECISION to decide whether we can satisfy this constraint given R, β, λ and κ.

$$\text{RS}(\upsilon)\text{-DECISION}(R, \beta, \lambda, \kappa) = \left\{ \begin{array}{c} \exists x \text{ with} \\ \sum_{r\in R} x_r \geq \kappa \\ \sum_{r\in R} \beta_{\upsilon r} x_r \leq \lambda \\ x_r \in \{0, 1\}, \forall r \in R \end{array} \right\} \tag{4.9}$$

Thus if we find an algorithm that solves the above decision problem in time T then we can solve RS-DECISION in time nT, where $n = |H|$.
We can solve RS(υ)-DECISION as follows: Without loss of generality we assume that $\beta_{\upsilon r}$ are ordered such that $\beta_{\upsilon 1} \leq \ldots \leq \beta_{\upsilon m}$, where $m = |R|$. Then set $x_1 = 1, \ldots, x_\kappa = 1$ and $x_{\kappa+1} = 0, \ldots, x_m = 0$ Then it follows that we have a yes-instance if and only if

$$\sum_{j=1}^{m} \beta_{\upsilon j} x_j = \sum_{j=1}^{\kappa} \beta_{\upsilon j} \leq \lambda.$$

Since the running time is $O(m)$ we have a polynomial-time algorithm to solve (υ)RS-DECISION and thus we can solve RS-DECISION in $O(nm)$-time for $c = 1$. □

4.2 Approximation

We will again consider RS, which was Π_{MFS} with $Q = \emptyset$ and binary variables. Since the RS problem is $\mathcal{NP}$-hard for $c \geq 2$, it is very unlikely (unless $\mathcal{P} = \mathcal{NP}$) that there is an algorithm that can solve RS efficiently. Therefore we are interested in how well this problem can be approximated in polynomial-time. Before we look into the approximation of RS, we first consider the following

Lemma 4.2.1. RS has an optimal solution with $\sum_{r\in R} x_r = \kappa$

Proof. Suppose we have an optimal solution to our problem RS which gives us `OPT` optional constraints satisfied with $\sum_{r\in R} x_r = \kappa + \eta$. Now select η of the x_r, $r \in R$, for which it holds that $x_r = 1$. Call this selection $R(\eta)$. Then for all optional constraints satisfied it holds that

$$\sum_{r\in R} \beta_{hr}x_r - \sum_{r\in R(\eta)} \beta_{hr}x_r \leq \sum_{r\in R} \beta_{hr}x_r \leq \lambda$$

since $\beta_{hr} \geq 0, \forall h \in H$ and $\forall r \in R$.
Thus by setting $x_r = 0, \forall r \in R(\eta)$, we get a solution satisfying `OPT` equations and with $\sum_{r\in R} x_r = \kappa$. Note that the number of satisfied equations is never greater than `OPT` since we assumed this to be the optimal amount satisfied. □

Now we will look at approximability of RS.

Theorem 4.2.2. The problem RS is as hard to approximate as maximum independent set.

Proof. We proof this by using a cost/measure preserving polynomial transformation from maximum independent set (MIS).
Let $G(V, E)$ be a graph of an arbitrary instance of MIS and let $|V| = \eta$ be the cardinality of V. For every node $v \in V$ let $N(v)$ denote the nodes incident to v, i.e., $N(v) := \{u \in V : \{u, v\} \in E\}$. We then consider the following $|V|$ inequalities

$$\begin{array}{lll} -x_v + \sum_{u\in N(v)} x_u \leq -1 & \forall v \in V & \text{(Optional constraints)} \\ x_v \in \{0,1\} & \forall v \in V & . \end{array}$$

We see that we satisfy the inequality for v if and only if $x_v = 1$ and $x_u = 0, \forall u \in N(v)$. Thus if we satisfy s inequalities, we obtain an independent set of size s by taking all vertices v for which $x_v = 1$. Conversely if we have an independent set I of size s we can satisfy s inequalities by setting $x_v = 1$ if $v \in I$ and setting $x_v = 0$ if $v \in V \setminus I$.
Now consider the following instance of RS. Let $R = V \cup \bar{V} = \{v_1, \dots, v_\eta\} \cup \{\bar{v}_1, \dots, \bar{v}_\eta\}$, $H = V$, $\kappa = |V|$ and $\lambda = |V| - 1$. Note that $|R| = 2|V|$ and $|H| = |V|$. Furthermore we take β_{hr} as follows for every $v \in H$

$$\beta_{vr} = \begin{cases} 0 & r = v \\ 2 & r = u \in N(v) \\ 1 & r = u \in V \setminus (N(v) \cup v) \\ 1 & r = \bar{u} \in \bar{V} \end{cases} \tag{4.10}$$

Then we have the following set of inequalities

$$\begin{array}{lll} 0x_v + \sum_{r\in N(v)} 2x_r + \sum_{r\in V\setminus(N(v)\cup v)} x_r + \sum_{r\in\bar{V}} x_r \leq \lambda & \forall h = v \in H & \text{(Optional constraint)} \\ x_r \in \{0,1\} & \forall r \in R & \end{array}$$

We can assume by Lemma 4.2.1 that $\sum_{r\in R} x_r = \kappa$. But we could also see this from the fact that we can only satisfy an optional equation if $\sum_{r\in R} x_r = \kappa$. To see this suppose that $\sum_{r\in R} x_r = \kappa + 1$. Then

$$0x_v + \sum_{r\in N(v)} 2x_r + \sum_{r\in V\setminus(N(v)\cup v)} x_r + \sum_{r\in\bar{V}} x_r \geq \kappa > \lambda$$

Thus we can substitute the following equality in our optional constraints

$$\sum_{r\in\bar{V}} x_r = \kappa - \sum_{r\in V} x_r$$

Then we get the following set of optional constraints

$$\begin{array}{lll} -x_v + \sum_{r\in N(v)} x_r \leq \lambda - \kappa = -1 & \forall h \in H & \text{(Optional constraint)} \\ x_r \in \{0,1\} & \forall r \in R & \end{array}$$

Which is in one to one correspondence to the MIS-problem and can be constructed in polynomial time. Thus we have a cost preserving transformation from MIS to RS.
For every independent set I of size s we can satisfy s inequalities. We do this by setting $x_r = 1$ if $r \in I$ and by setting $\kappa - s$ times an arbitrary $x_r = 1$ where $r \in \bar{V}$. $\square$

Note that this proof also holds for $\beta_{hr} \in \{0,1,2\}$, $C \in \mathbb{Z}$ and $\lambda \in \mathbb{Z}$. Since RS is as hard to approximate as maximum feasible subsystem, the best approximation algorithm we can hope has an approximation ratio of n, i.e., ALG $\geq \frac{1}{n}$OPT, Zuckerman [27].
An easy way of attaining the approximation ratio for RS found is doing the following: Select the first constraint of RS. Try to satisfy this constraint and $\sum_{r\in R} x_r = \kappa$. If the constraint is satisfied, output the x_r, otherwise one picks the next constraint and tries again. If one would continue in this way, one will satisfy at least 1 constraint or none. When one satisfies a single constraint one has an n-approximation. If one satisfies zero constraints, there were no constraints that could be satisfied, hence one has the optimal solution.

However we can do even (a little) better than this approximation bound. Take RS with $x_r \in \mathbb{Z}_+, \forall r \in R$ instead of $x_r \in \{0,1\}, \forall r \in R$. Consider the following algorithm.

Algorithm 1 for RS-problem

Input: Houses H, routines R, noise level $\beta_{hr}, \forall h \in H, \forall r \in R$ and an integer κ
Output: $x_r, \forall r \in R$ and an integer s denoting the number of satisfied constraints

```
1. Initialize $s = 0$, $x_r = 0, \forall r \in R$
2. for $i = 1, \ldots, m$ do
3.     Let $\hat{x}_r$ s.t. $\hat{x}_r = k$ if $r = i$ and $\hat{x}_r = 0$ if $r \neq i$
4.     Let $\hat{s}$ be the number of constraints satisfied by $\hat{x}_r$
5.     if $\hat{s} \geq s$ set $s = \hat{s}$ and $x_r = \hat{x}_r, \forall r \in R$
6. end
7. return $x_r$ and $s$
```

Then we can prove the following

Proposition 4.2.3. Algorithm 1 gives an approximation for the route scheduling problem with an approximation ratio of max $\{\frac{1}{n}, \frac{1}{m}\}$.

Proof. Let $\beta_{h_{\min}}$ be the minimum β_{hr} for a $h \in H$, i.e.,

$$\forall h \in H, \beta_{h_{\min}} = \min_{r\in R} \beta_{hr}$$

Then we know for every optional constraint that either $\kappa \cdot \beta_{h_{\min}} \leq \lambda$ or the constraint cannot be satisfied. Since during our algorithm we consider all $r \in R$ in step 3, every constraint, that can be satisfied, will satisfied at least once in step 4. If for it least one of the optional constraints it holds that$\kappa \cdot \beta_{h_{\min}} \leq \lambda$, our algorithm will obtain a solution with $s \geq 1$ and we get an n-approximation.

Now suppose that in total a constraints can be satisfied. Then we will satisfy each of these constraints at least once during our m iterations. Per iteration we will satisfy on average $\frac{a}{m}$ constraints and due to the pigeonhole principle there will be at least one iteration for which it satisfies at least $\frac{a}{m}$, thus $s \geq \frac{a}{m}$. Since $\texttt{OPT} \leq a$ we also have $\texttt{ALG} \geq \frac{a}{m} \geq \frac{1}{m}\texttt{OPT}$, i.e., an m-approximation. $\square$

4.3 Maximum Covered Edges and Maximum Coverage

The approximation result for the route scheduling problem RS in Section 4.2 is fairly negative. It implies that the best approximation ratio is obtained by satisfying 1 optional constraint. In Theorem 4.1.3 we saw that this can be done in polynomial time. Therefore it would be interesting to see what happens if we restrict our input variables further.

We will first consider our route scheduling problem rewritten as the *maximum covered edges problem* (MCE). We define the maximum covered edges problem as follows: Suppose that we have a graph $G = (V, E)$ with vertices $v \in V$ and edges $e \in E$, we want to know how many edges we can cover if we color c vertices. An edge $e = \{u, v\}$ is *covered* if u or v is colored. Furthermore let m denote the number of vertices and n the number of edges.

More formally given a graph $G = (V, E)$ considering the following constraints

$$\begin{array}{lll} \sum_{v \in V} x_v \leq k & & \text{(Binding constraint)} \\ x_u + x_v \geq 1, & \forall \{u, v\} \in E & \text{(Optional constraints)} \\ x_v \in \{0, 1\}, & \forall v \in V & \text{(Binding constraints)} \end{array}$$

where we want to maximize the number of optional constraints satisfied. The following theorem shows that given certain restriction on input variables of our route scheduling problem, RS is equal to MCE.

Theorem 4.3.1. Given the route scheduling problem with

$$Q = \emptyset \tag{4.11}$$

$$\lambda = 1 \tag{4.12}$$

$$\beta_{hr} \in \{0, 1\} \qquad \forall h \in H, \forall r \in R \tag{4.13}$$

$$\sum_{r \in R} \beta_{hr} = 2 \qquad \forall h \in H \tag{4.14}$$

the route scheduling problem is equivalent to the maximum covered edges problem.

Proof. Given (4.11)-(4.14) we can rewrite constraints of the route scheduling problem as follows by taking $x_r = 1 - y_r$

$$\begin{aligned} \sum_{r \in R} x_r \geq \kappa &\Leftrightarrow \sum_{r \in R} (1 - y_r) \geq \kappa \\ &\Leftrightarrow n - \sum_{r \in R} y_r \geq \kappa \\ &\Leftrightarrow \sum_{r \in R} y_r \leq n - \kappa \end{aligned}$$

$$\sum_{r \in R} \beta_{hr} x_r \leq \lambda \Leftrightarrow \sum_{r \in R} \beta_{hr}(1 - y_r) \leq \lambda$$

$$\Leftrightarrow \sum_{r\in R} \beta_{hr} - \sum_{r\in R} \beta_{hr} y_r \leq \lambda$$
$$\Leftrightarrow \sum_{r\in R} \beta_{hr} y_r \geq \sum_{r\in R} \beta_{hr} - \lambda$$
$$\Leftrightarrow \sum_{r\in R} \beta_{hr} y_r \geq 1$$

If we set $k = n - \kappa$, $V = R$ and $E = H$, we get the maximum covered edges problem and thus given (4.11)-(4.14) MFS and RS are equivalent. □

4.3.1 Complexity and approximability

If we restrict the input variables of RS according to (4.11)-(4.14) we get a one to one correspondence to the maximum covered edges problem. By doing these restrictions we hope to get better approximability results. We will first proof the maximum covered edges problem is $\mathcal{NP}$-complete. In order to do this we need a decision version of MCE. Let $S_{\text{MCE}}(x)$ be the set of optional constraints satisfied given a specific $x \in \{x_v \in \{0,1\}, \forall v \in V\}$, i.e., $S_{\text{MCE}}(x) := \{\{u,v\} \in E : x_u + x_v \geq 1\}$ and let again $|S_{\text{MCE}}(x)|$ denote the cardinality of $S_{\text{MCE}}(x)$. The corresponding decision problem for MCE is defined as follows

$$\text{MCE-DECISION}(V,E,k,c) = \left\{ \begin{array}{c} \exists x \text{ with} \\ \sum_{v\in V} x_v \leq k \\ x_v \in \{0,1\}, \forall v \in V \\ \text{such that} \\ |S_{\text{MCE}}(x)| \geq c \end{array} \right\}. \tag{4.15}$$

Theorem 4.3.2. For $c = |E|$ the maximum covered edges problem is $\mathcal{NP}$-complete.

Proof. First we will show that MCE is in $\mathcal{NP}$. Suppose we have a yes-instance of MCE. Then to verify that this is indeed a yes-instance, one needs to check whether or not $\sum_{v\in V} x_v \leq k$ and whether or not c of the n linear constraints for the edges are satisfied, i.e., $|S_{\text{MCE}}(x)| \geq c$. This can be done in polynomial time, thus MCE is in $\mathcal{NP}$.
Now we will show that MCE is $\mathcal{NP}$-hard by reducing *minimum vertex cover problem* (MVC) to it. Given a graph $G(V,E)$ with vertices V and edges E, a *vertex cover* is a subset V' of the vertices, such that each edge of the graph is incident to at least one vertex of the set V'. Minimum vertex cover is one of the 21 $\mathcal{NP}$-complete problems proved by Karp [17]. We can formulate the minimum vertex cover problem as the following decision problem

$$\text{MVC-DECISION}(V,E,d) = \left\{ \begin{array}{c} \exists x \text{ with} \\ x_u + x_v \geq 1, \forall \{u,v\} \in E \\ x_v \in \{0,1\}, \forall v \in V \\ \text{such that } \sum_{v\in V} x_v \leq d \end{array} \right\}. \tag{4.16}$$

Thus this decision problem tells us whether or not a graph has an edge cover of size d or less. Now we want to show the following: there is a vertex cover with cardinality d or less if and only if there is a solution to the MCE with $k = d$ and satisfying $c = |E|$ constraints.
Take for both the same graph G and set the variables as above. Suppose we have a yes-instance of MVC-DECISION. Then by setting $x_v = 1$ in MCE if $x_v = 1$ in MVC, we get a yes-instance of MCE-DECISION with $k = d$ and $c = |E|$.

Now suppose we have a yes-instance of MCE-DECISION with $k = d$ and satisfying $c = |E|$ of the optional constraints. Then by setting $x_v = 1$ in MVC if $x_v = 1$ in MCE, we get a yes-instance of MVC-DECISION with $d = k$.
We conclude that for large c the MCE is $\mathcal{NP}$-hard and thus $\mathcal{NP}$-complete. □

Since the maximum covered edges problem is $\mathcal{NP}$-complete, it is of interest to know how well it can be approximated. For this we look at the following approximation algorithm, which is a greedy type algorithm.

```
Greedy Algorithm for MCE
Input: Undirected graph $G = (V, E)$ and integer $k$
Output: Colored vertices $V'$ and covered edges $E'$
1. Initialize $V' = \emptyset$ and $E' = \emptyset$
2. for $i = 1, \dots, k$ do
3.     Let $v$ be a vertex of maximum degree in $G' = (V \setminus V', E \setminus E')$
4.     Add $v$ to $V'$
5.     Add all edges $e = \{v, u\}$ incident to $v$ to $E'$
6. end
7. return $V'$ and $E'$
```

Theorem 4.3.3. For all k the greedy algorithm for the MCE-problem gives an approximation with ratio of $2 - \frac{|S|}{k}$.

Proof. Let $V_{\texttt{ALG}} = \{v_1, v_2, \dots v_k\}$ denote the vertices we color in steps $1, 2, \dots, k$ of our algorithm and let $d_1, d_2, \dots d_k$ denote the number of edges we cover in steps $1, 2, \dots, k$, respectively. Furthermore let $\texttt{OPT}$ denote the optimal amount of edges that could be covered and let $V_{\texttt{OPT}} = \{\hat{v}_1, \hat{v}_2, \dots, \hat{v}_k\}$ be the vertices who need to be picked to obtain this amount.
Let

$$\begin{aligned} S &:= V_{\texttt{OPT}} \cap V_{\texttt{ALG}} = \{v_i : \exists j \text{ s.t. } v_i = \hat{v}_j\} \\ S'_{\texttt{OPT}} &:= V_{\texttt{OPT}} \setminus S = \{\hat{v}_j : \hat{v}_j \neq v_i, \forall i \in \{1, \dots, k\}\} \end{aligned}$$

We define the *remaining cardinality* of a vertex v after k steps of the algorithm, as the number of edges incident to node v in the graph $G_k = (V \setminus V_{\texttt{ALG}}, E \setminus E_{\texttt{ALG}})$, where $E_{\texttt{ALG}}$ contains all edges that are incident to the vertices in $V_{\texttt{ALG}}$. Then after k steps of the algorithm the remaining cardinality of the nodes in S is zero. Furthermore the remaining cardinality of the nodes in $S'_{\texttt{OPT}}$ after k steps of the algorithm is at most d_k, because this is the remaining cardinality we may have removed as much as $\sum_{i=1}^k d_i$ from the original cardinality of the nodes in $V_{\texttt{OPT}}$. Thus we get the following bound for our optimal solution

$$\texttt{OPT} \leq |S'_{\texttt{OPT}}| d_k + \sum_{i=1}^k d_i.$$

Since $d_1 \geq d_2 \geq \dots \geq d_k$ we obtain

$$\frac{\sum_{i=1}^k d_i}{\texttt{OPT}} \geq \frac{\sum_{i=1}^k d_i}{|S'_{\texttt{OPT}}| d_k + \sum_{i=1}^k d_i} \tag{4.17}$$

It holds that $|S'_{\texttt{OPT}}| = k - |S|$. Thus we can rewrite (4.17) as

$$\frac{\sum_{i=1}^k d_i}{\texttt{OPT}} \geq \frac{\sum_{i=1}^k d_i}{(k - |S|) d_k + \sum_{i=1}^k d_i} = \frac{1}{\frac{(k-|S|)d_k}{\sum_{i=1}^k d_i} + 1} \geq \frac{1}{\frac{(k-|S|)d_k}{k d_k} + 1} \geq \frac{1}{2 - \frac{|S|}{k}}$$

Thus we have our approximation ratio. □

Note that because $|S| \geq 0$ we have at least an $\frac{1}{2}$-approximation. Furthermore also note that we took two very crude bounds for our approximation, namely $kd_k \leq \sum_{i=1}^{k} d_i$ and taking the difference between the cardinality and the remaining cardinality of $V_{\mathtt{OPT}}$ to be at most $\sum_{i=1}^{k} d_i$. Therefore one has the suspicion that the approximation ratio could be improved. We will see in the next section that the approximation ratio can be improved to $1 - \frac{1}{e}$, where e is the base of the natural logarithm.

4.3.2 Extended versions

In order to bridge the gap between the MCE-problem and the route scheduling problem we will be considering more general settings. These may give us insight in where the complexity of the problem comes from.

Hyper-edges

We again consider a graph $G = (V, E)$ but now the edges are not restricted to edges from one node to another, but we consider hyper-edges which may connect from 1 up to n nodes. Thus an edge $e \in E$ is a subset of the vertices, i.e., $e \subseteq V$. An edge e is covered if at least 1 node $v \in e$ is colored. In the literature this problem is know as the *maximum coverage problem* (MC). Note however that normally this is defined as a set U and a collection of subsets $\mathcal{S} = \{S_1, \ldots S_m\}$ where $S_i \subseteq U, \forall i \in \{1, \ldots, m\}$. But by taking $U = V$ and $\mathcal{S} = E$ we can stick to our graph representation.
Formally we can write our problem as follows. Given a hyper-graph $G = (V, E)$, where E is a set of hyper-edges, considering the following constraints

$$\begin{array}{lll} \sum_{v \in V} x_v \leq k & & \text{(Binding constraint)} \\ \sum_{v \in e} x_v \geq 1, & \forall e \in E & \text{(Optional constraints)} \\ x_v \in \{0,1\}, & \forall v \in V & \text{(Binding constraints)} \end{array}$$

we want to maximize the number of optional constraints satisfied. Now we will continue by showing that given certain restriction on the input parameters of the route scheduling problem, RS and MC are equivalent. The prove will be similar to the prove of Theorem 4.3.1.

Theorem 4.3.4. Given the route scheduling problem with

$$Q = \emptyset \tag{4.18}$$

and

$$\beta_{hr} = \beta_{eu} = \begin{cases} \frac{\lambda}{|e|-1} & \text{if } r = u \in e \\ 0 & \text{if } r = u \notin e \end{cases} \qquad \forall h \in H, \forall r \in R \tag{4.19}$$

the route scheduling problem is equivalent to the maximum coverage problem.

Proof. If we take $\lambda \in \mathbb{Z}_+$ arbitrary and set $x_r = 1 - y_r$, then given (4.18) and (4.19) we can rewrite our problem constraints as follows

$$\sum_{r \in R} x_r \geq \kappa \Leftrightarrow \sum_{r \in R} y_r \leq n - \kappa$$

$$\begin{aligned}\sum_{r\in R}\beta_{hr}x_r \leq \lambda &\Leftrightarrow \sum_{r\in R}\beta_{hr}(1-y_r) \leq \lambda \\ &\Leftrightarrow \sum_{r\in R}\beta_{hr}y_r \geq \sum_{r\in R}\beta_{hr} - \lambda \\ &\Leftrightarrow \frac{\lambda}{|e|-1}\sum_{r\in R}\mathbb{1}_{r\in e}\ y_r \geq |e|\frac{\lambda}{|e|-1} - \lambda \\ &\Leftrightarrow \sum_{r\in R}\mathbb{1}_{r\in e}\ y_r \geq \frac{|e|-1}{\lambda}\left(|e|\frac{\lambda}{|e|-1} - \lambda\right) \\ &\Leftrightarrow \sum_{r\in R}\mathbb{1}_{r\in e}\ y_r \geq 1\end{aligned}$$

Where $\mathbb{1}_{r\in e}$ is the indicator function for the event that $e \in R$. By again taking $k = n - \kappa$, $H = E$ and $R = V$ we obtain the maximum coverage problem. □

Note that we can write the MCE-problem as the maximum coverage problem, thus also this problem is $\mathcal{NP}$-hard. Since also yes instances can be checked in polynomial time because MC is a restricted version of RS, we can conclude that MC is in $\mathcal{NP}$-complete. Therefore we are again interested in the approximation ratio for this problem. For the maximum coverage problem we have the following greedy algorithm, which is similar to the greedy algorithm for MCE

Greedy Algorithm for MC

Input: Undirected hyper-graph $G = (V, E)$ and integer k
Output: Colored vertices V' and covered hyper-edges E'

```
1. Initialize $V' = \emptyset$ and $E' = \emptyset$
2. for $i = 1, \dots, k$ do
3.     Let $v$ be a vertex of maximum degree in $G' = (V \setminus V', E \setminus E')$
4.     Add $v$ to $V'$
5.     Add all hyper-edges $e$ incident to $v$ to $E'$
6. end
7. return $V'$ and $E'$
```

The maximum coverage problem is a known problem and therefore analysis has been done on its approximation. From Hochbaum and Pathria [15] we get the following theorem.

Theorem 4.3.5. Hochbaum and Pathria [15] For all k the greedy algorithm for the MC-problem gives an approximation for the maximum coverage problem with an approximation ratio of $\frac{e-1}{e}$, where e is the basis number of the natural logarithm.

Proof. Again let $V_{\texttt{ALG}} = \{v_1, v_2, \dots v_k\}$ denote the vertices we colour in steps $1, 2, \dots, k$ of our algorithm and let $d_1, d_2, \dots d_k$ denote the number of hyper-edges we cover in steps $1, 2, \dots, k$, respectively. Let $\texttt{OPT}$ be the optimal amount of edges covered by coloring k vertices $V_{\texttt{OPT}} = \{\hat{v}_1, \hat{v}_2, \dots, \hat{v}_k\}$.
After $l \leq k$ steps of our greedy algorithm we will have covered $\sum_{i=1}^{l} d_i$ edges. Thus the remaining uncovered edges incident to $V_{\texttt{OPT}}$ will be at least

$$\texttt{OPT} - \sum_{i=1}^{l} d_i.$$

Since $V_{\texttt{OPT}}$ contains k vertices, then due to the pigeon hole principle there is at least one vertex $v_i \in V_{\texttt{OPT}}$ with at least the following number of uncovered edges incident to it

$$\frac{\texttt{OPT} - \sum_{i=1}^{l} d_i}{k}.$$

Since our greedy algorithm picks a vertex with maximum remaining cardinality we have

$$d_l \geq \frac{\texttt{OPT} - \sum_{i=1}^{l-1} d_i}{k} \tag{4.20}$$

for all $l \in \{1, \dots, k\}$. We will proceed by proving that

$$\sum_{i=1}^{l} d_i \geq (1 - (1 - \tfrac{1}{k})^l)\texttt{OPT}. \tag{4.21}$$

First for $l = 1$ we know that $d_1 \geq \frac{1}{k}\texttt{OPT}$, thus (4.21) holds. Suppose that for $l \geq 1$ equation (4.21) holds, then for $l + 1$ we know from (4.20) that

$$\begin{aligned}
\sum_{i=1}^{l+1} d_i &= d_{l+1} + \sum_{i=1}^{l} d_i \\
&\geq \frac{\texttt{OPT} - \sum_{i=1}^{l} d_i}{k} + \sum_{i=1}^{l} d_i \\
&= \frac{\texttt{OPT}}{k} + (1 - \tfrac{1}{k}) \sum_{i=1}^{l} d_i \\
&\geq \frac{\texttt{OPT}}{k} + (1 - \tfrac{1}{k})(1 - (1 - \tfrac{1}{k})^l)\texttt{OPT} \\
&= (1 - (1 - \tfrac{1}{k})^{l+1})\texttt{OPT}
\end{aligned}$$

By setting $l = k$ it follows that

$$\begin{aligned}
\sum_{i=1}^{k} d_i &\geq (1 - (1 - \tfrac{1}{k})^k)\texttt{OPT} \\
&> (1 - \tfrac{1}{e})\texttt{OPT} \\
&= \frac{1}{\frac{e-1}{e}}\texttt{OPT}
\end{aligned}$$

□

Multi-covering

We consider the maximum coverage problem but now an edge e is covered if at least l vertices of e are colored. We call this problem the *maximum multi-coverage problem.* Thus given a hypergraph $G = (V, E)$, with vertices V and hyper-edges E, considering the following constraints

$$\begin{array}{lll}
\sum_{v \in V} x_v \leq k & & \text{(Binding constraint)} \\
\sum_{v \in e} x_v \geq l & \forall e \in E & \text{(Optional constraint)} \\
x_v \in \{0, 1\}, & \forall v \in V & \text{(Binding constraint)}
\end{array}$$

we want to maximize the number of optional constraints satisfied with $l \in \mathbb{Z}^+$. Note that a hyperedge e needs to have at least cardinality l, i.e., $|e| \geq l$, otherwise it could never be covered. We will show that given certain restriction on the input parameters of the route scheduling problem, RS and the maxumum multi-coverage problem are equivalent. The prove will again be similar to the prove of Theorem 4.3.1.

Theorem 4.3.6. Given the route scheduling problem with

$$Q = \emptyset \tag{4.22}$$

and

$$\beta_{hr} = \beta_{eu} = \begin{cases} \frac{\lambda}{|e|-l} & \text{if } r = u \in e \\ 0 & \text{if } r = u \notin e \end{cases} \qquad \forall h \in H, \forall r \in R \tag{4.23}$$

$$\tag{4.24}$$

the route scheduling problem is equivalent to the maximum multi-coverage problem.

Proof. We take $\lambda \in \mathbb{Z}^+$ arbitrary, and set $x_r = 1 - y_r$. Then given (4.22) and (4.22) we can rewrite our problem constraints as

$$\begin{aligned} \sum_{r\in R} x_r \geq \kappa &\Leftrightarrow \sum_{r\in R}(1-y_r) \geq \\ &\Leftrightarrow \sum_{r\in R} y_r \leq n-\kappa \\ \sum_{r\in R} \beta_{hr} x_r \leq \lambda &\Leftrightarrow \sum_{r\in R} \beta_{hr}(1-y_r) \leq \lambda \\ &\Leftrightarrow \sum_{r\in R} \mathbb{1}_{r\in e} y_r \geq \frac{|e|-1}{\lambda}\left(|e|\frac{\lambda}{|e|-l} - \lambda\right) \\ &\Leftrightarrow \sum_{r\in R} \mathbb{1}_{r\in e} y_r \geq l \end{aligned}$$

Setting $\kappa = n - k$, $H = E$ and $R = V$ we get the desired maximum multi-coverage problem. □

Note that this problem is $\mathcal{NP}$-hard because we can reduce our MCE-problem to it. Since also yes instances can be checked in polynomial time because MC is a restricted version of RS, we can conclude that MC is in $\mathcal{NP}$-complete.

Now if we consider our greedy algorithm for the MC-problem to approximate the maximum multi-coverage problem, we run into problems. Consider the following example:
We have the graph $G = (V, E)$ with $V = \{v_1, \dots v_6\}$ and $E = \{e_1, \dots e_6\}$ with

$$\begin{aligned} e_1 &= \{v_1, v_3, v_4\} \\ e_2 &= \{v_1, v_5, v_6\} \\ e_3 &= \{v_1, v_3, v_5\} \\ e_4 &= \{v_2, v_3, v_4\} \\ e_5 &= \{v_2, v_5, v_6\} \\ e_6 &= \{v_2, v_4, v_6\} \end{aligned}$$

Furthermore we have $k = 2$ and $\lambda = 2$. Then our greedy algorithm for the MC-problem could color v_1 and v_2, covering none of the edges. The optimal solution would color the pair v_3 and v_4 or the pair v_5 and v_6, yielding 2 covered edges. Hence our approximation ratio would be infinite. Thus we need another algorithm to approximate the maximum multi-coverage problem.

4.3.3 Integrality gap

One of the techniques often used to construct approximation algorithms for optimization problems is to construct an integer linear program (ILP) for the problem and relax the integrality constraints to make it a linear program (LP-relaxation), i.e., one changes $x_i \in \mathbb{Z}_+$ to $x_i \in \mathbb{R}_+$, which makes it easier to solve. The LP-relaxation can be solved to optimality in polynomial time and gives an upper bound for the optimal solution of the ILP.
In general one is interested in how tight the LP-relaxation bounds the ILP. Therefore we will introduce the notion of the integrality gap of an LP-relaxation. Given an LP-relaxation for a maximization problem Π, let $\texttt{OPT}(I)$ be the optimal integer valued solution to instance I of problem Π and let $\texttt{OPT}_{\text{LP}}(I)$ be the optimal solution in real numbers of instance I. Then the integrality gap is defined to be

$$\inf_I \frac{\texttt{OPT}(I)}{\texttt{OPT}_{\text{LP}}(I)}$$

In order to construct an approximation algorithm via LP-relaxation for our maximum multi-coverage problem (MMC-problem) we first need an ILP-formulation.

$$\begin{array}{rll} \max & \sum_{e\in E} z_e & \\ \text{such that} & \sum_{v\in V} x_v \leq k & \\ & \sum_{v\in V} \mathbb{1}_{\{v\in e\}} x_v \geq l z_e & \forall e \in E \\ & x_v \in \{0,1\} & \forall v \in V \\ & z_e \in \{0,1\} & \forall e \in E \end{array}$$

By relaxing the binary constraints for x_v and z_e we get the following LP-relaxation for the MMC-problem

$$\begin{array}{rll} \max & \sum_{e\in E} z_e & \\ \text{such that} & \sum_{v\in V} x_v \leq k & \\ & \sum_{v\in V} \mathbb{1}_{\{v\in e\}} x_v \geq l z_e & \forall e \in E \\ & 0 \leq x_v \leq 1 & \forall v \in V \\ & 0 \leq z_e \leq 1 & \forall e \in E \end{array}$$

Now we want to know the size of the integrality gap. For this we consider the following instance I_1. Let $E = \{e_1, \dots, e_n\}$ and $V = \{v_1, \dots, v_{n+2}\}$, where n is an even integer. Set $k = 2$, $l = 2$ and

$$e_i = \begin{cases} e_i = \{v_i, v_{n+1}\}, & \text{if } i \in \{1, \dots, \frac{1}{2}n\} \\ e_i = \{v_i, v_{n+2}\}, & \text{if } i \in \{\frac{1}{2}n + 1, \dots, n\} \end{cases}.$$

Then for the ILP we can satisfy at most one constraint, hence $\text{OPT}(I_1) = 1$. In the LP-relaxation we set $v_{n+1} = 1$ and $v_{n+2} = 1$. Then we can set $z_e = \frac{1}{2}$, $\forall e \in E$. Thus it follows that $\texttt{OPT}_{\text{LP}}(I_1) = \frac{1}{2}n$. We conclude that the integrality gap of the MMC-problem is

$$\inf_I \frac{\texttt{OPT}(I)}{\texttt{OPT}_{\text{LP}}(I)} \leq \frac{\texttt{OPT}(I_1)}{\texttt{OPT}_{\text{LP}}(I_1)} = \frac{2}{n}$$

Since we know that the approximation ratio is bounded by the integrality gap, if we base our approximation algorithm on the LP-relaxation, we conclude that this is not the way to go.

4.3.4 Dense k-Subgraph problem

In the previous section we have seen that if we assume certain bounds for our route scheduling problem, this problem can be written as maximum coverage problem. After relaxing $l = 1$ to $l \in \mathbb{Z}_+$ we could not use our greedy algorithm to approximate it. Because we want to know more about the approximability of maximum multi-coverage problem, we will now look at the *dense k-subgraph problem* (DkS) and show that for it is equivalent to the maximum multi-coverage problem with $l = 2$.
The dense-k-subgraph problem is the problem of finding a subgraph of k-vertices which has maximum average degree. In other words, given a graph $G = (V, E)$ and a parameter k, we want to find k vertices such that the subgraph induced by these vertices is of maximum average degree. In other words we search for a subgraph of k vertices of which the average degree is maximized. An integer linear program formulation of this problem would be

$$\begin{array}{rll} \max & \frac{1}{k}\left(\sum_{e\in E} z_e\right) & \\ \text{such that} & \sum_{v\in V} x_v = k, & \\ & x_u + x_v \geq 2z_e, & \forall e = \{u,v\} \in E \\ & x_v \in \{0,1\}, & \forall v \in V \\ & z_e \in \{0,1\}, & \forall e \in E \end{array}$$

For the dense-k-subgraph problem Feige [11] proves that there is no polynomial time approximation scheme (PTAS). More recently, Alon et al. [1] show that dense-k-subgraph is hard to approximate within any constant factor in polynomial time. They actually show that assuming no $n^{O(\log n)}$ algorithm solves the hidden CLIQUE problem, DkS cannot be approximated up to a factor of $2^{(\log n)^{2/3}}$ in polynomial time. The currently best know approximation is by Bhaskara et al. [6]. The algorithm they present approximates DkS for every $\epsilon > 0$ within a ratio of $n^{1/4+\epsilon}$ in time $n^{O(1/\epsilon)}$.

Proposition 4.3.7. The maximum multi-coverage problem is at least as hard to approximate as the dense-k-subgraph problem.

Proof. For the dense-k-subgraph problem we are given a graph $G = (V, E)$ and an integer k. First notice that by multiplying the objective function of DkS by k we do not change the optimal solution only its optimal value. Hence we take the following objective function

$$\max \sum_{e\in E} z_e$$

We want to reduce DkS to the maximum multi-coverage problem. To do this we consider the same graph $G = (V, E)$, set $l = 2$ and we take k for the maximum multi-coverage problem equal to the k from DkS. By using Lemma 4.2.1 we get the following instance of the multi-coverage problem

which is in one to one correspondence with the DkS problem, where the objective function is multiplied by k. □

$$\begin{array}{rll} \max & \sum_{e\in E'} z_e & \\ \text{such that} & \sum_{v\in V} x_v = k & \\ & \sum_{v\in V} I_{\{v\in e\}} x_v \geq 2z_e & \forall e \in E \\ & x_v \in \{0,1\} & \forall v \in V \\ & z_e \in \{0,1\} & \forall e \in E \end{array}$$

4.4 Summary

In this chapter we proved that we $Q = \emptyset$ the route scheduling problem is strongly $\mathcal{NP}$-complete and that it can be approximated with an approximation of n, the number of optional constraint. Afterwards we looked RS with more restriction on the input variables. We saw that for certain restrictions, our route scheduling problem was equivalent to the *maximum covered edges problem*, to the *maximum coverage problem* and to the *maximum multi-coverage problem*. All these problems were $\mathcal{NP}$-complete. Table 4.1 summarizes our results for the route scheduling problem.

Restrictions	Equivalent to	Approximation ratio
$Q = \emptyset$	-	$\alpha = n$
$Q = \emptyset$ $\beta_{hr} \in \{0,1,2\}$	-	$\alpha = n$ [1]
$Q = \emptyset$ $\lambda = 1$ $\beta_{hr} \in \{0,1\}$ $\sum_{r\in R} \beta_{hr} = 2$	maximum covered edges problem	$\alpha = \frac{e}{e-1}$
$Q = \emptyset$ $\beta_{hr} = \beta_{eu} = \begin{cases} \frac{\lambda}{\lvert e\rvert - 1} & \text{if } r = u \in e \\ 0 & \text{if } r = u \notin e \end{cases}$	maximum coverage problem	$\alpha = \frac{e}{e-1}$
$Q = \emptyset$ $\beta_{hr} = \beta_{eu} = \begin{cases} \frac{\lambda}{\lvert e\rvert - l} & \text{if } r = u \in e \\ 0 & \text{if } r = u \notin e \end{cases}$	maximum multi-coverage problem	$\alpha > 2^{(\log n)^{2/3}}$ [2]

Table 4.1: Theoretical results for the route scheduling problem

Chapter 5

Pareto optimality

The model we described in Chapter 3 is a multi-objective optimization problem with conflicting objective functions. The conflict in our route scheduling problem is that by scheduling more flights, there is the possibility that less of the houses remain below the noise pollution threshold. Because of this conflict there will be no global optimal solution for our problem. There will always be a trade-off between the two objectives. Therefore we will construct for our problem a set of solutions which are *Pareto optimal*. In this Chapter we will first consider multiobjective optimization problems and we will define Pareto optimality. After this we will consider three different methods of constructing a Pareto optimal set.

5.1 Definitions

We consider a multi-objective optimization problem Π_{multi} in its most general form

$$\begin{array}{ll} \max_x & F(x) = [f_1(x), \ldots, f_s(x)] \\ \\ \text{subject to} & \\ g_j(x) \leq 0, & j \in \{1, \ldots, n_1\} \\ h_l(x) = 0, & l \in \{1, \ldots, n_2\} \\ l_i \leq x_i \leq u_i, & i \in \{1, \ldots, m\}. \end{array}$$

Note that we take $s \geq 2$, otherwise we would not have multiple objective functions. All variables x_i have lower bounds l_i and upper bounds u_i with $l_i \leq u_i$. Furthermore all functions f_i, g_j and h_l are functions from $\mathbb{R}^m$ to $\mathbb{R}$. Lastly x denotes the vector of all variables, thus $x = \{x_1, \ldots, x_m\}$. Define X as the *feasible space*, i.e.,

$$\begin{aligned} X \;=\; & \{x : g_j(x) \leq 0, j \in \{1, \ldots, n_1\} \text{ and } h_l(x) = 0, l \in \{1, \ldots, n_2\}\} \\ & \text{and } l_i \leq x_i \leq u_i, i \in \{1, \ldots, m\}\} \,. \end{aligned}$$

We assume that X is non empty. We also define the *feasible criterion space* Z as

$$Z \;=\; \{F(x) : x \in X\} \,.$$

Thus Z is the space in $\mathbb{R}^s$ of all possible combinations of objective values. As discussed before Π_{multi} in general does not have a single optimal solution. The most used definition for defining an optimal point is that of *Pareto optimality* [24], it is defined as follows:

Definition 5.1.1. A point x^* is said to be *Pareto optimal* if and only if there does not exist another point $x \in X$ such that $f_i(x) \geq f_i(x^*), \forall i \in \{1, \ldots, s\}$ and there is at least one $i \in \{1, \ldots, s\}$ such that $f_i(x) > f_i(x^*)$.

All Pareto optimal points (or Pareto points for short) will lay on the boundary of Z. Choosing which Pareto point to use as a final solution is up to a decision maker. Therefore our goal is not to find a single Pareto optimal point, but is to construct a set of Pareto optimal points from which the decision maker can choose. In order to do this we will look at three different methods of constructing this set of Pareto optimal points. These three methods all have different properties. For a method that generates Pareto points the following properties are important:

1. The method generates an evenly distributed set of Pareto points in Z. It does not neglect any region.
2. The method should have the ability to generate all possible Pareto points if necessary.
3. The method should generate only Pareto solutions.
4. The method should be relatively easy to apply.

From these four properties the first three are the most important ones. The third one may seem obvious but there are many methods that do not guarantee this. These methods often generate *weakly Pareto optimal* points instead of Pareto optimal points.

Definition 5.1.2. A point x^* is said to be *weakly Pareto optimal* if and only if there does not exist another point $x \in X$ such that $f_i(x) > f_i(x^*), \forall i \in \{1, \ldots, s\}$.

Thus a weakly Pareto optimal point is a point for which there is at least one objective function that cannot be improved. One may be able to improve all others.

5.2 Weighted sum method

One of the most used techniques for generating a Pareto optimal set is the weighted sum method. It is especially useful if one already knows the preferences of the decision maker. In the weighted sum method we introduce weights $w_i \in [0, 1]$ for our objective functions, such that $\sum_{i=1}^{s} w_i = 1$. With these weights we transform our multi-objective function to a single objective function $f(x)$:

$$\max_x f(x) = \sum_{i=1}^{s} w_i f_i \tag{5.1}$$

If we then optimize this single objective function we get an optimal point according to the decision makers preferences. If we do not know his preferences we can also generate a Pareto optimal set by systematically varying the weights.

The weighted sum method is very easy to implement and if all objective functions are convex it can generate all Pareto points. Also if $w_i > 0, \forall i \in \{1, \ldots, s\}$, it will only generate Pareto optimal points. On the other hand we cannot guarantee that the weighted sum method will generate an evenly distributed set of Pareto points if the weights are uniformly distributed. Furthermore if the set Z is not convex, there may be regions of Z which cannot generated, see Marler and Arora [19].

In our route scheduling problem we have integer valued variables. This implies that our solution spaces are not convex and thus some points we may not be able to generate. Another drawback of the weighted sum method for us is the fact that it does not generate an even spread of Pareto points. Note that we can only use this method for the ILP-formulated problem $\Pi_{\text{Big}M}$. If we use this method our objective function becomes

$$\max w_1 \left(\sum_{r \in R} x_r \right) - w_2 \left(\sum_{h \in H} \omega_h z_h \right).$$

With $w_1 + w_2 = 1$.

5.3 Bounded objective function method

In the bounded objective function method one only maximizes the single most important objective function f_I. All the other objective functions are bounded by a constant and form additional constraints. Thus we transform problem Π_{multi} to

$$\begin{array}{ll} \max_x & f_I(x) \\ & \\ \text{subject to} & \\ f_i(x) \geq \kappa_i & i \in \{1, \ldots, I-1, I+1, \ldots, s\} \\ g_j(x) \leq 0, & j \in \{1, \ldots, n_1\} \\ h_l(x) = 0, & l \in \{1, \ldots, n_2\} \\ l_i \leq x_i \leq u_i, & i \in \{1, \ldots, m\}. \end{array}$$

Here $\kappa_i \in \mathbb{R}, \forall i \in \{1, \ldots, I-1, I+1, \ldots, s\}$. The decision maker can choose the κ_i's as minimal values for the associated objective functions. If one wants to generate a Pareto optimal set one has to vary the κ_i. In each iteration one then obtains a weakly Pareto optimal solution.
The bounded objective function method is easy to implement and if the feasible space X is convex and all objective functions are quasi convex then from Miettinen [21] we know that we can generate any weakly Pareto optimal point. On the downside we can not guarantee that we have a Pareto optimal point. Only if the found point x^* is unique, then it is a Pareto optimal point (Miettinen [21]). Another drawback is that if we vary the κ_i's uniformly, we will not get an evenly distributed Pareto set.
In our formulation Π_{MFS} we actually already used the bounded objective function method. We could also use the bounded objective function method on $\Pi_{\text{Big}M}$. Then we either introduce the constraint $\sum_{r \in R} x_r \geq \kappa$ or the constraint $\sum_{h \in H} \omega_h z_h \leq \kappa$.
Since we only have two objective functions we can iteratively construct a Pareto optimal set. We do this by first setting $\sum_{r \in R} x_r \geq \kappa_1$ and maximizing the number of satisfied housing constraints. Then after obtaining a set of satisfied constraints, we fix this set and maximize the number of flights. In this way we obtain a number of flights κ_2. We have then obtained our first (weakly) Pareto point. Then we return to maximizing the number of satisfied housing constraints, but now with $\sum_{r \in R} x_r \geq \kappa_2 + 1$. We have to violate the same amount of housing constraints or more then in our first iteration to obtain a feasible optimal solution. We continue in this way until we have found the entire weakly Pareto optimal set, containing all Pareto optimal solutions. If in the final step we remove all weakly Pareto optimal points, which are not Pareto optimal, we end up with a Pareto optimal set.

5.4 Normalized normal constraint method

The normalized normal constraint method was introduced by Messac et al. [20]. It has the advantage that it satisfies all the four properties for a method for generating a Pareto set, if one

has convex objective functions and real valued variables. A downside is that it requires a little more notation and insight than the weighted sum and the bounded objective function method. First let us introduce the concept of *anchor points*, the *Utopia point* and the *Nadir point*.
An *anchor point* or optimal vertex f_i^* is obtained by maximizing over only the objective function f_i. In this way we obtain the maximal value of every objective function given the constraints. If we put this differently $f_i^* = [f_1(x^{i*}), \dots, f_s(x^{i*})]$, where x^{i*} is the unique solution of

$$\begin{array}{ll} \max_x & f_i(x) \\ & \\ \text{subject to} & \\ g_j(x) \leq 0, & j \in \{1, \dots, n_1\} \\ h_l(x) = 0, & l \in \{1, \dots, n_2\} \\ l_i \leq x_i \leq u_i, & i \in \{1, \dots, m\}. \end{array}$$

The *Utopia point* f^U is the point consisting of all maximal values of the s objectives, i.e.,

$$f^U = [f_1(x^{1*}), \dots, f_s(x^{s*})].$$

The *Nadir point* f^N is the point consisting of all minimal values of the objective functions, if we would optimize over the other objective functions. In other words

$$f^N = [f_1^N, \dots, f_s^N],$$

where $f_i^N = \min\{f_i(x^{1*}), \dots, f_i(x^{s*}\}, \forall i \in \{1, \dots, s\}$. The *Utopian plane* is the plane in the objective space between the anchor points. We can describe any point on the Utopian plane as a convex combination of the anchor points.
Let us define $l_i = f_i(x^{i*}) - f_i^N$, thus l_i is the distance between the maximum and the minimum optimal value. Then with this distance we can normalize our objective functions as follows:

$$\bar{f}_i(x) = \frac{f_i((x^{1*})) - f_i(x)}{l_i}$$

With these normalize objective function we define the vectors $\bar{f}^{1*}, \dots \bar{f}^{s*}$ as

$$\bar{f}^{i*} = [\bar{f}_1(x^{i*}), \dots, \bar{f}_s(x^{i*})], \forall i \in \{1, \dots, s\}.$$

These vectors are the corners (or vertices) of the Utopian plane. Now the normalized normal constraint method comprises of five steps

1. The anchor points $f_1^*, \dots, f_s^*$ are computed and the Utopia point f^U, the Nadir point f^N and the Utopia plane constructed.

2. Using the Utopia and and Nadir point normalize the objective functions to obtain $\bar{f}_1, \dots, \bar{f}_s$.

3. Define the direction vector $N_1, \dots N_{s-1}$ which are the vectors from the anchor point f_s^* to the other anchor points, i.e.,

$$N_i = \bar{f}_s^* - \bar{f}_i^* = [f_1(x^{s*}), \dots, f_s(x^{s*})] - [f_1(x^{i*}), \dots, f_s(x^{i*})], \forall i \in \{1, \dots, s-1\}$$

4. Define a set of points $X = \{X_1, \dots, X_k\}$ on the Utopian plane. One takes X_p for all $p = \{1, \dots, k\}$ such that

$$\begin{array}{l} X_p = \sum_{i=1}^s \alpha_{pi} \bar{f}^{i*} \\ \text{with} \sum_{i=1}^s \alpha_{pi} = 1, \\ 0 \leq \alpha_{pi} \leq 1, \forall i \in \{1, \dots, s\}. \end{array}$$

   Note that one can choose the α_{pi} in such a way that the points are evenly distributed on the Utopian plane.

5. Generate a set of well distributed Pareto solutions by solving the following problem for each $X_P \in X$.

$$\begin{array}{ll} \max_x & f_s(x) \\ \\ \text{subject to} & \\ g_j(x) \leq 0, & j \in \{1, \dots, n_1\} \\ h_l(x) = 0, & l \in \{1, \dots, n_2\} \\ l_i \leq x_i \leq u_i, & i \in \{1, \dots, m\}. \\ N_k \left(\bar{f}(x) - X_P\right)^T \leq 0, & k \in \{1, \dots, s-1\} \end{array}$$

Where $\bar{f}(x) = [\bar{f}_1(x), \dots, \bar{f}_s(x)]$.

The power of the method lies in the fact that a constraint $N_k \left(\bar{f}(x) - X_P\right)^T \leq 0$ defines a half space of which the boundary is perpendicular to the Utopian plane. This makes sure that for convex objective functions it will only generate Pareto optimal points. Furthermore if the points are evenly distributed on the Utopian plane, they will also be evenly distributed on the Pareto front [20].

We will now consider the Normalized normal constraint method for our route scheduling problem. Because we need an explicit formula for the objective functions we again only can use formulation $\Pi_{\text{Big-}M}$. To write the second objective function as a maximization instead of a minimization we multiply by -1. Thus we consider the following problem Apllying the method as described above

$$\begin{array}{lll} \max & [f_1, f_2] = [\sum_{r \in R} x_r,\ \sum_{h \in H} -\omega_h z_h] & \\ \\ \text{s.t.} & \sum_{r \in R} \alpha_{qr} x_r \leq \theta, & \forall q \in Q \\ & \sum_{r \in R} \beta_{hr} x_r \leq \lambda + M_h z_h, & \forall h \in H \\ & x_r \in \mathbb{Z}_+, & \forall r \in R \\ & z_h \in \{0,1\}, & \forall h \in H \end{array}$$

we obtain two anchor point $f_1^* = [f_1(x^{1*}, z^{1*}), f_2(x^{1*}, z^{1*})]$ and $f_2^* = [f_1(x^{2*}, z^{2*}), f_2(x^{2*}, z^{2*})]$. The Utopian point will be $f^U = [f_1(x^{1*}, z^{1*}), f_2(x^{2*}, z^{2*})]$ and the Nadir point will be $f^N = [f_1(x^{2*}, z^{2*}), f_2(x^{1*}, z^{1*})]$. Note that $f_2(x^{2*}, z^{2*}) = 0$ and $f_1(x^{1*}, z^{1*})$ could be infinite. If one would end up with an infinite amount of flights for $f_1(x^{1*}, z^{1*})$ it is better to set this value as small as possible while keeping $f_2(x^{1*}, z^{1*})$ housing constraints violated.
With these anchor points we normalize our objective function to

$$\bar{f}_1(x, z) = \frac{f_1(x^{1*}, z^{1*}) - f_1(x, z)}{f_1(x^{1*}, z^{1*}) - f_1(x^{2*}, z^{2*})} = \frac{f_1(x^{1*}, z^{1*}) - \sum_{r \in R} x_r}{f_1(x^{1*}, z^{1*}) - f_1(x^{2*}, z^{2*})}$$

and to

$$\bar{f}_2(x, z) = \frac{f_2(x^{2*}, z^{2*}) - f_2(x, z)}{f_2(x^{2*}, z^{2*}) - f_2(x^{1*}, z^{1*})} = \frac{\sum_{h \in H} \omega_h z_h)}{-f_2(x^{1*}, z^{1*})}$$

We have

$$N_1 = \left[\begin{array}{c} -1 \\ 1 \end{array} \right].$$

Furthermore since we only have two objective values we have for all $p = \{1, \dots, k\}$

$$X_p = \left[\begin{array}{c} \alpha_p \\ 1 - \alpha_p \end{array} \right], \text{with } 0 \leq \alpha_p \leq 1.$$

We can evenly distribute points on our Pareto front by varying α uniformly. We now only have to construct the addition constraint for our problem.

$$\begin{aligned}
N_1\left(\bar{f}(x,z)-X_P\right)^T &\leq 0\\
\begin{bmatrix}-1\\1\end{bmatrix}\left(\begin{bmatrix}\frac{f_1(x^{1*},z^{1*})-\sum_{r\in R}x_r}{f_1(x^{1*},z^{1*})-f_1(x^{2*},z^{2*})}\\ \frac{\sum_{h\in H}\omega_h z_h}{-f_2(x^{1*},z^{1*})}\end{bmatrix}-\begin{bmatrix}\alpha\\1-\alpha\end{bmatrix}\right)^T &\leq 0\\
\frac{-f_1(x^{1*},z^{1*})+\sum_{r\in R}x_r}{f_1(x^{1*},z^{1*})-f_1(x^{2*},z^{2*})}+\alpha+\frac{\sum_{h\in H}\omega_h z_h}{-f_2(x^{1*},z^{1*})}-1+\alpha &\leq 0\\
\frac{-f_1(x^{1*},z^{1*})+\sum_{r\in R}x_r}{f_1(x^{1*},z^{1*})-f_1(x^{2*},z^{2*})}+\frac{\sum_{h\in H}\omega_h z_h}{-f_2(x^{1*},z^{1*})} &\leq 1-2\alpha
\end{aligned}$$

Hence we can obtain a weakly Pareto optimal set by varying $\alpha\in[0,1]$ uniformly, while solving the following problem

$$\begin{array}{lll}
\min & \sum_{h\in H}\omega_h z_h & \\
 & & \\
\text{s.t.} & \sum_{r\in R}\alpha_{qr}x_r\leq\theta, & \forall q\in Q\\
 & \sum_{r\in R}\beta_{hr}x_r\leq\lambda+M_h z_h, & \forall h\in H\\
 & \frac{-f_1(x^{1*},z^{1*})+\sum_{r\in R}x_r}{f_1(x^{1*},z^{1*})-f_1(x^{2*},z^{2*})}+\frac{\sum_{h\in H}\omega_h z_h}{-f_2(x^{1*},z^{1*})}\leq 1-2\alpha & \\
 & x_r\in\mathbb{Z}_+, & \forall r\in R\\
 & z_h\in\{0,1\}, & \forall h\in H
\end{array}$$

Chapter 6

Algorithms

In Chapter 4 we have seen several negative results about our route scheduling problem. We know that in general the problem is $\mathcal{NP}$-hard and from Theorem 4.2.2 we know that it cannot be approximated well. Because of these results it is very unlikely to find an algorithm that will solve the route scheduling problem optimally in polynomial time or to find a good solution in polynomial time that comes with a provable approximation guarantee. Therefore we will consider an algorithm that solves it optimally in exponential time and use heuristics that work in polynomial time, but do not have any approximation guarantee. In this chapter, we will first look at an integer linear program solver to obtain optimal solutions. After this, we will consider two heuristics to obtain solutions, namely Chinnecks algorithms and a dynamic approximation algorithm.

6.1 Integer linear program (exact)

An integer linear programming problem can be solved exactly, but it might take exponential time to do this. Using one of the Pareto methods as described in Chapter 5 we can use our problem formulation $\Pi_{\text{Big}M}$ to construct an input for an ILP solver. We will then use the solver `lp_solve`[1] to solve this ILP and obtain a (weakly) Pareto optimal solution. `lp_solve` uses a 'branch and bound' method to do this. In this method the integer linear program is first solved without the integer restrictions. Then we investigate if there are variables which are non-integral. If such a variable x_i exists we split the problem in two subproblems. Suppose that $x_i = \gamma$, then in the first subproblem we require that $x_i \geq \lceil\gamma\rceil$ and in the second subproblem we require that $x_i \leq \lfloor\gamma\rfloor$. We choose one of these subproblems and then we investigate if there are still other fractional variables. If so we split the problem again in a similar fashion. We continue this process until we have found an integer solution. This solution is stored and we remember its objective value as the current best. We continue then with a subproblem we did not explore yet until we find an integer solution or until its objective value falls below the current best. We continue this process until there are no subproblems left to explore. Since the number of subproblems grows exponentially with the number of integer variables, this method can take exponential time. Often if we solve our problem in this way, we will refer to it as the *Big-M method*.

[1] http://lpsolve.sourceforge.net/5.5/

6.2 Chinneck heuristics

We will consider a heuristic for solving the maximum feasible subsystem problem (MFS) by Chinneck [7]. We will afterwards change this heuristic slightly in order to use it to solve Π_{MFS}. As said before in MFS one is given an infeasible linear system $Ax \diamond b$ of linear (in)equalities and we want to find a subsystem of this system with maximum cardinality, that is feasible. Here again $A \in \mathbb{R}^{n \times m}$, $b \in \mathbb{R}^n$ and we will consider $\diamond = \{\leq, \geq, =\}$.
A closely related problem is the problem of finding a *minimum cardinality irreducible infeasible subsystem cover*. An *irreducible infeasible subsystem* of constraints or IIS is a set of constraints that is infeasible and has the property that any proper subsystem is feasible. In other words an IIS is an infeasible set of constraints, with the property that if we would remove a single constraint from the IIS it would be feasible. Thus if we could identify in an infeasible set of constraints all IISs we could solve a covering problem and by removing the cover we would obtain a feasible set. This is the *minimum cardinality IIS cover problem.*
The heuristic we will be considering will make extensive use of the *standard elastic program.* In the standard elastic program we consider the elasticized version of our constraints. This is done by adding nonnegative elastic variables $\epsilon_i, \forall i \in \{1, \dots, n\}$. The following table summarizes how this is done. The associated elastic objective function seeks to minimize the sum of the

Original constraint	Elasticized constraint
$\sum_{j=1}^m a_{ij}x_j \geq b$	$\sum_{j=1}^m a_{ij}x_j + \epsilon_i \geq b$
$\sum_{j=1}^m a_{ij}x_j \leq b$	$\sum_{j=1}^m a_{ij}x_j - \epsilon_i \leq b$
$\sum_{j=1}^m a_{ij}x_j = b$	$\sum_{j=1}^m a_{ij}x_j + \epsilon_i - \epsilon_i' = b$

elastic variables. We will call it the *sum of the infeasibilities* ($SINF$). The set of elastic constraints is always feasible, but the values of the ϵ_i's give insight in where the infeasibility could come from in the original set of constraints. Each nonzero elastic variable indicates a violated constraint. Another measure of the infeasibility in the original set of constraints is the *number of infeasibilities* ($NINF$). In MFS we want to minimize $NINF$.
In the heuristic we consider, we will iteratively remove constraints from $Ax \diamond b$ to end up with a feasible subsystem of the constraints. In the heuristic we will solve the standard elastic program and use its solution to find possible candidates for removal. We use the following six observations as described in Chinneck [7]:

1. The number of violated constraints in the original standard elastic program is an upper bound on the cardinality of the IIS set cover. The set of violated constraints is in itself an IIS set cover.

2. If an original standard elastic program has only one nonzero elastic variable ϵ_i, then the constraint associated with this elastic variable is the minimum cardinality IIS cover.

3. Removing a constraint that is part of the minimum cardinality IIS cover should reduce $SINF$ more, than removing a constraint that is not part of this cover.

4. Constraints to which the elastic objective function is not sensitive (i.e., their reduced cost is zero) do not reduce $SINF$ when removed from the set of constraints.

Having a nonzero elastic variable ϵ_i in the elastic program, actually means a change of b by an amount of ϵ_i in the original set of constraints. Following from the sensitivity analysis in Section 2.1.2 we then know that if we remove constraint i, a good estimator for the change in the elastic objective function would be $\epsilon_i p_i$, where p_i is the variable associated with constraint i in the dual problem. This leads to the following observation

5. For a constraint in the elastic program, a good estimator for the drop in $SINF$, if we would remove the constraint, is $\epsilon_i p_i$.

Lastly there may be constraints, that are not violated in the elastic program but that do give a large drop in $SINF$ when removed. This leads to the last observation.

6. For a constraint which has $\epsilon_i = 0$ in the elastic program, a good estimator for the drop in $SINF$, if we would remove the constrain, is p_i.

With these six observations we can define a heuristic that solves the maximum feasible subsystem problem. In the article by Chinneck [7] there are actually three algorithms that are considered. The difference of these algorithms lies in the way we choose the candidates for deletion $S_{\text{Candidate}}$. In the first algorithm, which we will call `Chinneck1`, the set of candidates is composed of all constraints with nonnegative reduced cost (this follows from Observation 4). In the second algorithm, which we will call `Chinneck2(k)`, $S_{\text{Candidate}}$ consist of the k constraints for which $\epsilon_i p_i$ is the greatest. We only consider the first k constraints because often the constraint, which constitutes the biggest drop in $SINF$, is among these first k and considering only the first k can speed up the algorithm. In the third algorithm, which we will call `Chinneck3(k)`, the candidates set $S_{\text{Candidate}}$ consists of the k constraints for which $\epsilon_i p_i$ is the greatest and the k constraints for which p_i is the greatest.
We now state the algorithm.

Chinneck Algorithm for MFS-problem

Input: A set of (infeasible) linear constraints $Ax \diamond b$ with $\diamond = \{\leq, \geq, =\}$.
Output: Set S of constraints to be removed.

```
1. Initialize Set $S = \emptyset$, $S_{\text{Hold}} = \emptyset$ and set up elastic LP.
2. Solve elastic LP
3. If $SINF = 0$ EXIT.
4. If $NINF = 1$ do
5.    Set $S$ to violated constraint and EXIT.
6. Else
7.    Set $S_{\text{Hold}}$ to currently violated constraints.
8. While $SINF > 0$ and $|S| < |S_{\text{Hold}}|$ do
9.    Set $MINSINF = \text{inf}$
10.   Set $S_{\text{Candidate}}$ according to algorithm.
11.   For each member in $S_{\text{Candidate}}$ do
12.      Delete constraint
13.      Solve elastic LP
14.      If $SINF = 0$ do
15.         Add constraint to $S$
16.         EXIT
17.      If $SINF < MINSINF$ do
18.         Set winner to current constraint
19.         Set $MINSINF = SINF$
20.         If $NINF = 1$ do
21.            Set Nextwinner equal to violated constraint
22.         Else
23.            Set Nextwinner equal $\emptyset$
24.   Add winner to $S$
25.   Delete winner constraint permanently
26.   If Nextwinner $\neq \emptyset$ do
27.      Add nextwinner to $S$
28.      EXIT
29. If $|S| < |S_{\text{Hold}}|$ do
30.   Return $S$
31. Else
32.   Return $S_{\text{Hold}}$
```

Note that in each iteration we solve a number of linear programming problems, which is at most m (for `Chinneck1`). Furthermore since there are n constraints we will have at most n iterations. Thus in total we will solve at most nm linear programs. For `Chinneck2(k)` this will be at most km LPs and for `Chinneck3(k)` at most $2km$. Since we can solve a linear program in polynomial time, the heuristic runs in polynomial time.

Adaption to route scheduling problem

In order to solve Π_{MFS} we need to adapt the heuristic. First we need to introduce the binding constraints. We do this by adding a set of constraints $Cx\Diamond d$ to the elastic program with $C \in \mathbb{R}^{(l+1)\times m}$, $d \in \mathbb{R}^{l+1}$, which we do not elasticize. Furthermore we need to take into account the weights ω_h associated with each optional constraint. We can consider the following two elastic objective functions

$$\min \sum_{h\in H} \omega_h \epsilon_h \tag{6.1}$$

or

$$\min \sum_{h\in H} (\omega_{\max} - \omega_h)\epsilon_h \tag{6.2}$$

where $\omega_{\max} = \max_{h\in H}\ \omega_h$. The objective function in (6.1) works well, because it makes sure that the elastic program violates those constraints that have lower weights. The objective function in (6.2) works well, because we use the drop in $SINF$ as a measure for dropping a constraint. Therefore we want the drop in $SINF$ to be high if ω_h for the constraint is low. Lastly we relax the integer variables to real valued variables, since otherwise the algorithms would probably not have polynomial running time.

6.3 Dynamic approximation algorithm

Although our route scheduling problem cannot be approximated well in general, we can formulate an algorithm for which we can obtain a bound dependent on the instance. Since this bound is instance dependent it does not fall in the category of approximation algorithms. To emphasize that we can give a bound, we will call it a dynamic approximation algorithm, although technically it is a heuristic.
For the algorithm we consider $\Pi_{\text{Big}M}$ with real valued $x_r \geq 0$, where we use the bounded objective function method to find a Pareto optimal solution. We bound the objective function of maximizing the flights by κ. Furthermore we also relax our variables z_h, demanding that $z_h \in [0,1], \forall h \in H$ instead of $z_h \in \{0,1\}$. Then we have to following linear program

$$\begin{array}{lll} \min & \sum_{h\in H} \omega_h z_h & \\ & & \\ \text{s.t.} & \sum_{r\in R} x_r \geq \kappa, & \\ & \sum_{r\in R} \alpha_{qr} x_r \leq \theta, & \forall q \in Q \\ & \sum_{r\in R} \beta_{hr} x_r - M_h z_h \leq \lambda, & \forall h \in H \\ & x_r \geq 0 & \forall r \in R \\ & z_h \in [0,1], & \forall h \in H. \end{array} \tag{6.3}$$

Now suppose that we consider $\Pi_{\text{Big}M}$ with real valued $x_r \geq 0$ and know the set of constraints violated, i.e., we know $z_h, \forall h \in H$. Then we only have to maximize the number of flights to obtain a solution. We thus have the following LP

$$\begin{array}{lll} \max & \sum_{r\in R} x_r & \\ & & \\ \text{s.t.} & \sum_{r\in R} \alpha_{qr} x_r \leq \theta, & \forall q \in Q \\ & \sum_{r\in R} \beta_{hr} x_r \leq \lambda, & \forall h \in H \text{ with } z_h = 0 \\ & x_r \geq 0 & \forall r \in R \end{array} \tag{6.4}$$

With the above two linear programs we can formulate the following algorithm.

Dynamic approximation algorithm for route scheduling problem
Input: An instance I_{RS} of the route scheduling problem as defined in Chapter 3 and a constant κ. **Output:** A Pareto solution ($x_r, \forall r \in R$ and $z_h, \forall h \in H$ of the route scheduling problem with real valued x_r.
1. *Initialize* Set $M_h = (\max_{r \in R} \beta_{hr})\,\kappa - \lambda$ and set up LP as in (6.3). 2. Solve LP obtaining $\bar{x}_r, \forall r \in R$ and $\bar{z}_h, \forall h \in H$ 3. Set $z_h = 1$ if $\bar{z}_h > 0$ and $z_h = 0$ otherwise 4. Given $z_h, \forall h \in H$, set up LP as in (6.4) 5. Solve LP obtaining $x_r, \forall r \in R$ 6. Return $x_r, \forall r \in R$ and $z_h, \forall h \in H$

Theorem 6.3.1. For all κ the dynamic approximation algorithm for the route scheduling problem gives in polynomial time for the route scheduling problem an approximation for the number of houses receiving an excess amount of noise pollution with

$$\frac{\texttt{ALG}^H}{\texttt{OPT}^H} \leq \frac{\sum_{h \in H} \omega_h z_h}{\sum_{h \in H} \omega_h \bar{z}_h},$$

where $\texttt{ALG}^H$ is the number of houses receiving an excess amount of noise pollution calculated by the algorithm given that at least κ flights are scheduled, i.e., if z is the solution calculated by our algorithm then $\texttt{ALG}^H = \sum_{h \in H} \omega_h z_h$ with $\sum_{r \in R} x_r \geq \kappa$, and $\texttt{OPT}^H$ is the minimum number of houses receiving an excess amount of noise pollution given that at least κ flights are scheduled. Where $\bar{z}_h, \forall h \in H$ are the valued obtained by solving (6.3) in step 2 of the algorithm.

Proof. The algorithm runs in polynomial time. This is due to the fact that it consists of constructing and solving two linear programs. Given a linear program optimal solution $\texttt{OPT}_{\text{LP}}$ and an integer linear program optimal solution $\texttt{OPT}_{\text{ILP}}$ for the same problem we know that $\texttt{OPT}_{\text{LP}} \leq \texttt{OPT}_{\text{ILP}}$ for a minimization problem.
Suppose that for an instance of $\Pi_{\text{Big}M}$ with real valued $x_r \geq 0, \forall r \in R$. $\texttt{OPT}^H$ is the minimum number of houses receiving an excess amount of noise pollution given at least κ flights. Furthermore suppose $\texttt{ALG}^H$ is the number of houses receiving an excess amount of noise pollution computed by our algorithm given at least κ flights. Let $\texttt{OPT}_{\text{LP}}$ be the optimal value of the LP (6.3). Then since $\Pi_{\text{Big}M}$ is an integer linear program we know that

$$\texttt{OPT}_{\text{LP}} \leq \texttt{OPT}^H \leq \texttt{ALG}^H.$$

From this we can conclude that

$$\frac{\texttt{ALG}^H}{\texttt{OPT}^H} \leq \frac{\texttt{ALG}^H}{\texttt{OPT}_{\text{LP}}} = \frac{\sum_{h \in H} \omega_h z_h}{\sum_{h \in H} \omega_h \bar{z}_h} \tag{6.5}$$

□

Note that the linear program in (6.3) is actually the elastic LP defined for Chinnecks algorithm only now with its variables scaled to be between zero and one. Since Chinnecks algorithm uses the amount of constraints violated in the initial elastic LP as an upper bound, Chinnecks heuristics will always perform at least as good as the dynamic approximation algorithm. Thus equation (6.5) will also hold for Chinnecks algorithms.
For stability purposes we might also like to consider the elastic LP instead of the linear program

in (6.3). We do this by removing M_h in our constraints and taking $z_h \geq 0, \forall h \in H$. Then due to Theorem 4.2.1 equation (6.5) becomes

$$\frac{\mathtt{ALG}^H}{\mathrm{OPT}^H} \leq \frac{\mathtt{ALG}^H}{\mathrm{OPT}_{\mathrm{LP}}} = \frac{\sum_{h\in H} M_h \omega_h z_h}{\sum_{h\in H} M_h \omega_h \bar{z}_h}$$

with $M_h = (\max_{r\in R} \beta_{hr})\, \kappa - \lambda$.

6.4 Pre- and postprocessors

In this section we consider a few algorithms that we will run either before or after our main algorithm. These pre- and postprocessors will tackle small problems or deficiencies. One of the problems with all our algorithms is that given sets of binding and optional constraints, there may be optional constraints that are always satisfied due to the bindings ones. Therefore when solving the problem we can consider a smaller problem by removing these optional constraints. For this we use the following preprocessor.

```
Preprocessor for constrained MFS-problem
Input: A set of (infeasible) linear constraints $A\underline{x}\diamond b$ with $\diamond = \{\leq, \geq\}$,
which are optional and a set of (feasible) linear constraints $C\underline{x}\diamond d$.
Output: Set $S$ of optional constraints that could be violated
1. Initialize Set $S = \emptyset$
2. For every optional $\leq$ ($\geq$)-constraint $i$ do
3.     $f = \max_{\underline{x}\geq 0} \left\{\sum_{j=1}^{n} a_{ij}x_j | C\underline{x}\diamond d\right\}$ $\left(\text{ or } f = \min_{\underline{x}\geq 0} \left\{\sum_{j=1}^{n} a_{ij}x_j | C\underline{x}\diamond d\right\}\right)$
4.     If $f > b_i$ ($f < b_i$) do
5.         Add constraint $i$ to $S$
6. Return $S$.
```

The main idea behind the preprocessor is that one tries to maximize the value of a constraint, pushing it towards violation. If this maximum amount is still smaller then the allowed amount b_i one removes the constraint from the set of optional constraints considered.
Furthermore we will also make use of a postprocessor. It may happen in our algorithms (especially our heuristics) that it violates or removes constraints, that are in the solution given the flights are not violated. Thus these constraints can be reinserted. Especially Chinnecks algorithm may select a number of constraints in the initial iterations that are not violated in the final solution. Thus we have the following postprocessor

```
Postprocessor for MFS-problem
Input: A set of optional (infeasible) linear constraints $A\underline{x}\diamond b$ with $\diamond = \{\leq, \geq\}$,
a feasible (optimal) solution $x^* \in \mathbb{R}^n$ and a set of violated constraints $S$.
Output: Set $S$ of optional constraints that could be violated
1. For every $\leq$-constraint $i \in S$ (or $\geq$-constraint) do
2.     $f = \sum_{j=1}^{n} a_{ij}x_j^*$
3.     If $f \leq b_i$ ($f \geq b_i$) do
4.         remove constraint $i$ from $S$
5. Return $S$.
```

Another deficiency that may occur is when we try to produce a Pareto front using our heuristics. Since we are making use of a heuristic, we will in general not produce a set of solutions that are Pareto optimal. Therefore it is quite common that in the set of solution there is a solution $\sigma^1 = \{x^1, z^1\}$ and a solution $\sigma^2 = \{x^2, z^2\}$ such that solution σ^2 has more flights than σ^1 and less houses exceeding their threshold noise pollution. In other words for these two solution σ^1 and σ^2 it holds that $\sum_{r\in R} x_r^1 < \sum_{r\in R} x_r^2$ and $\sum_{h\in H} \omega_h z_h^1 > \sum_{h\in H} \omega_h x_h^2$. We therefore could use a Pareto filter that removes these solutions as proposed by Messac et al. [20]. We state the Pareto filter for our route scheduling problem, but it can easily be adapted for any set of solutions for a multi-objective optimization problem.

```
Pareto filter for RS-problem
Input: A instance $I_{RS}$ of the route scheduling problem as defined in
Chapter 3 and set of solutions to the instance $S_{\text{sol}} = \{\sigma^1, \dots, \sigma^q\}$, with
$\sigma_i = \{x^i, z^i\}, \forall i \in \{1, \dots, q\}$.
Output: A set of solutions $S'_{\text{sol}}$ to $I_{RS}$ that for which it hold that there
are no $\sigma, \hat{\sigma} \in S'_{\text{sol}}$ such that $\sum_{r\in R} \hat{x}_r < \sum_{r\in R} x_r$ and $\sum_{h\in H} \omega_h \hat{z}_h > \sum_{h\in H} \omega_h z_h$.
1. Initialize Set $S'_{\text{sol}} = S_{\text{sol}}$
1. For $i = 1, \dots, q$ do
3.    For $j = i + 1, \dots, q$ do
4.       If $\sum_{r\in R} x_r^i < \sum_{r\in R} x_r^j$ and $\sum_{h\in H} \omega_h z_h^i > \sum_{h\in H} \omega_h z_h^j$ do
5.          remove $\sigma_i$ from $S'_{\text{sol}}$
6.       If $\sum_{r\in R} x_r^i < \sum_{r\in R} x_r^j$ and $\sum_{h\in H} \omega_h z_h^i > \sum_{h\in H} \omega_h z_h^j$ do
7.          remove $\sigma_j$ from $S'_{\text{sol}}$
8. Return $S'_{\text{sol}}$.
```

Note that we can only conclude that the removed points are not weakly Pareto optimal. The solutions that remain in the set, do not have to be weakly Pareto optimal.

Chapter 7

Experimental results

In this chapter we will examine three types of airports for of our route scheduling problem. We will consider an abstract linear airport, a fictitious airport based on real world data and consider the same fictitious airport where random locations for the houses are taken. We will compute solutions to these instances using our exact algorithm and the heuristics described in Chapter 6. We will evaluate the performance of our algorithms in terms of optimality and computation time.

7.1 Linear airport

We will consider the abstract linear airport. We will first describe the linear airport. Then we will solve two problem instances; one with integer valued flights and one with real valued flights.

7.1.1 Description

We consider an airport which has two runways, one runway to the west of length l and a second one to the east also of length l. Furthermore we have two flight routines with associated variables x_1 and x_2. Flight routine x_1 takes the lane to the west and flight route x_2 takes the lane to the east. After take-off these flights will continue in a straight line. The plane around the airport considered is the plane $[0, X] \times [0, Y]$, where the airport is located in the middle of this plane, i.e., $(\frac{1}{2}X, \frac{1}{2}Y)$.

This set up is described in Figure 7.1 and Figure 7.2.

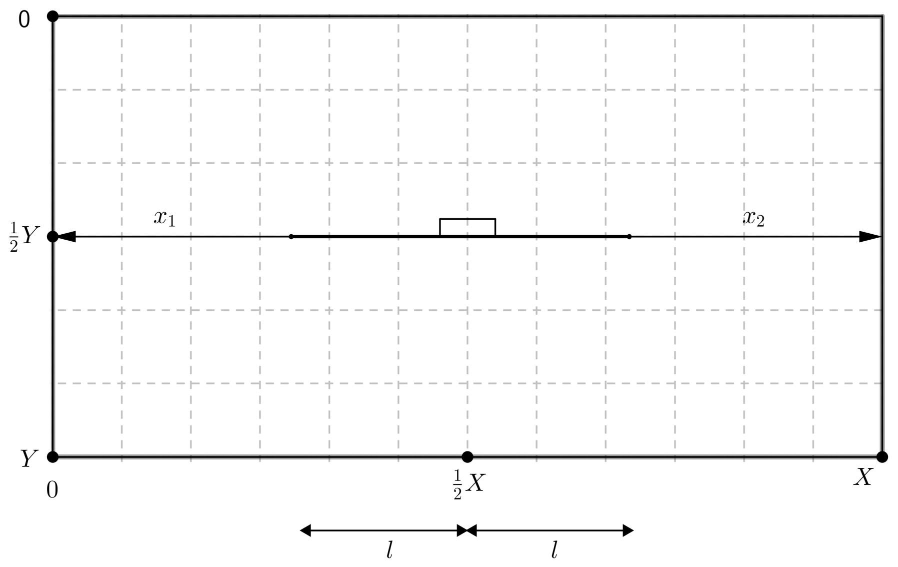


Figure 7.1: Linear airport top view

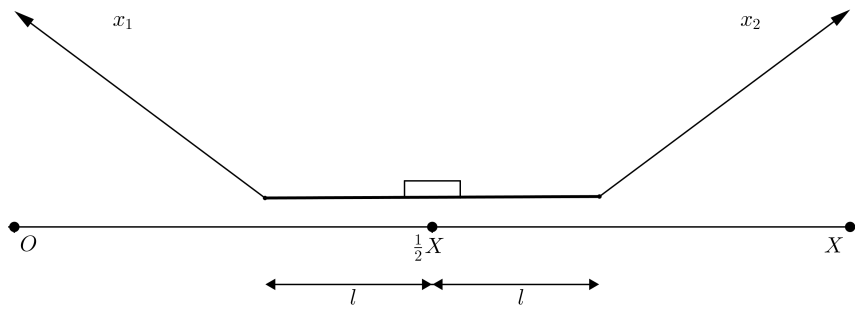


Figure 7.2: Linear airport side view

The rectangle in the middle of Figure 7.1 and Figure 7.2 represents the terminal, i.e., the location where the flight routines begin. We take $X \in \mathbb{Z}_+$ and $Y \in \mathbb{Z}_+$. Then we divide the plane into X parts on the horizontal axis and Y parts on the vertical axis. This way we obtain XY squares of size one. In the center of these squares we define our housing locations with one house per housing location, i.e., $\omega_h = 1, \forall h \in H$. What remains is to define the noise pollution for each flight routine per flight. We do this in such a way that the two flights only create noise pollution on the half of the plane which they fly through. In other words, flight routine x_1 creates noise pollution on $[0, \frac{1}{2}X] \times [0, Y]$ and flight routine x_2 on $[\frac{1}{2}X, X] \times [0, Y]$. Furthermore, in their half of the plane a flight will give noise pollution of 1 on the runways and 0 on the boundaries of the plane. In between the noise pollution drops linear in the distance horizontal and vertical distance to the runway. If we formalize this we get the following.

For a house h on location $[g_X^h, g_Y^h]$, we define $\gamma(g_Y)$ with $g_Y \in [0, Y]$ as

$$\gamma(g_Y) = \begin{cases} \frac{2g_Y}{Y}, & 0 \leq g_Y \leq \frac{1}{2Y} \\ 2 - \frac{2g_Y}{Y}, & \frac{1}{2Y} < g_Y \leq Y \end{cases} .$$

For flight routine x_1 we have the following noise pollution on house h with location $[g_X^h, g_Y^h]$

$$\beta_{h1} = \begin{cases} \gamma(g_Y^h)\frac{2g_X^h}{X-2l}, & 0 \leq g_X^h \leq \frac{X}{2} - l \\ \gamma(g_Y^h), & \frac{X}{2} - l \leq g_X^h \leq \frac{X}{2} \\ 0, & \frac{X}{2} < g_X^h \leq X \end{cases} .$$

For flight routine x_2 given $\epsilon > 0$ we have the following noise pollution on house h with location $[g_X^h, g_Y^h]$

$$\beta_{h2} = \begin{cases} 0, & 0 \leq g_X^h < \frac{X}{2} \\ \gamma(g_Y^h) + \epsilon, & \frac{X}{2} \leq g_X^h \leq \frac{X}{2} + l \\ \gamma(g_Y^h)\left(\frac{2g_X^h}{-X+2l} + \frac{2X}{X-2l}\right) + \epsilon, & \frac{X}{2} + l < g_X^h \leq X \end{cases} .$$

For example if we take $X = 26$, $Y = 11$ and $l = 4$ we would get the following noise pollution for the flight routines.

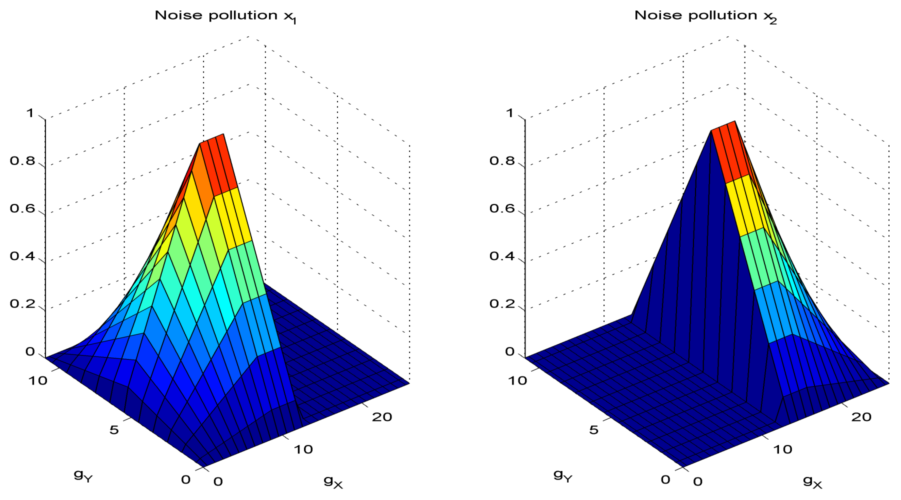


Figure 7.3: Noise pollution for flight routines for linear airport

Note that we introduce the increase of $\epsilon > 0$ in the noise pollution for flight routine x_2, because otherwise we could always interchange the values of x_1 and x_2 in our solutions.

7.1.2 Results

Integer values

We first consider the linear airport with integer valued flights, i.e., $x_r \in \mathbb{Z}_+, \forall r \in R$. We will consider a smaller instance than for the real valued case, because of computation time. Therefore we take $X = 12$, $Y = 4$, $l = 2$ and $\lambda = 100$. We solve the problem optimally with Big-M integer linear programming and heuristically with the three Chinneck algorithms and the dynamic approximation algorithm. We use $k = 3$ with Chinnecks second and third algorithm. We can use any weight rule for our Chinneck algorithms, since the two are identical given weights of one.
For each algorithm we construct a set of solutions by using the bounded objective function method as described in Section 5.3. We use the iterative scheme, described in that section. For our five algorithms we obtain a set of solutions. In case of the Big-M method, these are all Pareto optimal, but using the iterative scheme we can also construct all weakly Pareto optimal solutions. We get the following solutions for the different algorithms

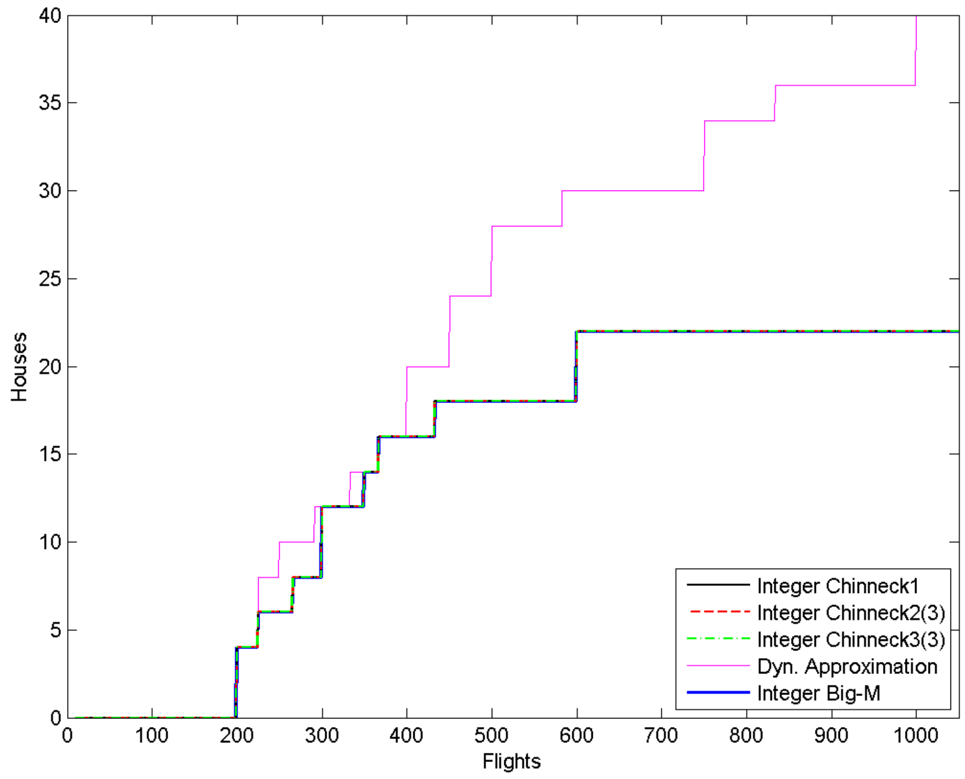


Figure 7.4: Fronts of maximum flight against minimum houses

Note that we interpolated between the different solutions to construct the fronts. We see that the three Chinneck algorithms perform optimally for this instance. We already see from the above that the dynamic approximation algorithms performs to within a factor of two of the optimal solution. The Table 7.1 sums up the different performance statistics.

In terms of computation time, all algorithms finish within a second. In Appendix B.1 a figure

Approximation ratio flights		
	mean	max
Chinneck 1	1.0003	1.0023
Chinneck 2(3)	1.0003	1.0023
Chinneck 3(3)	1.0003	1.0023
Dyn. Approximation	1.0866	1.3341

Approximation ratio houses		
	mean	max
Chinneck 1	1	1
Chinneck 2(3)	1	1
Chinneck 3(3)	1	1
Dyn. Approximation	1.3648	1.8182

Computation time (sec)		
	mean	max
Chinneck 1	0.011823	0.01547
Chinneck 2(3)	0.012185	0.014167
Chinneck 3(3)	0.015123	0.032565
Dyn. Approximation	0.012588	0.016534
Big-M	0.036667	0.10471

Table 7.1: Performance of algorithms for linear Airport
with integer valued flights

can be found of the computation time per solution. Also an animation can be found in Appendix C, which rotates through the found solutions for the Big-M method. Looking at this animation, one can see that between certain subsequent Pareto points there is a large change in value of x_1 and x_2. In other words the flight schedule changes from favoring one flight to favoring the other. This can also be seen in Figure 7.5, which shows the values of the flight routines for different Pareto points found by the Big-M method.

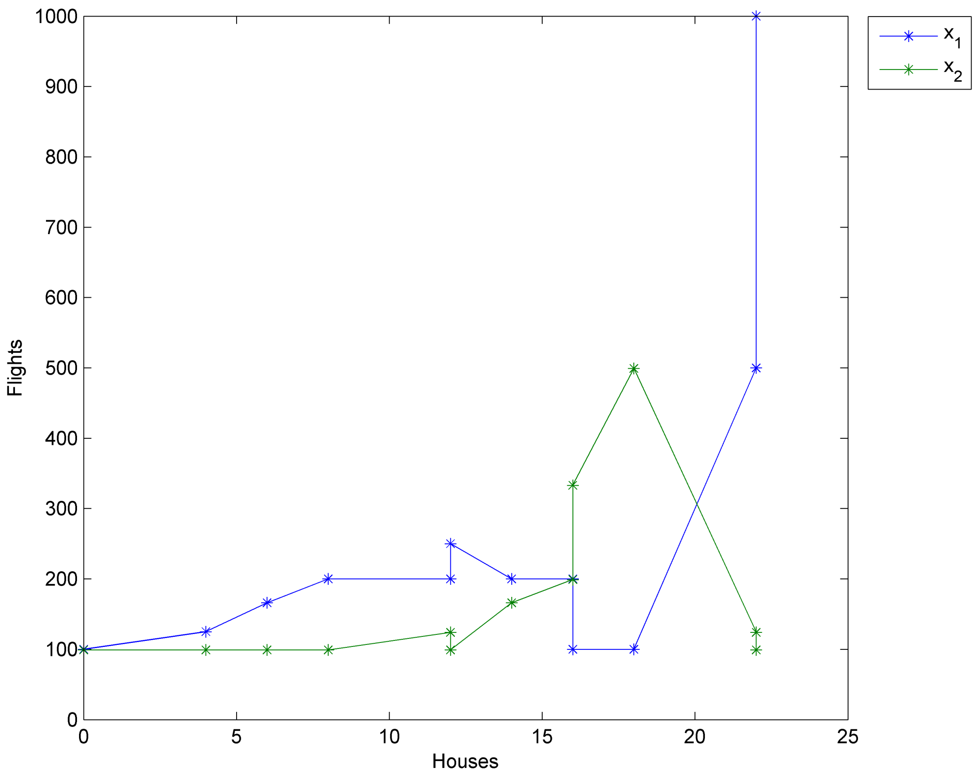


Figure 7.5: Values of x_1 and x_2 for solutions generated by the Big-M method

For example from the Pareto point with 18 violated housing constraints to the Pareto point with 22 housing constraints violated we see a huge change in the values of x_1 and x_2. Thus the solution may greatly depend on the number of housing constraints that are violated or in the same sense the minimum number of flights that have to be scheduled. Note that from a sensitivity analysis point of view, both x_1 and x_2 stay in the basis the entire time, thus x_1 and x_2 are not sensitive from this point of view. However, since their values chance there is also a big change in which constraints are violated. This is due to the way we set up or problem. Scheduling flights x_1 will violate constraints on the left, while scheduling x_2 will violate constraints on the right. Due to the show changes in the flights schedule, the variables associated with which constraint is violated and which not, can be very sensitive. To make this statement more clear, one should look at the animation in Appendix C.

Real values

We will now consider the linear airport with real valued x_r, i.e., $x_r > 0, \forall r \in R$. We have the $X = 26$, $Y = 11$, $l = 4$ and $\lambda = 100$. We solve the problem optimally with Big-M integer linear programming and solve it heuristically with the three Chinneck algorithms and the dynamic approximation algorithm. We use $k = 3$ with Chinnecks second and third algorithm. We construct sets of solutions for our algorithms in a similar fashion as in the integer valued case. We then get the following set of solutions given in Figure 7.6.

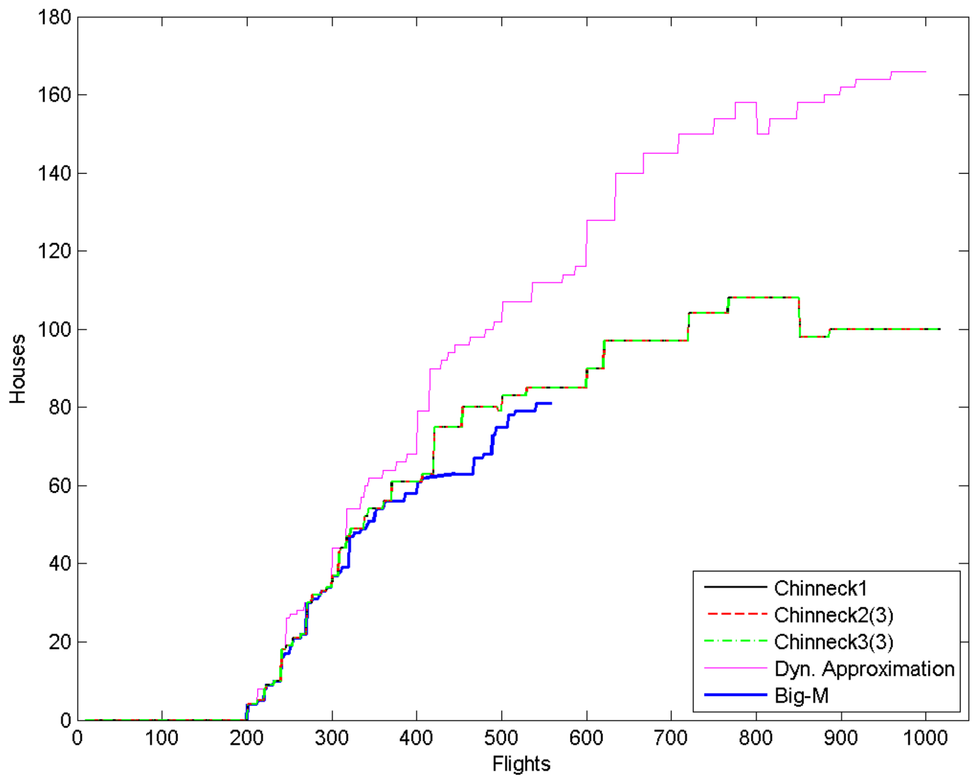


Figure 7.6: Fronts of solutions of maximum flight against minimum houses

Note that we did not compute a solution with the Big-M method for $\sum_{r\in R} x_r \geq 558$. The computation for the next solution in the set run for more then 8 hours, before we cut it off. From Figure 7.6 of the solution sets we see that in this instance the Chinneck heuristics are not optimal anymore, but remain very close to the optimal solution. Another thing that we see in the solution front of the Chinneck heuristics, is that they have a drop in the number of houses suffering an excess amount of noise pollution while they can schedule more flights. This can occur because we are using heuristics to solve the problem. If we would use the postprocessor as described in Section 6.4, we would remove these solutions that are clearly suboptimal. Figure 7.6 with postprocessor can be found in Appendix B.1. We can also see these results from the Table 7.2. Note that for the approximation ratios we only consider solutions found by the heuristic for which $\sum_{r\in R} x_r \leq 558$, because otherwise we would not have an optimal solution to compare it to.

Approximation ratio flights		
	mean	max
Chinneck 1	1.0003	1.0023
Chinneck 2(3)	1.0003	1.0023
Chinneck 3(3)	1.0003	1.0023
Dyn. Approximation	1.0866	1.3341

Approximation ratio houses		
	mean	max
Chinneck 1	1.0199	1.1193
Chinneck 2(3)	1.0197	1.1193
Chinneck 3(3)	1.0205	1.1193
Dyn. approximation	1.0765	1.3035

Computation time (sec)		
	mean	max
Chinneck 1	23.092	82.5625
Chinneck 2(3)	1.9742	4.1542
Chinneck 3(3)	3.0838	7.119
Dyn. approximation	0.021893	0.0544
Big-M	1759.1206	24779.8285

Table 7.2: Performance of algorithms for linear Airport with real valued flights

In Table 7.2 we see that given a number of violated housing constraints, the Chinneck algorithms approximate the optimal number of flights scheduled quite well where the dynamic approximation algorithm performs bad, compared to them. If we look at the approximation of the number of violated housing constraints given the number of flights, we see that the Chinneck heuristics on average are within 2.05 percent of the optimal solution. In the worst case they are within 12 percent. The dynamic approximation algorithm performs worse then the Chinneck heuristics. It does however perform much better then its theoretical approximation bound as found in Theorem 6.3.1, since this bound has a mean of 26.3630 and a maximum of 1909.9.
In terms of computation time we see the behavior we expect. The dynamic approximation algorithm for each solution only solves one linear program, thus its computation time is constant and small. For the second Chinneck algorithm we solve 3 linear programs before removing a constraint and the third will solve 6 before doing so. The first Chinneck algorithm solves a number of LP's equal to the number of constraints with nonnegative dual variables. This will in general be more LP's than the second and third Chinneck algorithms. The Big-M method may take time exponential in the input size and we see this reflected in its computation time.

Figure 7.7 further emphasizes the computation time behavior of our algorithms.

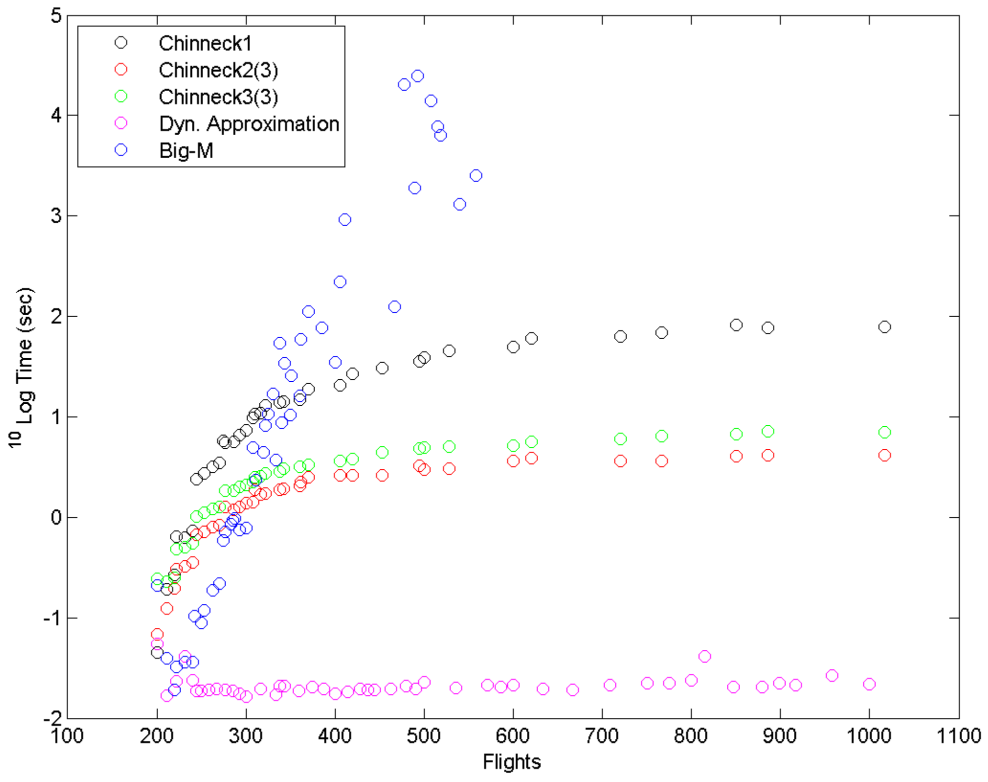


Figure 7.7: Computation time per solution of our algorithms for the linear airport

As in the case with integer values, we can also look at the values of x_1 and x_2 for the solutions found by the Big-M method.

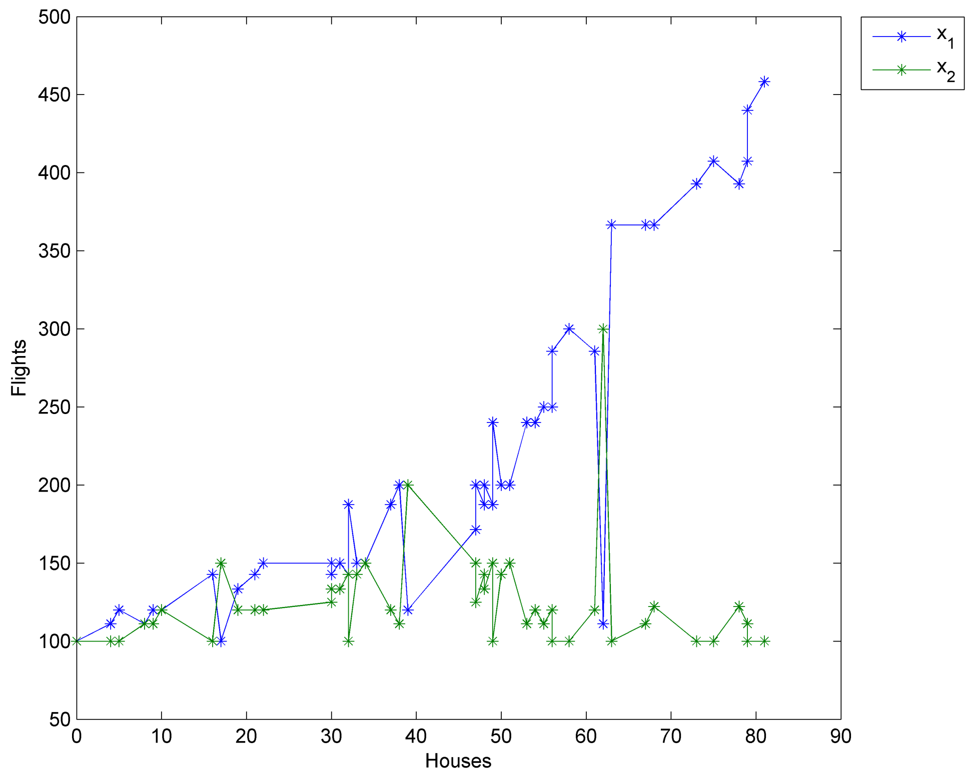


Figure 7.8: Values of x_1 and x_2 for solutions generated by the Big-M method

We see that between certain subsequent Pareto points there is a large change in value of x_1 and x_2. These are found, for example, between the solution which violates 38 constraints and the solution which violates 39 constraints constraints. Another example where this can be found is between the solution which violates 61 constraints, the solution which violates 62 constraints constraints and the solution which violates 63 constraints constraints. Thus, again we see that the constraints that are violated are very sensitive to the amount of violated constraint or equally to the amount of flights scheduled.

A remarkable thing is that the solutions found by the Chinneck heuristics show this behavior a lot less extreme. We get a set of solutions for which the change in value of x_1 and x_2 between subsequent Pareto points seems to be more due to an increase in flights scheduled, see Figure 7.9.

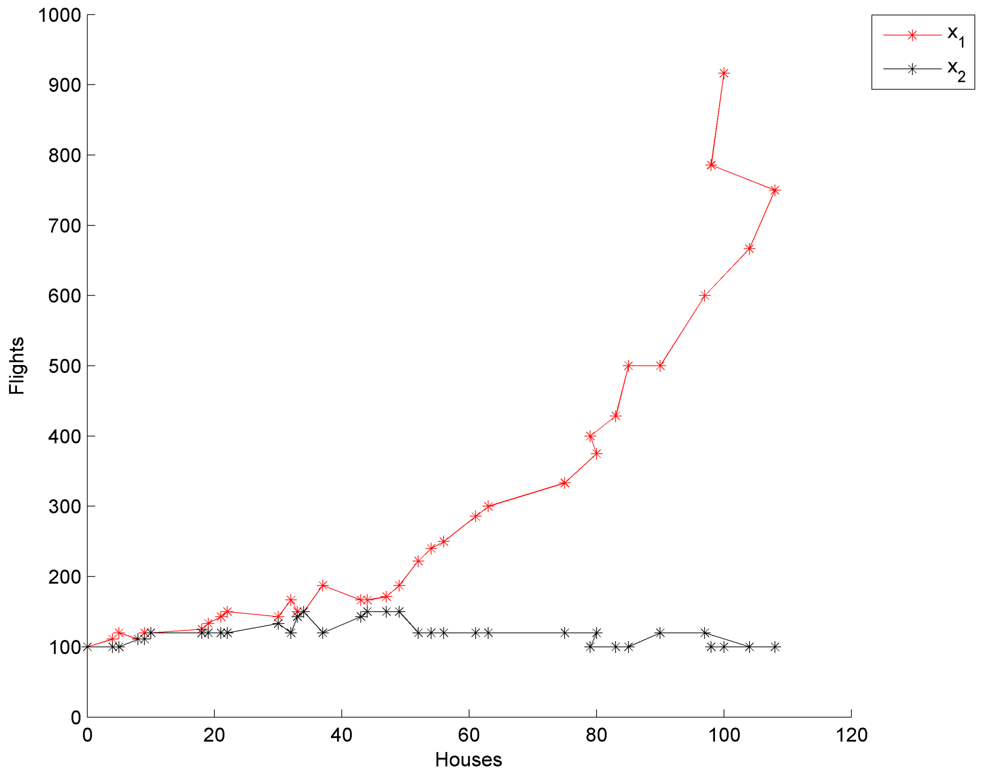


Figure 7.9: Values of x_1 and x_2 for solutions generated by the Chinneck heuristics

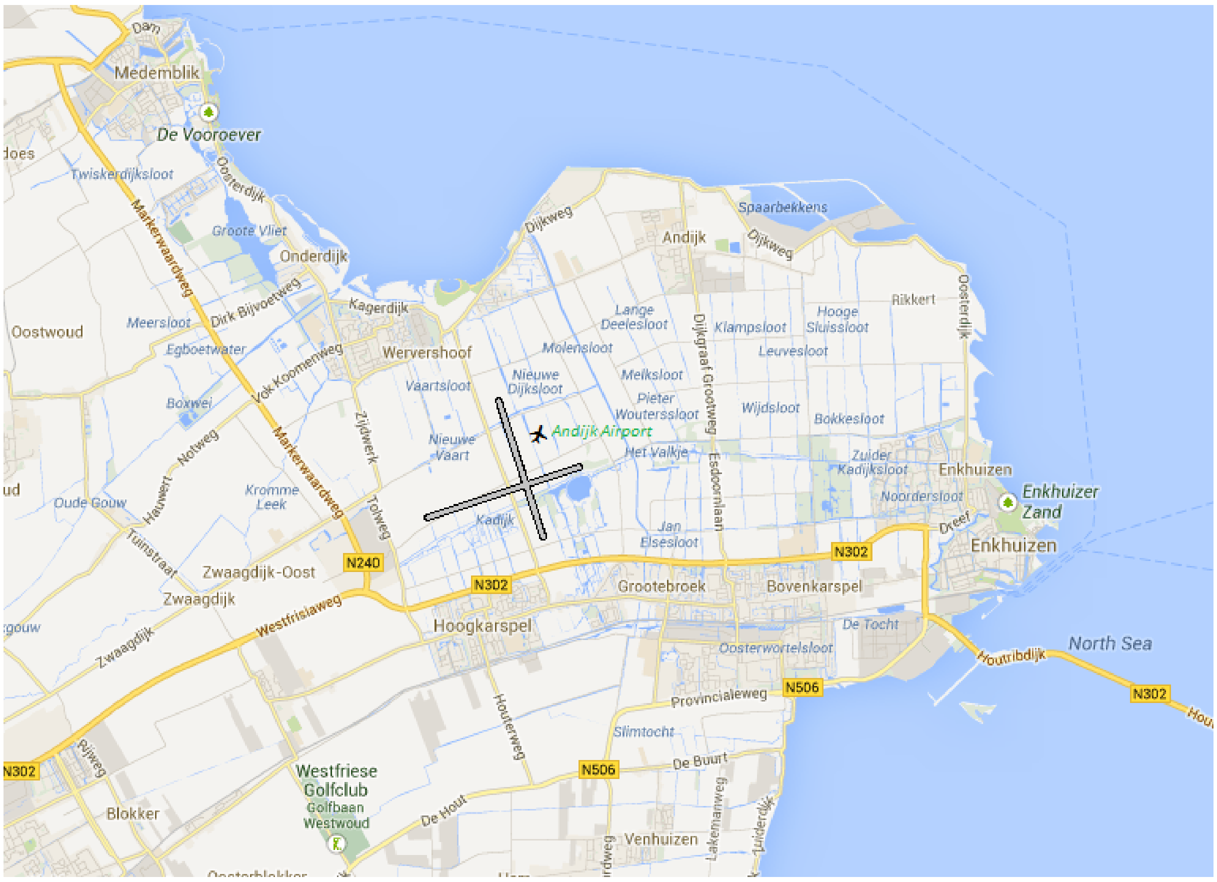
An animation rotating through the found solutions for the Big-M method and those found by the Chinneck algorithms can be found in Appendix C.

7.2 Andijk airport

We will consider the fictitious Andijk airport. We will first describe the airport. Then we will solve three problem instances; Andijk airport consider on a 500 meter grid with measure points, Andijk airport consider on a 500 meter grid without measure points and Andijk airport consider on a 250 meter grid with measure points.

7.2.1 Description

We consider a fictitious airport in the neighborhood of Andijk. Andijk is a village in the North-West of the Netherlands. The fictitious Andijk airport lays South-West of Andijk and has 4 runways.



Although this airport does not exist in real life, it resembles a real life airport quite well. The airport layout, the types of airplanes and the flight routines could be realized in the real world. The locations of the houses are based on the real world. The amount of noise pollution per flight routine scheduled these locations suffer are obtained through the same calculations currently used for real airports. Thus, we can conclude that calculations done on Andijk airport although fictitious, could also be obtained by doing calculations on a real airport.
For Andijk airport we have 112 governmentally imposed measure points. In the proximity of Andijk airport there are 33999 houses. We will group these houses using a 500 by 500 meter grid into 420 housing locations. Given the housing locations we have calculated the noise pollution on these locations for all 2408 flight routines. We will consider this amount expressed in 'Kosten

eenheid' (Ke)[1], which is one of the noise measures used by the Dutch government. The maximum amount of noise pollution a housing location may receive is $H_{\max} = 35$ Ke. In this abstract form the airport will be as in Figure 7.10.

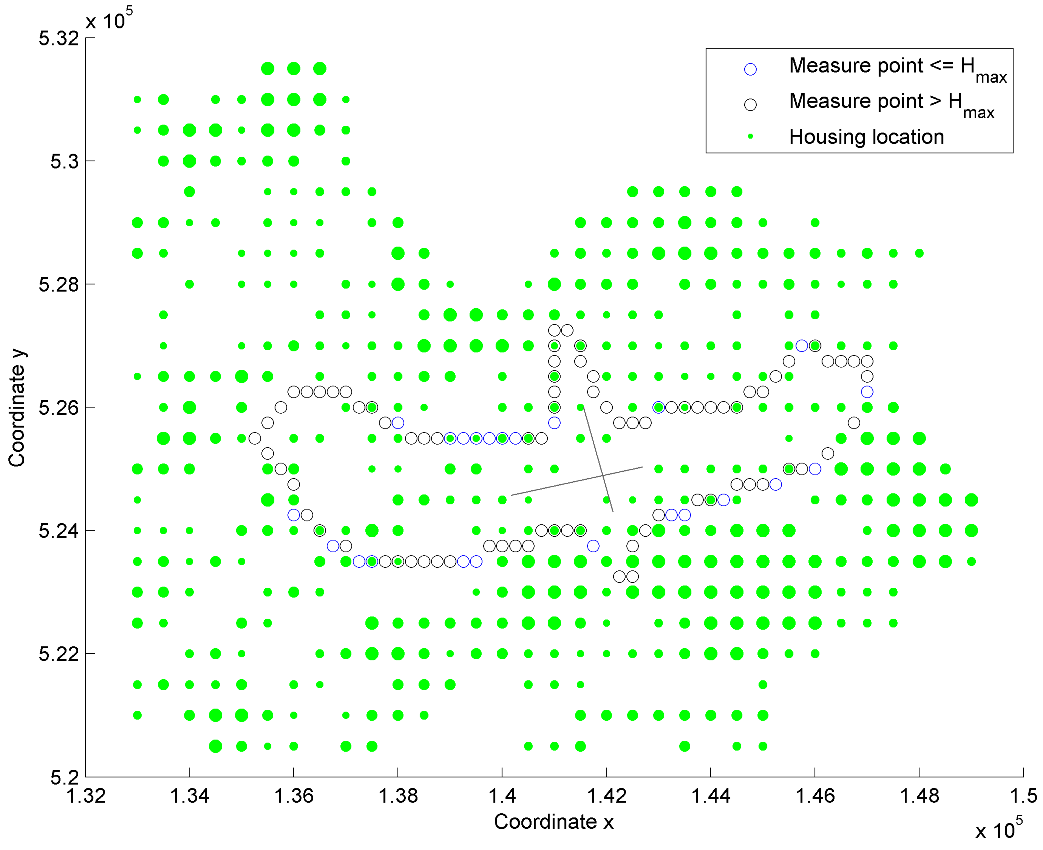


Figure 7.10: Andijk airport layout

In Figure 7.10 the size of the green dots indicates the order of magnitude of the amount of houses on every location. The real number of houses can be found in Figure B.3 which can be found in Appendix B.2. We make a distinction between the government imposed measure points. Some have a imposed noise level smaller than 35 Ke, thus a housing location in the same region cannot receive an excess amount of noise pollution, while for others the level is more than 35 Ke, thus a housing location in the same region can receive too much noise pollution.

7.2.2 Results

For Andijk airport we only solve the problem with real valued flights, i.e., $x_r \geq 0, \forall r \in R$. This is because solving the problem for integer valued flights would take to much computation time. We will solve the problem using the five algorithms described in Chapter 6. We solve it optimally with Big-M integer linear programming and heuristically with the three Chinneck algorithms and the dynamic approximation algorithm. We used $k = 3$ with Chinnecks second and third algorithm and for all three Chinneck algorithm used the weighting as described in (6.1). For all five algorithms we use the bounded objective function method as described in Section 5.3 to

[1]'Kosten eenheid' is named after prof.dr.ir. C.W. Kosten who introduced the measure in the sixties.

obtain solutions. We use the iterative scheme we described in that section to obtain a set of solutions for every algorithm. After solving we get for every algorithm a set of solutions. If we interpolated between the different solutions to construct fronts, we get Figure 7.11. This front is actually the (weak) Pareto front in the Big-M methods case.

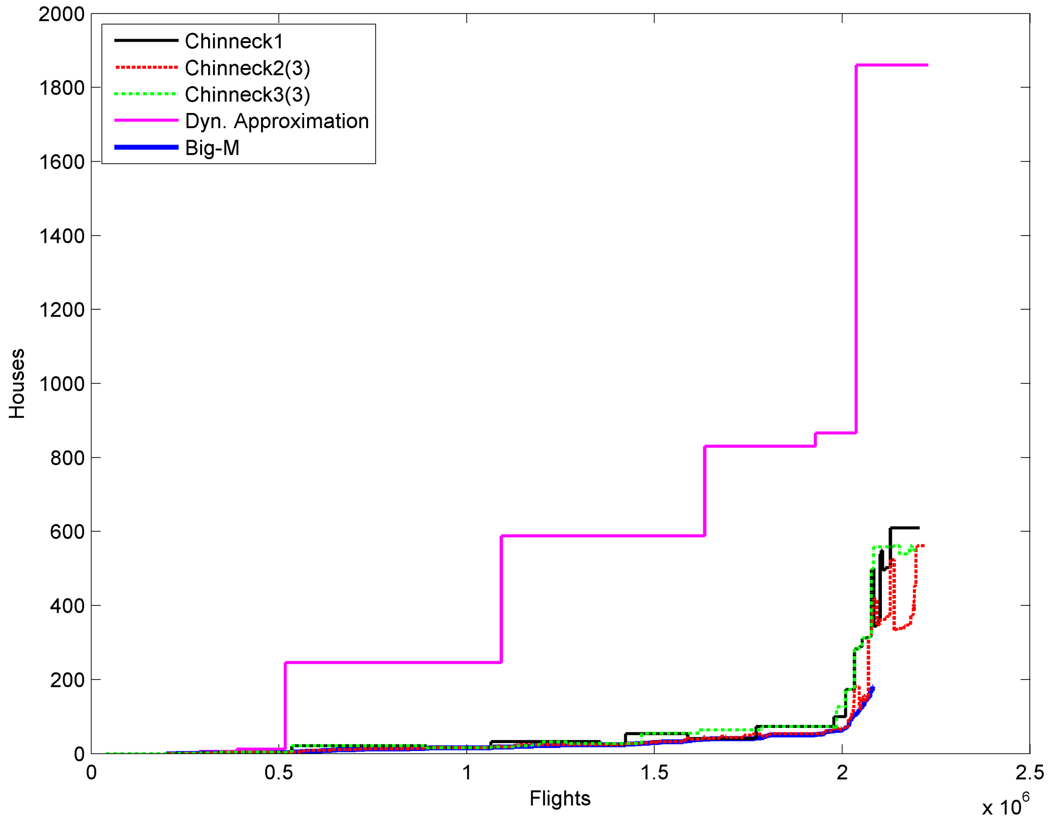


Figure 7.11: Fronts of maximum flight against minimum houses for Andijk airport

Note that up to this point we did not use any pre- or postprocessors. Furthermore note that we stopped computation for the Big-M method for $\sum_{r \in R} x_r \geq 2081520$. This because the next computation took more than 8 hours to complete. We see that indeed the Big-M method produces a real Pareto front. Chinnecks algorithms are quite close and the dynamic approximation algorithm can be quite off. Table 7.3 illustrates this further.

From Table 7.3 we see that the Chinneck heuristics are on average performing much better than the dynamic approximation algorithm. From the three Chinneck heuristics the second algorithm has average and in the worst case the best approximation ratios. The approximation ratio of the Chinneck algorithms per solution can be found in Figure B.4 in Appendix B.2. There we see that the worst case performance of the first and third algorithm in approximating the number of houses receiving an excess amount of noise pollution, is the solution found for 53431 flights. The Figure 7.12 looks more closely at this solution.

Approximation ratio flights		
	mean	max
Chinneck 1	1.0751	1.4212
Chinneck 2(3)	1.0303	1.4212
Chinneck 3(3)	1.0683	1.4212
Dyn. Approximation	1.2203	1.4737

Approximation ratio houses		
	mean	max
Chinneck 1	1.7149	4.2
Chinneck 2(3)	1.3166	2.2176
Chinneck 3(3)	1.6323	4.2
Dyn. Approximation	20.1463	82

Computation time (sec)		
	mean	max
Chinneck 1	109.3784	219.0582
Chinneck 2(3)	18.1761	44.585
Chinneck 3(3)	20.856	40.8931
Dyn. Approximation	0.37338	0.43565
Big-M	154.2396	1867.882

Table 7.3: Performance of algorithms for Andijk airport

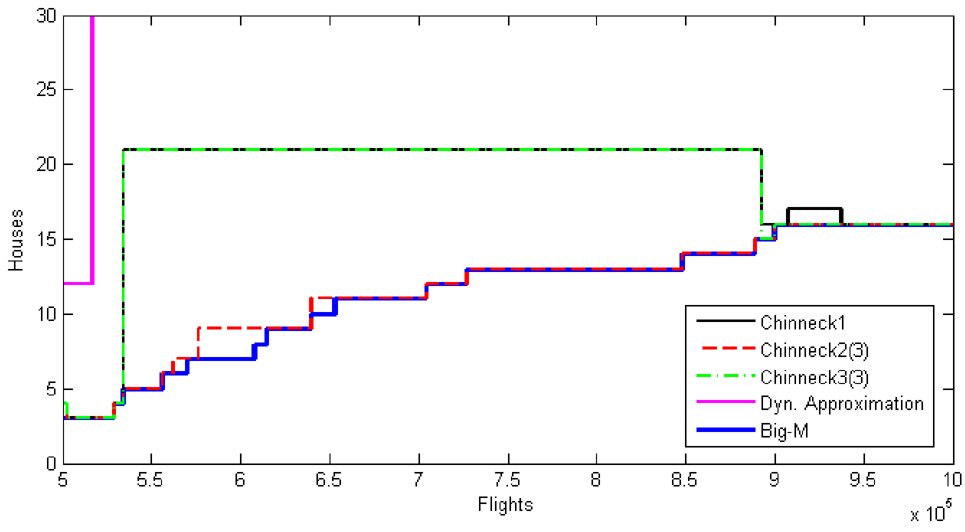


Figure 7.12: Fronts of maximum flight against minimum houses for Andijk airport for $5 \cdot 10^5 \leq \sum_{r \in R} x_r \leq 10^6$

Looking at Figure 7.12 we see that we can improve this worst case performance by using a Pareto filter. The solutions is obviously not a Pareto optimal solution, since we can also schedule $9.0 \cdot 10^5$ flights using the same algorithms with only 15 instead of 21 houses receiving an excess amount of noise pollution. For the dynamic approximation algorithm, its approximation ratio for the number of houses is still much better than its theoretical approximation bound as found in Theorem 6.3.1. This bound has a mean of $8.44 \cdot 10^8$ and a maximum of $7.59 \cdot 10^9$.

If we look at the computation time of our five algorithms we see results similar to those of the linear airport with real valued flights. Next we comment on the computation time of our algorithms. The computation time for the dynamic approximation algorithm for each solution is almost constant and small, due to the fact that the algorithm solves only a single LP. For the second Chinneck algorithm we solve 3 linear programs before removing a constraint and the third will solve 6 before doing so. The first Chinneck algorithm solves a number of LP's equal to the number of constraints with nonnegative dual variables, which in general will be more than for the second and third Chinneck algorithms. The Big-M method may take exponential time and this is reflected in its computation time, which is the largest computation time of all our algorithms, even though we stopped its computation at 2081520 scheduled flights. Figure 7.13 shows the computation time per solution. It further illustrates the computation time behavior of our algorithms.

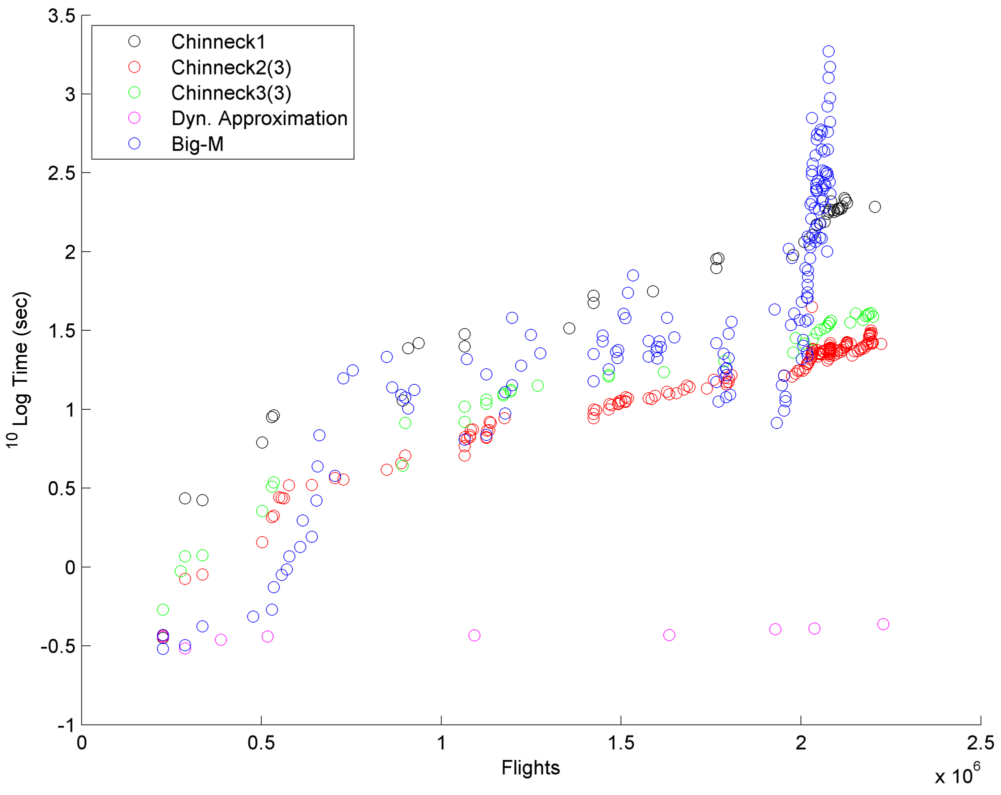


Figure 7.13: Solutions - time needed for computation

We will now look at the flight routines which are used to schedule flights. For the Big-M method a total of 84 different flight routines have values greater than zero in at least one of the found solutions. To get an insight in the solutions found by the Big-M method we can look at the Figure 7.14. It shows for every solution the fraction of total flights scheduled per flight routine. We see, that from the solutions where we have 22 or more houses getting an excess amount of noise pollution, the flight routines 5, 157, 690, 763, 776, 823 and 887 are always greater then zero and account on average for 65% of the total flights scheduled. Thus, after a certain point these flight routines are not sensitive to the amount of flights scheduled or the amount of constraints violated. Because these flight routines are not sensitive there also will be a set of constraints that is not, and will be violated in the solutions, where 22 or more houses get an excess amount of noise pollution. On the other hand there are also a few flight routines which are sensitive. For example routines 1431, 1435, 1479 and 1483 are switching between equal to zero and greater to zero for the solution with 27 upto 37 houses having an excess amount of noise pollution. Thus their are some flight routines, which are very sensitive to the amount of flights scheduled or the amount of constraints violated.

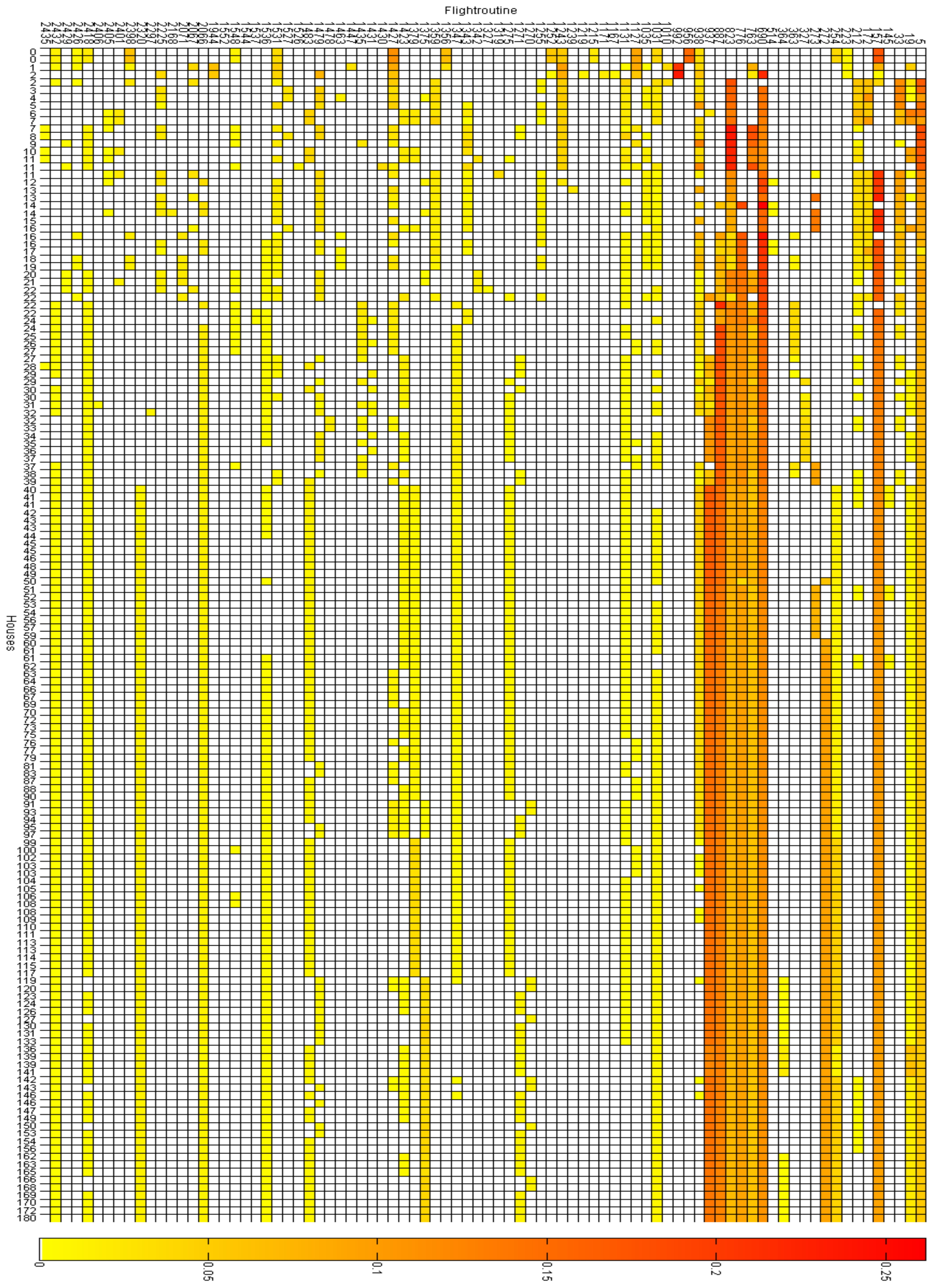


Figure 7.14: Relative of amount of flights per flight routine per solution

For the algorithms we obtain a real valued solution $x_r, \forall r \in R$. Of course we are also interested in an integer valued solution. One way of obtaining an integer solution is by rounding down every x_r to the nearest integer. Since all our constraints are less-or-equal constraints this procedure produces a feasible solution. When doing this we lose at most 30.01 flights. With a minimum of 226168.17 flights scheduled this is a loss of less than 0.014 %. Thus we can get an integer valued solution, which is close to its optimal solution quite easily for Andijk airport.

Pre and postprocessors

If we consider only the binding and not the optional constraints, we obtain the Pareto point (i.e., the anchor point for this objective function) given in Figure 7.15.

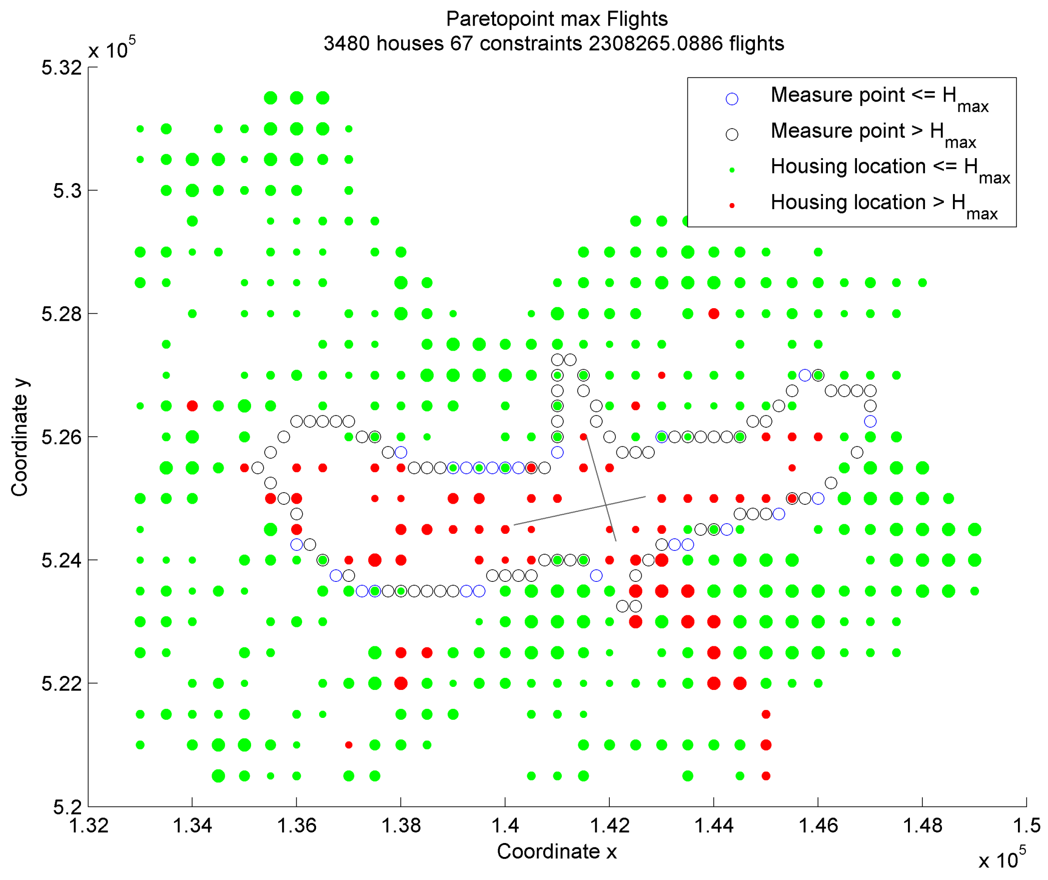


Figure 7.15: Pareto point with maximum number of flights

From this plot we see that a lot of housing locations are not violated. Thus, it may be that some optional constraints cannot be violated at all due to the binding ones and that we should use the preprocessor as described in Section 6.4 to remove some constraints from our computation. If we test which can and which cannot be violated we get the plot in Figure 7.16

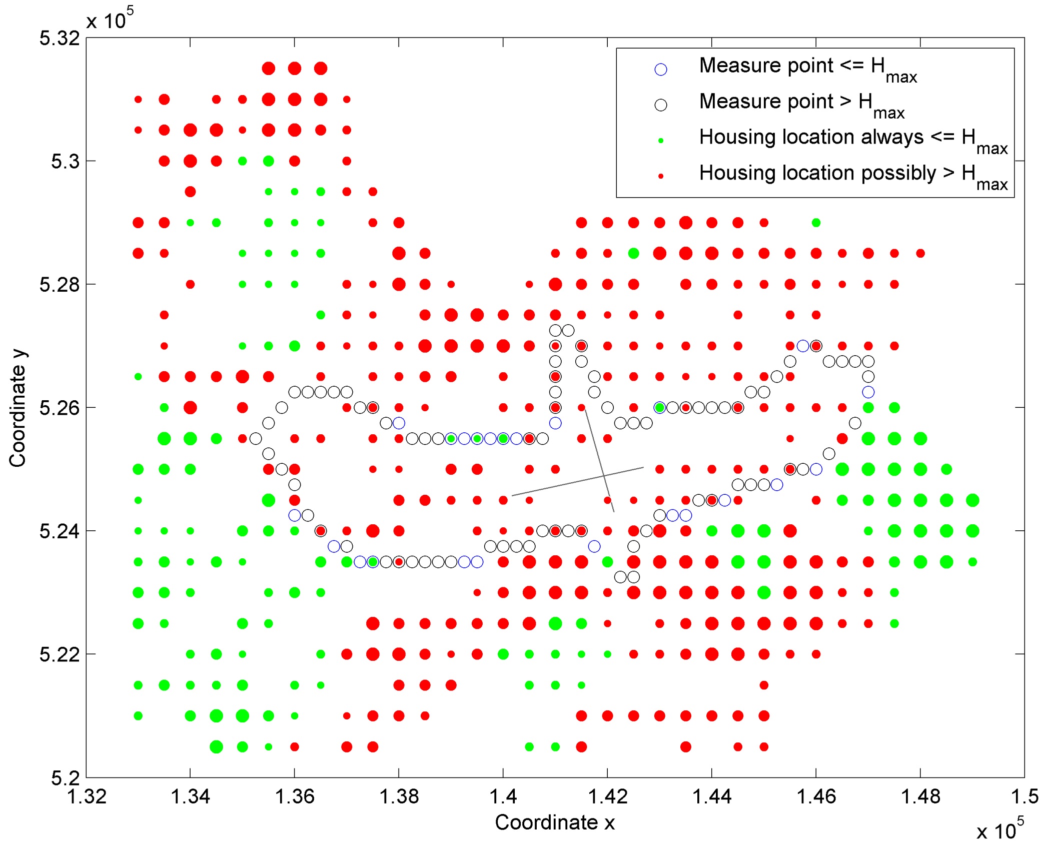


Figure 7.16: Possible violated constraints for Andijk airport

We see that there are 127 locations for which the constraint is always satisfied, thus, we could remove these constraints to obtain faster algorithms. We compute solutions with the preprocessor for $\sum_{r\in R} x_r \leq 1800000$

Computation time (sec)				
	Without preprocessor		**With preprocessor**	
	mean	max	mean	max
Chinneck 1	36.3625	95.4999	30.5803	71.5745
Chinneck 2(3)	8.68754	16.5397	6.7131	12.3754
Chinneck 3(3)	9.7793	22.9615	8.6452	26.9939
Dyn. Approximation	0.35965	0.40393	0.30556	0.35554
Big-M	17.1556	70.6871	14.035	42.9641

Table 7.4: Computation time with and without preprocessor

The preprocessor itself takes 82 seconds to complete. Thus we can conclude that the preprocessor is effective in reducing the computation time. We also implemented the postprocessor as described in Section 6.4 but this was not effective for Andijk airport. It only reinstated one constraint in total (this was for a solution found by the third Chinneck heuristic).
We already saw that in the solution sets found by the Chinneck algorithms there are several obvious non optimal solutions. We can remove this by using our Pareto filter. If we again interpolated between the different points in the set, given in Figure 7.17.

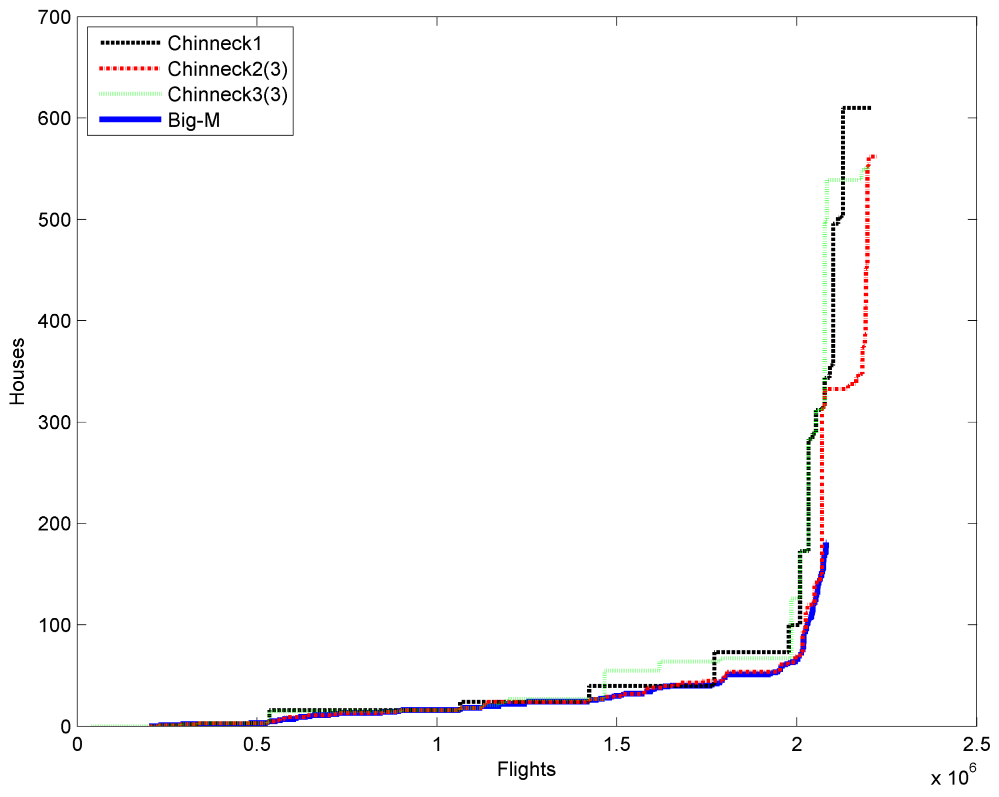


Figure 7.17: Fronts of maximum flight against minimum houses with Pareto filter

Note that we did not plot the solutions for the dynamic approximation algorithm, since the Pareto filter did not remove any solutions. We see from this figure that all algorithms now have a increasing set of solutions. If for the filtered solutions we look at their approximation ratios we get the results given in Table 7.5.

Approximation ratio flights				
	Before Pareto filter		**After Pareto filter**	
	mean	max	mean	max
Chinneck 1	1.0751	1.4212	1.0571	1.4212
Chinneck 2(3)	1.0303	1.4212	1.0209	1.4212
Chinneck 3(3)	1.0683	1.4212	1.0541	1.4212
Dyn. Approximation	1.2203	1.4737	1.2203	1.4737

Approximation ratio houses				
	Before Pareto filter		**After Pareto filter**	
	mean	max	mean	max
Chinneck 1	1.7149	4.2	1.5577	3.2
Chinneck 2(3)	1.3166	2.2176	1.2446	2.0458
Chinneck 3(3)	1.6323	4.2	1.5711	3
Dyn. Approximation	20.1463	82	20.1463	82

Table 7.5: Performance of algorithms for Andijk airport after Pareto filter

We see that given the number of houses getting an excess amount of noise pollution, the average approximation ratio of the number of flights is improved for the three Chinneck algorithms. Their worst case performance is not improved. Given the number of flights the average approximation ratio for the number of houses is improved for all three Chinneck algorithms as well as their worst case performance. The second Chinneck algorithm is still performing the best.

Without measure points

One of the questions posed before is whether or not we could schedule more flights while giving less houses to much noise pollution (i.e., more than its threshold amount) if we would remove the government imposed measure points. Therefore we will now consider Andijk airport without the measure points, i.e., $Q = \emptyset$. This also means that we do not have any binding constraints, which implies that there will be a solution for which we can schedule an infinite amount of flights. We use the same algorithms as for our original airport to obtain solutions for this instance of Andijk airport. If we again interpolated between the different solutions to construct fronts, we get the results presented in Figure 7.18

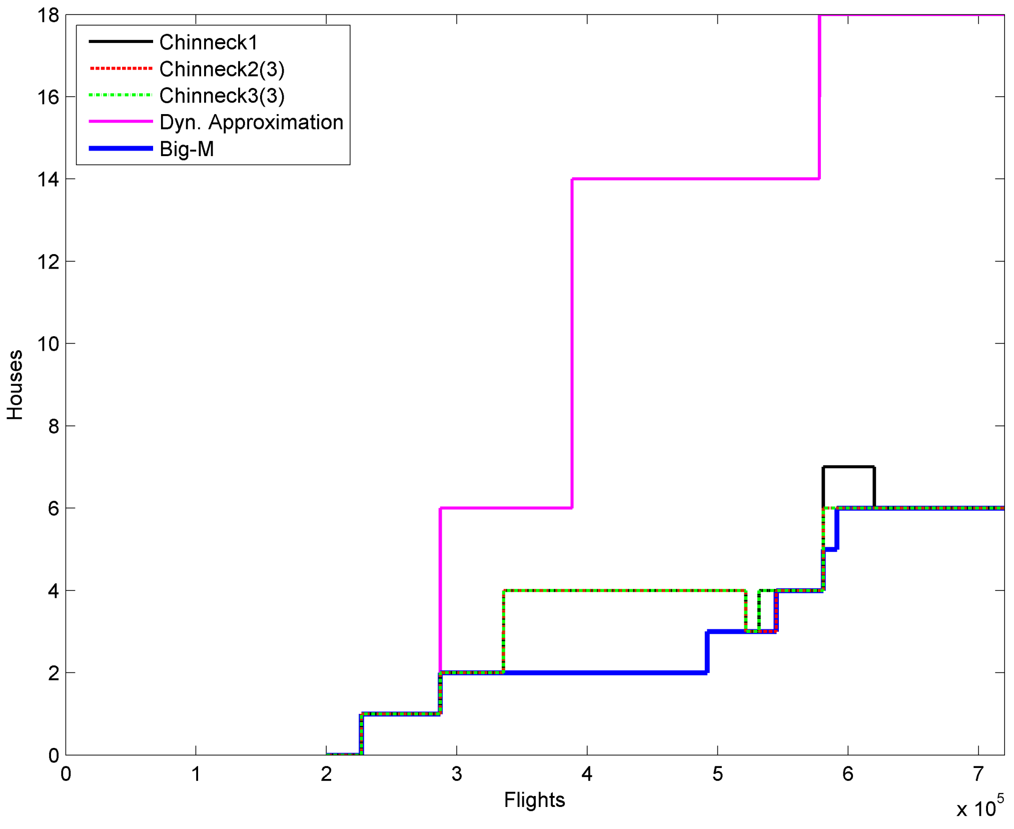


Figure 7.18: Fronts of maximum flight against minimum houses

Note that our algorithms find solutions that violate the same constraints for $7 \cdot 10^5$ flights scheduled as for an infinite number of flights scheduled. We see that we could schedule an infinite amount of flights while only giving 6 houses more than the threshold amount of noise pollution. This infinite amount of flights is scheduled by using flight routine 823 and 928 by the Big-M method and the three Chinneck heuristics. The dynamic approximation algorithm uses flight routine 157. The Pareto point for the infinite amount of flights found by the Big-M method (and the three Chinneck heuristics) is shown in Figure 7.2.2.

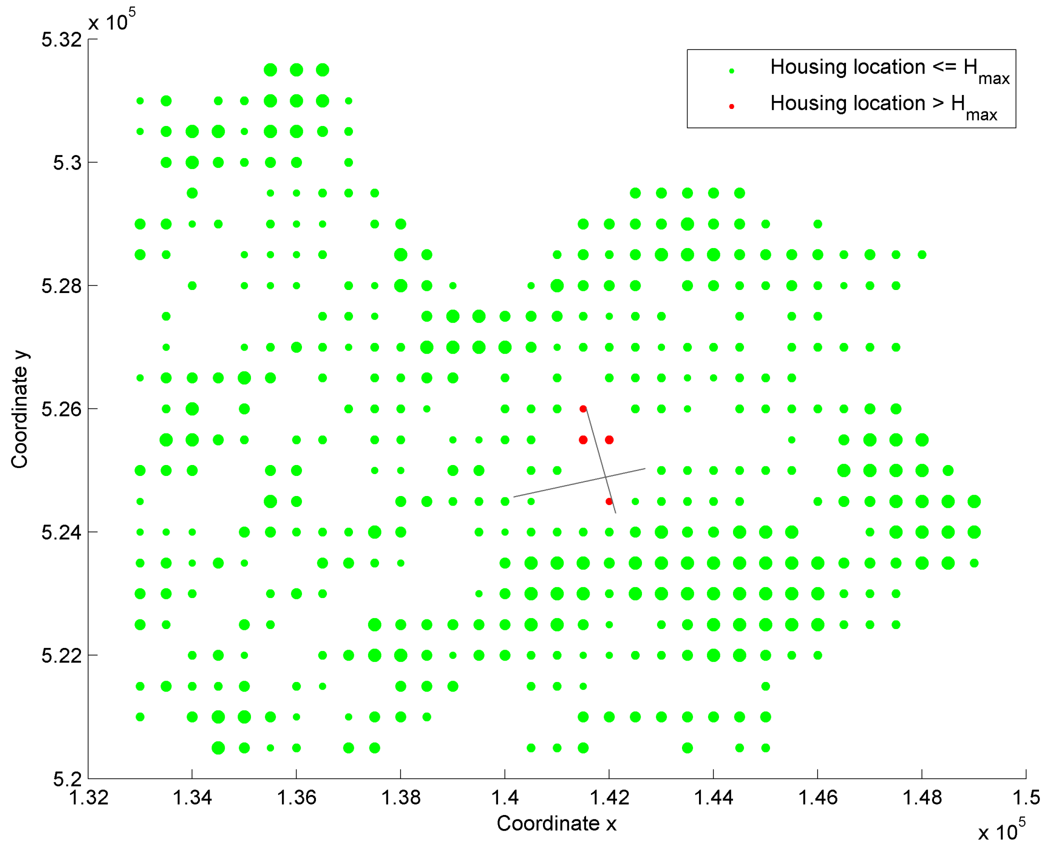


Figure 7.19: Pareto point Andijk airport with infinite amount of flights scheduled

Table 7.6 summarisses the performance of our algorithms for this instance of Andijk airport.

Approximation ratio flights[2]				
	Before Pareto filter		After Pareto filter	
	mean	max	mean	max
Chinneck 1	1.1007	1.4653	1.0892	1.4653
Chinneck 2(3)	1.0864	1.4653	1.0765	1.4653
Chinneck 3(3)	1.0922	1.4653	1.0823	1.4653
Dyn. Approximation	NA	NA	NA	NA

Approximation ratio houses				
	Before Pareto filter		After Pareto filter	
	mean	max	mean	max
Chinneck 1	1.2167	2	1.1762	1.5
Chinneck 2(3)	1.12	2	1.0875	1.5
Chinneck 3(3)	1.1533	2	1.1292	1.5
Dyn. Approximation	3.3	7	3	7

Computation time (sec)		
	mean	max
Chinneck 1	10.8933	21.4314
Chinneck 2(3)	2.4724	3.8327
Chinneck 3(3)	3.8496	6.3904
Dyn. Approximation	0.51497	0.54709
Big-M	1.1083	2.2462

Table 7.6: Performance of algorithms for Andijk airport without measure points

We did not consider the preprocessor because since $Q = \emptyset$ all constraints could be violated. We see that for the approximation ratio of the number of flights, the three Chinneck algorithms have similar results as those for the Andijk airport instance with measure points. For the dynamic approximation algorithm we give no approximation ratio since it has a solution for which it cannot schedule an infinite amount of flights while it does give 6 houses more than their threshold amount of noise pollution.
For the approximation ratio, for the number of houses which receive more than their threshold amount of noise pollution, we see that all algorithm perform better than for the instance with measure points. Especially the second Chinneck algorithm performs well. On average, it is only 9% away from the optimum if we use our Pareto filter. The dynamic approximation algorithm is again performing the worst for approximating the number of houses, although it is still better then its theoretical approximation bound, which has a mean of $1.5 \cdot 10^9$ and a maximum of $7.6 \cdot 10^9$ for this instance of Andijk airport.
In terms of computation time we see that the Big-M method is fastest after the dynamic approximation algorithm. This is the case because it has to violate at most 4 constraints to obtain a solution. When comparing this to figure 7.13 this is not unexpected, also there the Big-M method is faster than the Chinneck heuristics if the number violated constraints is less than 6 (which is for 652922 flights scheduled and 10 houses receiving more than their threshold amount of noise pollution).

[2]We did not consider the last solutions found per algorithm for this approximation ratio (the solutions having an infinite amount of flights), since this would make this ratio arbitrary bad for two of our heuristics.

For this instance if we would round down our real valued solution to an integer valued solution we would lose at most 30.08. With a minimum of again 226168.17 flights scheduled this is a loss of less then 0.014 %.
In Appendix B.2.1 a figure can be found with the solutions after using a Pareto filter, furthermore in this Appendix there are also figures showing the flight routines which are used to schedule flights for the different solutions found by the Big-M method. In Appendix C their is an animation available which shows the solutions found by the Big-M method. One thing that is interesting, is that the flights scheduled in the solutions of the Big-M method for this instance of Andijk airport have a lot of similarities with the flights scheduled for the first 7 solutions of the Big-M method for Andijk airport with measure points, i.e., the mostly use the same routines to construct these solutions.

Fine grid

Up until now we have considered Andijk airport where we calculated noise pollution levels for our flight routines on a grid of 500 by 500 meters. Of course this grid can have an influence on the solutions found because we will group houses together in housing locations using this grid. Therefore we will now consider Andijk airport only now on a grid of 250 by 250 meters. This will give use the airport layout found in Figure 7.20.

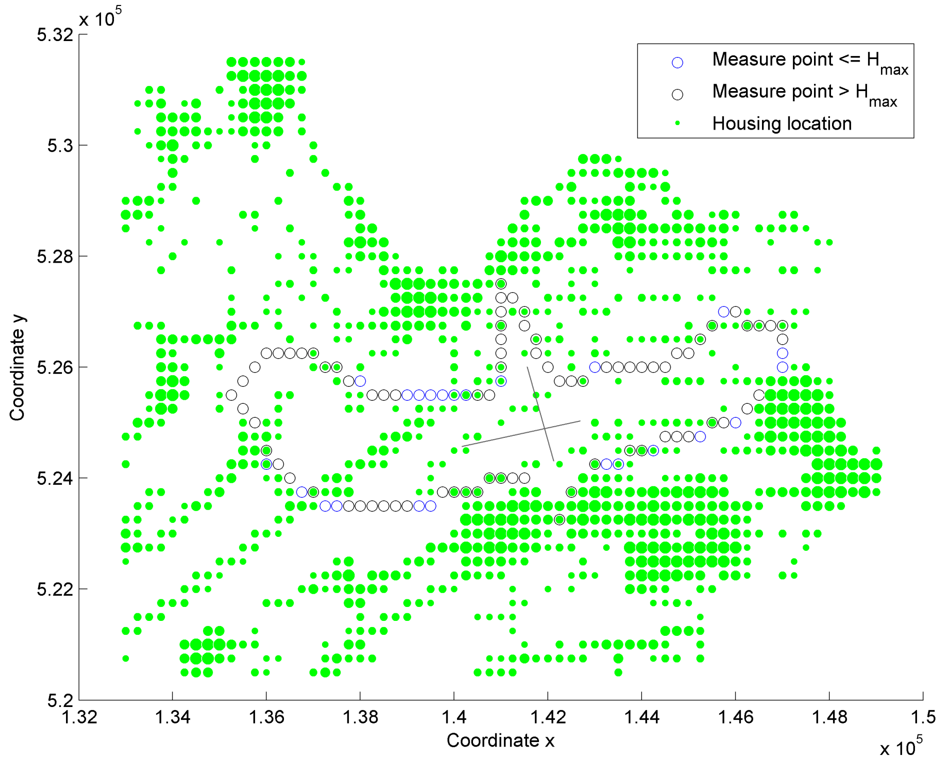


Figure 7.20: Andijk airport layout with 250 meter grid

It is obvious that we have 960 housing locations which is 540 more than in the original set up. We solve this instance using the same algorithms as before and with our preprocessor active. Our preprocessor removes 411 constraints from the computation. For all five algorithms we use the bounded objective function method as described in Section 5.3 to obtain solutions and we use the iterative scheme we also described in that section to obtain a set of solutions. If we then interpolate between the different solutions to construct fronts, we get Figure 7.21.

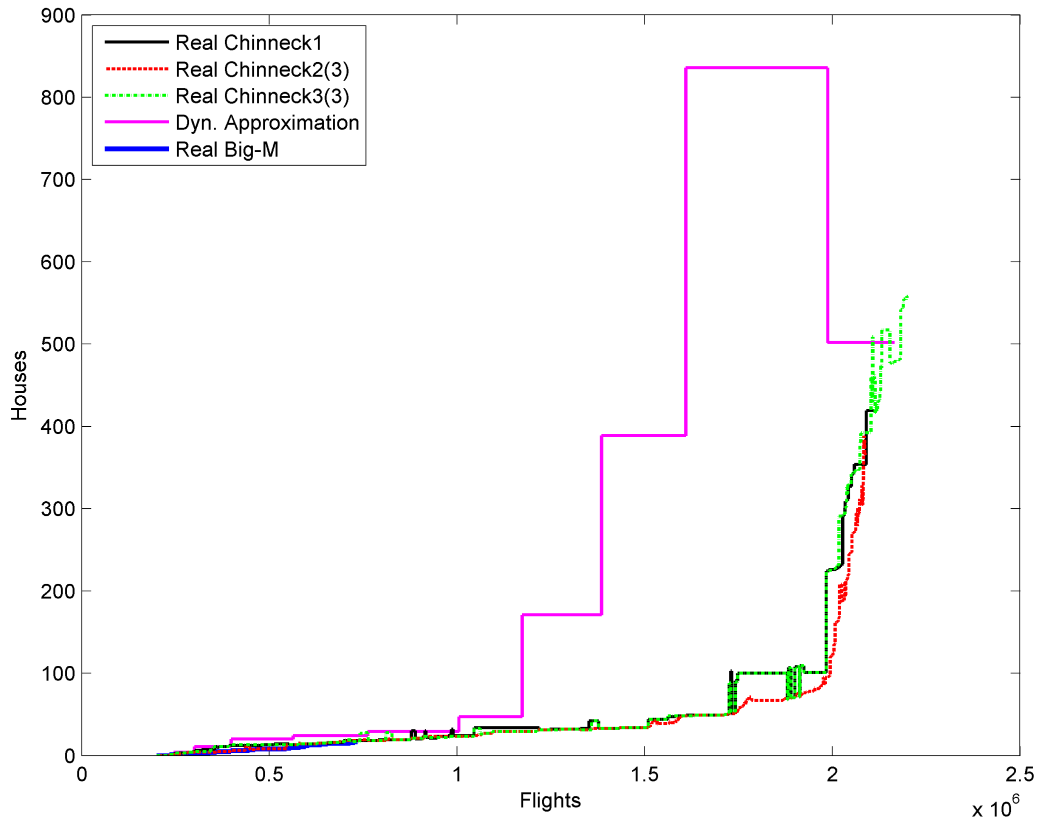


Figure 7.21: Fronts of maximum flight against minimum houses for Andijk airport with 250 meter grid

Note that we again stop the computation of the Big-M method because it took more than 8 hours to complete. This is for $\sum_{r\in R} x_r \geq 734317$. We see that the three Chinneck algorithm are quite close to the optimal solution for $\sum_{r\in R} x_r \leq 734317$ and the dynamic approximation algorithm is also closer compared to Figure 7.11. If we compare our results for Andijk airport on a 250 meter grid as to those for Andijk airport on a 500 meter grid with preprocessor we get the Table 7.7. Note that for the results on the 500 meter grid we only take solutions into account with a number of flights less or equal then the maximum number of flights found by our algorithms on the 250 meter grid. Thus, for Big-M method for Andijk airport on a 500 meter grid the results are based on the solutions found with $\sum_{r\in R} x_r \geq 734317$.

Approximation ratio flights				
	500 meter grid		250 meter grid	
	mean	max	mean	max
Chinneck 1	1.0751	1.4212	1.2371	1.6585
Chinneck 2(3)	1.0303	1.4212	1.0804	1.3326
Chinneck 3(3)	1.0683	1.4212	1.2295	1.6768
Dyn. Approximation	1.2203	1.4737	1.1838	1.4968

Approximation ratio houses				
	500 meter grid		250 meter grid	
	mean	max	mean	max
Chinneck 1	1.7149	4.2	1.6904	3
Chinneck 2(3)	1.3166	2.2176	1.2583	2
Chinneck 3(3)	1.6323	4.2	1.65	3
Dyn. Approximation	20.1463	82	2.5667	4

Computation time (sec)				
	500 meter grid		250 meter grid	
	mean	max	mean	max
Chinneck 1	109.3784	219.0582	326.8861	1147.4636
Chinneck 2(3)	18.1761	44.585	51.9041	97.5534
Chinneck 3(3)	20.856	40.8931	66.3837	151.9274
Dyn. Approximation	0.37338	0.43565	1.02	1.1901
Big-M	154.2396	1867.882	402.1131	3869.1713

Table 7.7: Performance of algorithms for Andijk airport for 250 and 500 meter grid size

If we look at the approximation ratio for the number of flights scheduled, we see that in general, the heuristic perform better for the 500 meter grid than for the 250 meter grid. Only the mean of dynamic approximation algorithm and the maximum of the second Chinneck algorithm are a little better.
For the approximation ratio of the number of houses receiving an excess amount of noise pollution it is the complete opposite. Here the first Chinneck algorithm, the second Chinneck algorithm and the dynamic approximation algorithm perform better on the 250 meter, than on the 500 meter grid. Especially the dynamic approximation algorithm sees a great improvement. Also the worst case performance of all heuristics is better on the 250 meter grid. The theoretical bound for the dynamic approximation algorithm has an average of $1.93 \cdot 10^9$ and a maximum of $2.32 \cdot 10^{10}$.
In terms of computation time we see an increase, which is to be expected. We had 297 housing locations remaining after the preprecessor with the 500 meter grid and we now have 549 with the 250 meter grid after the preprecessor. Since we do not have more houses, more constraints will probably need to be violated to attain the same amount of flights. So computation times increase. We see that they increase with a factor between 2 and 6 depending on the algorithm.

In Figure 7.21 we have seen that our heuristic produce some obvious non optimal solutions. We therefore use our Pareto filter to remove these for the solutions sets. We get the following results given in Table 7.8.

Approximation ratio flights				
	500 meter grid		250 meter grid	
	mean	max	mean	max
Chinneck 1	1.0571	1.4212	1.2297	1.6195
Chinneck 2(3)	1.0209	1.4212	1.0591	1.2112
Chinneck 3(3)	1.0541	1.4212	1.2235	1.6768
Dyn. Approximation	1.2203	1.4737	1.1838	1.4968

Approximation ratio houses				
	500 meter grid		250 meter grid	
	mean	max	mean	max
Chinneck 1	1.5577	3.2	1.6714	3
Chinneck 2(3)	1.2446	2.0458	1.2218	2
Chinneck 3(3)	1.5711	3	1.6357	3
Dyn. Approximation	20.1463	82	2.5667	4

Table 7.8: Performance of algorithms for Andijk airport for 250 and 500 meter grid size with Pareto filter

We see that the Pareto filter improves the approximation ratios for our Chinneck heuristics a little for the 250 meter grid. On the 500 meter grid we see small improvements for the approximation ratio of the number of flights scheduled, but we see a bigger improvement for the the approximation ratio for the houses getting an excess amount of noise pollution given the number of flights scheduled.
In Appendix B.2.2 figures can be found of the computation time needed per solution found and of the solutions when filtered by the Pareto filter. Furthermore in Appendix C their are animations available which show the solutions found by the Big-M method and by the second Chinneck algorithm.

Seeing the results from Table 7.8 one might wonder whether or not one can improve the approximation ratio of our heuristics for the 500 meter grid by solving the problem heuristically for a 250 meter grid. Thus, we want to compare the solutions found by the heuristics for the 250 meter grid to the solutions found by the Big-M method for the 500 meter grid. Figure 7.22 depicts the sets of solutions we are comparing.

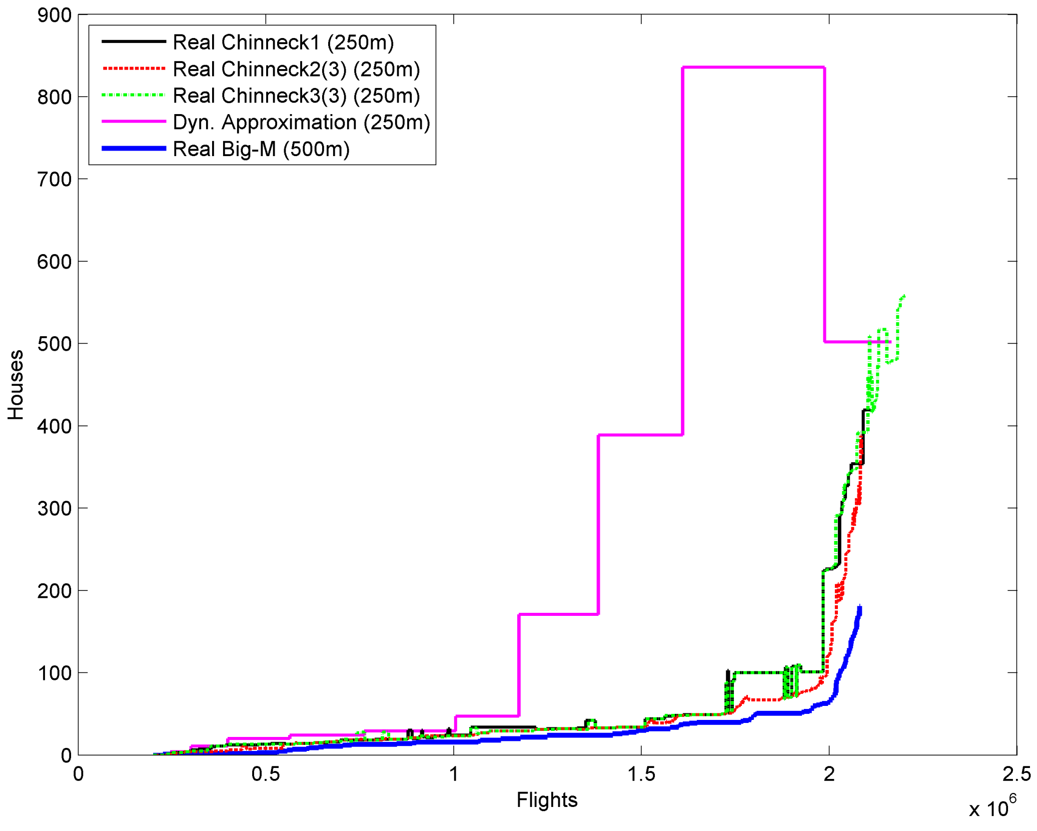


Figure 7.22: Fronts of heuristics solutions with 250 meter grid and the front of solutions of the Big-M method on the 500 meter grid

We look at the solutions obtained by our heuristics for Andijk airport with a 250 meter grid and with a 500 meter grid to approximate the the solutions for Andijk airport with a 500 meter grid found by the Big-M method. Table 7.9 summarizes the results. Note that we consider our solutions without Pareto filter.

We see in Table 7.9 that for the three Chinneck algorithms this approach does not work to improve the approximation ration. For the dynamic approximation algorithm, on the other hand, we see quite an improvement in the approximation ratio for the number of houses getting an excess amount of noise pollution. Since we already saw that the dynamic approximation algorithm was still fast on calculation with the 250 meter grid this may be an option to improve the performance of the algorithm. Note that the theoretical bounds found for the dynamic approximating algorithm are worse and therefore there are no guarantees that this will work.

In Appendix A.2.1 we also look at Andijk airport with a 250 meter grid but now without measure points as also done with the 500 meter grid. We see that it produces results similar to those with the 500 meter grid, i.e., the second Chinneck has the best approximation ratios and we can schedule an infinite amount of flights by using flight routine 823 and 928. On the other hand we do see a change in that the Big-M method takes the most time to complete its computations (this was not the case on the 500 meter grid without measure points) and that we will give 7

[3]The approximation ratios are for approximating the optimal solution of Andijk airport with 500 meter grid

Approximation ratio flights[3]				
	500 meter grid		250 meter grid	
	mean	max	mean	max
Chinneck 1	1.0751	1.4212	1.3544	1.9587
Chinneck 2(3)	1.0303	1.4212	1.1758	1.8743
Chinneck 3(3)	1.0683	1.4212	1.3398	1.97
Dyn. Approximation	1.2203	1.4737	1.5734	2.0896

Approximation ratio houses[3]				
	500 meter grid		250 meter grid	
	mean	max	mean	max
Chinneck 1	1.7149	4.2	2.2593	6
Chinneck 2(3)	1.3166	2.2176	1.7126	5.5
Chinneck 3(3)	1.6323	4.2	2.2549	6.5
Dyn. Approximation	20.1463	82	7.1658	22.5946

Table 7.9: Performance of algorithms for 250 and 500 meter grid for approximating Andijk airport with a 500 meter grid

houses an excess amount of noise pollution to schedule this infinite amount of flight instead of 6.

7.3 Random locations

For the linear airport and Andijk airport we considered the experimental approximation ratios for our heuristics. We saw that the Chinneck algorithms had approximation ratios for the number of houses receiving an excess amount of noise pollution given the number of flights scheduled, were often below 2 and in the worst case the ratio was 4.2. In order to get even more insight in the performance of our heuristics, we want to test these on more problem instances. We will therefore consider the airport as described in Section 7.2 only now with different housing locations. We will generate these new housing locations randomly with also a random amount of houses per location. We will generate multiple of these instances and look at the approximation ratios of our heuristics for the number of houses receiving an excess amount of noise pollution given the number of flights scheduled.

7.3.1 Description

As mentioned before we will consider the same airport as described in Section 7.2. Thus, we have an airport with 4 runways. Around the airport we have 112 governmentally imposed measure points. There are 2408 flight routines which can be scheduled and for every flight routine the amount of noise pollution per flight per (possible) housing location is given. Again we set the maximum amount of noise pollution a housing location may receive to 35Ke. We will divide the area around the airport into squares by using a 250 meter grid. It is divided into 69 parts on the east-west axis and in 45 parts on the north-south axis. We take our possible housing locations in the center of each square. In this way their are 3105 possible housing locations.
For each instance, we will consider, we will randomly determine which possible locations will contain houses and, if so, how much houses are on that housing locations. A possible housing

location will contain houses with probability p_1 , i.e. $P(\omega_h > 0) = p_1, \forall h \in H$. Thus a housing locations will not have any houses with probability $(1 - p_1)$ and we will omit these locations from our calculations.
If for a certain h we know that $\omega_H > 0$ then ω_h follows a bounded Pareto distribution. The bounded Pareto distribution is the Pareto distribution as defined by Arnold [2] truncated to the interval $[\rho, \tau]$. We choose this distribution because, for Andijk airport, the number of houses per housing location seem to follow a power law probability distribution. It has the following probability density function

$$f_{\text{Par}}(x) = \Pr(X = x) = \frac{a\rho^a x^{-a-1}}{1 - \left(\frac{\rho}{\tau}\right)^a} \tag{7.1}$$

where $\rho \leq x \leq \tau$ and $a > 0$. And is has the following cumulative distribution function

$$F_{\text{Par}}(x) = \Pr(X \leq x) = \frac{1 - \rho^a x^{-a}}{1 - \left(\frac{\rho}{\tau}\right)^a} \tag{7.2}$$

where $\rho \leq x \leq \tau$ and $a > 0$. Since we have $\rho = 1$ we can rewrite (7.1) as

$$f_{\text{Par}}(x) = \Pr(X = x) = \frac{ax^{-a-1}}{1 - \frac{1}{\tau^a}}$$

and we can rewrite (7.2) as

$$F_{\text{Par}}(x) = \Pr(X \leq x) = \frac{1 - x^{-a}}{1 - \frac{1}{\tau^a}}.$$

Since the bounded Pareto distribution is a continuous distribution, we will can obtain non integer values when sampling from this distribution, while we need integer values for our houses. We will therefore round the sampled values to the nearest integer. Thus, given that a housing location has houses associated with it, i.e., $\omega_h > 0$, we will have the following probability mass function

$$\Pr(\omega_h = y) = F_{\text{Par}}\left(y + \tfrac{1}{2}\right) - F_{\text{Par}}\left(y - \tfrac{1}{2}\right) = \frac{\left(y - \frac{1}{2}\right)^{-a} - \left(y + \frac{1}{2}\right)^{-a}}{1 - \frac{1}{\tau^a}}$$

where $y \in \{1, \ldots, \tau\}$. Note that we take $F_{\text{Par}}\left(\frac{1}{2}\right) = 0$ and $F_{\text{Par}}\left(\tau + \frac{1}{2}\right) = 1$ for the left and right boundaries of our support and thus the last equality will not hold for these values. Summarizing for a possible housing location h we have the following probability mass function for ω_h

$$f_{\text{Houses}}(y) = \Pr(\omega_h = y) \begin{cases} 1 - p_1, & \text{if } y = 0 \\ p_1 \cdot \frac{1 - \left(y + \frac{1}{2}\right)^{-a}}{1 - \frac{1}{\tau^a}}, & \text{if } y = 1 \\ p_1 \cdot \frac{\left(y - \frac{1}{2}\right)^{-a} - \left(y + \frac{1}{2}\right)^{-a}}{1 - \frac{1}{\tau^a}}, & \text{if } y = y \in \{2, \ldots, \tau - 1\} \\ p_1 \cdot \left(1 - \frac{1 - \left(y - \frac{1}{2}\right)^{-a}}{1 - \frac{1}{\tau^a}}\right), & \text{if } y = \tau \\ 0, & \text{otherwise} \end{cases} \tag{7.3}$$

Note that if for a possible housing location h we have $\omega_h = 0$ we will remove this location from our computations, i.e. h is not a housing location for that instance.

If we take for example $p_1 = 0.0451$, $\tau = 894$ and $a = 0.3800$ we could generate the following airport layout given in Figure 7.23.

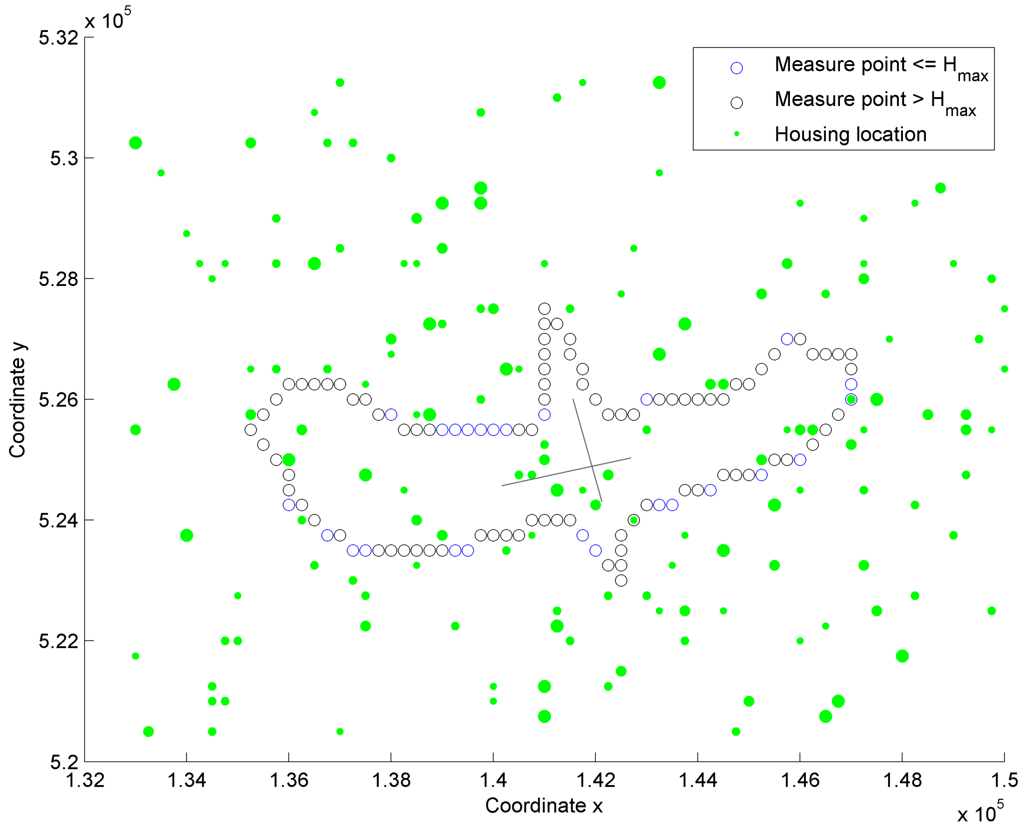


Figure 7.23: Aiport layout for airport with randomly generated housing locations

Note that for Andijk airport we saw housing locations often grouped together and locations where ω_h was big often where near other locations where ω_h was large. This is not the case for our randomly generated housing locations as we can see in this figure.

7.3.2 Results

In total we consider 14 instances of our airport with random locations. When generating our housing locations we take $p_1 = 0.0451$, $\tau = 894$ and $a = 0.3800$. We take $\tau = 894$, because this is the maximum number of houses for a single house locations for Andijk airport on the 500 meter grid. We take $a = 0.3800$, because this is equal to the maximum likelihood estimator of the Pareto distribution if we consider the housing locations of Andijk airport is random data, i.e. $a = \frac{n}{n\ln(\rho)-\sum_{h\in H}\ln(\omega_h)}$. This estimator is close to the maximum likelihood estimator for the bounded Pareto distribution since $\frac{\rho}{\tau}$ is small. Lastly p_1 is equal to one third of the number of housing locations for Andijk airport on the 500 meter grid divided by the total possible housing locations, i.e., $p_1 = \frac{1}{3} \cdot \frac{420}{3105}$. We take p_1 smaller than $\frac{420}{3105}$, because otherwise we would generate solutions, that the Big-M method cannot solve in an acceptable amount of time. The following figure shows the empirical distribution of the houses per housing location for Andijk airport, the probability density function of the fitted bounded Pareto distribution and the probability mass function of the rounded fitted bounded Pareto distribution is given in Figure 7.24.

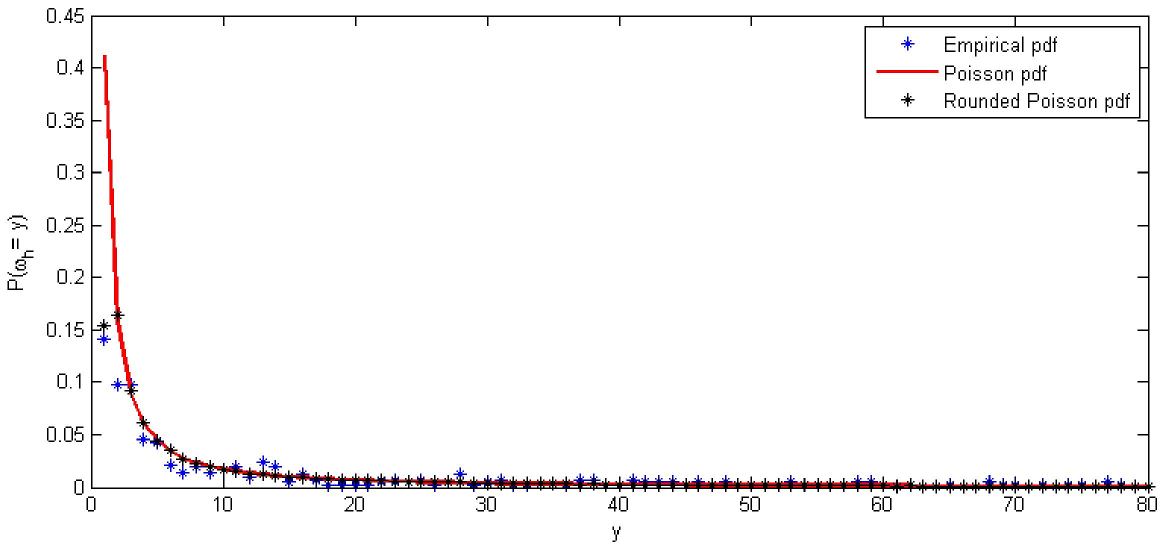


Figure 7.24: Empirical distribution and fitted distributions for the number of houses per housing location

Note that the figure only runs up until $y = 80$ while the actual distribution functions actually run up until $y = 834$.
Since we are primarily interested in the approximation ratio of the number of houses getting an excess amount of noise pollution given the number of flights scheduled, we will generate solution for our algorithm by using a method different to the method we used for Andijk airport. We will first use the normalized normal constraint method on $\Pi_{\text{Big-}M}$. By using our Big-M method in combination with our preprocessor we can obtain 11 weakly Pareto optimal points by setting $\alpha = \{0, \frac{1}{10}, \ldots, \frac{9}{10}, 1\}$. In this way we obtain 11 values for the number of flights $\kappa_1, \ldots, \kappa_{11}$. We continue by using bounded objective value method to obtain Π_{MFS}. We then solve this problem using our three Chinneck heuristics and the dynamic approximation algorithm for $\kappa = \kappa_1, \ldots, \kappa_{11}$. This way we obtain 11 solutions which schedule at least $\kappa_1, \ldots, \kappa_{11}$ respectively. Using the solutions obtained by our heuristics and those obtained by the Big-M method we can compute the experimental approximation ratios for our heuristics. Table 7.10 summarizes these.

Approximation ratio houses			
	mean	mean of max per instance	max over all instances
Chinneck 1	1.5091	3.4692	4.3636
Chinneck 2(3)	1.0872	1.768	3.2338
Chinneck 3(3)	1.4902	3.4692	4.3636
Dyn. Approximation	10.3849	49.3651	103.5833
Dyn. Approx. (Theoretical)	$1.0132 \cdot 10^5$	$3.4372 \cdot 10^5$	$6.8205 \cdot 10^5$

Computation time (sec)			
	mean	mean of max per instance	max over all instances
Chinneck 1	22.6881	52.2413	65.9736
Chinneck 2(3)	7.2277	14.0074	17.1042
Chinneck 3(3)	10.1134	23.9163	29.6909
Dyn. Approximation	0.43491	0.49303	0.85011
Big-M	272.1062	2960.7794	8079.8936

Table 7.10: Performance of algorithms for random housing locations

The second Chinneck algorithm again performs quite will. Most of the time it is not more than a factor 2 away from the optimum and in the worst case it has an experimental approximation ratio of 3.2338. After looking into this worst case performance we saw that only for 3 of the 154 solutions found the approximation ratio of the second Chinneck algorithm is above 2. The first and third Chinneck algorithm perform less but are still within a factor of 3.5 of the optimum for most of the time. Our dynamic approximation algorithm performs the worst. In the worst case it is a factor of 103.5833 away from the optimum, which is still much better than its theoretical approximation bound.

In terms of computation time we get similar results as Andijk airport. The dynamic approximation algorithm is the fastest, then Chinnecks second algorithm, followed by Chinnecks third and Chinnecks first algorithm. The Big-M method is again slow compared to the others. Its average computation time is already longer than the worst case performance of the heuristics.

Chapter 8

Conclusion and further research

We have considered the route scheduling problem, which is a multi-objective optimization problem where given an airport one wants to maximizing the number of flights, while minimize the number of houses suffering more than an acceptable amount of noise pollution. We have modeled this problem as a multi-objective optimization problem. We proved that the route scheduling problem is $\mathcal{NP}$-complete and proved that it is as hard to approximated as *maximum independent set*. Because of these features of the problem, it is unlikely that we can find an efficient algorithm with an approximation ratio smaller than the input size.

We discussed the concept of Pareto optimality and devised methods to rewrite our multi-objective optimization problem to single objective optimization problem. By using these methods we could rewrite our problem to an *integer linear programming* problem and to the *maximum weighted feasible subsystem* problem. Using this single objective formulations of our problem, we devised algorithms to solve it. We saw that we can solve the route scheduling problem optimally using an integer linear programming solver, which may take exponential time to find this (Pareto) optimal solution, and we saw that we can solve it efficiently using one of the three Chinneck heuristics or the dynamic approximation heuristic. We tested these algorithms on three different problem types. We saw that the Chinneck heuristics perform well in practice, especially the second Chinneck algorithm. It stays within a factor 2 of the optimal solution for most of the instances considered.

Further research can be done on the Chinneck heuristics to better incorporate the weights associated with optional constraints. We saw that we could incorporate these in two ways in our elastic program. The first one had the advantage that the elastic program will violates those constraints that have lower weights. While the second one had the advantage that removing a constraint with high valued weight is expensive. Thus if one would alter the algorithm to incorporate both, this could improve the performance of the Chinneck heuristics.

We now only computed exact solutions for our problems using the Big-M method, which we use to solve $\Pi_{\text{Big-}M}$. But there is also an the exact algorithm by Parker and Ryan [25] for the *weighted maximum feasible subsystem* problem. If one would adapt this algorithm to also incorporate binding constraints, one could use this algorithm to solve Π_{MFS}.

In terms of the problems we considered up until this point, we did not take into account any operational requirements. For example, we did not require that we have the same amount of take-offs as landings. Therefore it could be interesting to consider our route scheduling problem only now also with operational constraints incorporated. In this way one could obtain more realistic flights schedules. One could obtain such a solutions that better represent real world operations by introducing more binding constraints.

When considering the problem instance of Andijk airport we considered an instance with the current government regulations, i.e., the zone around the airport, and without the current regulation. Our experiments on the instance implicate that if one would remove the government regulations one would be able to schedule more flights while giving less houses an excess amount noise pollution. Thus one might argue from these results, that by removing the current regulations and imposing them on the number of houses suffering more than the acceptable amount of noise pollution, one could improve the overall quality of life of people living around an airport. More research needs to be done to prove this claim.

Bibliography

[1] Noga Alon, Sanjeev Arora, Rajsekar Manokaran, Dana Moshkovitz, and Omri Weinstein. On the inapproximability of the densest k-subgraph problem. *Manuscript*, 3(4):7, 2011.

[2] Barry C Arnold. *Pareto distribution.* Wiley Online Library, 1985.

[3] Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. Proof verification and intractibility of approximation problems. In *Proceedings of the 33rd IEEE Symposium on Foundations of Computer Science, IEEE Computer Science Press, Los Alamitos, CA*, pages 14–23, 1992.

[4] Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. Proof verification and the hardness of approximation problems. *J. ACM*, 45 (3):501–555, May 1998. ISSN 0004-5411. doi: 10.1145/278298.278306. URL `http://doi.acm.org/10.1145/278298.278306`.

[5] Dimitri Bertsimas and J Tsitsiklis. Introduction to linear programming. *Athena Scientific*, 1:997, 1997.

[6] Aditya Bhaskara, Moses Charikar, Eden Chlamtac, Uriel Feige, and Aravindan Vijayaraghavan. Detecting high log-densities: an $o(n1/4)$ approximation for densest k-subgraph. In *Proceedings of the 42nd ACM symposium on Theory of computing*, pages 201–210. ACM, 2010.

[7] John W Chinneck. Fast heuristics for the maximum feasible subsystem problem. *INFORMS Journal on Computing*, 13(3):210–223, 2001.

[8] Guy de Ghellinck and Jean-Philippe Vial. A polynomial newton method for linear programming. *Algorithmica*, 1(1-4):425–453, 1986.

[9] dr. F.W.J. van Deventer. *Basiskennis geluidshinder luchtvaart.* 2004.

[10] EUROCONTROL. *Seven years flights service units forecast 2013-2019.* 2013.

[11] Uriel Feige. Relations between average case complexity and approximation complexity. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 534–543. ACM, 2002.

[12] Uriel Feige, Shafi Goldwasser, Laszlo Lovász, Shmuel Safra, and Mario Szegedy. Interactive proofs and the hardness of approximating cliques. *Journal of the ACM (JACM)*, 43(2): 268–292, 1996.

[13] MR Garey and DS Johnson. Computers and intractability-a guide to the theory of np-completeness, 1979.

[14] Johan Håstad. Clique is hard to approximate within $n^{1-\epsilon}$. 1997.

[15] Dorit S Hochbaum and Anu Pathria. Analysis of the greedy approach in problems of maximum k-coverage. *Naval Research Logistics (NRL)*, 45(6):615–627, 1998.

[16] TL Jonker. Multiobjective optimization of aircraft noise and operational flexibility around airports. Master's thesis, University of Amsterdam.

[17] Richard M Karp. *Reducibility among combinatorial problems.* Springer, 1972.

[18] Irvin J Lustig. Feasibility issues in a primal-dual interior-point method for linear programming. *Mathematical Programming*, 49(1-3):145–162, 1990.

[19] R Timothy Marler and Jasbir S Arora. Survey of multi-objective optimization methods for engineering. *Structural and multidisciplinary optimization*, 26(6):369–395, 2004.

[20] Achille Messac, Amir Ismail-Yahaya, and Christopher A Mattson. The normalized normal constraint method for generating the pareto frontier. *Structural and multidisciplinary optimization*, 25(2):86–98, 2003.

[21] Kaisa Miettinen. *Nonlinear multiobjective optimization*, volume 12. Springer, 1999.

[22] Christos H Papadimitriou and Kenneth Steiglitz. *Combinatorial optimization: algorithms and complexity.* Courier Dover Publications, 1998.

[23] Christos H Papadimitriou and Mihalis Yannakakis. Optimization, approximation, and complexity classes. *Journal of computer and system sciences*, 43(3):425–440, 1991.

[24] Vilfredo Pareto. *Manuale di economia politica*, volume 13. Societa Editrice, 1906.

[25] Mark Parker and Jennifer Ryan. Finding the minimum weight iis cover of an infeasible system of linear inequalities. *Annals of Mathematics and Artificial Intelligence*, 17(1):107–126, 1996.

[26] Vijay V Vazirani. *Approximation algorithms.* springer, 2001.

[27] David Zuckerman. Linear degree extractors and the inapproximability of max clique and chromatic number. *Theory OF Computing*, 3:103–128, 2007.

Appendix A

Additional results

A.1 Model of predecessor

Before the *route scheduling* problem as we discuss in this thesis was considered, T. L. Jonker in her thesis looked at a different model for noise minimization around airports. In this model Π_{pre} we want to maximize the number of flights, while also maximizing the gap between the used amount and the amount allowed of noise pollution on the measure points imposed by the government. We have the following given

Flight routines	$\{r_1, \ldots, r_m\} = R$
Measure points	$\{q_1, \ldots, q_l\} = Q$

With this we have the following constants

$\alpha_{qr} \in \mathbb{R}_+$	$\forall q \in Q, \forall r \in R$	Representing the noise pollution of flight routine r on measure point q
$\theta \in \mathbb{R}_+$		Constant for the maximum noise allowed by the government (mostly 35 Ke)

Now we introduce the variables $\{x_{r_1}, \ldots, x_{r_m}\} = X$, with $x_r \in \mathbb{Z}_+, \forall r \in R$, which represent the amount of times routine r is used. Furthermore we introduce $g \in \mathbb{R}_+$, which represents the gap between the used amount and the amount allowed by the government. We have the following formulation Π_{pre}

$$\begin{array}{lllll} \max & \sum_{r \in R} x_r & & \textit{and} & \max \quad g \\ \\ \text{s.t.} & \sum_{x \in X} \alpha_{qr} x_r \leq \theta - g & \forall q \in Q & & \\ & x \in \mathbb{Z}_+, g \in \mathbb{R}_+ & & & \end{array}$$

If we use one of the method as described in Chapter 5 one could rewrite Π_{pre} as an *mixed integer linear program* and solve it using an MILP solver. Afterwards one can analyze the solutions found and impose more constraints to find a more realistic flight schedule. This is in short what is done in the thesis by Jonker [16].

A.2 Relaxing from integer to real valued variables

Suppose we want to turn our maximum feasible ILP problem into a maximum feasible LP problem. Then we want to relax $x_r \in \mathbb{Z}+$ to $x_r \in \mathbb{R}+$. We could after computing all x_r round every value down, which gives us a solution satisfying the same amount of constraints and having $\Sigma_{r \in R} x_r \geq \kappa - m$. Thus we have two options. We could solve the LP with $\kappa' = \kappa$ and then round down. This would give us $\Sigma_{r \in R} x_r \geq \kappa - m$. Which can be zero in the worst case when $m > \kappa$. Or we could solve for $\kappa' = \kappa + \sigma$ where $0 < \sigma \leq m$ and then round down. This will result in $\Sigma_{r \in R} x_r \geq \kappa - m + \sigma$. For large σ one can easily see that this may give very bad estimate for the number of constraints satisfied. Let $\sigma = \epsilon$, where $0 < \epsilon << 1$. Then consider the following example with $R = \{r_1, r_2\}$.

$$\begin{array}{cccccl} 42x_{r_1} & + & \frac{1}{\kappa+\epsilon}x_{r_2} & \leq & 1 & \text{for } h_1 \\ \frac{1}{\kappa}x_{r_1} & + & 42x_{r_2} & \leq & 1 & \text{for } h_2, \ldots, h_n \end{array} \tag{A.1}$$

In this example one could only satisfy 1 constraint when $\sum_{r \in R} x_r \geq \kappa + \sigma$, but one could satisfy $n-1$ when $\sum_{r \in R} x_r \geq \kappa$. Thus for every choice of rounding, one will encounter examples, where the LP gives a bad approximation of the ILP.

A.2.1 Andijk airport without measure points on a fine grid

We already saw that for Andijk airport with a the 500 meter grid, that if we would remove the measure point we could schedule an infinite amount of flights, while only having 6 houses suffering more then their threshold of noise pollution. We could do the same analysis for Andijk airport on the 250 meter grid. We use the bounded objective function method to transform the multi-objective optimization problem in a single objective optimization problem. We again us the Big-M method to solve it exactly and use the first Chinneck algorithm, the second Chinneck algorithm with $k = 3$, the third Chinneck algorithm with $k = 3$ and the dynamic approximation algorithm to gain approximate solutions. For all five algorithms we use the iterative scheme as described in Section 5.3 to obtain sets of solutions with an increasing number of flights. After obtaining these sets we interpolated between these different solutions to construct fronts and we get Figure A.1.

In figure A.1 we see that the Big-M method again constructs a (weak) Pareto front. The second Chinneck algorithm performs again the best of the four heuristics. One remarkable feature of this instance is that the dynamic approximation algorithm for certain number of flights performs better than the first and third Chinneck heuristic.
Just as with the 500 meter grid we can schedule an infinite amount of flights. In order to schedule this infinite amount of flights the Big-M, the first Chinneck algorithm and the second Chinneck algorithm use again flight routine 823 and 928 by and the three Chinneck heuristics. The dynamic approximation algorithm uses again flight routine 157, which is now also used by the third Chinneck algorithm (for the 500 meter grid it used flight routine 823 and 928).

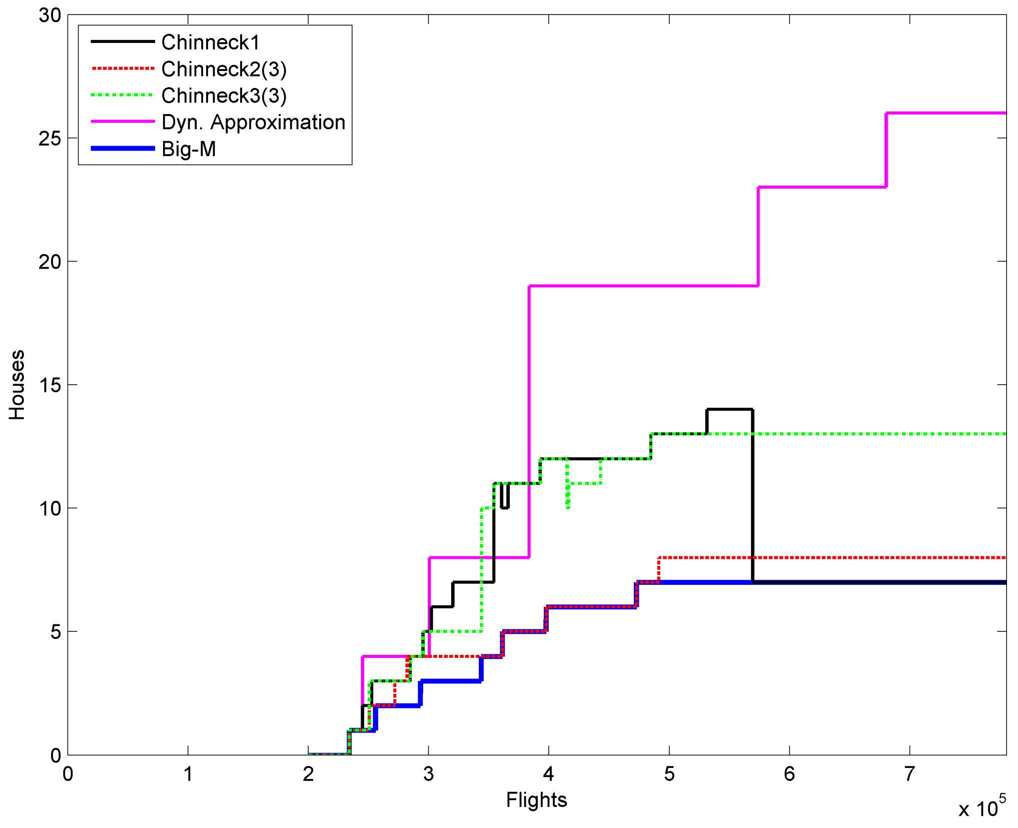


Figure A.1: Fronts of maximum flight against minimum houses for Andijk airport with 250 meter grid and no measure points

The Pareto point found by the Big-M method (and first two Chinneck heuristics) which schedules the infinite amount of flights is shown in Figure A.2.

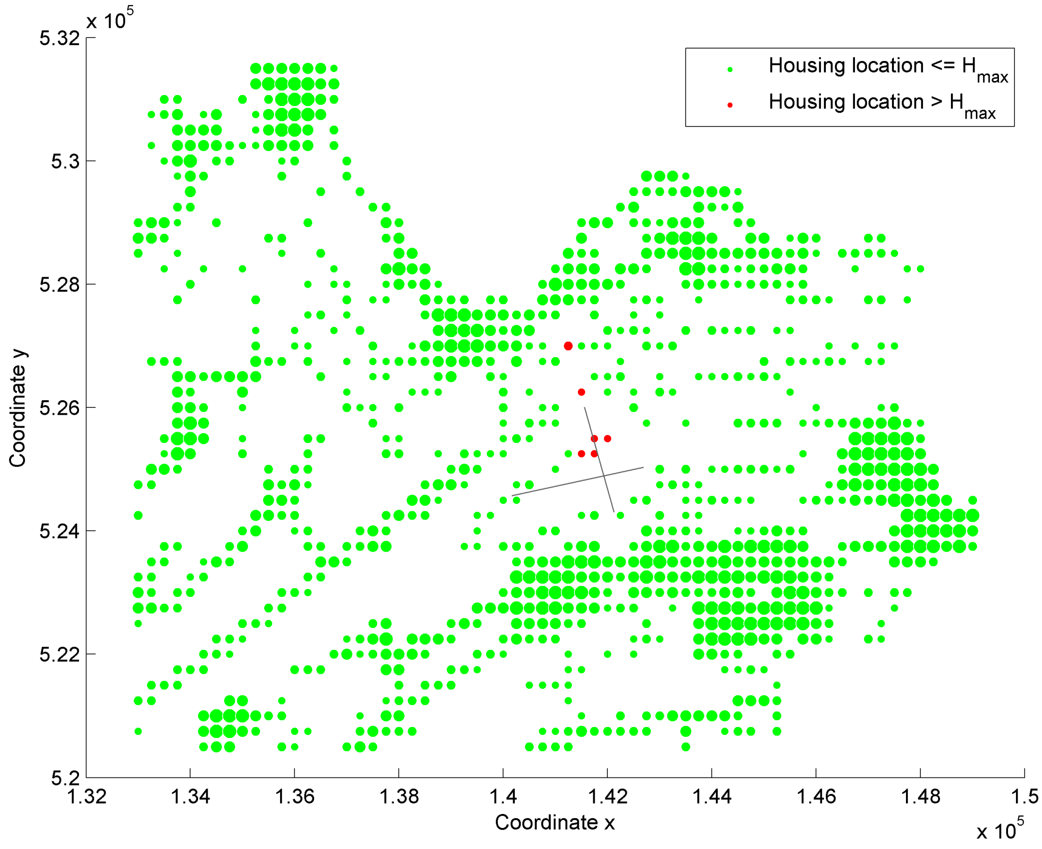


Figure A.2: Pareto point Andijk airport with infinite amount of flights scheduled

While scheduling the infinite amount of flights we now give 7 houses more than thir threshold amount of noise pollution. This is one more than for Andijk airport on a 500 meter grid. Table A.1 summarized the results for the five algorithms for Andijk airport without measure points on a 500 meter and a 250 meter grid.

Approximation ratio flights[4]				
	500 meter grid		250 meter grid	
	mean	max	mean	max
Chinneck 1	1.1007	1.4653	1.19	1.4768
Chinneck 2(3)	1.0864	1.4653	1.0526	1.2181
Chinneck 3(3)	1.0922	1.4653	1.1086	1.2237
Dyn. Approximation	NA	NA	1.0822	1.2026

Approximation ratio houses				
	500 meter grid		250 meter grid	
	mean	max	mean	max
Chinneck 1	1.2167	2	1.9123	3
Chinneck 2(3)	1.12	2	1.2403	2
Chinneck 3(3)	1.1533	2	1.9504	3
Dyn. Approxation	3.3	2	2.781	4

Computation time (sec)				
	500 meter grid		250 meter grid	
	mean	max	mean	max
Chinneck 1	10.8933	21.4314	45.7576	107.295
Chinneck 2(3)	2.4724	3.8327	6.0849	12.3344
Chinneck 3(3)	3.8496	6.3904	13.8179	25.4905
Dyn. Approximation	0.51497	0.54709	1.1043	1.1849
Big-M	1.1083	2.2462	122.712	761.609

Table A.1: Performance of algorithms for Andijk airport without measure points for 250 and 500 meter grid size

First note the the approximation ratio for the dynamic approximation algorithm does not say much. Only the first few solutions can be used to obtain the bound, since for $3 \cdot 10^5$ it already has 7 houses receiving an excess amount of noise pollution. Thus again we see that the second Chinneck algorithm has the best approximation ration for the number of flights given the number of houses receiving more than their threshold amount of noise pollution. For Chinnecks second and thrid algorithm, the ration is better than for the 500 meter grid case, while for the first Chinneck algorithm it is worse.
The approximation ratio for the houses receiving an excess amount of noise pollution given the number of flights, is in general worse than for Andijk airport on a 500 meter grid. Although the values for the second Chinneck heuristic are still quite close to those for the 500 meter grid and thus still quite good. In terms of computation time we see an increase of time needed of a factor between 3 and 6 for our Chinneck algorithms. For the dynamic approximation algorithm it is about doubled, while for the Big-M method we see a huge increase. Its valued for the 250 meter grid is more in line with what we saw for other instances (other than Andijk airport without measure points on a 500 meter grid).
We can again also use our Pareto filter to try to improve on the approximation ratios and obtain better sets of solution for our heuristics. If we do so we get the approximation ratios shown in Table A.2.

[4]We did not consider the last solutions found per algorithm for this approximation ratio (the solutions having an infinite amount of flights), since this would make this ratio arbitrary bad for three of our heuristics.

Approximation ratio flights 3				
	500 meter grid		250 meter grid	
	mean	max	mean	max
Chinneck 1	1.0892	1.4653	1.19	1.4768
Chinneck 2(3)	1.0765	1.4653	1.0526	1.2181
Chinneck 3(3)	1.0823	1.4653	1.1086	1.2237
Dyn. Approximation	NA	NA	1.0822	1.2026

Approximation ratio houses				
	500 meter grid		250 meter grid	
	mean	max	mean	max
Chinneck 1	1.1762	1.5	1.5917	3
Chinneck 2(3)	1.0875	1.5	1.25	2
Chinneck 3(3)	1.1292	1.5	1.8973	3
Dyn. Approxation	3	7	2.4933	4

Table A.2: Performance of algorithms for Andijk airport without measure points for 250 and 500 meter grid size with Pareto filter

We see that the filter has no influence on the approximation ratio for the number of flights, but it does improve the mean approximation ratio for the number of houses getting an excess amount of noise pollution.
In Appendix C their is an animation available which shows the solutions found by the Big-M method for Andijk airport without measure point on a 250 meter grid.

Appendix B

Additional plots

B.1 Addition plots linear airport

For the linear airport with integer values we did not plot the computation time because for all our algorithms these we less then a second. Figure B.1 gives the full picture.

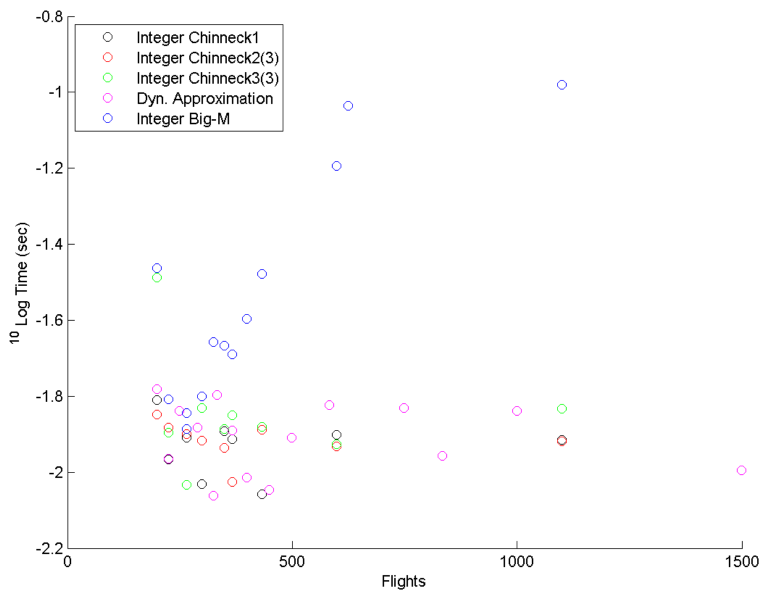


Figure B.1: Solutions - time needed for computation

For the linear airport with real values we saw that the Chinneck algorithms sometimes compute solutions which are clearly suboptimal. These solutions can be removed by using the postprocessor in Section 6.4. We would then obtain the following solution sets.

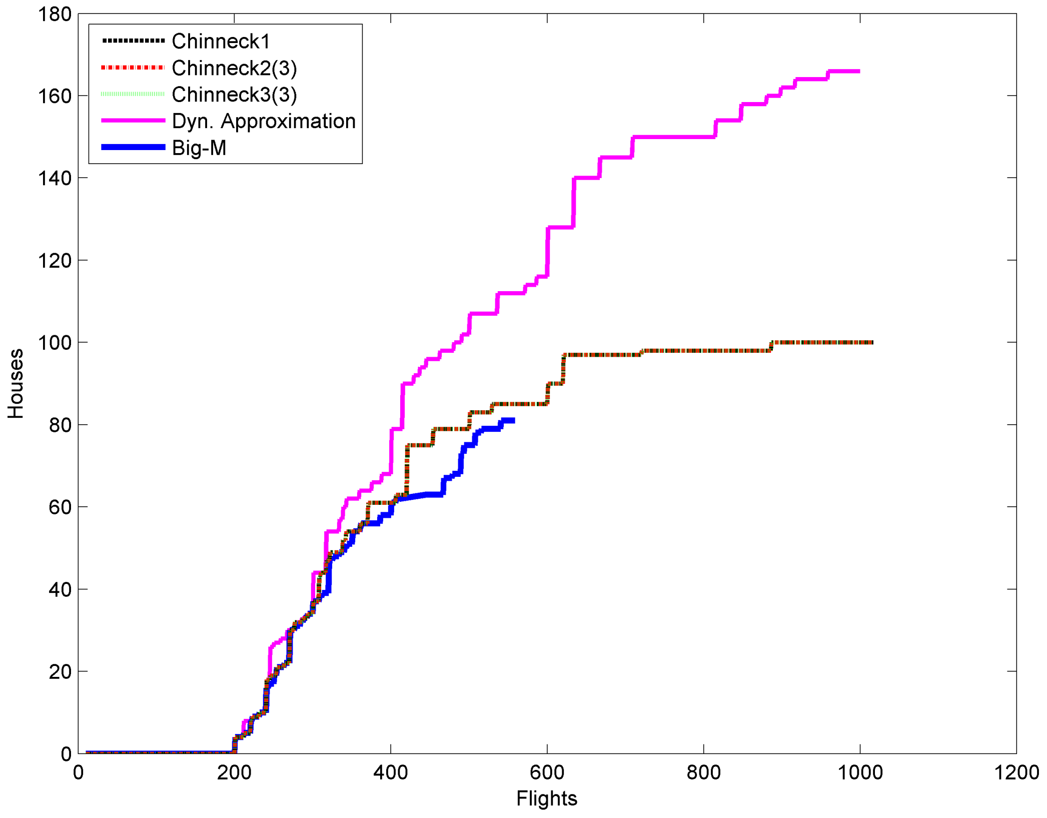


Figure B.2: Fronts of solutions linear airport with postprocessor

B.2 Addition plots Andijk airport

To give further insight in the values obtained for Andijk airport we have listed the number of houses per location in the following figure.

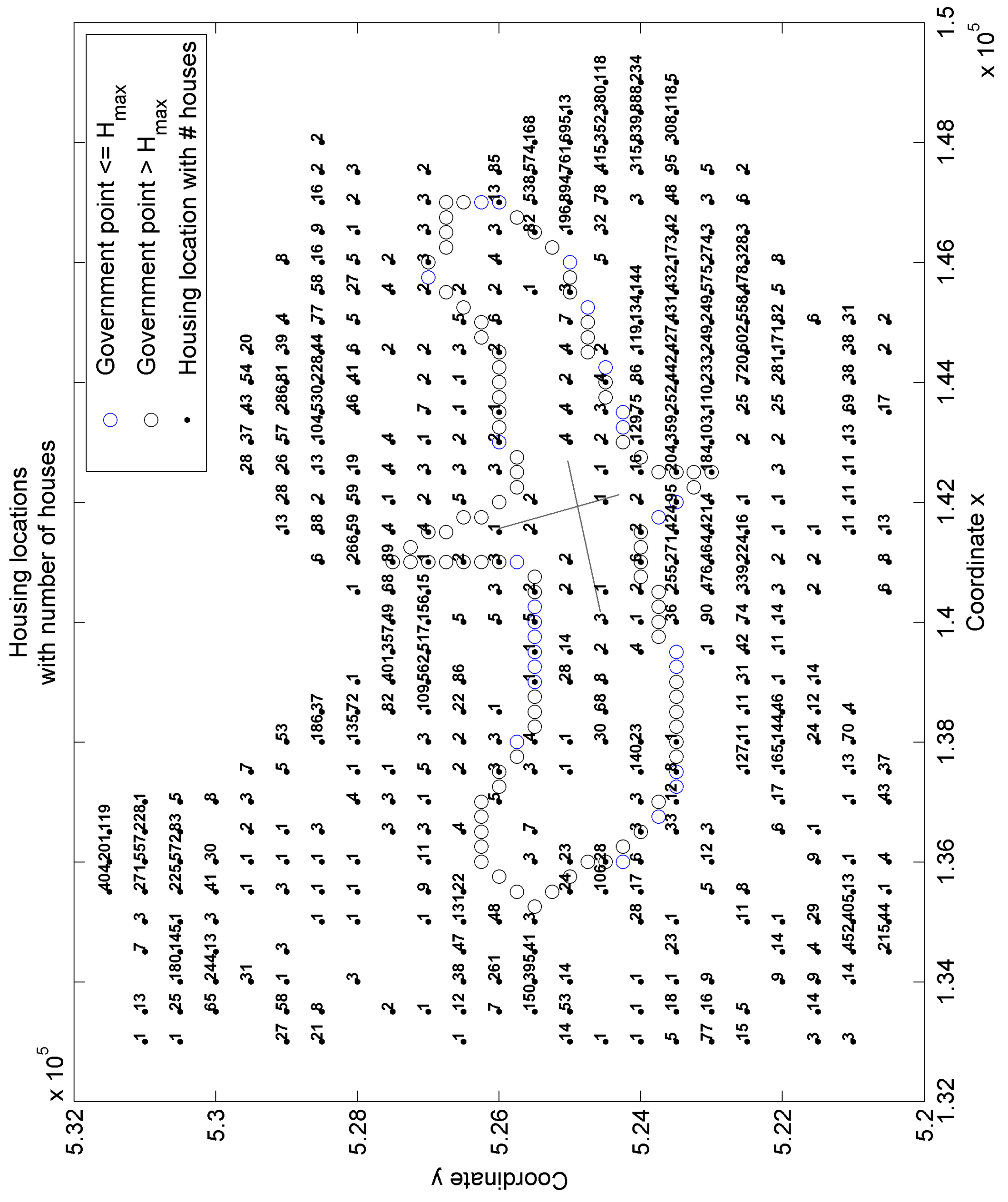


Figure B.3: Housing location with number of houses per location

We already saw the average and worst case performance of our Chinneck heuristics. To give even more insight in their performance the following figures show their performance per found solution.

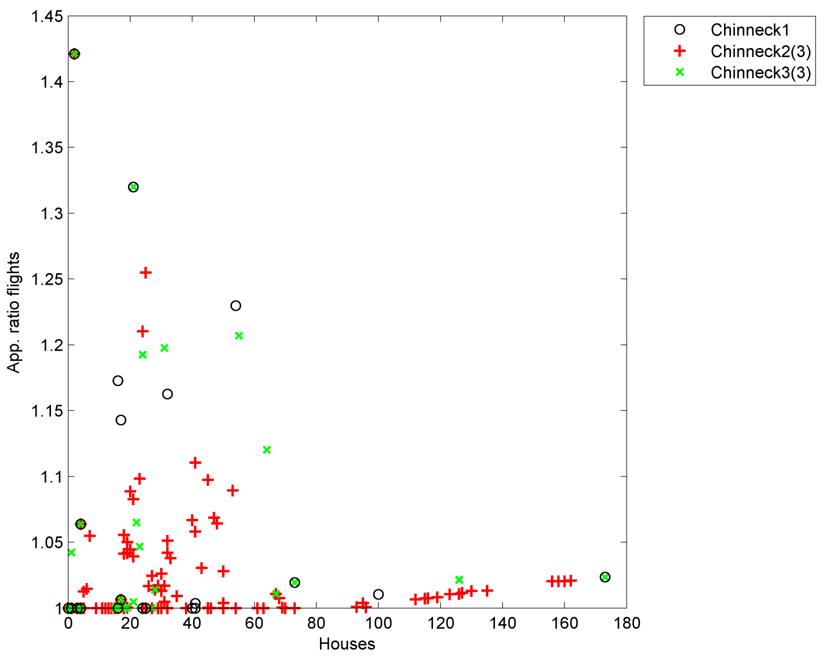


Figure B.4: Approximation ratio of the number of flights scheduled for the three Chinneck algorithms

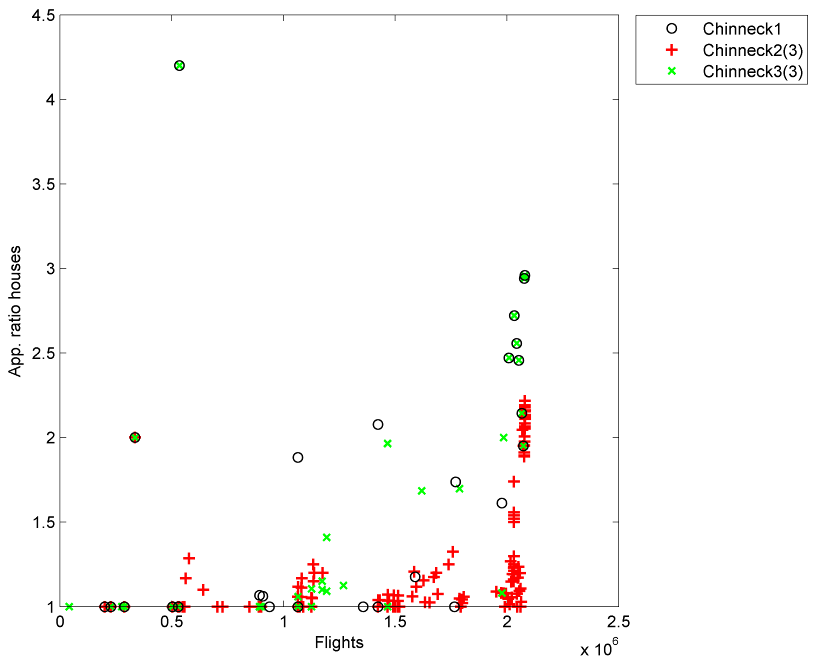


Figure B.5: Approximation ratio of the number of houses with more then the threshold amount of noise pollution for the three Chinneck algorithms

B.2.1 Without measure points

Figure B.6 shows the solutions found for Andijk airport after we have use a Pareto filter.

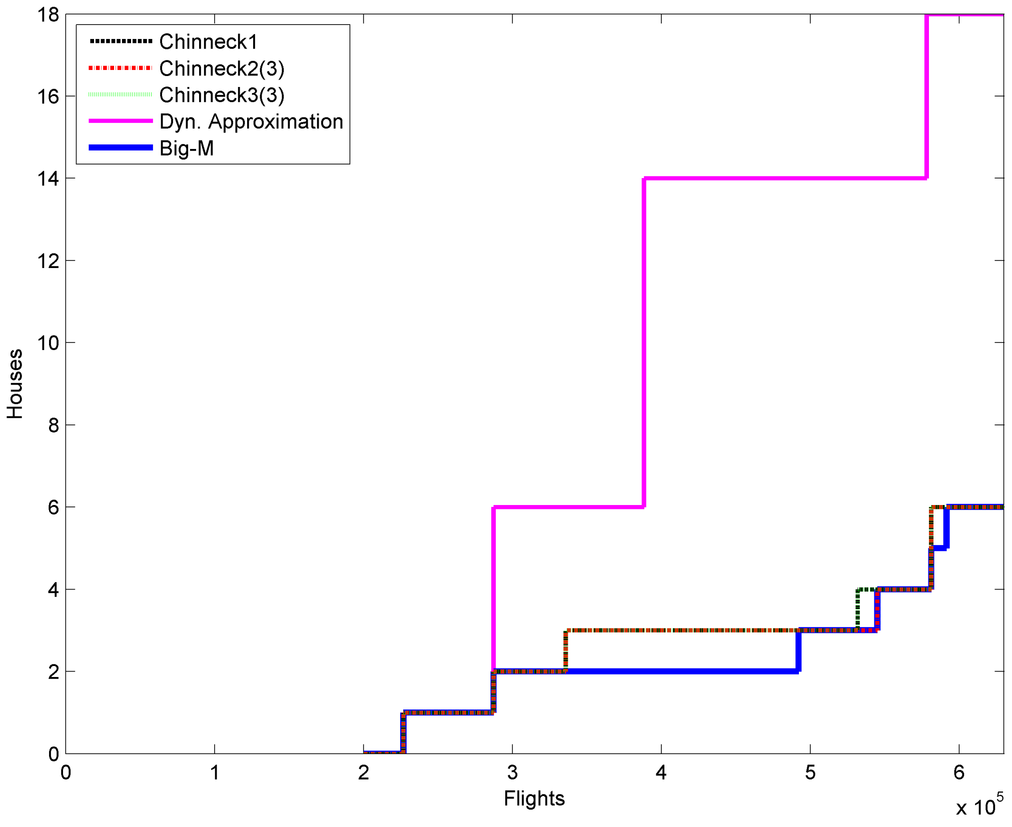


Figure B.6: Fronts of maximum flight against minimum houses with Pareto filter

We can also look at which flight routines are scheduled per weakly Pareto point found by the Big-M method. Figure B.7 shows the relative amount of flights scheduled per flight routine per weakly Pareto point. Note that we did not plot the last point that schedules an infinite amount of flights, because it would greatly increase the range of the plot. This last point schedules flight 823 and 928 an infinite amount of times.

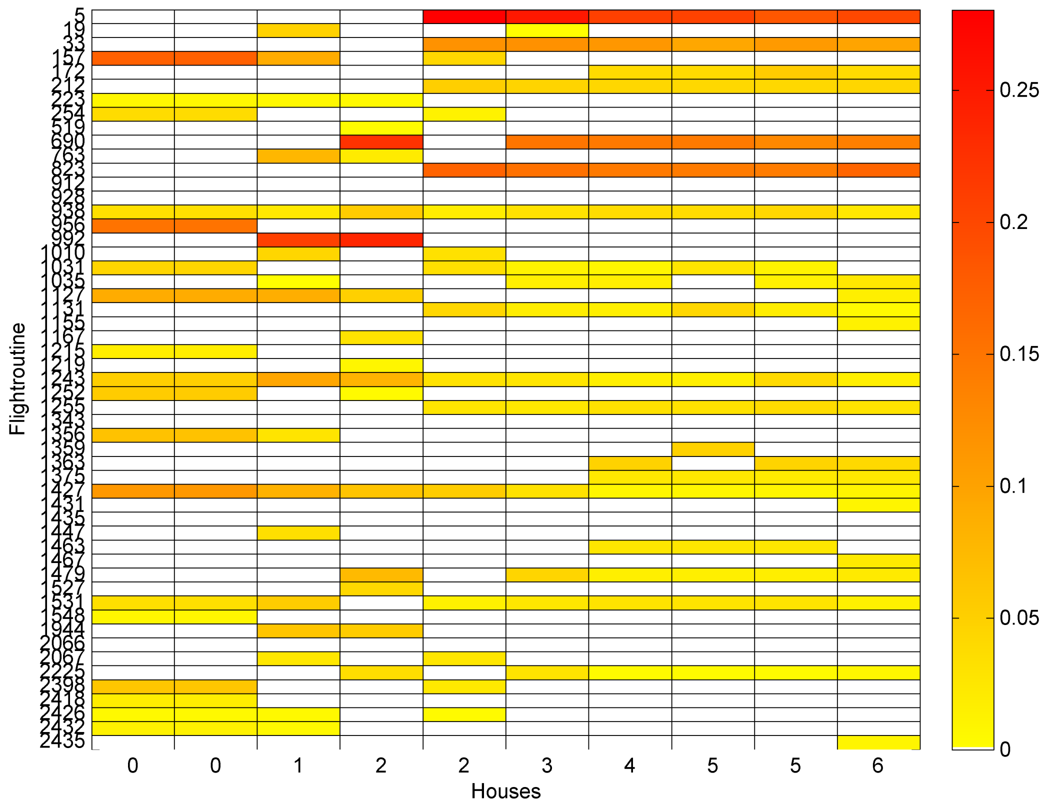


Figure B.7: Relative of amount of flights per flight routine per solution of Andijk airport without measure points

B.2.2 Fine grid

We also consider Andijk airport on a grid of 250 meters instead of 500 meters. This will give different solutions since housing locations will have different weights and there will be more options available. This will increase the computation time of our algorithms. Figure B.8 shows the computation time per solution found.

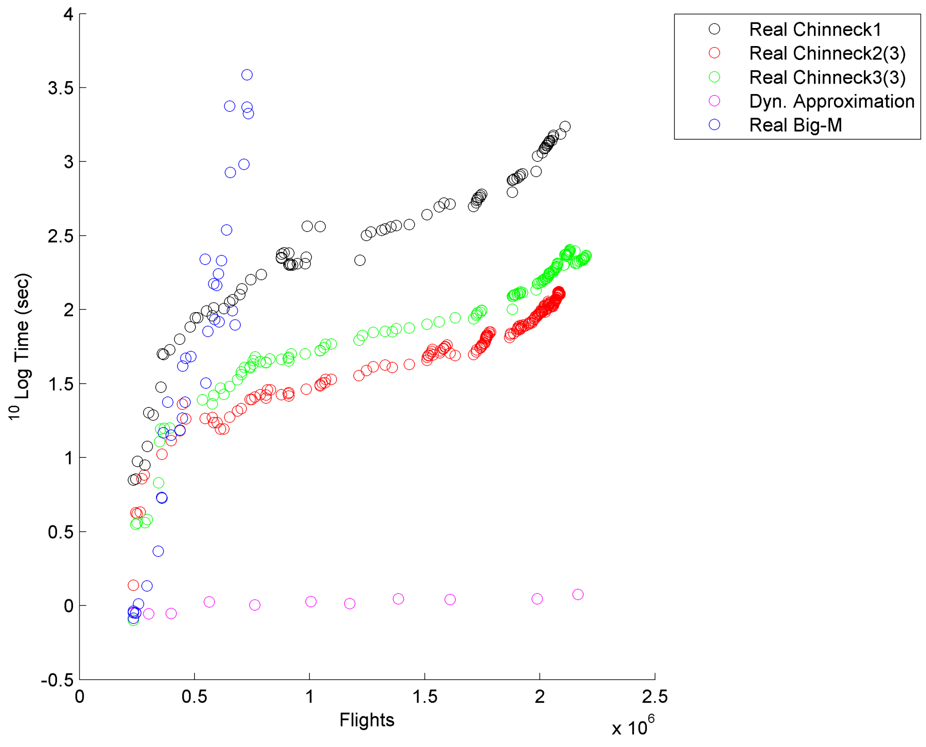


Figure B.8: Solutions - time needed for computation for Andijk airport with a 250 meter grid

Also for this problem instance we saw that the heuristics generate some obvious non optimal solutions. Thus we can use a Pareto filter to remove these. Figure B.9 we interpolated between the solutions in these filtered sets to obtain fronts for all our algorithms.

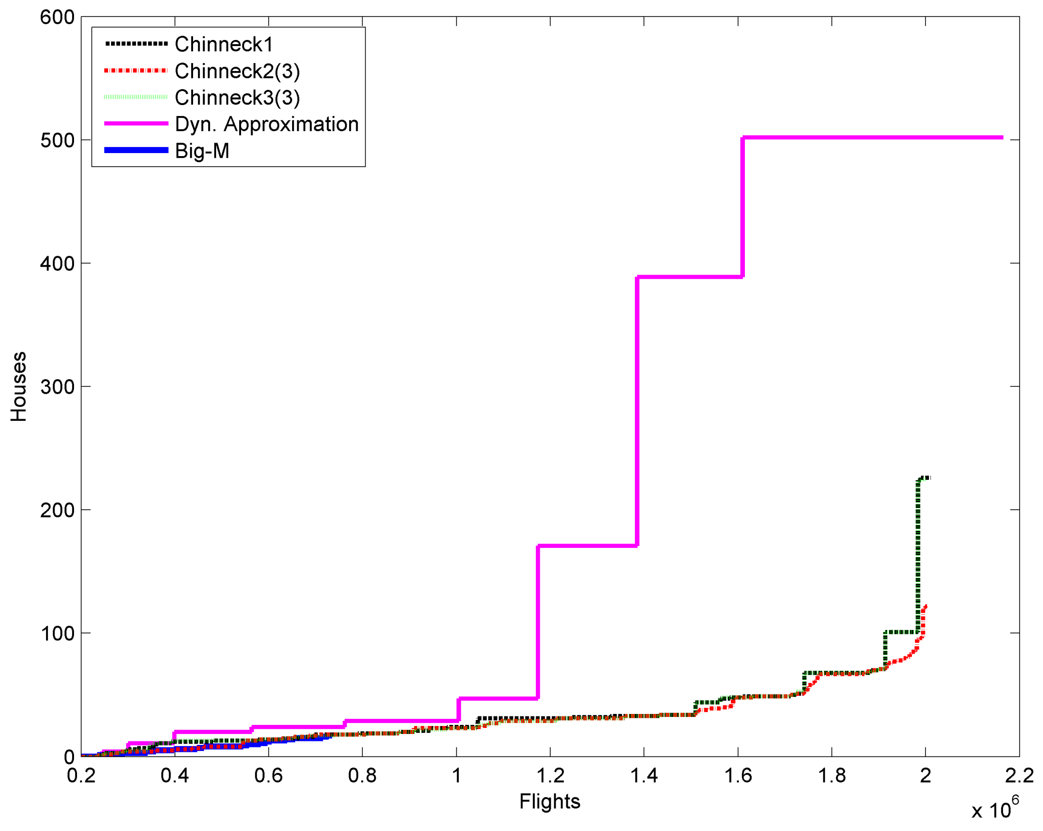


Figure B.9: Fronts of maximum flight against minimum houses with Pareto filter for Andijk airport with a 250 meter grid

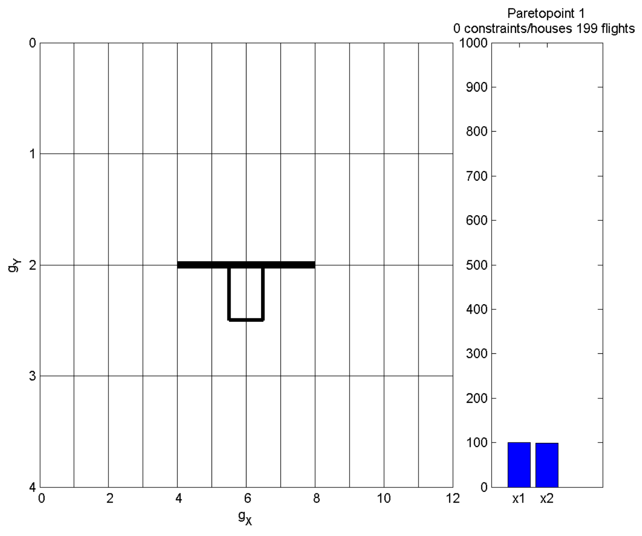
Appendix C

Animations

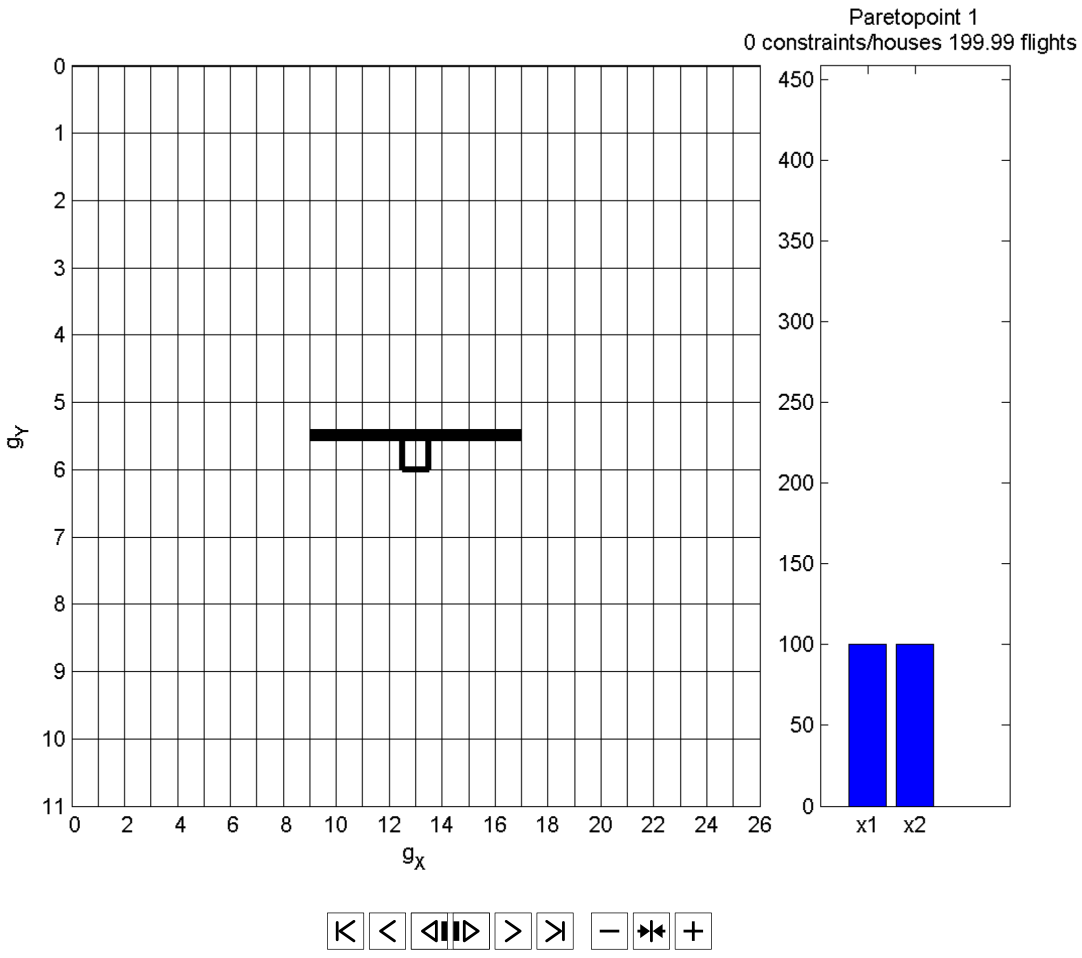
This appendix contains animations of the different solutions obtained for the problem instances we considered. These can only be viewed in its digital format.

C.0.3 Linear airport

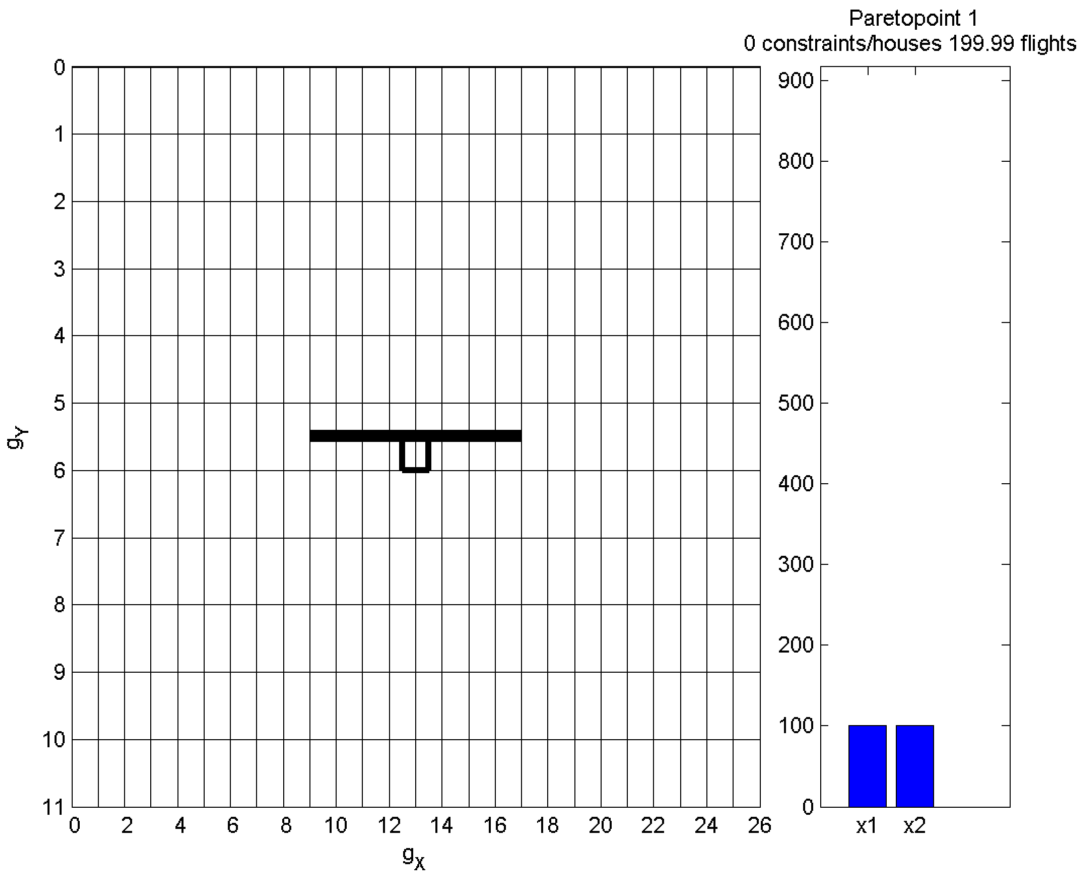
The following animation shows the solutions found by the Big-M method. These are Pareto optimal solutions. Note that since the Chinneck heuristics found the same solutions as the Big-M method. An animation of these would be the same.



If we look at our second problem, where we considered real valued flight we get the following animation showing the solutions found by the Big-M method. Note that we also took $X = 26$ and $Y = 11$ instead of 12 and 4 respectively.

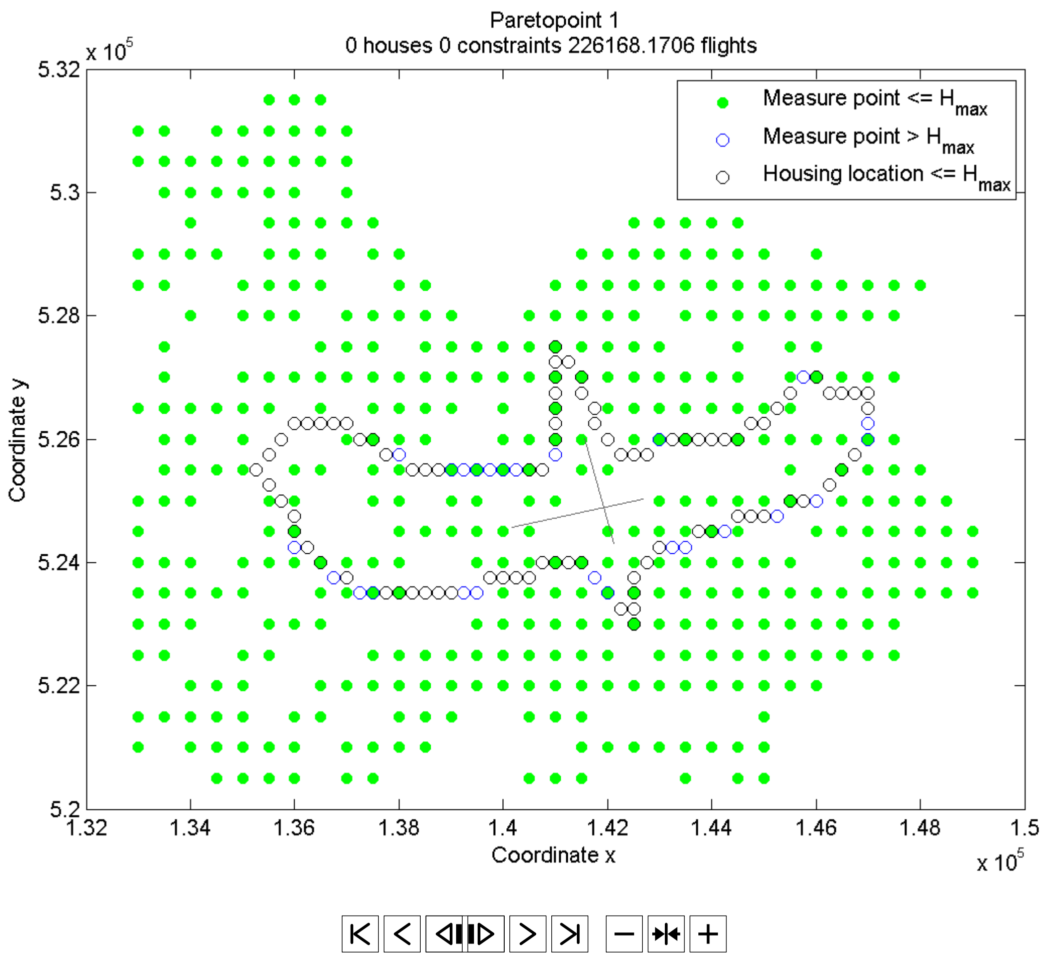


For the Chinneck algorithm we got a different set of solutions then those found by the Big-M method. The following animation shows those.

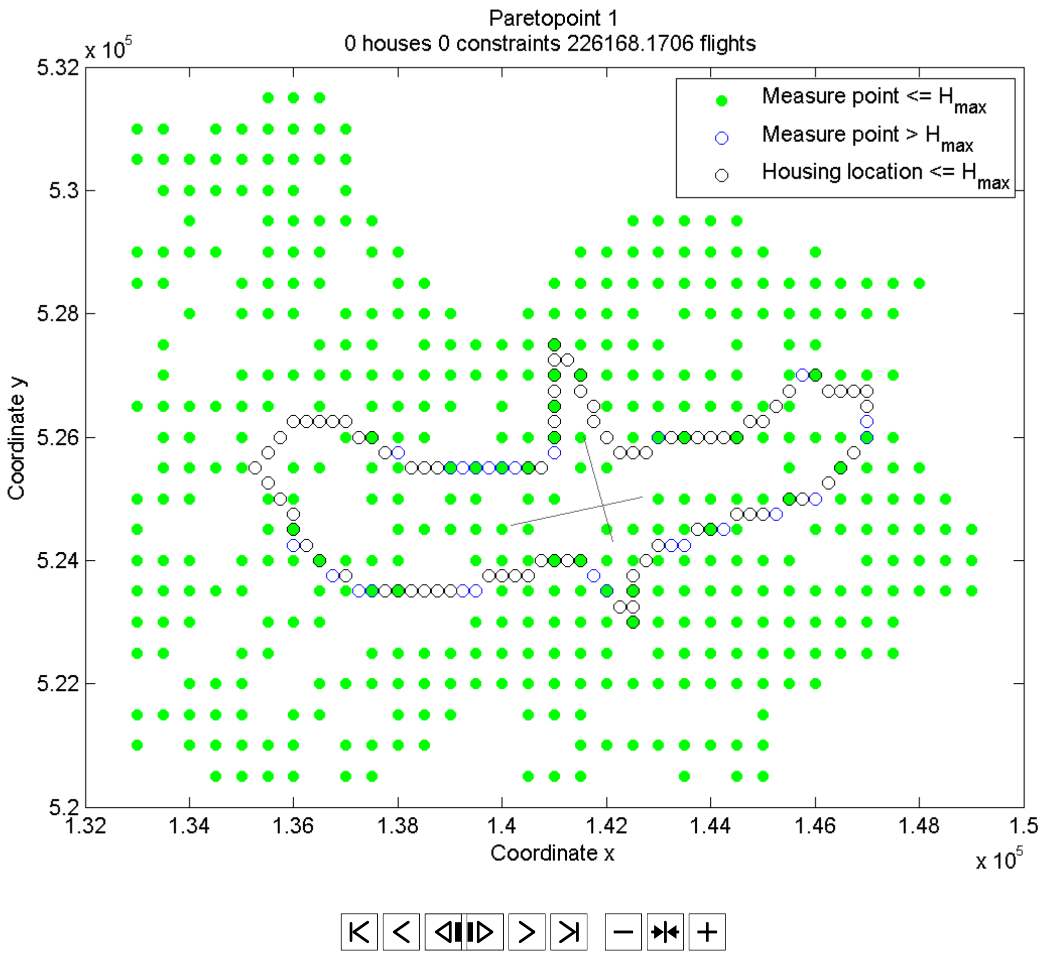


C.0.4 Andijk airport

Visualizing the solutions of the Big-M method for every Pareto point, we would obtain the following animation

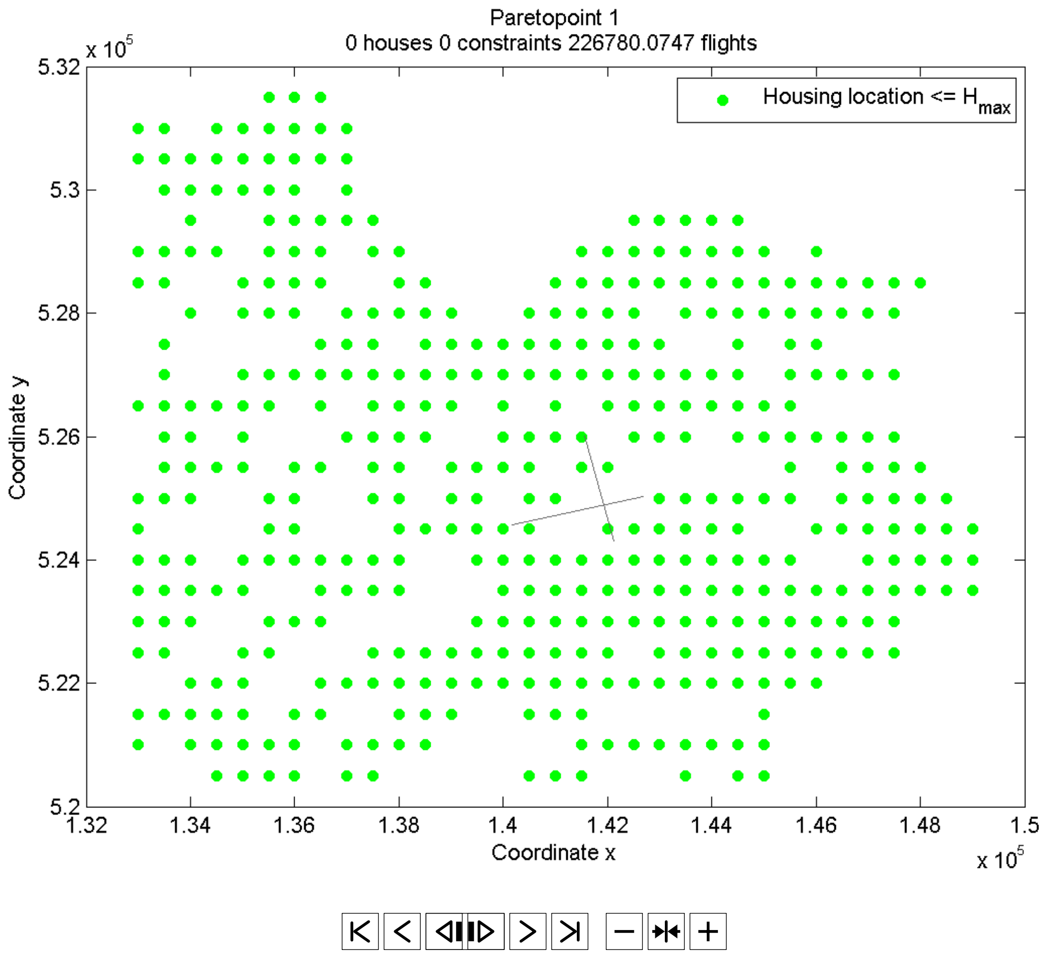


For the Chinneck algorithm we got a different set of solutions then those found by the Big-M method. The following animation visualizes those.



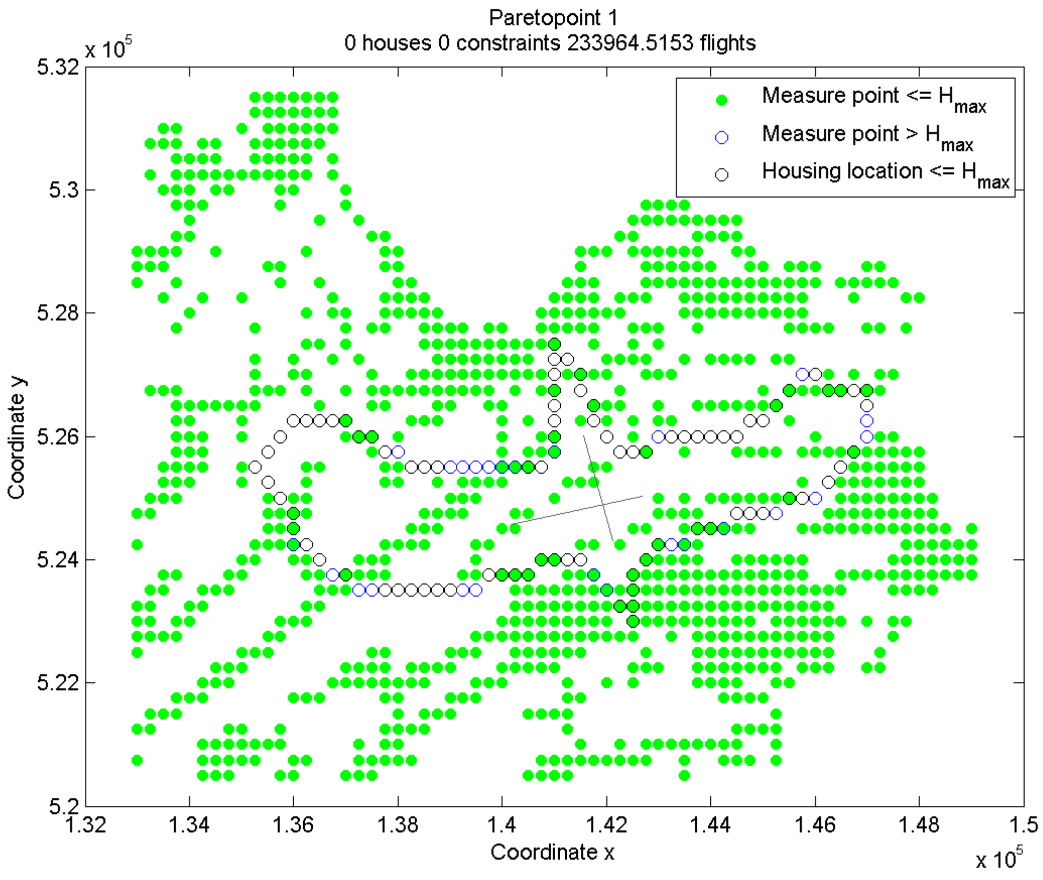
Without measure points

We also look at Andijk airport without the measure points. If we do this we can schedule an infinite amount of flights. The following animation shows the solutions found by the Big-M method for this instance of Andijk airport.

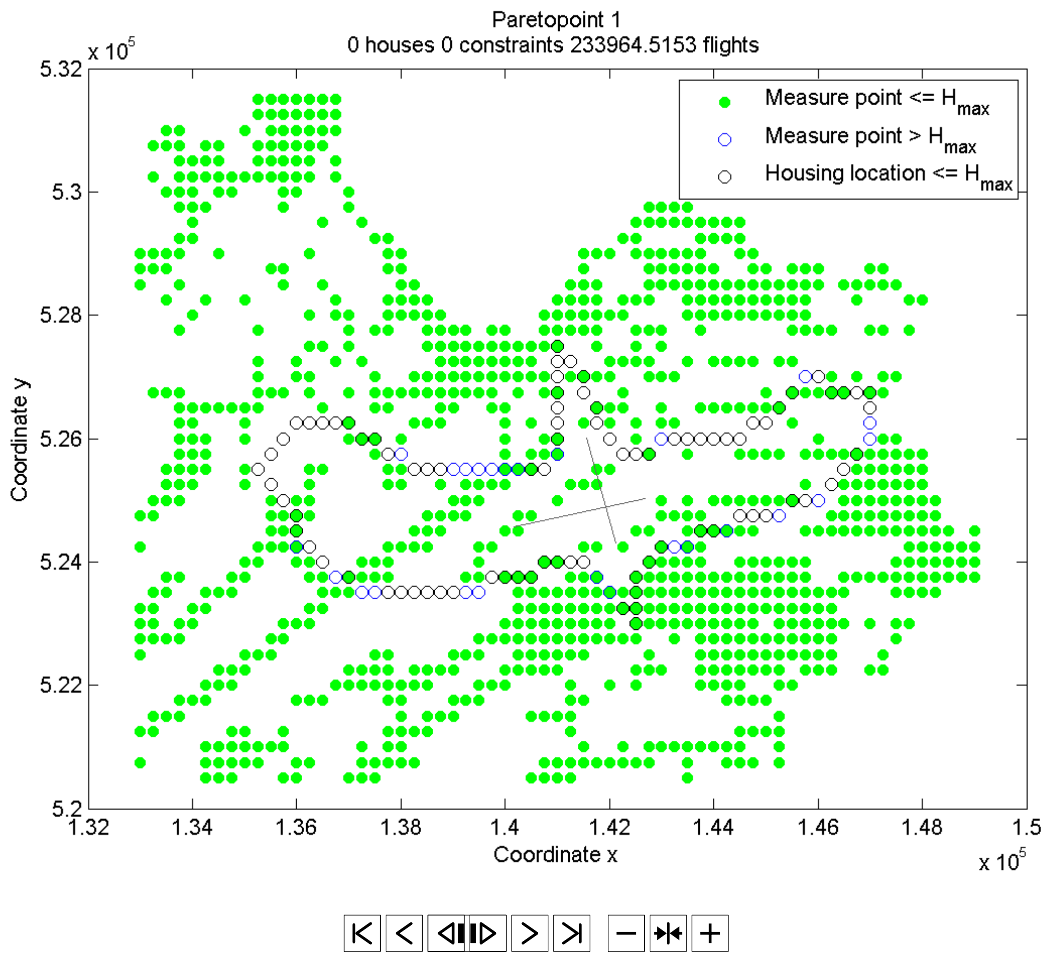


Fine grid

In the original Andijk airport instance we have 500 by 500 meter grid on which we group houses to housing locations. We look at a finer grid of 250 by 250 meters and solve this new problem instance. This will give different solutions since housing locations will have different weights and there will be more options available. The following animation show the solutions found by the Big-M method

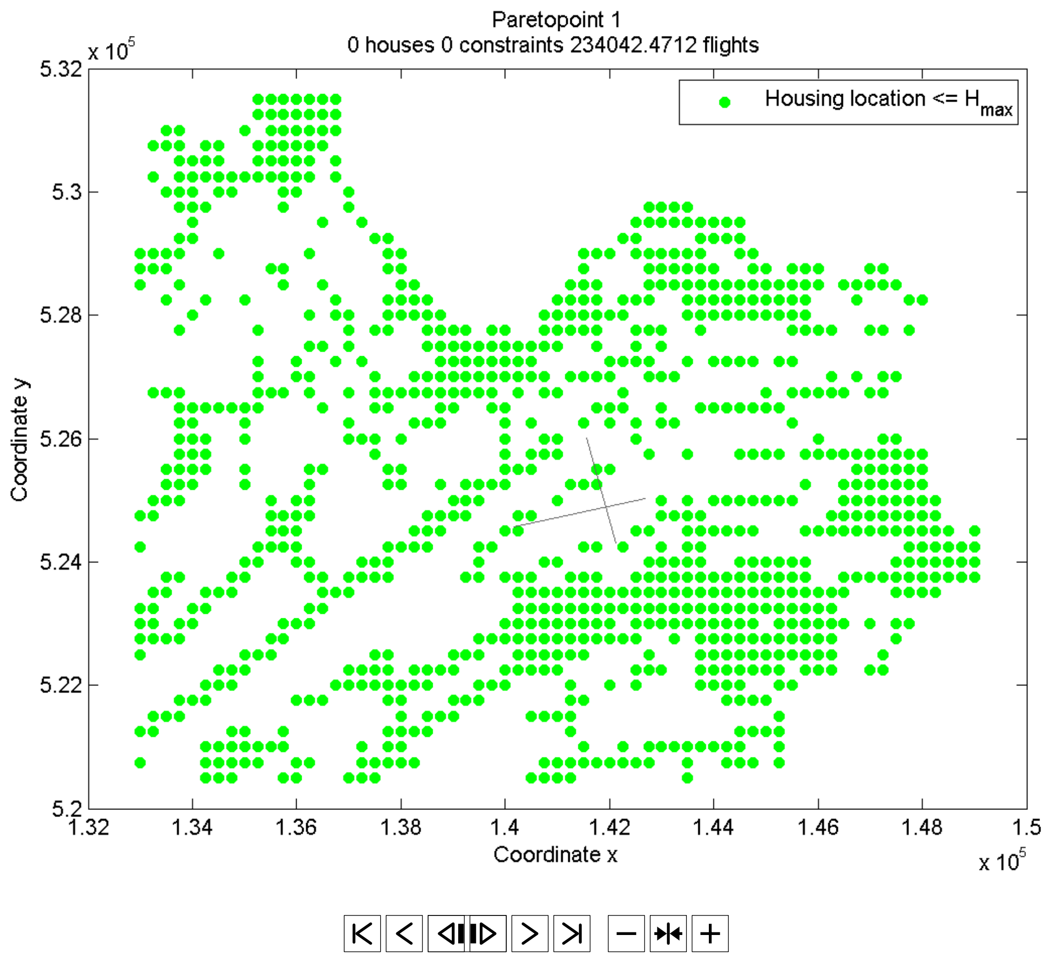


The following animation show the solutions found the second Chinneck algorithm for this problem instance.



Without measure points on a 250 meter grid

Also with a 250 meter grid we looked at Andijk airport without measure points in Appendix A.2.1. For this instance we can also schedule an infinite amount of flight, but now with 7 houses getting more then their threshold amount of noise pollution. The following animation shows the solutions found by the Big-M method for this instance.


Paretopoint 1
0 houses 0 constraints 234042.4712 flights
Housing location <= H_max
Coordinate y
Coordinate x
x 10^5