A maximum feasible linear subsystem problem for modeling noise pollution around airports

Sandra Hommersom
(3032256)

Master thesis in Mathematics

Radboud university Nijmegen
The Netherlands

Supervisor:
Prof. dr. Guido Schäfer
Networks & Optimization
CWI

Supervisor:
Dr. Wieb Bosma
Algrebra & Topology
Radboud University

December, 2014

Abstract

This thesis is the product of a graduation project at CWI, the Dutch national research centre for mathematics and computer science. We study the problem of noise pollution around airports. The main problem that is investigated is to minimize the number of houses suffering from noise pollution, while maximizing either the number of executed flights, or the number of passengers carried by the executed flights. We define this problem as a maximum feasible linear subsystem problem with integer variables. The theoretical part of this thesis investigates the complexity of several decision variants of the problem with binary, integer and real variables. Thereafter we introduce a simulation platform, which describes an imaginary airport environment. The goal is then to see which instances of the main problem can be solved efficiently.

Preface

This thesis is the product of my last project as a Mathematics student at Radboud University in Nijmegen. Within these five and a half years, I had the chance to discover the beauty of mathematics and moreover, I got to know many new people. I want to thank my friends in Nijmegen for sharing my enthusiasm for mathematics, studying together, cooking together and all the other things we did together. Also I would like to to thank my family and non-mathematical friends for supporting me during these years and tolerating me talking about mathematics, although they were not always very interested in the topic themselves.

This project was supposed to be a joint project between CWI (the Dutch national research centre for mathematics and computer science) and NLR (the Dutch National Aerospace Laboratory). Sadly, due to circumstances, we were not able to continue this collaboration. I would like to thank CWI, especially Guido, to give me the opportunity to finish the project nonetheless by extending my position as an intern at CWI.

I would like to express my thanks to my supervisors Guido (CWI) and Wieb (RU). Although Wieb was not much involved in the content related to the project, he always showed a great interest in the progress of my thesis and regularly provided me with useful advice. I want to thank Guido for our nearly weekly meetings, which always lasted longer than expected. During these meetings, he showed his enthusiasm for my results and always came up with new interesting questions. Also when the collaboration with NLR terminated, he immediately provided some new proposals for continuing the project at CWI, taking into account my wishes. Altogether, also by answering my e-mails late in the evenings or even in the weekends, he has put a huge effort in supervising me during this project.

Finally, I would like to thank my boyfriend Marcel. You were always there for me during my Master and in particular during this project. Especially at moments when I did not feel motivated for studying anymore, you cheered me up, encouraged me to look at things from a positive side and made me think of the nice prospects ahead of me. You were a great support.

Sandra Hommersom
Nijmegen, The Netherlands
December 3, 2014

Contents

Introduction

Over the last decades there has been a large increase of air traffic. Considering the largest commercial airport in the Netherlands, Amsterdam Schiphol Airport, the annual number of flight movements has grown from 105.466 in 1970, to 414.928 in 2000 and to 425.565 in 2013. Consequently, Amsterdam Schiphol Airport is fourth in the ranking of European airports with respect to the annual number of flight movements. At the same time, the passenger volume has also seen an enormous growth: from 5.172.000 in 1970, to 39.607.000 in 2000 and to 52.569.000 in 2013. In the upcoming years, the total European air traffic is expected to have an annual growth of 2,7% on average.

The increase of air traffic usually also yields an increase of the environmental pollution. This thesis focuses on noise pollution. For defining standards on this topic, the Dutch government considers two aspects. First, it tries to determine which amount of noise pollution is still acceptable and which is not. The question then is how to respect these thresholds.

The Dutch government currently defines a zone around every airport, beyond which a maximum level of annual noise pollution is imposed, that is not to be exceeded. To respect the given threshold, several measure points are placed at the boundary of this zone. Besides these strict measure points, the standards of noise pollution of air traffic focus on the protection of people in the direct habitat of an airport. Therefore, the noise pollution in residential areas surrounding an airport is also kept track of. The idea is to try to keep the amount of pollution acceptable for the people living in these areas.

Collecting the above information, one would like to keep the environmental noise pollution at a minimum, while satisfying the growing demand of air traffic. The goal of this thesis is to study this problem from a mathematical point of view. This investigation was started in an earlier thesis on this topic ([Jan13]).

From a theoretical point of view, we shall import the mathematical model for this problem that was proposed in [Jan13]. This model is a multi-objective optimization problem with linear constraints. The goal stated in this model is to maximize the number of houses receiving an acceptable amount of noise pollution, while also maximizing the demand of air traffic. Concerning the demand, we will consider both the total number of flight movements and the passenger volume. This extends the scope of [Jan13], where only the total number of flight movements was studied.

Considering the complexity of the given problem, we already know that the problem belongs to a class of hard problems. In [Jan13], several heuristics to solve the multi-objective optimization problem were proposed and tested. In this thesis we will deviate from this approach by trying to solve the problem optimally. This will be done by creating an imaginary airport environment and by feeding the corresponding optimization problem to a solver called SCIP.

As the above discussion suggests, this thesis is divided into two parts. We start the theoretical part in Chapter 1 by reviewing all basic mathematical concepts that are needed to understand the rest of the thesis. Chapter 2 introduces the problem under study in more detail, produces a mathematical model of it, and discusses the main theoretical results that are known about this problem or similar problems. In Chapter 3 we will present our additional complexity results on the problem.

The practical part reporting on simulations consists of Chapters 4 and 5. Chapter 4 explains the design of the imaginary airport environment and the source code for simulating this situation. In Chapter 5 the results of solving specific instances by SCIP are discussed.

Part I

THEORETICAL ASPECTS

Chapter 1

Preliminaries

This chapter introduces the reader to all basic definitions and concepts that will be used in this thesis. We shall introduce the basic theory of linear programming and then discuss the basic concepts of complexity theory and approximation algorithms.

The theory of linear programming is based on the first four chapters in [BT97]. The concepts of complexity theory and approximation algorithms are based on parts in [PS82], [Vaz01] and [Sch14].

1.1 Linear programming

In this thesis we will study several optimization problems. Our focus will mainly be put on linear programming problems and (mixed) integer programming problems. We shall introduce the basic theory of linear programming in this section.

Definition 1.1. An *optimization problem* Π is given by a set $\mathcal{I}$ of instances. An *instance* $I \in \mathcal{I}$ of an optimization problem specifies the following data:

1. a set $\mathcal{F}$ of feasible solutions for I,
2. a cost function $c : \mathcal{F} \to X$, where X is usually one of the sets $\mathbb{R}$, $\mathbb{Z}$ or $\mathbb{N}$.

Optimization problems can be either *maximization* problems or *minimization* problems. This terminology refers to the cost function. Given an instance $I = (\mathcal{F}, c)$ of Π, the goal of the problem is to find a feasible solution $f^* \in \mathcal{F}$ which optimizes the cost function. This means that f^* must satisfy the following:

- $c(f) \leq c(f^*)$ for all $f \in \mathcal{F}$, when Π is a maximization problem;
- $c(f^*) \leq c(f)$ for all $f \in \mathcal{F}$, when Π is a minimization problem.

Such an $f^* \in \mathcal{F}$ is called an *optimal solution* for I. The optimal cost $c(f^*)$ is denoted by $\text{opt}(I)$.

The *Linear programming problem* is an optimization problem of which instances are called *linear programs*. In a linear program, one wants to optimize a linear objective function over a set of feasible solutions, which are defined by a system of linear constraints.

A linear program can always be written in the following form.

$$\begin{array}{ll} \text{minimize} & \boldsymbol{c}^{\mathsf{T}}\boldsymbol{x} \\ \text{subject to} & \boldsymbol{A}\boldsymbol{x} \geq \boldsymbol{b} \\ & \boldsymbol{x} \geq \boldsymbol{0} \end{array} \tag{1.1}$$

Here $\boldsymbol{c} \in \mathbb{R}^n$ is the *cost vector*, $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ is the *constraint matrix*, $\boldsymbol{b} \in \mathbb{R}^m$ is the *constraint vector* and the entries $x_1, \ldots, x_n$ of $\boldsymbol{x}$ are the *variables*. We shall assume that the matrix $\boldsymbol{A}$ has linearly independent rows. Row i is the vector $\boldsymbol{a}_i$ and column j is denoted by $\boldsymbol{A}_j$.

A vector $\boldsymbol{x} \in \mathbb{R}^n$ is called a *feasible solution* (or just *solution*) to (1.1) if it satisfies all linear constraints. A vector $\boldsymbol{x}^* \in \mathbb{R}^n$ is called an *optimal solution* to (1.1) if it minimizes the objective function, i.e. $\boldsymbol{c}^{\mathsf{T}}\boldsymbol{x}^* \leq \boldsymbol{c}^{\mathsf{T}}\boldsymbol{x}$ for all feasible solutions $\boldsymbol{x}$ to (1.1). If such an optimal solution exists, program (1.1) is called *bounded*. If there exists a feasible solution, but there does not exist an optimal solution, the program is called *unbounded*. If there does not exist any feasible solution, the program is called *infeasible*.

1.1.1 Geometry

The set of vectors $\boldsymbol{x}$ satisfying some inequality $\boldsymbol{a}^{\mathsf{T}}\boldsymbol{x} \geq b$ is called a *halfspace*. Its boundary $\{\boldsymbol{x} \in \mathbb{R}^n \mid \boldsymbol{a}^{\mathsf{T}}\boldsymbol{x} = b\}$ is said to be a *hyperplane*. The intersection of a finite number of halfspaces is called a *polyhedron*. An *extreme point* of a polyhedron P is a vector $\boldsymbol{x} \in P$ which cannot be written as a convex combination of two other elements of P. That is, if $\boldsymbol{x} \in P$ and there do not exist $\boldsymbol{y}, \boldsymbol{z} \in P$ (both different from $\boldsymbol{x}$) and $\lambda \in [0, 1]$ such that $\boldsymbol{x} = \lambda\boldsymbol{y} + (1 - \lambda)\boldsymbol{z}$.

Let P be any polyhedron defined by linear (in)equality constraints. If $\boldsymbol{x} \in \mathbb{R}^n$ satisfies an (in)equalilty constraint with equality, then this constraint is called *active*. Thus if $\boldsymbol{x} \in P$, then all equality constraints are active. We call $\boldsymbol{x} \in \mathbb{R}^n$ a *basic solution* if

(i) all equality constraints are active;

(ii) out of the constraints that are active at $\boldsymbol{x}$, there are n of them that are linearly independent.

Here we say that the inequality $\boldsymbol{a_1}^{\mathsf{T}}\boldsymbol{x} \geq b_1$ is *linearly independent* from the inequality $\boldsymbol{a_2}^{\mathsf{T}}\boldsymbol{x} \geq b_2$ if the vectors $\boldsymbol{a_1}$ and $\boldsymbol{a_2}$ are linearly independent. Notice that a basic solution does not need to be an element of P. If a basic solution $\boldsymbol{x}$ satisfies $\boldsymbol{x} \in P$, then it is called a *basic feasible solution*.

In an extreme point of a polyhedron $P \subset \mathbb{R}^n$ there are n linearly independent active inequalities. It can be proven that the extreme points of a polyhedron P are exactly its vertices and that these are exactly its basic feasible solutions.

1.1.2 Standard form programs and optimality

In the case of a linear program, the set $\{\boldsymbol{x} \in \mathbb{R}^n \mid \boldsymbol{A}\boldsymbol{x} \geq \boldsymbol{b}, \boldsymbol{x} \geq \boldsymbol{0}\}$ of feasible solutions to (1.1) is a polyhedron. For determining optimal solutions, we consider a different formulation of a linear program. By adding *surplus variables* to denote the amount of surplus in the linear constraints given by $\boldsymbol{A}$ and $\boldsymbol{b}$, any linear program can be written in its *standard form*, which is the following.

$$\begin{aligned} \text{minimize} \quad & \boldsymbol{c}^\mathsf{T}\boldsymbol{x} \\ \text{subject to} \quad & \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} \\ & \boldsymbol{x} \geq \boldsymbol{0} \end{aligned} \tag{1.2}$$

As every equality can be written equivalently as two inequalities, its set of feasible solutions

$$\mathcal{F} = \{\boldsymbol{x} \in \mathbb{R}^n \mid \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}, \boldsymbol{x} \geq \boldsymbol{0}\}$$

is a polyhedron. We call this a polyhedron *in standard form*. We shall now explain how to find basic solutions of such a polyhedron.

Since in any basic solution all equality constraints must be satisfied, the system $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ provides us with m linearly independent active constraints. This is because we assumed the rows of $\boldsymbol{A}$ to be linearly independent. In order to obtain n active constraints, we must choose $n-m$ variables and set them equal to 0. These variables are called *non-basic variables*. Let us denote the indices of the other variables, the so-called *basic variables*, by $B(1), \ldots, B(m)$. It is known that for the n active constraints to be linearly independent, we must choose the non-basic variables in such a way that the columns $\boldsymbol{A}_{B(1)}, \ldots, \boldsymbol{A}_{B(m)}$ are linearly independent.

Thus far we have only created n linearly independent active constraints. To obtain a basic solution, we still must determine the values of the basic variables, which are given by the vector $\boldsymbol{x}_B := (x_{B(1)}, \ldots, x_{B(m)})^\mathsf{T}$. These values must be chosen in such a way that $\boldsymbol{x}$ satisfies $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$. We shall denote the square matrix with columns $\boldsymbol{A}_{B(1)}, \ldots, \boldsymbol{A}_{B(m)}$ by $\boldsymbol{B}$. This is called the *basis matrix* associated with $\boldsymbol{x}$. As its columns are linearly independent, $\boldsymbol{B}$ is invertible. Therefore, we can determine a basic solution by computing the basic variables as the unique solution of $\boldsymbol{B}\boldsymbol{x}_B = \boldsymbol{b}$.

If this procedure results in a vector $\boldsymbol{x}$ such that $\boldsymbol{x}_B \geq \boldsymbol{0}$, then $\boldsymbol{x}$ is a basic feasible solution (hence an extreme point or a vertex of S). It does make sense to search for basic feasible solutions by this procedure, because one can prove that for every feasible linear program (1.2):

(i) the set S has at least one basic feasible solution;

(ii) either the program is unbounded or there exists a basic feasible solution that is optimal.

Most optimization algorithms solve linear programs in the following way: they start with finding some feasible solution and then search for nearby 'better solutions', that

is, feasible solutions with lower cost. Whenever it is not possible to find such better solutions anymore, the current feasible solution is a local optimum and the algorithm terminates. Because in linear programming one minimizes a linear function over a linear set, this local optimum is a global optimum and therefore an optimal solution is found.

Moving to nearby feasible solutions can be done by making one non-basic variable basic and removing another basic variable (that is, set this one to 0). Suppose that we move in some direction by choosing variable x_j to become basic instead of non-basic, then the rate of cost change along this direction is called the *reduced cost* of variable x_j. This is defined in the following way.

Definition 1.2. Let $\boldsymbol{x}$ be a basic solution and let $\boldsymbol{B}$ be its associated basis matrix. Let $\boldsymbol{c}_B := (c_{B(1)}, \ldots, c_{B(m)})^\mathsf{T}$ be the vector of costs of the basic variables. For each j, we define the *reduced cost* $\bar{c}_j$ of variable x_j by

$$\bar{c}_j = c_j - \boldsymbol{c}_B^\mathsf{T} \boldsymbol{B}^{-1} \boldsymbol{A}_j.$$

In order to move into a 'good' direction from a basic feasible solution, we must choose a non-basic variable with negative reduced cost to become basic. As already mentioned, an optimal solution is always one of the basic feasible solutions of the polyhedron S. We now have enough ingredients to check for a basic feasible to be optimal.

Definition 1.3. Let $\boldsymbol{x}$ be a basic solution and let $\boldsymbol{B}$ be its associated basis matrix. Then $\boldsymbol{x}$ is an optimal solution if it satisfies the following *optimality conditions*:

1. it is feasible, i.e. $\boldsymbol{x}_B = \boldsymbol{B}^{-1}\boldsymbol{b} \geq \boldsymbol{0}$;
2. it has non-negative reduced costs, i.e. $\bar{\boldsymbol{c}}^\mathsf{T} = \boldsymbol{c}^\mathsf{T} - \boldsymbol{c}_B^\mathsf{T}\boldsymbol{B}^{-1}\boldsymbol{A} \geq \boldsymbol{0}^\mathsf{T}$.

A commonly used method to solve linear programs is the *simplex method*. This algorithm searches for an optimal solution by moving along the edges of the polyhedron S from one basic feasible solution to another. Although this algorithm is not efficient in theory, it often works very well in practice.

1.1.3 Duality theory

A very important concept in the theory of linear programming is that of *duality*. In linear programming solvers, this is often used as a tool to check for optimality of feasible solutions. The explanation for this is captured in the theorem of *strong duality*. We now shall discuss this in more detail.

Linear program (1.2) has the following *dual program*.

$$\begin{aligned} \text{maximize} \quad & \boldsymbol{p}^\mathsf{T}\boldsymbol{b} \\ \text{subject to} \quad & \boldsymbol{p}^\mathsf{T}\boldsymbol{A} \leq \boldsymbol{c}^\mathsf{T} \end{aligned} \tag{1.3}$$

Here $\boldsymbol{p} \in \mathbb{R}^m$ is called the *dual vector*.

Now suppose $\boldsymbol{x}$ and $\boldsymbol{p}$ are feasible solutions to (1.2) resp. (1.3). Then we have

$$\boldsymbol{p}^\top \boldsymbol{b} = \boldsymbol{p}^\top \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{c}^\top \boldsymbol{x}. \tag{1.4}$$

The existence of feasible solutions to the dual program thus bounds the primal program from below. On the other hand, the existence of feasible solutions to the primal program gives us upper bounds on the dual program. Whenever there exist feasible solutions to both the primal and the dual program, equation (1.4) is called *weak duality*.

Theorem 1.4 (Weak duality)**.** If $\boldsymbol{x}$ is a feasible solution to the primal problem (1.2) and $\boldsymbol{p}$ is a feasible solution to the dual problem (1.3), then

$$\boldsymbol{p}^\top \boldsymbol{b} \leq \boldsymbol{c}^\top \boldsymbol{x}.$$

□

Notice that from this it follows that whenever the primal program is unbounded, the dual program must be infeasible (and vice versa). Another consequence of this insight is that if there exists a feasible solution to both the primal and the dual program and equation (1.4) is satisfied with equality, then these feasible solutions must both be optimal solutions.

The latter statement also holds the other way around.

Theorem 1.5 (Strong duality)**.** If a linear programming problem has an optimal solution, so does its dual, and the respective optimal costs are equal.

Proof. Suppose $\boldsymbol{x}$ is an optimal solution to (1.2) and let $\boldsymbol{B}$ be its optimal basis. Let $\boldsymbol{x}_B = \boldsymbol{B}^{-1}\boldsymbol{b}$ be the corresponding vector of basic variables. Since $\boldsymbol{x}$ is optimal, we know that $\boldsymbol{c}^\top - \boldsymbol{c}_B^\top \boldsymbol{B}^{-1}\boldsymbol{A} \geq \boldsymbol{0}^\top$. Consider $\boldsymbol{p}^\top = \boldsymbol{c}_B^\top \boldsymbol{B}^{-1}$. Then $\boldsymbol{p}^\top \boldsymbol{A} \leq \boldsymbol{c}^\top$ so that $\boldsymbol{p}$ is a feasible solution to (1.3). Furthermore, we have

$$\boldsymbol{p}^\top \boldsymbol{b} = \boldsymbol{c}_B^\top \boldsymbol{B}^{-1}\boldsymbol{b} = \boldsymbol{c}_B^\top \boldsymbol{x}_B = \boldsymbol{c}^\top \boldsymbol{x}.$$

By weak duality we see that $\boldsymbol{p}$ is an optimal solution. Moreover, we have proven that the respective optimal costs are equal. □

1.1.4 Optional and binding constraints

In this thesis, we shall consider linear programs in which a solution does not have to satisfy all linear constraints. In this case, the constraints that must be satisfied are called *binding* or *mandatory* constraints, while the others are called *optional* constraints. A suitable vector is then called a feasible solution if it satisfies all binding linear constraints. If the set of constraints contains optional constraints, this must be explicitly mentioned. All other constraints can be assumed to be binding.

1.1.5 Integer linear programming

We will consider programs that are closely related to linear programs: (mixed) integer linear programs and even binary linear programs. In an integer linear program, the variables are not only required to be non-negative, but also must be integral. A mixed integer linear program can contain both non-negative real and integer variables. In a binary linear program, the variables are required to be either 0 or 1.

Any integer linear program can be written in the following form.

$$\begin{aligned} \text{minimize} \quad & \boldsymbol{c}^{\mathsf{T}}\boldsymbol{x} \\ \text{subject to} \quad & \boldsymbol{A}\boldsymbol{x} \geq \boldsymbol{b} \\ & \boldsymbol{x} \in \mathbb{N} \end{aligned} \tag{1.5}$$

This seemingly small change in the problem turns out to have a large effect on the difficulty of the problem. While the Linear Programming Problem can be solved efficiently, the Binary Linear Programming Problem (and therefore also the (Mixed) Integer Linear Programming Problem) is NP-complete. The next section will explain what we exactly mean by this. Integer linear programming solvers often look into the *LP-relaxation* and use the solution to the latter problem to find an integral solution of the original problem. The reason for this is that in general, an LP-relaxation can be solved much more efficiently than its integer linear program.

Given a (mixed) integer linear program, the LP-relaxation relaxes on the integrality of the variables. The LP-relaxation of problem (1.5) is the following linear program.

$$\begin{aligned} \text{minimize} \quad & \boldsymbol{c}^{\mathsf{T}}\boldsymbol{x} \\ \text{subject to} \quad & \boldsymbol{A}\boldsymbol{x} \geq \boldsymbol{b} \\ & \boldsymbol{x} \geq 0 \end{aligned} \tag{1.6}$$

1.2 Complexity theory

In complexity theory, one studies the 'hardness' of computational problems. The hardness of a problem is based on the time and memory needed to solve it. Complexity theory usually focuses on *decision problems*. These are problems for which the solution is an answer 'yes' or 'no'. Formally, decision problems are defined as follows.

Definition 1.6. A *decision problem* Π is given by a set $\mathcal{I}$ of instances. An *instance* $I \in \mathcal{I}$ of a decision problem specifies the following data:

1. a set $\mathcal{F}$ of feasible solutions for I;
2. a cost function $c : \mathcal{F} \to X$, where X is usually one of the sets $\mathbb{R}$, $\mathbb{Z}$ or $\mathbb{N}$;
3. an integer K.

Given an instance $I = (\mathcal{F}, c, K)$ of a decision problem Π, the goal is to decide if there exists a feasible solution $f \in \mathcal{F}$ which satisfies some statement $S_I(f)$ involving $c(f)$ and K. If there exists $f \in \mathcal{F}$ such that $S_I(f)$, then I is called a *yes-instance*. Otherwise, I is called a *no-instance*.

In order to investigate the hardness of optimization problems, we have to turn these into decision problems. Given an instance I of an optimization problem Π, this can be done by adding an integer K to the data which I specifies. We shall denote the decision variant of an optimization problem Π by DEC-Π.

For maximization problems, a natural goal for its decision variant is to find a solution $f \in \mathcal{F}$ which satisfies $c(f) \geq K$. Similarly, finding a solution $f \in \mathcal{F}$ satisfying $c(f) \leq K$ is a natural goal to set for minimization problems. For approximation purposes, which shall be explained later on, it is useful that the cost $c(f)$ takes *exactly* some value K.

Given an optimization problem Π, unless stated otherwise, we will consider its decision variant DEC-Π, where given an instance $I = (\mathcal{F}, c, K) \in \mathcal{I}$, the goal is to find $f \in \mathcal{F}$ such that $c(f) = K$ (that is, $S_I(f) \equiv c(f) = K$).

1.2.1 Complexity classes

Decision problems and optimization problems are examples of computational problems. For a computer to solve such a problem, it needs a sequence of instructions that describe a specific computational procedure solving every instance of the problem. Such a sequence of instructions is called an *algorithm*.

We are interested in *efficient* algorithms. These are algorithms which are not 'too time-consuming'. The overall *running time* of an algorithm can be expressed by the number of elementary operations that the algorithm needs to solve a problem. There are several ways to determine the running time of an algorithm, but we shall represent it by the *worst-case* running time. This is the running time of the algorithm on the worst possible input instance.

The number of elementary operations used in an algorithm depends on the size of the input. The *size* $|I|$ of an instance I is determined by its number of objects. For graphs, for example, this involves the number of nodes and the number of edges. The size of a linear program consists of the dimensions of the constraint matrix and the values of the cost vector, the constraint vector and the constraint matrix.

Now we are ready to formally explain what we mean by efficient algorithms. We say that a problem Π can be solved in *polynomial time* (or equivalently, is *polynomial-time solvable*), if there exists an algorithm whose (worst-case) running time is bounded by a polynomial in the input size. That is, if there exists an algorithm $\mathcal{A}$ and a polynomial p such that for every instance I of Π, $\mathcal{A}$ solves the problem in running time bounded by $p(|I|)$.

Definition 1.7. The complexity class P consists of all decision problems that are solvable in polynomial time.

Examples of problems in P are the resp. decision variants of *Shortest path problem* and the *Maximum flow problem*. Even though the simplex method is not a polynomial-time algorithm, the *Linear programming problem* is a problem in P.

We shall now introduce another complexity class. This class contains decision problems that are *verifiable* in polynomial time. Given a decision problem Π and a yes-instance $I = (\mathcal{F}, c, K) \in \mathcal{I}$, we say that f is a *certificate* for I if $f \in \mathcal{F}$ and $S_I(f)$ holds. Notice that by definition, every yes-instance must have a certificate.

Definition 1.8. The complexity class NP consists of all decision problems Π for which every yes-instance $I \in \mathcal{I}$ admits a certificate whose validity can be verified in polynomial time.

It is not hard to see that P $\subseteq$ NP. For decades, researchers have been trying to prove that there are problems in NP that are significantly harder than problems in P. No one ever succeeded. However, there also exist problems in NP that are not known to admit a polynomial-time algorithm. Examples of such problems are the *Traveling salesman problem* and the *Integer programming problem*. The question whether P $\neq$ NP is one of the seven 'Millennium Prize Problems' (http://www.claymath.org/millennium-problems). Anyone who succeeds in solving one of these open problems receives an award of one million dollars.

The class NP contains an important subclass, the so-called NP-*complete* problems. This subclass contains the most difficult problems of NP. In order to define this complexity class, we need the notion of a *polynomial-time reduction*.

1.2.2 Reductions and NP-completeness

In this section, we shall introduce several kinds of reductions. In complexity theory, reductions are used to prove hardness of decision problems. A reduction from one problem to another usually means that the second problem is at least as hard to solve as the first one.

As already announced, we shall need the notion of a polynomial-time reduction.

Definition 1.9. Let Π_1 and Π_2 be two decision problems. A *polynomial-time reduction* or *Karp reduction* from Π_1 to Π_2 is a function $\phi : \mathcal{I}_1 \to \mathcal{I}_2$ that maps every instance $I_1 \in \mathcal{I}_1$ of Π_1 to an instance $I_2 := \phi(I_1) \in \mathcal{I}_2$ of Π_2 such that

1. the mapping can be done in time that is polynomially bounded in the size of I_1;
2. I_1 is a yes-instance of Π_1 if and only if I_2 is a yes-instance of Π_2.

If there exists such a reduction, we say that Π_1 is *polynomial-time reducible* to Π_2 and write $\Pi_1 \preceq \Pi_2$. As polynomial-time reductions can be composed, the relation $\preceq$ is transitive.

Definition 1.10. Two decision problems Π_1 and Π_2 are said to be *polynomial-time equivalent* if $\Pi_1 \preceq \Pi_2$ and $\Pi_2 \preceq \Pi_1$.

Suppose $\Pi_1 \preceq \Pi_2$. Let us see why problem Π_2 is then understood to be at least as hard as problem Π_1. Suppose that we are in possession of an algorithm that solves Π_2 in polynomial time. Then we can solve problem Π_1 by the following procedure: take an instance I_1 of Π_1 and apply ϕ to obtain an instance I_2 of Π_2. Apply the algorithm to answer the question whether I_2 is a yes-instance or a no-instance. By Property 2 of Definition 1.9, we can now determine whether I_1 is a yes-instance or a no-instance. If follows by Property 1 of Definition 1.9 that this procedure is a polynomial-time algorithm for Π_1.

From this argument we can immediately conclude that when Π_1 and Π_2 are equivalent and one of the problems can be solved efficiently, so could the other.

We are now able to define the complexity class containing the most difficult problems in NP.

Definition 1.11. A decision problem Π is called *NP-complete* if

1. $\Pi \in$ NP;
2. every problem in NP is polynomial-time reducible to Π.

A decision problem that only satisfies Property 2 is called *NP-hard*.

From the discussion above we deduce that finding a polynomial-time algorithm for only one NP-complete problem would prove NP $\subseteq$ P, hence P = NP. Up to this moment, no one succeeded in doing this, but also no one succeeded in proving that such algorithms cannot exist for NP-complete problems.

There are two kinds of NP-completeness. Let therefore $d(I)$ denote the largest data value appearing in an instance I. Sometimes it is possible for an NP-complete problem to find an algorithm whose running time is bounded by a polynomial in $|I|$ and $d(I)$. Such an algorithm is called *pseudo-polynomial*. If a decision problem remains NP-complete even when restricted to instances $I \in \mathcal{I}$ such that $d(I)$ is polynomially bounded by $|I|$, then the problem is called *strongly* NP-complete. This implies in particular that there cannot exist a pseudo-polynomial algorithm for strongly NP-complete problems. Problems that are NP-complete, but not in the strong sense, are called *weakly* NP-complete.

Let us spend a few words on a method of proving that a decision problem Π is NP-complete. Property 2 of Definition 1.11 indeed requires us to show that *every* problem in NP polynomial-time reduces to Π. Using the transitivity of polynomial-time reductions we can actually do something smarter. Suppose that we are able to reduce one other NP-complete problem Π' to Π in polynomial time. Then by definition 1.11, every problem in NP is polynomial-time reducible to Π'. Hence, by transitivity, every problem in NP is also polynomial-time reducible to Π. Therefore, in order to prove that Π is NP-hard, we only have to reduce from *one* NP-complete problem.

From this we can also understand that if one wants to prove that a decision problem is weakly (resp. strongly) NP-complete, one must reduce from another weakly (resp. strongly) NP-complete problem. An example of a weakly NP-complete problem is the *Knapsack Problem*. The *Maximum Independent Set Problem*, which we will soon introduce in more detail, is strongly NP-complete.

We shall now introduce a generalization of the concept of polynomial-time reducibility, which is *Turing reducibility*. While polynomial-time reductions can only have one call to solve another problem, Turing reductions may have polynomially many calls like this.

Definition 1.12. A computational problem Π_1 is called *Turing reducible* or *Cook reducible* to a computational problem Π_2 if there exists a polynomial-time algorithm $\mathcal{A}$ that solves problem Π_1 using polynomially many calls to an oracle that solves problem Π_2.

Here *polynomial* means polynomial in the size of Π_1. Since an oracle is used at every call of Π_2, the time needed to solve Π_2 is not counted in the total running time of $\mathcal{A}$. This definition tells us that once we have a polynomial-time algorithm to solve problem Π_2, problem Π_1 can be solved in polynomial time too.

This was also the case for polynomial-time reductions. The main important difference lies in the use of an oracle to solve problem Π_2 instead of a one-to-one correspondence of yes-instances. Turing reductions therefore do not preserve membership in NP, while polynomial-time reductions do. For some kind of reduction, *preserving membership in NP* means that whenever Π_1 is reducible to Π_2 and $\Pi_2 \in \text{NP}$, then $\Pi_1 \in \text{NP}$.

We deliberately defined Turing reductions for general computational problems, so that they can be applied to both optimization problems and decision problems.

1.3 Approximation algorithms

Many decision variants of optimization problems are known to be NP-complete. As it is widely believed that $\text{P} \neq \text{NP}$, this means that it is unlikely to find an efficient algorithm that solves such an optimization problem. An alternative way of tackling such problems is by finding algorithms that do run in polynomial time, but compute suboptimal solutions. However, if such an algorithm computes a solution that is 'close to' the optimum, we are often content with it.

Definition 1.13. Let Π be an optimization problem and let $\alpha \geq 1$. An algorithm $\mathcal{A}$ is said to be an *α-approximation algorithm* if it computes in polynomial time for every instance $I = (\mathcal{F}, c) \in \mathcal{I}$ a solution $f \in \mathcal{F}$ such that

- $c(f) \leq \alpha \cdot \text{opt}(I)$, whenever Π is a minimization problem,
- $c(f) \geq \frac{1}{\alpha} \cdot \text{opt}(I)$, whenever Π is a maximization problem.

We alternatively say that Π is *approximable within α* if there exists a α-approximation algorithm.

We shall now introduce another kind of reduction, which is called a *cost-preserving polynomial-time reduction*. Cost-preserving polynomial-time reductions between optimization problems are useful to find new approximation algorithms or to show an inapproximability result. These are the simplest *approximation preserving* reductions.

Definition 1.14. Let Π_1 and Π_2 be two optimization problems. A *cost-preserving polynomial-time reduction* from Π_1 to Π_2 is a function $\phi : \mathcal{I}_1 \to \mathcal{I}_2$ that maps every instance $I_1 = (\mathcal{F}_1, c_1) \in \mathcal{I}_1$ of Π_1 to an instance $\phi(I_1) = I_2 = (\mathcal{F}_2, c_2) \in \mathcal{I}_2$ of Π_2 such that

1. the mapping can be done in time that is polynomially bounded in the size of I_1;
2. $\text{opt}(I_1) = \text{opt}(I_2)$;
3. for every $f_2 \in \mathcal{F}_2$, there exists $f_1 \in \mathcal{F}_1$ such that $c_1(f_1) = c_2(f_2)$.

Whenever there exists such a cost-preserving polynomial-time reduction, we say that Π_2 is Π_1-*hard*.

Whenever we construct a cost-preserving polynomial-time reduction in this thesis, we will provide a proof that first creates an instance I_2 of Π_2, given an instance I_1 of Π_1. This defines the polynomial mapping ϕ. Then we shall show that there exists a solution $f_1 \in \mathcal{F}_1$ to I_1 with cost γ if and only if there exists a solution $f_2 \in \mathcal{F}_2$ to I_2 with cost γ. It is clear that such a construction satisfies all requirements of a cost-preserving polynomial-time reduction.

We shall now explain in more detail what it means for a cost-preserving polynomial-time reduction to be approximation preserving. Suppose therefore that Π_1 and Π_2 are two minimization problems, that there exists a cost-preserving polynomial-time reduction from Π_1 to Π_2 and that Π_2 is approximable within α. Call this α-approximation algorithm $\mathcal{A}_2$.

Consider algorithm $\mathcal{A}_1$ which is defined by the following procedure. Take any instance $I_1 \in \mathcal{I}_1$ and compute $I_2 = \phi(I_1)$. Apply algorithm $\mathcal{A}_2$ to obtain a solution to I_2 with cost ALG_2. By Property 3 of Definition 1.14, we can find a solution to I_1 with the same cost ALG_2. Let this solution be the output of algorithm $\mathcal{A}_1$ and denote its cost by ALG_1.

By Property 1 of Definition 1.14, algorithm $\mathcal{A}_1$ runs in polynomial time. Furthermore, we have $\mathsf{ALG}_1 = \mathsf{ALG}_2$ by definition and by Property 2 of Definition 1.14, we even have $\text{opt}(I_1) = \text{opt}(I_2)$. Altogether this means that we have

$$\mathsf{ALG}_1 = \mathsf{ALG}_2 \leq \alpha \cdot \text{opt}(I_2) = \alpha \cdot \text{opt}(I_1).$$

Hence, algorithm $\mathcal{A}_1$ is an α-approximation algorithm for Π_1.

On the other hand, this also means that if there exists a cost-preserving polynomial-time reduction from Π_1 to Π_2 and Π_1 is not approximable within α, then also Π_2 is not approximable within α. A similar argument holds for maximization problems.

A well-known optimization problem that brings a bad inapproximability result is the *Maximum independent set problem*. Given a graph $G = (V, E)$, an *independent set* in G is a

subset $V' \subseteq V$ such that for each $u, v \in V'$, $\{u, v\} \notin E$. The Maximum independent set problem is defined as follows.

Maximum independent set problem (MIS)

Instance:	Graph $G = (V, E)$.
Goal:	Find an independent set V' of G such that $\|V'\|$ is maximized.

The Maximum independent set problem will often be used to prove complexity results in Chapter 3. Unfortunately, the problem brings a very bad inapproximability result.

Theorem 1.15 ([Hås96])**.** MIS cannot be approximated within $|V|^{1-\varepsilon}$, for any $\varepsilon > 0$, unless any problem in NP can be solved in probabilistic polynomial time.

Chapter 2

The problem

This chapter introduces the reader to the problem mentioned in the Introduction. We shall describe the problem in more detail and formulate it in a mathematical model. Thereafter, we summarize the existing complexity results related to the problem.

2.1 Formulation of the problem

In this section, we formulate the problem mentioned in the Introduction in detail. We shall explain its practical origin and formulate it as a multi-objective optimization problem. Then we will set a bound on one of the objective functions to change the problem into an integer linear programming problem with one objective function.

2.1.1 Practical origin

We consider an airport, of which we are given its runways. Each runway consists of two endings. The endpoint of a runway represents the position where the aircraft is starting from or coming to a halt. For example, considering a runway pointing from East to West, the runway splits in a East-end runway and a West-end runway. At the East-end runway, we can have departures to the West and arrivals from the West. Similarly, at the West-end runway, we can have arrivals from the East and departures to the East. In this section, by 'runway' we will allways mean a runway ending.

Any flight routine depends on four aspects:

- The type of aircraft used for the routine, for example 'Boeing 737'.
- The runway used for the routine.
- The procedure the aircraft is following. This involves its ascend or descend, its speed profile and the accompanying thrust settings.
- The route the aircraft is following. This is the coordinate set in the plane, i.e. the top view of the routine.

We are given all possible flight routines at the airport.

Our interest concerns the environment of the airport. In the surroundings of the airport, we are given the locations of houses and some measure points from the government. At each of the government measure points we are given a threshold, which represents the maximum amount of noise pollution per year that is allowed at that point. These thresholds may definitely not be exceeded. Concerning the locations of houses, we are given the number of houses at the location and a threshold representing the maximum acceptable amount of noise pollution per year. Airports must try to not exceed these thresholds in their flight schedules.

Every flight routine contributes an amount of noise pollution to each of the described housing locations and government measure points. These amounts can be measured and are therefore given to us. The total noise pollution at a given point is obtained by summing up all the amounts of noise pollutions to that point by the executed flight routines. The goal of the problem is to maximize the number of flights scheduled, while minimizing the number of houses suffering from too much (that is, more than the given theshold) noise pollution.

2.1.2 Mathematical formulation

To convert the problem described above into a mathematical model, we are given the sets of flight routines, government measure points and housing locations.

R	Set of flight routines.
Q	Set of government measure points.
H	Set of housing locations.

Next, we create parameters for the amounts of noise pollution at measure points, the number of houses at housing locations and the thresholds. Except for the number of houses, which is a natural number, we assume that all these parameters take rational values.

$\alpha_{hr} \in \mathbb{Q}^+$	$h \in H, r \in R$	Noise pollution of flight routine r on housing location h.
$\beta_h \in \mathbb{Q}^+$	$h \in H$	Threshold of acceptable noise pollution at housing location h.
$\gamma_{qr} \in \mathbb{Q}^+$	$q \in Q, r \in R$	Noise pollution of flight routine r on government measure point q.
$\delta_q \in \mathbb{Q}^+$	$q \in Q$	Threshold of allowed noise pollution at government measure point q.
$\omega_h \in \mathbb{N}$	$h \in H$	Number of houses at housing location h.

Variables are denoted by vectors $\boldsymbol{x}$ of dimension $|R|$. An entry x_r denotes the number of flights scheduled that follow routine r. Therefore, we have $\boldsymbol{x} \in \mathbb{N}^{|R|}$. Given these data, we can write down the following set of inequalities. Notice the difference between

binding and optional constraints. The relations for the government measure points are binding, since these thresholds may not be exceeded. As violation of the thresholds for the housing locations is allowed (but not preferable), these relations are optional.

$$\begin{aligned} \textstyle\sum_{r\in R} \gamma_{qr} x_r &\leq \delta_q \quad \forall q \in Q \quad \text{(binding constraint)} \\ \textstyle\sum_{r\in R} \alpha_{hr} x_r &\leq \beta_h \quad \forall h \in H \quad \text{(optional constraint)} \\ x_r &\in \mathbb{N} \quad \forall r \in R \end{aligned} \tag{2.1}$$

Following the style of previous articles on these kinds of problems, we do not count the restriction on each of the variables as binding constraints. Of course, these restrictions are binding implicitly.

Every optional constraint h is assigned a *weight* ω_h. Recall that a vector $\boldsymbol{x} \in \mathbb{N}^{|R|}$ is called a *feasible solution* to the above system if it satisfies all binding constraints. The goal of the problem is to find a feasible solution $\boldsymbol{x}$ to (2.1) such that the sum of the entries $\sum_{r\in R} x_r$ is maximized *and* such that the total weight of the satisfied optional constraints is maximized.

We see that this problem is a *multiple objective optimization problem*, i.e. an optimization problem with multiple (in this case two) objective fucntions. To be able to write down the objective function concerning the total weight of satisfied optional constraints explicitely, we use a trick called the *Big-M method* ([ST08]). In this method, every optional constraint h is assigned a new binary variable z_h and a sufficiently large number M_h. This is done in such a way that if constraint h is satisfied, then $z_h = 1$ and otherwise $z_h = 0$, which means that the constraint is always satisfied. This way, the total weight of the satisfied optional constraints is equal to $\sum_{h\in H} \omega_h z_h$.

Our problem is then formulated as the following 2-objective optimization problem.

$$\begin{aligned} \max \quad & \textstyle\sum_{r\in R} x_r & & \\ & \textstyle\sum_{h\in H} \omega_h z_h & & \\ \text{subject to} \quad & \textstyle\sum_{r\in R} \gamma_{qr} x_r & \leq \delta_q & \forall q \in Q \\ & \textstyle\sum_{r\in R} \alpha_{hr} x_r & \leq \beta_h + M_h(1 - z_h) & \forall h \in H \\ & x_r & \in \mathbb{N} & \forall r \in R \\ & z_h & \in \{0, 1\} & \forall h \in H \end{aligned} \tag{2.2}$$

In order to make this approach work, we must choose the numbers M_h such that $M_h \geq \sum_{r\in R} \alpha_{hr} x_r - \beta_h$ for all $h \in H$. Next we formulate problem (2.2) as an integer linear program (that is, remove one the objective functions). Then we can elaborate on the choise of the M_h more.

The two objective functions in problem (2.2) are conflicting: if more flights are scheduled, then also more noise pollution is produced and thus less housing locations meet

their threshold. In order to optimize this problem by (mixed) integer linear programming solvers, only one objective function is allowed. For this purpose, we apply the *bounded objective function method* ([MA04]) to the first objective function. This means that we put a lower bound on the objective function and add this as a binding contraint to the problem. Similarly, if the objective function was of the type 'minimization', then one would have put an upper bound on it.

Introducing a new parameter κ, representing the lower bound on the first objective function, our problem becomes the following. In [Jan13], this problem is referred to as the Route Scheduling Problem.

$$\begin{array}{rlll} \max & \sum_{h\in H} \omega_h z_h & & \\ \text{subject to} & \sum_{r\in R} x_r & \geq \kappa & \\ & \sum_{r\in R} \gamma_{qr} x_r & \leq \delta_q & \forall q \in Q \\ & \sum_{r\in R} \alpha_{hr} x_r & \leq \beta_h + M_h(1 - z_h) & \forall h \in H \\ & x_r & \in \mathbb{N} & \forall r \in R \\ & z_h & \in \{0,1\} & \forall h \in H \end{array} \tag{2.3}$$

In this formulation we observe that for finding a feasible solution, there is no need for a variable x_r to exceed κ. Therefore we see that for the choice of the M_h, it is sufficient to have $M_h \geq \kappa \cdot \sum_{r\in R} \alpha_{hr} - \beta_h$ for all $h \in H$.

When investigating the complexity of this problem, as is done in the next chapter, we consider a special case of the problem, in which there are no government measure points. Furthermore, we assume that for every flight routine, we have a sufficient amount of copies of that routine. In this way, the variables can be taken binary instead of natural. We also assume that the thresholds at the housing locations are equal everywhere. As a last restriction, we assume that there is only one house at every housing location, so that we are actually maximizing the number of satisfied optional constraints. This special case of the problem is redefined at the beginning of the next chapter.

2.2 Problem notation

The system of optional constraints in our main problem is an example of a maximum feasible linear subsystem problem. These problems involve a set of optional linear relations and a set of binding linear relations. The objective is then to find a solution satisfying all binding relations and as much optional ones as possible. Hereby we follow the general approach and therefore not include the range of the variables into the set of binding constraints.

In this section we will introduce all maximum feasible linear subsystem problems that we consider throughout this thesis. To denote the different problems, we use a triplet $A \mid B \mid C$. Here the A field describes the main problem, the B field shows the relations

involved in the problem and the C field is used to add special requirements to the problem, like restrictions on the coefficients. We shall always consider problems where the relation of the optional constraints is $\leq$, as in (2.1). The B field then gives us the relation of the mandatory relations (if there are any). If we consider a decision variant, the B field also shows the required relation in the question to decide.

2.2.1 General versions

Let X and Y be subsets of $\mathbb{R}$. When choosing the set of binding relations to be empty, we define

$$\text{Max-}X\text{-FLS} \mid\mid Y$$

to be the problem of maximizing the number of satisfied optional constraints in the following problem.

$$\begin{aligned} \boldsymbol{A}\boldsymbol{x} &\leq \boldsymbol{b} \quad \text{(optional)} \\ \boldsymbol{x} &\in X^n \end{aligned} \tag{2.4}$$

Here FLS stands for *feasible linear subsystem*, X specifies the range of the variables and Y the range of the entries of the constraint matrix A and the constraint vector b so that we have $\boldsymbol{A} \in Y^{m \times n}$ and $\boldsymbol{b} \in Y^m$.

If there are mandatory relations, thus we are considering a *constrained* maximum feasible linear subsystem, this is denoted by adding a C in the first field:

$$\text{Max-}X\text{-C FLS} \mid \diamond_b \mid Y.$$

In this case, $\diamond_b$ is the relation of the binding constraints and Y also denotes the range of the coefficients in the mandatory relations.

When considering a decision variant, we remove the phrase Max from the first field and add to the middle field the required condition for the decision. Let therefore denote $\diamond_d$ the decision relation and c the decision parameter. When for example $\diamond_d$ is the relation $\geq$, then

$$X\text{-FLS} \mid \diamond_d c \mid Y$$

is the problem of deciding if there exists a solution $\boldsymbol{x} \in X^n$ which satisfies at least c optional constraints. In this case,

$$X\text{-C FLS} \mid \diamond_b, \diamond_d c \mid Y$$

is the problem of deciding if there exists a solution $\boldsymbol{x} \in X^n$ which satisfies all binding constraints and at least c optional ones.

2.2.2 Specifying our version

As explained in the previous section, we will consider a specific maximum feasible linear subsystem problem for theoretical purposes. Let us therefore be given two finite index sets I and J, where I is used to index the optional constraints and J the variables. In Part I, m is defined to be the number of optional constraints, that is $m = |I|$, and n is the number of variables, $n = |J|$. In the upcoming chapter, we will consider the linear program

$$\begin{aligned} \sum_{j\in J} p_j x_j \diamond_b k & \quad && \text{(binding)} \\ \sum_{j\in J} a_{ij} x_j \leq b_i & \quad \forall i \in I && \text{(optional)} \\ x_j \in X & \quad \forall j \in J \end{aligned} \tag{2.5}$$

where we must have $a_{ij}, b_i \in Y$. Notice the generalized version of the binding constraint, where we have introduced *profits* p_j. The reason for this will become clear later.

Considering the notation, we observe some special features of this problem. First of all, the problem has exactly one binding constraint. We shall emphasize this in the notation by writing C1 in the A field instead of just C. We shall also specify the value of k, which is always assumed to be a natural number.

In the C field, the following properties will occur:

- $\boldsymbol{b}$ *uni*, meaning that all entries b_i of $\boldsymbol{b}$ are equal,
- $\boldsymbol{p} = 1$, meaning that all profits p_j are equal to 1.

For X and Y defined similarly as before, the decision problem corresponding to program (2.5) is then denoted by

$$X\text{-C1 FLS} \mid \diamond_b k, \diamond_d c \mid Y,$$

where c again is the decision parameter. The optimization variant of the problem is denoted by

$$\text{Max-}X\text{-C1 FLS} \mid \diamond_b k \mid Y.$$

2.3 Related work

In this section we will give an overview of the research that has been done on maximum feasible linear subsystem problems. Sometimes we will compare these results to the specific version of the problem we are considering.

In this specific version, we assume that the constraint matrix A and the constraint vector b, constituting the optional relations, have non-negative rational entries. This is because these coefficients denote noise pollutions and thresholds. By multiplying each optional constraint by a sufficiently large number, we can even assume that these entries are natural numbers. Furthermore, we consider a variant having exactly one binding constraint. The variables, denoting the number of flight routines, are natural numbers,

but can assumed to be binary for theoretical purposes. Therefore, we shall sometimes compare previous results about maximum feasible linear subsystem problems to the problem

$$\text{Max-0/1-C1 FLS} \mid \geq k \mid \mathbb{N}.$$

The use of a non-negative constraint matrix $\boldsymbol{A}$ and a non-negative constraint vector $\boldsymbol{b}$ in linear programming problems is by a few articles referred to as *positive linear programming*, see e.g. [LN93]. However, no articles are known that combine maximum feasible subsystem problems with this concept.

2.3.1 Research done by Amaldi and Kann

Amaldi and Kann have investigated the complexity and approximability of maximum feasible linear subsystem problems intensively. In their article [AK95a] they treat many different versions of the general optimization problem Max-$\mathbb{R}$-FLS $\mid\mid \mathbb{R}$. First only the complexity of this problem is investigated, while thereafter also versions of the problem that have a set of mandatory relations are considered. Finally, Amaldi and Kann consider versions of the problem that have binary or bipolar variables (with and without a set of mandatory relations). The latter are especially interesting, as these are more related to our maximum feasible subsystem problem (having binary variables and a specific mandatory relation).

While Amaldi and Kann treat the complexity of problems with the relation on the optional constraints being any in the set $\{=, \leq, <, \neq\}$, we shall focus in the discussion below on the case when this relation is $\leq$, since the optional constraints in our version are stated with this relation. It is proven ([AK95a, Thm. 5]) that Max-$\mathbb{R}$-FLS $\mid\mid \mathbb{Z}$ is APX-hard. This means that it cannot be approximated within aribitrary constants, unless P=NP. Furthermore, a 2-approximation is given ([AK95a, Prop. 9]), which exploits the possibility of having negative variables. In particular, the algorithm does not work if one considers non-negative variables only. Moreover, it is proven that Max-0/1-FLS $\mid\mid \mathbb{R}$ is at least as hard to approximate as the Maximum Independent Set Problem.

Yet considering Max-0/1-FLS $\mid\mid \mathbb{N}$, the unconstrained version in our case, the problem suddenly becomes trivial: the vector $\boldsymbol{x} = \boldsymbol{0}$ satisfies all relations, since $0 \leq b_i$ for $1 \leq i \leq m$. Even when we do not allow the trivial solution, the problem remains easy. Therefore we notice that whenever $\boldsymbol{x} \in \{0,1\}^n$ satisfies the relation $\sum_{j=1}^{n} a_{ij}x_j \leq b_i$, then also every unit vector $\boldsymbol{e^l}$ such that $x_l = 1$ satisfies $\sum_{j=1}^{n} a_{ij}e_j^l \leq b_i$. Hence, the optimal solution is one of the n possible unit vectors. We conclude that our version of the problem is interesting only when the mandatory relation is included.

The constrained version of Max-$\mathbb{R}$-FLS $\mid\mid \mathbb{R}$, denoted as Max-$\mathbb{R}$-C FLS $\mid \diamond_b \mid \mathbb{R}$, also includes a set of mandatory relations. It is proven ([AK95a, Thm. 10]) that Max-$\mathbb{R}$-C FLS $\mid \leq \mid \mathbb{R}$ is at least as hard to approximate as the Maximum Independent Set Problem. Furthermore, in the article it is argued that every instance of Max-$\mathbb{R}$-C FLS $\mid = \mid \mathbb{R}$ can be transformed to an equivalent instance of Max-$\mathbb{R}$-FLS $\mid\mid \mathbb{R}$ by eliminating variables in the set of optional relations using the set of mandatory

equations. This is interesting, because we just argued that our corresponding problem MAX-0/1-FLS | | $\mathbb{N}$ is an easy one.

We could eliminate one variable by using the binding constraint. However, the non-negativity of $\boldsymbol{A}$ and $\boldsymbol{b}$ may not be preserved during this elimination. We can therefore not conclude that MAX-0/1-C1 FLS | $=k$ | $\mathbb{N}$ is an easy problem by the approach of [AK95a].

Finally, the constrained version of MAX-$\mathbb{R}$-FLS | | $\mathbb{R}$ with binary variables is considered. This is then the problem MAX-0/1-C FLS | $\diamond_b$ | $\mathbb{R}$. It is proven that MAX-0/1-C FLS | $\diamond_b$ | $\mathbb{R}$ is NPO PB-complete for $\diamond_b \in \{=, \leq\}$. Here NPO stands for 'NP optimization' and refers to a class of optimization problems to which these maximum feasible subsystem problems belong. The phrase PB stands for 'polynomially bounded' and it is a property of objective functions of NPO problems. The proof of the statement implies that, unless P=NP, MAX-0/1-C FLS | $\diamond_b$ | $\mathbb{R}$ with $\diamond_b \in \{=, \leq\}$ is not approximable within $m^{1-\varepsilon}$ for any $\varepsilon > 0$.

The article leaves some open questions for the unconstrained version of the problem. As said, this version is not very interesting in our case and therefore we will not answer those open questions. But we will obtain many inapproximability results for the specific version of the problem we are considering.

In [AK95b], Amaldi and Kann tackle the general maximum feasible linear subsystem problem from a different view. Instead of finding the largest possible set of relations that can be satisfied simultaneously, one could also try to find a solution violating as few relations as possible while satisfying all the others. This problem is referred to as MIN-$\mathbb{R}$-ULR | | $\mathbb{R}$ (ULR standing for *unsatisfied linear relations*) and the different versions of the problem are defined similarly as for MAX-$\mathbb{R}$-FLS | | $\mathbb{R}$.

The problems MIN-$\mathbb{R}$-ULR | | $\mathbb{R}$ and MAX-$\mathbb{R}$-FLS | | $\mathbb{R}$ are clearly equivalent in terms of optimization: whenever OPT is the optimal objective value of MAX-$\mathbb{R}$-FLS | | $\mathbb{R}$, then $m -$ OPT is the optimal objective value for MIN-$\mathbb{R}$-ULR | | $\mathbb{R}$ (and vice versa). However, it turns out that MIN-$\mathbb{R}$-ULR | | $\mathbb{R}$ is harder than MAX-$\mathbb{R}$-FLS | | $\mathbb{R}$ in terms of approximation. While a 2-approximation exists for MAX-$\mathbb{R}$-FLS | | $\mathbb{R}$, it is proven in [AK95b] that MIN-$\mathbb{R}$-ULR | | $\mathbb{R}$ cannot be approximated within any constant. By using elimination of variables, the same result holds for MIN-$\mathbb{R}$-C ULR | $=$ | $\mathbb{R}$. Furthermore, it is proven that MIN-$\mathbb{R}$-C ULR | $\leq$ | $\mathbb{R}$ is equally hard as the unconstrained version MIN-$\mathbb{R}$-ULR | | $\mathbb{R}$ and thus cannot be approximated within any constant. The problem MIN-0/1-ULR | | $\mathbb{R}$ is NPO PB-complete and cannot be approximated within $\max(n, m)^{1/2-\varepsilon}$ for any $\varepsilon > 0$. As a final result of [AK95b], MIN-0/1-C ULR | $\diamond$ | $\mathbb{R}$, for $\diamond \in \{=, \leq\}$, is NPO PB-complete and cannot be approximated within $\max(n, m)^{1-\varepsilon}$ for any $\varepsilon > 0$. All these inapproximability results hold unless P=NP.

2.3.2 Other complexity results and presented heuristics

In [ERRS09], Elbassioni, Raman, Ray and Sitters investigate the approximabilily of the maximum feasible subsystem problem when $\boldsymbol{A}$ only consists of binary entries. Except

for the upper bound $\boldsymbol{b}$, also lower bounds are imposed on all relations, giving by a vector $\boldsymbol{l} \in \mathbb{R}^m$. Furthermore, only non-negative solutions $\boldsymbol{x}$ are allowed.

As a special case they consider so-called *interval matrices*, which means that every row of $\boldsymbol{A}$ contains at most one sequence of ones. Let IM denote the set of interval matrices. For $\alpha, \beta \geq 1$, an (α, β)-*approximation* is defined as an α-approximation where the satisfied constraints may violate the upper bound by a factor of β, i.e. all constraints i that contribute to the objective function must satisfy $l_i \leq \boldsymbol{a}_i\boldsymbol{x} \leq \beta b_i$, where $\boldsymbol{a}_i$ is the i-th row of $\boldsymbol{A}$.

For $L = \max(l_1, \ldots, l_m)$ bounded by a polynomial in m and n and $\boldsymbol{A}$ being a general 0/1-matrix, we are thus considering the problem

$$\text{Max-}\mathbb{R}^+\text{-FLS} \mid\mid \mathbb{R}, \boldsymbol{A} \in \{0,1\}^{m \times n}, \boldsymbol{l} \leq \boldsymbol{A}\boldsymbol{x}, L \leq p(n,m).$$

It is proven that for this problem, there exists a $(\log(nL/\varepsilon), 1+\varepsilon)$-approximation for any $\varepsilon > 0$, but the running time of the corresponding algorithm is polynomial in $\frac{1}{\varepsilon}$. Having no restrictions on L, i.e. for the problem

$$\text{Max-}\mathbb{R}^+\text{-FLS} \mid\mid \mathbb{R}, \boldsymbol{A} \in \{0,1\}^{m \times n}, \boldsymbol{l} \leq \boldsymbol{A}\boldsymbol{x},$$

there exists a constant μ such that the problem is not approximable within $O(\log^\mu n)$, i.e. there does not exist a $(O(\log^\mu n), O(1))$-approximation.

For interval matrices, it is shown that when no violations are allowed in the upper bounds, the problem is APX-hard. That is, there does not exist a $(O(1), 1)$-approximation for

$$\text{Max-}\mathbb{R}^+\text{-FLS} \mid\mid \mathbb{R}, \boldsymbol{A} \in \text{IM}, \boldsymbol{l} \leq \boldsymbol{A}\boldsymbol{x}.$$

There does exist a $(\sqrt{\text{OPT}} \log n, 1)$-approximation. When violations in the upper bounds are allowed, a $(\log^2 n \log\log(nL/\varepsilon), 1+\varepsilon)$-approximation is given. Again, the running time is a polynomial in $\frac{1}{\varepsilon}$. If L is bounded by a polynomial in n and m, then there exists a $(1, 1+\varepsilon)$-approximation for any $\varepsilon > 0$, whose running time is quasi-polynomial.

Another way to get an equivalent formulation of the maximum feasible linear subsystem problem is by the use of *irreducible infeasible sets* of constraints or *irreducible inconsistent subsystems* (ISS). An ISS is a system of relations, which is infeasible, but has the property that every proper subsystem is feasible. Hence, by deleting any relation in an ISS, the system becomes feasible. Suppose we were given all ISS's in a system of optional constraints. Then one could find a feasible subsystem by deleting one relation from every ISS, i.e. delete a cover of all ISS's. This leads to the minimum ISS cover problem, denoted as Min ISS Cover.

In [Cha94], it is proven that Min ISS Cover is NP-hard. It is also shown that an infeasible system of relations may contain exponentially many ISS's. An exact algorithm with possible non-polynomial running time is given in [PR96], yet this algorithm is very hard to implement. Chinneck presented several heuristics to solve Min ISS Cover. The

first heuristics can be found in [Chi96]. These could be slow for some instances, but are relatively simple to implement. In [Chi01], he improved his original heuristics to obtain significantly faster ones. Some additional work has been done on finding infeasible subsystems in mixed-integer and integer linear programs ([GC99]).

Unlike all heuristics based on ISS's, Amaldi presents a heuristic based on a bilinear formulation of the maximum feasible subsystem problem in [ABC08]. This heuristic works for instances with bounded variables and instances with one single unbounded variable. An open question remains whether the heuristic also works when all variables are unbounded.

2.3.3 Previous results on our specific case

There has been a thesis on noise pollution around airports before ([Jan13]). We have incorporated the maximum feasible linear subsystem model from this text. Therefore, we here repeat the main results about this specific version of the maximum feasible linear subsystem problem:

$$\text{Max-0/1-C1 FLS} \mid \geq k \mid \mathbb{Q}^+, \boldsymbol{b}\ uni, \boldsymbol{p} = \mathbf{1}.$$

In the next chapter, we shall import some results from [Jan13] and whenever needed, we will repeat its proving methods.

From [Jan13] we know that satisfying at least one optional constraint is polynomial-time solvable, i.e. for $c = 1$,

$$\text{0/1-C1 FLS} \mid \geq k, \geq c \mid \mathbb{Q}^+, \boldsymbol{b}\ uni, \boldsymbol{p} = \mathbf{1}$$

can be solved in polynomial time. On the other hand, satisfying all optional constraints ($c = m$) is strongly NP-complete. It is actually claimed that weak NP-completeness holds for the satisfaction of any number of at least two optional constraints. We however do not agree with this statement, because the proof uses optional constraints that are not really optional (they will later on be defined as *redundant* optional constraints). Which optional constraints one allows is of course a matter of taste.

For the entries of $\boldsymbol{A}$ only taking values in $\{0, 1, 2\}$, it is proven that instances of

$$\text{Max-0/1-C1 FLS} \mid = k \mid \mathbb{Q}^+, \boldsymbol{b}\ uni, \boldsymbol{p} = \mathbf{1}$$

of a specific form are at least as hard to approximate as the Maximum Independent Set Problem. Together with this result, an $\max(m, n)$-approximation is given. Therefore we know that the more general problem

$$\text{Max-0/1-C1 FLS} \mid = k \mid \mathbb{Q}^+, \boldsymbol{b}\ uni$$

is strongly NP-complete. Furthermore, it is shown in [Jan13] that under another specific assumption about the values of the entries of $\boldsymbol{A}$, an $\frac{e}{e-1}$-approximation exists, while

under slightly different circumstances the problem is not approximable within $2^{(\log n)^{2/3}}$.

We shall look deeper into the conclusion of MIS-hardness in the next chapter. We will therefore consider other specific instances of the problem to see when this MIS-hardness still holds and in what cases we can prove a better result. We will do this by investigation the complexity for specific values of k and c.

Main problem	**Additional features**	**Complexity**	**Approximation**	**Inapproximable within**
MAX-$\mathbb{R}$-FLS $\mid\mid \mathbb{Z}$		APX-hard	2	
MAX-0/1-FLS $\mid\mid \mathbb{R}$		MIS-hard		
MAX-0/1-FLS $\mid\mid \mathbb{N}$		P		
MAX-$\mathbb{R}$-C FLS $\mid \leq \mid \mathbb{R}$		MIS-hard		
MAX-$\mathbb{R}$-C FLS $\mid = \mid \mathbb{R}$		APX-hard	2	
MAX-0/1-C FLS $\mid \diamond_b \mid \mathbb{R}$		NPO PB-complete		$m^{1-\varepsilon}\ \forall \varepsilon > 0$
MIN-$\mathbb{R}$-ULR $\mid\mid \mathbb{R}$				$O(1)$
MIN-$\mathbb{R}$-C ULR $\mid = \mid \mathbb{R}$				$O(1)$
MIN-$\mathbb{R}$-ULR $\mid\mid \mathbb{R}$				$O(1)$
MIN-0/1-C ULR $\mid \diamond \mid \mathbb{R}$		NPO PB-complete		$\max(n,m)^{1-\varepsilon}\ \forall \varepsilon > 0$
0/1-C1 FLS $\mid \geq\ k, \geq\ 1 \mid \mathbb{Q}^+$	$\boldsymbol{b}$ *uni*, $\boldsymbol{p} = \mathbf{1}$	P		
0/1-C1 FLS $\mid \geq\ k, \geq\ m \mid \mathbb{Q}^+$	$\boldsymbol{b}$ *uni*, $\boldsymbol{p} = \mathbf{1}$	NP-complete		
MAX-0/1-C1 FLS $\mid = k \mid \mathbb{Q}^+$	$\boldsymbol{b}$ *uni*, $\boldsymbol{p} = \mathbf{1}$	MIS-hard	$\max(m,n)$	
MAX-0/1-C1 FLS $\mid = k \mid \mathbb{Q}^+$	$\boldsymbol{b}$ *uni*	MIS-hard		
MAX-0/1-C1 FLS $\mid = k \mid \mathbb{Q}^+$	$\boldsymbol{b}$ *uni*, $\boldsymbol{p} = \mathbf{1}$, $\boldsymbol{A}$ specific		$\frac{e}{e-1}$	
MAX-0/1-C1 FLS $\mid = k \mid \mathbb{Q}^+$	$\boldsymbol{b}$ *uni*, $\boldsymbol{p} = \mathbf{1}$			$2^{(\log n)^{2/3}}$
MAX-$\mathbb{R}^+$-FLS $\mid\mid \mathbb{R}$	$\boldsymbol{A} \in \{0,1\}^{m \times n}$, $\boldsymbol{l} \leq \boldsymbol{Ax}, L \leq p(n,m)$		$(\log(nL/\varepsilon), 1+\varepsilon)\ \forall \varepsilon > 0$	
MAX-$\mathbb{R}^+$-FLS $\mid\mid \mathbb{R}$	$\boldsymbol{A} \in \{0,1\}^{m \times n}$, $\boldsymbol{l} \leq \boldsymbol{Ax}$			$O(\log^{\mu} n)$ for some μ
MAX-$\mathbb{R}^+$-FLS $\mid\mid \mathbb{R}$	$\boldsymbol{A} \in$ IM, $\boldsymbol{l} \leq \boldsymbol{Ax}$	APX-hard	$(\sqrt{\mathsf{OPT}} \log n, 1)$ $(\log^2 n \log\log(nL/\varepsilon), 1+\varepsilon)$	
MAX-$\mathbb{R}^+$-FLS $\mid\mid \mathbb{R}$	$\boldsymbol{A} \in$ IM, $\boldsymbol{l} \leq \boldsymbol{Ax}, L \leq p(n,m)$		$(1, 1+\varepsilon)\ \forall \varepsilon > 0$	
MIN ISS COVER		NP-hard		

Table 2.1: Previous work on maximum feasible linear subsystem problems.

Chapter 3

Complexity issues

In this chapter, we are investigating the complexity of problem (2.3) as an instance of the maximum feasible linear subsystem problem. We consider a special case of the problem, where there are no government measure points ($Q = \emptyset$), there is one threshold that applies for all housing locations ($\boldsymbol{\beta}$ is a constant vector) and all housing locations contain only one house ($\boldsymbol{\omega} = \mathbf{1}$). We shall consider the problem for binary, natural and non-negative real variables.

As a generalization of problem (2.3), we introduce non-negative integer *profits* on the binding constraint. A motivation for this is that a profit represents the passenger capacity of the aircraft belonging to a flight routine. Then the binding constraint forces us to ensure the carrying of a minimum number of passengers instead of executing a minimum number of flight routines.

Let us recall our specific version of the maximum feasible linear subsystem problem, using the notation of Section 2.2.

$$\begin{aligned} &\sum\nolimits_{j\in J} p_j x_j \diamond_b k && && \text{(binding)} \\ &\sum\nolimits_{j\in J} a_{ij} x_j \leq b && \forall i \in I && \text{(optional)} \\ &x_j \in X && \forall j \in J \end{aligned} \tag{3.1}$$

where $|I| = m$, $|J| = n$ and we must have $a_{ij}, b_i \in Y$. According to the introduced notation in Section 2.2, this problem is called

$$\textsc{Max}\text{-}X\text{-C1 FLS} \mid \diamond_b k \mid Y, \boldsymbol{b}\ \textit{uni}.$$

Considering the practical origin of this problem, we let $\diamond_b$ be one of $\{=, \geq\}$.

For obtaining complexity results, we consider two decision variants of this problem. As $\textsc{Max}$-X-C1 FLS $\mid \diamond_b k \mid Y, \boldsymbol{b}\ \textit{uni}$ is a maximization problem, we consider the question whether we can satisfy *at least c* optional constraints. Another variant is to ask whether

we can satisfy *exactly c* optional constraints. The latter version is useful to obtain (in)approximability results for the optimization problem.

Earlier we argued that by multiplying the optional constraints by a sufficiently large natural number, the values a_{ij} and b can assumed to be natural numbers. Therefore, we shall focus on the case $Y = \mathbb{N}$. The set X will always be one of $\{0,1\}$, $\mathbb{N}$ or $\mathbb{R}^+$. Of course when $X = \{0,1\}$, the problem is only feasible when $0 \leq k \leq n$. We will also focus on instances with $\boldsymbol{p} = \mathbf{1}$, as this property describes the original problem of scheduling (at least) k flight routines.

Summarizing, we shall mainly investigate the decision problems

$$X\text{-C1 FLS} \mid \diamond_b k, \diamond_d c \mid \mathbb{N}, \boldsymbol{b}\ uni, \boldsymbol{p} = \mathbf{1}$$

for $\diamond_b, \diamond_d \in \{=, \geq\}$. If X is not specified, the statement holds for all $X \in \{\{0,1\}, \mathbb{N}, \mathbb{R}^+\}$. Sometimes we will deviate from these problems, for example by dropping the property $\boldsymbol{p} = \mathbf{1}$.

For each feasible solution $\boldsymbol{x} \in X^n$ of (3.1), we define the set

$$S(\boldsymbol{x}) = \left\{ i \in I \mid \sum\nolimits_{j \in J} a_{ij} x_j \leq b \right\}$$

to be the set of indices corresponding to optional constraints that are satisfied.

In [Jan13] it is proven that the problem 0/1-C1 FLS $\mid \geq k, \geq c \mid \mathbb{N}, \boldsymbol{b}\ uni, \boldsymbol{p} = \mathbf{1}$ is strongly NP-complete when $c = m$. In this chapter, we will see that this statement also holds for other variations of $\diamond_b$ and $\diamond_d$. Therefore we also take a look at specific instances by specifying k and c in order to see which instances might be solved efficiently and which instances are still hard to solve.

This chapter is structured as follows: first we discuss the requirements of a constraint in order for it to be optional. Then we draw some connections and equivalences between the four main decision problems that we are focussing on ($\diamond_b, \diamond_d \in \{=, \geq\}$). Hereafter we investigate the complexity of these problems, distinguishing polynomial-time and NP-complete cases. We conclude the chapter by looking into the consequences of our results for the corresponding optimization problem Max-X-C1 FLS $\mid \diamond_b k \mid \mathbb{N}, \boldsymbol{b}\ uni, \boldsymbol{p} = \mathbf{1}$.

3.1 Defining redundancy

In this section we shall impose some restrictions on the shape of the optional constraints. We require the constraints $\sum_{j \in J} a_{ij} x_j \leq b$ to be real optional constraints. That is, we do not wish to be able to decide if the constraint can be satisfied or not without considering the entire linear program. Therefore, we are given some requirements on the coefficients a_{ij} and b. These rules are based on the case of unit profits.

When we consider the problem Max-0/1-C1 FLS $\mid \diamond_b k \mid \mathbb{N}, \boldsymbol{b}\ uni, \boldsymbol{p} = \mathbf{1}$, i.e. when having binary variables, the following observations provide reasonable rules.

- For every $i \in I$, it should be possible to satisfy the constraint $\sum_{j\in J} a_{ij}x_j \leq b$. Taking the binding constraint into account, we should have that the sum of the k smallest values a_{ij} is less then or equal to b. Mathematically, if we sort the coefficients such that $a_{i1} \leq \ldots \leq a_{in}$, we should have

$$\sum_{j=1}^{k} a_{ij} \leq b.$$

- For every $i \in I$, it should be possible to violate the constraint $\sum_{j\in J} a_{ij}x_j \leq b$. For that, it suffices that the k largest values a_{ij} sum up to something larger than b. Thus if we again have $a_{i1} \leq \ldots \leq a_{in}$, we should have

$$\sum_{j=n-k+1}^{n} a_{ij} > b.$$

  Instead of this inequality, we will also allow the more flexible rule

$$\sum_{j=1}^{n} a_{ij} > b.$$

For $x_j \in \mathbb{N}$ or $x_j \in \mathbb{R}^+$ there is no upper bound on the values of the variables. This means that it is always possible to violate a constraint. On the other hand, we will soon prove that Max-X-C1 FLS $\mid \diamond_b k \mid \mathbb{N}, \boldsymbol{b}$ *uni*, $\boldsymbol{p} = \mathbf{1}$ for $\diamond_b \in \{=, \geq\}$ and $X \in \{\{0,1\}, \mathbb{N}, \mathbb{R}^+\}$ has an optimal solution satisfying $\sum_{j\in J} x_j = k$. Therefore it is still desirable to impose a redundancy rule.

We also have to make sure that we would be able to satisfy each constraint. Notice that we need to satisfy the binding constraint, which is $\sum_{j\in J} x_j \geq k$. Considering an optional constraint $\sum_{j\in J} a_{ij}x_j \leq b$ for some $i \in I$, where the variables are sorted such that $a_{i1} \leq \ldots \leq a_{in}$, we obtain the smallest possible value of $\sum_{j\in J} a_{ij}x_j$ by setting x_{j_1} equal to k. The largest needed value of $\sum_{j\in J} a_{ij}x_j$ is given by $k \cdot a_{in}$. Therefore, the non-redundancy rules for Max-X-C1 FLS $\mid \diamond_b k \mid \mathbb{N}, \boldsymbol{b}$ *uni*, $\boldsymbol{p} = \mathbf{1}$ with $X \in \{\mathbb{N}, \mathbb{R}^+\}$ are:

- for every $i \in I$, when $a_{i1} \leq \ldots \leq a_{in}$, we must have $k \cdot a_{i1} \leq b$;
- for every $i \in I$, when $a_{i1} \leq \ldots \leq a_{in}$, we must have $k \cdot a_{in} > b$.

From now on, we will always assume that the coefficients a_{ij} and b satisfy these relations. If an optional constraint does not satisfy both of the relations for a certain variable range, it will be called *redundant*.

3.2 Connections and equivalences

In this section we will draw connections and equivalences between the decision problems in consideration. Using these, we do not have to derive complexity results for every version seperately, but properties might directly pass through a connection.

Let us first state some trivial properties. These are based on the following observations:

- if a solution satisfies the binding constraint $\sum_{j\in J} p_j x_j \diamond_b k$ with equality, then the binding constraint is also satisfied for $\diamond_b$ being $\geq$,
- if a solution satisfies exactly c of the optional constraints, then this solution also satisfies at least c optional constraints.

Translating these instance-dependent properties to the decision problems in general, we obtain the following.

Fact 3.1. Let $\diamond$ be one of $\{=, \geq\}$. Then for any instance U of X-C1 FLS $\mid \diamond_b k, \diamond_d c \mid \mathbb{N}, \boldsymbol{b}$ *uni*, $\boldsymbol{p} = \mathbf{1}$:

(i) if U is a yes-instance of X-C1 FLS $\mid = k, \diamond c \mid \mathbb{N}, \boldsymbol{b}$ *uni*, then U understood as an instance of X-C1 FLS $\mid \geq k, \diamond c \mid \mathbb{N}, \boldsymbol{b}$ *uni* is also a yes-instance;

(ii) if U is a yes-instance of X-C1 FLS $\mid \diamond k, = c \mid \mathbb{N}, \boldsymbol{b}$ *uni*, then U understood as an instance of X-C1 FLS $\mid \diamond k, \geq c \mid \mathbb{N}, \boldsymbol{b}$ *uni* is also a yes-instance;

(iii) if U is a yes-instance of X-C1 FLS $\mid = k, = c \mid \mathbb{N}, \boldsymbol{b}$ *uni*, then U understood as an instance of X-C1 FLS $\mid \geq k, \geq c \mid \mathbb{N}, \boldsymbol{b}$ *uni* is also a yes-instance.

Let us have a closer look at the problems X-C1 FLS $\mid \diamond_b k, \geq c \mid \mathbb{N}, \boldsymbol{b}$ *uni*, $\boldsymbol{p} = \mathbf{1}$. Consider a linear program U as an instance of both X-C1 FLS $\mid = k, \geq c \mid \mathbb{N}, \boldsymbol{b}$ *uni*, $\boldsymbol{p} = \mathbf{1}$ and X-C1 FLS $\mid \geq k, \geq c \mid \mathbb{N}, \boldsymbol{b}$ *uni*, $\boldsymbol{p} = \mathbf{1}$. Clearly, if U is a yes-instance of X-C1 FLS $\mid = k, \geq c \mid \mathbb{N}, \boldsymbol{b}$ *uni*, $\boldsymbol{p} = \mathbf{1}$, then U is also a yes-instance of X-C1 FLS $\mid \geq k, \geq c \mid \mathbb{N}, \boldsymbol{b}$ *uni*, $\boldsymbol{p} = \mathbf{1}$. This is one of the trivial observations made above. The following lemmas prove that this statement also holds the other way around.

Lemma 3.2. If $\boldsymbol{x} \in \{0,1\}^n$ is a solution to U as an instance of 0/1-C1 FLS $\mid \geq k, \geq c \mid \mathbb{N}, \boldsymbol{b}$ *uni*, $\boldsymbol{p} = \mathbf{1}$, then we can find a solution $\boldsymbol{y} \in \{0,1\}^n$ to U as an instance of 0/1-C1 FLS $\mid = k, \geq c \mid \mathbb{N}, \boldsymbol{b}$ *uni*, $\boldsymbol{p} = \mathbf{1}$.

Proof. Let $\boldsymbol{x} \in \{0,1\}^n$ be a solution to U as an instance of 0/1-C1 FLS $\mid \geq k, \geq c \mid \mathbb{N}, \boldsymbol{b}$ *uni*, $\boldsymbol{p} = \mathbf{1}$ and suppose that $\sum_{j\in J} x_j = k + l$ for some $l > 0$. Renumber the components of $\boldsymbol{x}$ such that $x_1 = \ldots = x_{k+l} = 1$ and $x_{k+l+1} = \ldots = x_n = 0$. Define the vector $\boldsymbol{y} \in \{0,1\}^n$ by $y_1 = \ldots = y_l = 0$ and $y_j = x_j$ for $j = l+1, \ldots, n$. Then clearly $\sum_{j\in J} y_j = k$ and for each i such that $\boldsymbol{x}$ satisfies optional constraint i, we have

$$\sum\nolimits_{j\in J} a_{ij} y_j = \sum\nolimits_{j=l+1}^{n} a_{ij} y_j = \sum\nolimits_{j=l+1}^{n} a_{ij} x_j \leq \sum\nolimits_{j\in J} a_{ij} x_j \leq b.$$

Hence $S(\boldsymbol{x}) \subseteq S(\boldsymbol{y})$ and $\boldsymbol{y}$ satisfies at least c optional constraints. Thus $\boldsymbol{y}$ is a solution to U as an instance of 0/1-C1 FLS $\mid = k, \geq c \mid \mathbb{N}, \boldsymbol{b}$ *uni*, $\boldsymbol{p} = \mathbf{1}$. □

When the variables are non-negative integers or reals, we have to take a slightly different approach.

Lemma 3.3. Let $X \in \{\mathbb{N}, \mathbb{R}^+\}$. If $\boldsymbol{x} \in X^n$ is a solution to U as an instance of X-C1 FLS $\mid \geq k, \geq c \mid \mathbb{N}, \boldsymbol{b}$ *uni*, $\boldsymbol{p} = \mathbf{1}$, then we can find a solution $\boldsymbol{y} \in X^n$ to U as an instance of X-C1 FLS $\mid = k, \geq c \mid \mathbb{N}, \boldsymbol{b}$ *uni*, $\boldsymbol{p} = \mathbf{1}$.

Proof. Let $\boldsymbol{x} \in X^n$ be a solution to U as an instance of X-C1 FLS $| \geq k, \geq c \mid \mathbb{N}, \boldsymbol{b}\ uni, \boldsymbol{p} = \mathbf{1}$ and suppose that $\sum_{j\in J} x_j = k + l$ for some $l > 0$. Pick a subset $J' \subseteq J$ such that $\sum_{j\in J'} x_j \geq l$. For $j \in J'$, define numbers $y_j \in X$ such that $y_j \leq x_j$ and $\sum_{j\in J'} y_j = \sum_{j\in J'} x_j - l$. Define the vector $\boldsymbol{y} \in X^n$ by $y_j = x_j$ for $j \in J \setminus J'$. Then we have

$$\sum_{j\in J} y_j = \sum_{j\in J'} y_j + \sum_{j\in J\setminus J'} y_j = \sum_{j\in J'} x_j - l + \sum_{j\in J\setminus J'} x_j = \sum_{j\in J} x_j - l = k$$

and for each i such that $\boldsymbol{x}$ satisfies optional constraint i, we have

$$\sum_{j\in J} a_{ij} y_j \leq \sum_{j\in J} a_{ij} x_j \leq b.$$

Hence $S(\boldsymbol{x}) \subseteq S(\boldsymbol{y})$ and $\boldsymbol{y}$ satisfies at least c optional constraints. Thus $\boldsymbol{y}$ is a solution to U as an instance of X-C1 FLS $| = k, \geq c \mid \mathbb{N}, \boldsymbol{b}\ uni, \boldsymbol{p} = \mathbf{1}$. □

From these observations we now prove our first equivalence.

Theorem 3.4. The decision problems X-C1 FLS $| \geq k, \geq c \mid \mathbb{N}, \boldsymbol{b}\ uni, \boldsymbol{p} = \mathbf{1}$ and X-C1 FLS $| = k, \geq c \mid \mathbb{N}, \boldsymbol{b}\ uni, \boldsymbol{p} = \mathbf{1}$ are polynomial-time equivalent.

Proof. We have to show that the problems are polynomial-time reducible to each other. Notice that these problems are in NP: given a yes-instance of one of these problems, a certificate is a vector $\boldsymbol{x} \in X^n$. We can check whether $\boldsymbol{x}$ satisfies the binding constraint and count the the number of satisfied optional constraints in time polynomial in n and m.

Now if take an instance U of any one of the problems, we consider U as an instance of the other. Clearly, this transformation can done in polynomial time. Thus our observations above show that yes-instances correspond. □

If we look into the proof of Lemmas 3.2 and 3.3, then we see that it is enough for the solution $\boldsymbol{x}$ to satisfy exactly c optional constraints in order to let $\boldsymbol{y}$ be a solution to the instance of X-C1 FLS $| = k, \geq c \mid \mathbb{N}, \boldsymbol{b}\ uni, \boldsymbol{p} = \mathbf{1}$. Therefore we state the following observation without proof.

Corollary 3.5. If U is a yes-instance of X-C1 FLS $| \geq k, = c \mid \mathbb{N}, \boldsymbol{b}\ uni, \boldsymbol{p} = \mathbf{1}$, then U understood as an instance of X-C1 FLS $| = k, \geq c \mid \mathbb{N}, \boldsymbol{b}\ uni, \boldsymbol{p} = \mathbf{1}$ is also a yes-instance. □

Taking all connections together, we see the following 'chain of yes-instances'. For any instance U of X-C1 FLS $| \diamond_b k, \diamond_d c \mid \mathbb{N}, \boldsymbol{b}\ uni, \boldsymbol{p} = \mathbf{1}$, we have:

if	U is a yes-instance of X-C1 FLS $| = k, = c \mid \mathbb{N}, \boldsymbol{b}\ uni, \boldsymbol{p} = \mathbf{1}$,
then	U is a yes-instance of X-C1 FLS $| \geq k, = c \mid \mathbb{N}, \boldsymbol{b}\ uni, \boldsymbol{p} = \mathbf{1}$,
then	U is a yes-instance of X-C1 FLS $| = k, \geq c \mid \mathbb{N}, \boldsymbol{b}\ uni, \boldsymbol{p} = \mathbf{1}$,
iff	U is a yes-instance of X-C1 FLS $| \geq k, \geq c \mid \mathbb{N}, \boldsymbol{b}\ uni, \boldsymbol{p} = \mathbf{1}$.

The following example shows that the 'then' in the second line could not be replaced by 'iff'. Therefore, in general, the problems X-C1 FLS $\mid = k, = c \mid \mathbb{N}, \boldsymbol{b}$ *uni*, $\boldsymbol{p} = \mathbf{1}$ and X-C1 FLS $\mid \geq k, = c \mid \mathbb{N}, \boldsymbol{b}$ *uni*, $\boldsymbol{p} = \mathbf{1}$ are not equivalent.

Example 3.6. Consider the following set of non-redundant optional constraints.

$$\begin{aligned} x_1 + x_2 &\leq 1 \\ x_1 + x_3 &\leq 1 \\ x_1, x_2, x_3 &\in \{0, 1\} \end{aligned}$$

Suppose now we want to satisfy exactly one of the two constraints. If $x_1 = 0$, then both constraints are satisfied, so x_1 has to be set to 1. To violate exactly one of the constraints, one easily sees that we must satisfy $x_2 + x_3 = 1$. Hence we can satisfy exactly one optional constraint only if $\sum_{j \in J} x_j = 2$. Therefore, the problem X-C1 FLS $\mid \geq 1, = 1 \mid \mathbb{N}, \boldsymbol{b}$ *uni*, $\boldsymbol{p} = \mathbf{1}$ is solvable while X-C1 FLS $\mid = 1, = 1 \mid \mathbb{N}, \boldsymbol{b}$ *uni*, $\boldsymbol{p} = \mathbf{1}$ is not.

Another example shows that the 'then' in the third line could not be replaced by 'iff'. Therefore, the problems X-C1 FLS $\mid = k, \geq c \mid \mathbb{N}, \boldsymbol{b}$ *uni*, $\boldsymbol{p} = \mathbf{1}$ and X-C1 FLS $\mid \geq k, = c \mid \mathbb{N}, \boldsymbol{b}$ *uni*, $\boldsymbol{p} = \mathbf{1}$ are not equivalent.

Example 3.7. Consider the following set of non-redundant optional constraints.

$$\begin{aligned} x_1 + x_2 &\leq 1 \\ x_1 + x_3 &\leq 1 \\ x_2 + x_3 &\leq 1 \\ x_1, x_2, x_3 &\in \{0, 1\} \end{aligned}$$

If at most one variable is equal to 1, all optional constraints are satisfied. If two variables are equal to 1, then we satisfy exactly 2 optional constraints. If all variables are equal to 1, we satisfy none of the optional constraints. Suppose we impose the binding constraint $\sum_{j \in J} x_j \diamond_b 2$. Then it is possible to satisfy at least 1 optional constraint, but it is not possible to satisfy exactly 1. Therefore, the problem X-C1 FLS $\mid = 2, \geq 1 \mid \mathbb{N}, \boldsymbol{b}$ *uni*, $\boldsymbol{p} = \mathbf{1}$ is solvable while X-C1 FLS $\mid \geq 2, = 1 \mid \mathbb{N}, \boldsymbol{b}$ *uni*, $\boldsymbol{p} = \mathbf{1}$ is not.

We already know from Theorem 3.4 that X-C1 FLS $\mid \geq k, \geq c \mid \mathbb{N}, \boldsymbol{b}$ *uni*, $\boldsymbol{p} = \mathbf{1}$ is solvable in polynomial time if and only if X-C1 FLS $\mid = k, \geq c \mid \mathbb{N}, \boldsymbol{b}$ *uni*, $\boldsymbol{p} = \mathbf{1}$ is. Considering the other two decision variants, we can lift the preservation of yes-instances to the actual solvability of the decision problems. This argument holds only for binary variables.

Proposition 3.8. If 0/1-C1 FLS $\mid = k, = c \mid \mathbb{N}, \boldsymbol{b}$ *uni*, $\boldsymbol{p} = \mathbf{1}$ is polynomial-time solvable, then also 0/1-C1 FLS $\mid \geq k, = c \mid \mathbb{N}, \boldsymbol{b}$ *uni*, $\boldsymbol{p} = \mathbf{1}$ is polynomial-time solvable.

Proof. Let U be an instance of 0/1-C1 FLS $\mid \geq k, = c \mid \mathbb{N}, \boldsymbol{b}$ *uni*, $\boldsymbol{p} = \mathbf{1}$ and let $\mathcal{A}$ be an algorithm with polynomial running time that solves 0/1-C1 FLS $\mid = k, = c \mid \mathbb{N}, \boldsymbol{b}$ *uni*, $\boldsymbol{p} = \mathbf{1}$. Observe that U is a yes-instance of 0/1-C1 FLS $\mid \geq k, = c \mid \mathbb{N}, \boldsymbol{b}$ *uni*, $\boldsymbol{p} = \mathbf{1}$ if and only if there exists $k' \in \{k, \ldots, n\}$ such that U is a yes-instance of 0/1-C1 FLS $\mid = k', = c \mid$

$\mathbb{N}, \boldsymbol{b}\ uni, \boldsymbol{p} = \mathbf{1}$. We can therefore solve the problem 0/1-C1 FLS $\mid \geq k, = c \mid \mathbb{N}, \boldsymbol{b}\ uni, \boldsymbol{p} = \mathbf{1}$ on instance U by checking if U is a yes-instance of 0/1-C1 FLS $\mid = k', = c \mid \mathbb{N}, \boldsymbol{b}\ uni, \boldsymbol{p} = \mathbf{1}$ for $k' = k, \ldots, n$. This checking is done by the algorithm $\mathcal{A}$. If there exists such a k', then U is a yes-instance of 0/1-C1 FLS $\mid \geq k, = c \mid \mathbb{N}, \boldsymbol{b}\ uni, \boldsymbol{p} = \mathbf{1}$. Otherwise, U is a no-instance.

Because we have binary variables, n is the largest value of k' that needs to be considered. The procedure described above therefore needs at most n calls to the polynomial-time algorithm $\mathcal{A}$ and thus runs in polynomial time itself. □

A similar argument holds for the problems 0/1-C1 FLS $\mid \geq k, = c \mid \mathbb{N}, \boldsymbol{b}\ uni, \boldsymbol{p} = \mathbf{1}$ and X-C1 FLS $\mid = k, \geq c \mid \mathbb{N}, \boldsymbol{b}\ uni, \boldsymbol{p} = \mathbf{1}$. We will therefore state the next proposition without a proof.

Proposition 3.9. If 0/1-C1 FLS $\mid \geq k, = c \mid \mathbb{N}, \boldsymbol{b}\ uni, \boldsymbol{p} = \mathbf{1}$ is polynomial-time solvable, then also 0/1-C1 FLS $\mid = k, \geq c \mid \mathbb{N}, \boldsymbol{b}\ uni, \boldsymbol{p} = \mathbf{1}$ is polynomial-time solvable. □

For binary variables we can therefore see the following 'chain of solvability'.

if	0/1-C1 FLS $\mid = k, = c \mid \mathbb{N}, \boldsymbol{b}\ uni, \boldsymbol{p} = \mathbf{1}$	is polynomial-time solvable,
then	0/1-C1 FLS $\mid \geq k, = c \mid \mathbb{N}, \boldsymbol{b}\ uni, \boldsymbol{p} = \mathbf{1}$	is polynomial-time solvable,
then	0/1-C1 FLS $\mid = k, \geq c \mid \mathbb{N}, \boldsymbol{b}\ uni, \boldsymbol{p} = \mathbf{1}$	is polynomial-time solvable,
iff	0/1-C1 FLS $\mid \geq k, \geq c \mid \mathbb{N}, \boldsymbol{b}\ uni, \boldsymbol{p} = \mathbf{1}$	is polynomial-time solvable.

It would be convenient to find a polynomial-time algorithm for (special cases of) the problem X-C1 FLS $\mid = k, = c \mid \mathbb{N}, \boldsymbol{b}\ uni, \boldsymbol{p} = \mathbf{1}$, because finding yes-instances of this problem provides yes-instances of the other problems. When considering binary variables, this even yields the solvability of other problems. Therefore we will in the next sections focus on deriving complexity results for this problem.

We close this section by deriving another equivalence. Here we allow the entries of the constraint matrix $\boldsymbol{A}$ and the value b to be negative integers. This will be useful for later reductions. However, allowing negative values apparently does not change the problem as a whole.

Proposition 3.10. X-C1 FLS $\mid = k, \diamond c \mid \mathbb{N}, \boldsymbol{b}\ uni, \boldsymbol{p} = \mathbf{1}$ is polynomial-time equivalent to X-C1 FLS $\mid = k, \diamond c \mid \mathbb{Z}, \boldsymbol{b}\ uni, \boldsymbol{p} = \mathbf{1}$.

Proof. Notice that by the same argument as in Theorem 3.4, the two problems are in NP. Moreover, every instance U of X-C1 FLS $\mid = k, \diamond c \mid \mathbb{N}, \boldsymbol{b}\ uni, \boldsymbol{p} = \mathbf{1}$ can be understood as an instance of X-C1 FLS $\mid = k, \diamond c \mid \mathbb{Z}, \boldsymbol{b}\ uni, \boldsymbol{p} = \mathbf{1}$ and solutions coincide. Therefore, we have

$$X\text{-C1 FLS} \mid = k, \diamond c \mid \mathbb{N}, \boldsymbol{b}\ uni, \boldsymbol{p} = \mathbf{1} \quad \leq \quad X\text{-C1 FLS} \mid = k, \diamond c \mid \mathbb{Z}, \boldsymbol{b}\ uni, \boldsymbol{p} = \mathbf{1}.$$

We shall now prove that the converse is also true.
Suppose therefore we are given an instance of X-C1 FLS $| = k, \diamond c \mid \mathbb{Z}, \boldsymbol{b}\ uni, \boldsymbol{p} = \mathbf{1}$:

$$\begin{aligned} \sum_{j\in J} x_j &= k && \text{(binding)} \\ \sum_{j\in J} a_{ij}x_j &\leq b \quad \forall i \in I && \text{(optional)} \\ x_j &\in X \quad \forall j \in J \end{aligned} \tag{3.2}$$

Let then $l^* \in \mathbb{N}$ be minimal such that $a_{ij} + l^* \geq 0$ for all $i \in I$, $j \in J$ and such that $b + kl^* \geq 0$.

Using the binding constraint, optional constraint $\sum_{j\in J} a_{ij}x_j \leq b$ is equivalent to

$$\sum_{j\in J} a_{ij}x_j + k \cdot l^* \leq b + kl^*,$$

which is

$$\sum_{j\in J} a_{ij}x_j + \sum_{j\in J} l^* x_j \leq b + kl^*,$$

which is

$$\sum_{j\in J} (a_{ij} + l^*)x_j \leq b + kl^*.$$

Since the coefficients in the latter inequality are non-negative, we see that the above instance (3.2) of X-C1 FLS $| = k, \diamond c \mid \mathbb{Z}, \boldsymbol{b}\ uni, \boldsymbol{p} = \mathbf{1}$ is equivalent to the following instance of X-C1 FLS $| = k, \diamond c \mid \mathbb{N}, \boldsymbol{b}\ uni, \boldsymbol{p} = \mathbf{1}$.

$$\begin{aligned} \sum_{j\in J} x_j &= k && \text{(binding)} \\ \sum_{j\in J} (a_{ij} + l^*)x_j &\leq b + kl^* \quad \forall i \in I && \text{(optional)} \\ x_j &\in X \quad \forall j \in J \end{aligned} \tag{3.3}$$

By these instances being equivalent we mean that $\boldsymbol{x} \in X^n$ is a solution to (3.2) of X-C1 FLS $| = k, \diamond c \mid \mathbb{Z}, \boldsymbol{b}\ uni, \boldsymbol{p} = \mathbf{1}$ if and only if it is a solution to (3.3) of X-C1 FLS $| = k, \diamond c \mid \mathbb{N}, \boldsymbol{b}\ uni, \boldsymbol{p} = \mathbf{1}$. Thus yes-instances correspond and we can conclude

$$X\text{-C1 FLS} \mid = k, \diamond c \mid \mathbb{Z}, \boldsymbol{b}\ uni, \boldsymbol{p} = \mathbf{1} \quad \leq \quad X\text{-C1 FLS} \mid = k, \diamond c \mid \mathbb{N}, \boldsymbol{b}\ uni, \boldsymbol{p} = \mathbf{1}.$$

□

3.3 Polynomial-time cases

In this section, we will establish some cases which can be solved efficiently. Hereby we rely on certain values to be constant.

In [Jan13], the problem variant where one wishes to satisfy *at least c* optional constraints was investigated. If we do not allow redundant optional constraints, the thesis states results for the cases $c = 1$ and $c = m$. Only the first case can be solved efficiently.

Proposition 3.11 ([Jan13])**.** For $c = 1$, problem 0/1-C1 FLS $\mid \diamond k, \geq c \mid \mathbb{N}, \boldsymbol{b}$ *uni*$, \boldsymbol{p} = \mathbf{1}$ is polyniomial-time solvable.

This particular problem was solved by going through each optional constraints i of the given instance, and checking if it could be satisfied by setting the variables x_j corresponding to the k smallest values a_{ij} to 1.

A similar approach can be followed for non-binary variables.

Proposition 3.12. Let $X \in \{\mathbb{N}, \mathbb{R}^+\}$. For $c = 1$, problem X-C1 FLS $\mid \diamond k, \geq c \mid \mathbb{N}, \boldsymbol{b}$ *uni*$, \boldsymbol{p} = \mathbf{1}$ is polyniomial-time solvable.

Proof. Let U be an instance of X-C1 FLS $\mid \diamond k, \geq 1 \mid \mathbb{N}, \boldsymbol{b}$ *uni*$, \boldsymbol{p} = \mathbf{1}$. Consider the optional constraints $\sum_{j \in J} a_{ij} x_j \leq b$ for all $i \in I$ one by one. Given such an i, we would like to solve the following linear program, where $\diamond \in \{=, \geq\}$.

$$\begin{aligned} \sum\nolimits_{j \in J} x_j &\diamond k \\ \sum\nolimits_{j \in J} a_{ij} x_j &\leq b \\ x_j &\in X \quad \forall j \in J \end{aligned} \tag{3.4}$$

Reorder the variables x_j such that $a_{i1} \leq \ldots \leq a_{in}$. Then the two linear constraints above can be satisfied (by setting $x_1 = k$) if and only if $a_{i1} \cdot k \leq b$. If there exists $i \in I$ such that the latter holds, U is a yes-instance. Otherwise, U is a no-instance.

Since we run through a total of m optional constraints and reordering can be done in polynomial time, this procedure takes polynomial time. □

Notice that in the above proof, we cannot guarantee that exactly one optional constraint is satisfied. This is because when we solve linear program (3.4) for a particular i, we might accidentally satisfy more optional constraints than just i.

Let us consider the case $c = 0$. We can of course always satisfy at least zero optional constraints, so that every instance of X-C1 FLS $\mid \geq k, \geq 0 \mid \mathbb{N}, \boldsymbol{b}$ *uni*$, \boldsymbol{p} = \mathbf{1}$ is a yes-instance. So what about satisfying exactly zero optional constraints? For this we take a look at the redundancy rules. Let us therefore denote for every $i \in I$:

- A_i is the set of the k smallest values a_{ij};
- $a_i^* = \max\{a_{ij} \mid j \in J\}$.

Considering an instance U of the binary problem 0/1-C1 FLS $\mid \geq k, = 0 \mid \mathbb{N}, \boldsymbol{b}$ *uni*$, \boldsymbol{p} = \mathbf{1}$, then we construct $\boldsymbol{x} \in \{0,1\}^n$ as follows.

$$x_j = \begin{cases} 1 & \text{if there exists } i \in I \text{ such that } a_{ij} \in A_i \\ 0 & \text{otherwise.} \end{cases}$$

By the redundancy rules for binary variables, this vector $\boldsymbol{x}$ is a feasible solution of U violating all optional constraints: for all $i \in I$, we have

$$\sum\nolimits_{j\in J} a_{ij}x_j \geq \sum\nolimits_{j\in A_i} a_{ij} > b.$$

Considering non-binary variables, that is, U is an instance of X-C1 FLS $\mid \geq k, = 0 \mid \mathbb{N}, \boldsymbol{b}\ uni, \boldsymbol{p} = \mathbf{1}$ for $X \in \{\mathbb{N}, \mathbb{R}^+\}$, we construct $\boldsymbol{x} \in X$ as follows.

$$x_j = \begin{cases} k & \text{if there exists } i \in I \text{ such that } a_{ij} = a_i^* \\ 0 & \text{otherwise.} \end{cases}$$

Then by the redundancy rules for non-binary variables, the vector $\boldsymbol{x}$ is a feasible solution of U violating all optional constraints: for all $i \in I$, we have

$$\sum\nolimits_{j\in J} a_{ij}x_j \geq k \cdot a_i^* > b.$$

Since these vectors can be constructed in polynomial time, we have proven the following result.

Proposition 3.13. The problems X-C1 FLS $\mid \geq k, \diamond 0 \mid \mathbb{N}, \boldsymbol{b}\ uni, \boldsymbol{p} = \mathbf{1}$ are polynomial-time solvable. Furthermore:

- every instance of X-C1 FLS $\mid \geq k, \geq 0 \mid \mathbb{N}, \boldsymbol{b}\ uni, \boldsymbol{p} = \mathbf{1}$ is a yes-instance;
- every instance of X-C1 FLS $\mid \geq k, = 0 \mid \mathbb{N}, \boldsymbol{b}\ uni, \boldsymbol{p} = \mathbf{1}$ is a no-instance. □

The above results are true for any value of k. We shall look into cases where k is a either constant or n minus a constant. The results are then independent of the value of c.

Proposition 3.14. For any $k \in O(1)$, the problem 0/1-C1 FLS $\mid = k, \diamond c \mid \mathbb{N}, \boldsymbol{b}\ uni, \boldsymbol{p} = \mathbf{1}$ is polynomial-time solvable.

Proof. Let U be an instance of 0/1-C1 FLS $\mid = k, \diamond c \mid \mathbb{N}, \boldsymbol{b}\ uni, \boldsymbol{p} = \mathbf{1}$. Given a vector $\boldsymbol{x} \in \{0,1\}^n$ such that $\sum_{j\in J} x_j = k$, we can check all optional constraints and count how many of them are satisfied in polynomial time. If this number is (at least) c, then U is a yes-instance. Otherwise, U is a no-instance. We thus have to ensure that there are only polynomially many possible solutions to check.

Since we are dealing with binary variables, we know that exactly k variables have to be set to 1. Thus we have $\binom{n}{k}$ possible solutions $\boldsymbol{x}$. For $k = 0$, this is only 1. For $k \geq 1$, we use that for all natural numbers p, r such that $1 \leq r \leq p$, it holds that $\binom{p}{r} \leq p^r$, as follows simply from the formula

$$\binom{p}{r} = \frac{p \cdot (p-1) \cdot \ldots \cdot (p-r+1)}{r!} \leq \frac{p^r}{r!} \leq p^r.$$

Therefore, for $k \geq 1$, there are $\binom{n}{k} \leq n^k$ possible solutions to check, which, as k is assumed to be a constant, are polynomially many. □

The same trick can be done if we require $n-k$ variables to be set to 1, for any constant k. Therefore we either use the equality $\binom{n}{k} = \binom{n}{n-k}$ or just notice that setting $n-k$ variables to 1 is the same as setting k variables to 0.

Corollary 3.15. For any $k \in O(1)$, the problem 0/1-C1 FLS $\mid = n-k, \diamond c \mid \mathbb{N}, \boldsymbol{b}\ uni, \boldsymbol{p} = \mathbf{1}$ is polynomial-time solvable. □

Let us consider the problem 0/1-C1 FLS $\mid = k, \diamond c \mid \mathbb{N}, \boldsymbol{b}\ uni, \boldsymbol{p} = \mathbf{1}$ for $X \in \{\mathbb{N}, \mathbb{R}^+\}$ and $k \in O(1)$ and look into the proof of Proposition 3.14. The advantage of having binary variables there was that we knew exactly what our solution should look like: for some value k, exactly k variables had to be set to 1. Then in the proof of Proposition 3.14 we could run through all $\binom{n}{k}$ possibilities of vectors $\boldsymbol{x} \in \{0,1\}^n$ with $\sum_{j \in J} x_j = k$ in polynomial time.

Consider the case $X = \mathbb{N}$. For a certain k, we can obtain $\sum_{j \in J} x_j = k$ in many ways. For example, when $k = 5$, we can write $1+1+1+1+1 = 2+1+1+1 = 2+2+1 = 3+1+1 = 3+2 = 4+1 = 5$, so even without taking the order into account, there are seven possibilities to have $\sum_{j \in J} x_j = 5$. Writing k as an unordered sum of positive integers is called a *partition* of k. Taking the order into account (as we wish to) leads to a *composition* of the number k. Of course, a natural number has more compositions than partitions, as for 5, the number of compositions is already sixteen.

It is known that for $k \geq 1$, the number of compositions of k is equal to 2^{k-1}. Defining the *length* of a composition $\alpha_1 + \cdots + \alpha_l$ as l, it is also easily seen that the number of compositions of k of length l is $\binom{k-1}{l-1}$. Therefore we can prove the following.

Proposition 3.16. For any $k \in O(1)$, the problem $\mathbb{N}$-C1 FLS $\mid = k, \diamond c \mid \mathbb{N}, \boldsymbol{b}\ uni, \boldsymbol{p} = \mathbf{1}$ is polynomial-time solvable.

Proof. Let U be an instance of $\mathbb{N}$-C1 FLS $\mid = k, \diamond c \mid \mathbb{N}, \boldsymbol{b}\ uni, \boldsymbol{p} = \mathbf{1}$. Given a vector $\boldsymbol{x} \in \mathbb{N}^n$ such that $\sum_{j \in J} x_j = k$, we can check all optional constraints and count how many of them are satisfied in polynomial time. If this number is (at least) c, then U is a yes-instance. Otherwise, U is a no-instance. We thus have to ensure that there are only polynomially many possible solutions to check.

For $k = 0$, there is only one possible solution $\boldsymbol{x} = \mathbf{0}$. For $k = 1$, one variable must be set to 1, so there are n possible solutions to check.

For $k \geq 2$, we count the number of possible solutions as follows. For $l = 1, \ldots, k$, pick l variables and try all compositions of k of length l on these variables. The number of possible solutions is then

$$\sum_{l=1}^{k} \binom{n}{l}\binom{k-1}{l-1}.$$

Using $k \leq n$ and $\binom{p}{r} \leq p^r$ for $1 \leq r \leq p$, we have that the number of possible solutions is

$$\begin{aligned}
&\sum_{l=1}^{k} \binom{n}{l} \cdot \binom{k-1}{l-1} \\
&= n + \sum_{l=2}^{k} \binom{n}{l} \cdot \binom{k-1}{l-1} \\
&\leq n + \sum_{l=2}^{k} n^l (k-1)^{l-1} \\
&\leq n + \sum_{l=2}^{k} n^k (k-1)^{k-1} \\
&= n + (k-1) n^k (k-1)^{k-1} \\
&= n + n^k (k-1)^k \\
&\leq n + n^{2k}
\end{aligned}$$

which, since k is assumed to be a constant, is polynomial in n. Therefore, we can go through all possible solutions in polynomial time. □

For $X = \mathbb{R}^+$, we cannot follow this approach, since there are uncountably many ways to write $\sum_{j \in J} x_j = k$ when $x_j \in \mathbb{R}^+$.

Up until now, we were not able to find any more cases that can be solved efficiently. It would be interesting to see whether there exist polynomial-time cases where both k and c are not required to be in $O(1)$.

3.4 Establishing NP-completeness

In this section, we will construct several polynomial-time reductions from NP-complete problems to X-C1 FLS $\mid = k, = c \mid \mathbb{N}, \boldsymbol{b}$ *uni*. If these NP-complete problems are maximization problems originally, we consider its $=$-decision variant. We shall look into both the weighted variant, where $\boldsymbol{p}$ is a vector of non-negative integers, and the unweighted variant, where $\boldsymbol{p} = \mathbf{1}$.

This section follows a structure of building up results. This means that some results might be redundant, because stronger statements are proven later. Despite this property, we have chosen to present all results of our theoretical investigation.

First, we mention just one time that all problems X-C1 FLS $\mid \diamond_b k, \diamond_d c \mid \mathbb{N}, \boldsymbol{b}$ *uni* are in NP: given any certificate $\boldsymbol{x} \in X^n$ for a yes-instance of X-C1 FLS $\mid \diamond_b k, \diamond_d c \mid \mathbb{N}, \boldsymbol{b}$ *uni*, we can check the binding constraint and count whether (at least) c optional constraints are satisfied in time polynomial in m and n. When proving NP-completeness, we therefore only have to show that the decision problem in consideration is NP-hard. This is then done by constructing a polynomial-time reduction from a decision problem that is NP-complete.

3.4.1 Weighted variant

We start by investigating the weighted variant 0/1-C1 FLS $\mid = k, = c \mid \mathbb{N}, \boldsymbol{b}$ *uni*. Recall that in this version every variable x_j is assigned a *profit* p_j. Within the practical origin of the problem, one could think of p_j as the capacity of the aircraft belonging to flight routine x_j and of k as the minimum number of passengers that must be carried by all flight routines.

The first result is obtained by constructing a polynomial-time reduction from the *Equipartition problem*. This problem is defined as follows.

Equipartition problem (EP)

Instance:	Set $A \subset \mathbb{N}^{\geq 1}$ of even cardinality, such that $\sum_{a \in A} a = 2B$.
Question:	Does there exist a subset $A' \subseteq A$ such that $\lvert A' \rvert = \frac{1}{2}\lvert A \rvert$ and $\sum_{a \in A'} a = \sum_{a \notin A'} a = B$?

EP is known to be weakly NP-complete. A proof of this can be found in Appendix B.

Theorem 3.17. 0/1-C1 FLS $\mid = k, = c \mid \mathbb{N}, \boldsymbol{b}$ *uni* is weakly NP-complete for instances with m an even number and $c = \frac{m}{2}$. This holds even for $b = 0$ and $a_{ij} \in \{0, 1\}$.

Proof. To prove weak NP-hardness, we consider EP. An instance of EP consists of a set of $2r$ positive natural numbers $s_1, \ldots, s_{2r}$ such that $\sum_{j=1}^{2r} s_j = 2B$. From this, we create an instance for 0/1-C1 FLS $\mid = k, = \frac{m}{2} \mid \mathbb{N}, \boldsymbol{b}$ *uni* defined by $I = J = \underline{2r}$, $p_j = s_j$, $b = 0$, $k = B$ and

$$a_{ij} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{otherwise.} \end{cases}$$

This means that we are given the following binary linear program.

$$\begin{aligned} \sum\nolimits_{j \in J} s_j x_j &= B && \text{(binding)} \\ x_j &\leq 0 \qquad \forall j \in J && \text{(optional)} \\ x_j &\in \{0, 1\} \quad \forall j \in J \end{aligned} \tag{3.5}$$

We shall now prove that yes-instances correspond, that is: we will prove that we have a yes-instance of EP if and only if we can find a feasible solution to (3.5) satisfying half of the optional constraints.

Suppose we have a yes-instance for EP. Then we have a subset $S \subseteq \underline{2r}$ such that $\lvert S \rvert = r$ and $\sum_{j \in S} s_j = B$. Define $\boldsymbol{x} \in \{0, 1\}^n$ by $x_j = 1$ if and only if $j \in S$. Then we have $\sum_{j \in J} w_j x_j = \sum_{j \in S} s_j = B$. The optional constraint $x_j \leq 0$ is satisfied if and only if $j \notin S$. Since $\lvert S \rvert = r$, there are $r = \frac{1}{2}\lvert I \rvert = \frac{m}{2}$ optional constraints satisfied. Thus $\boldsymbol{x}$ is a solution to 0/1-C1 FLS $\mid = k, = \frac{m}{2} \mid \mathbb{N}, \boldsymbol{b}$ *uni*.

If on the other hand we have a solution $\boldsymbol{x}$ to (3.5) satisfying $\frac{m}{2}$ optional constraints, then we define a set $S \subseteq \underline{2r}$ by $j \in S$ if and only if $x_j = 1$. Since half of the optional constraints are satisfied, we have $|S| = \frac{m}{2} = r$ and furthermore, $\sum_{j\in S} s_j = \sum_{j\in S} s_j x_j = \sum_{j\in J} w_j x_j = B$. Hence S solves EP. □

The above theorem gives us a hardness result for 0/1-C1 FLS $| = k, = c \mid \mathbb{N}, \boldsymbol{b}$ *uni* only for one value of c. To obtain complexity results for more values of c, we consider another optimization problem.

Maximum constraint partition problem (MCP)

Instance:	Finite set A, subset $S \subseteq A$, sizes $s(a) \in \mathbb{N}^{\geq 1}$ $(\forall a \in A)$ such that $\sum_{a\in A} s(a) = 2B$, special element $a_0 \in A$.
Goal:	Find a subset $A' \subseteq A$ such that $a_0 \in A'$, $\sum_{a\in A'} s(a) = \sum_{a\notin A'} s(a) = B$ and such that $\lvert S \cap A'\rvert$ is maximized.

We turn MCP into a decision problem by fixing a natural number K and replacing the goal in the above definition by the question: does there exist a subset $A' \subseteq A$ such that $a_0 \in A'$, $\sum_{a\in A'} s(a) = \sum_{a\notin A'} s(a) = B$ and such that $|S \cap A'| = K$?

This decision variant of MCP is also weakly NP-complete. We prove this in Appendix B.

Theorem 3.18. 0/1-C1 FLS $| = k, = c \mid \mathbb{N}, \boldsymbol{b}$ *uni* is weakly NP-complete for instances with $c > 0$. This holds even for $b = 0$ and $a_{ij} \in \{0, 1, 2\}$.

Proof. We construct a polynomial-time reduction from the problem Dec-MCP to the problem 0/1-C1 FLS $| = k, = c \mid \mathbb{N}, \boldsymbol{b}$ *uni*. Suppose we are given an instance for MCP, that is a finite set A, a subset $S \subseteq A$, sizes $s(a) \in \mathbb{N}^{\geq 1}$ with $\sum_{a\in A} s(a) = 2B$ and an element $a_0 \in A$. Then we construct an instance of the problem 0/1-C1 FLS $| = k, = c \mid \mathbb{N}, \boldsymbol{b}$ *uni* in the following way: define $J = A$ and $I = S$. Furthermore, we define $k = B$ and $p_j = s(j)$ for all $j \in J$. We check whether $a_0 \in S$.

If $a_0 \in S$, then we define for each $i \in I$ and $j \in J$

$$a_{ij} = \begin{cases} 1 & \text{if } i \neq a_0 \text{ and } (i = j \text{ or } j = a_0) \\ 2 & \text{if } i = a_0 \\ 0 & \text{otherwise.} \end{cases}$$

If $a_0 \notin S$, then we define for each $i \in I$ and $j \in J$

$$a_{ij} = \begin{cases} 1 & \text{if } i = j \text{ or } j = a_0 \\ 0 & \text{otherwise.} \end{cases}$$

In both cases, we are given the following binary linear program.

$$\begin{aligned} \sum_{a\in A} s(a)x_a &= B && \text{(binding)} \\ x_{a_0} + x_a &\leq 0 \qquad \forall a \in S && \text{(optional)} \\ x_a &\in \{0,1\} \quad \forall a \in A \end{aligned} \tag{3.6}$$

For an arbitrary $c \in \{1,\dots,m\}$, we shall now prove that there exists a solution $A' \subseteq A$ to MCP satisfying $|S \cap A'| = c$ if and only if problem (3.6) is solved by satisfying c optional constraints, that is, if and only if the above instance is a yes-instance of 0/1-C1 FLS $| =k, =c \mid \mathbb{N}, \boldsymbol{b}$ *uni*.

Suppose we have a solution A' for MCP such that $|S \cap A'| = c$. Assign the variables x_a the following values:

$$x_a = \begin{cases} 0 & \text{if } a \in A' \\ 1 & \text{otherwise.} \end{cases}$$

Then we have $\sum_{a\in A} s(a)x_a = \sum_{a\notin A'} s(a) = B$, thus the binding constraint is satisfied. Notice that $a_0 \in A'$ so that $x_{a_0} = 0$. Moreover, for $a \in S$, the optional constraint $x_{a_0} + x_a \leq 0$ is satisfied if and only if $x_a = 0$, i.e. $a \in A'$. Hence there are $|S \cap A'| = c$ satisfied optional constraints.

Now suppose that we have a solution $\boldsymbol{x}$ to (3.6), which satisfies c of the optional constraints. Notice that since $c > 0$, we must have $x_{a_0} = 0$. We define a set $A' \subseteq A$ by $a \in A'$ if and only if $x_a = 0$. Then $a_0 \in A'$.

Using that the binding constraint is satisfied, we have

$$\sum_{a\notin A'} s(a) = \sum_{a\notin A'} s(a)x_a = \sum_{a\in A} s(a)x_a = B.$$

So A' is a solution to MCP. We still need to check its objective value: we have $a \in S$ if and only if there is a optional constraint $x_{a_0} + x_a \leq 0$, and $a \in A'$ if and only if $x_a = 0$. Hence, $a \in S \cap A'$ if and only if the optional constraint $x_{a_0} + x_a \leq 0$ is satisfied. Therefore, $|S \cap A'| = c$. □

3.4.2 Unweighted variant

In the previous section we have obtained weak NP-completeness for several cases of the weighted variant 0/1-C1 FLS $| =k, =c \mid \mathbb{N}, \boldsymbol{b}$ *uni*, where we especially considered the case of having binary variables. We continue by deriving complexity results for the more specific unweighted variant *X*-C1 FLS $| =k, =c \mid \mathbb{N}, \boldsymbol{b}$ *uni*, $\boldsymbol{p} = \mathbf{1}$. When considering binary variables in this problem, NP-completeness results also apply to to the weighted variant, since one could always create an polynomial-time reduction from a special case of a problem to the problem itself.

Besides having binary variables, we shall also investigate the problem when having non-negative natural variables (modeling the original problem) or even real variables (modeling the LP-relaxation). As explained in Chapter 1, the reason for considering an LP-relaxation is to find solutions to the integer program in an efficient way.

Maximum independent set problem

The problem that we will be using for constructing polynomial-time reductions is the Maximum independent set problem. We repeat here the definition given in Chapter 1. Given a graph $G = (V, E)$, an *independent set* in G is a subset $V' \subseteq V$ such that for each $u, v \in V'$, $\{u, v\} \notin E$. MIS is defined as follows.

Maximum independent set problem (MIS)

Instance: Graph $G = (V, E)$.
Goal: Find an independent set V' of G such that $|V'|$ is maximized.

In the problem MIS, we shall always assume that instances do not contain isolated nodes. This is because such a node could always be added to an independent set and we can thus reduce the instance to a smaller problem.

The decision variant of MIS is known to be strongly NP-complete and the optimization problem brings a very bad inapproximability result.

We can formulate MIS as a linear program with optional constraints. Let therefore $G = (V, E)$ be an instance of MIS. For any $v \in V$, let $N(v)$ denote the set of nodes adjacent to v, i.e.

$$N(v) = \{u \in V \mid \{u, v\} \in E\}.$$

Consider the following set of optional constraints.

$$\begin{aligned} -x_v + \sum\nolimits_{u \in N(v)} x_u \leq -1 \quad & \forall v \in V \quad \text{(optional)} \\ x_v \in X \quad & \forall v \in V \end{aligned} \tag{3.7}$$

The assumption of G having no isolated nodes ensures that the optional constraints in (3.7) are non-redundant.

When considering binary variables ($X = \{0, 1\}$), then optional constraint v is satisfied if and only if $x_v = 1$ and $x_u = 0$ for all $u \in N(v)$. For $\{u, v\} \in E$ we cannot have that both constraints u and v are satisfied, because this would imply $x_v = 1$ (since constraint v is satisfied) and $x_v = 0$ (since constraint u is satisfied). Therefore, given a solution $\boldsymbol{x} \in \{0, 1\}^n$ to (3.7), the set

$$\{v \in V \mid \text{constraint } v \text{ is satisfied by } \boldsymbol{x}\}$$

is an independent set in G.

When the variables are non-binary, the above argument does not hold anymore. Still we can prove that two optional constraints u and v cannot be satisfied simultaneously if u and v constitute an edge. Therefore we state the following important lemma. The proof only uses the non-negativity of the variables, instead of exploiting the exact range of the variables.

Lemma 3.19. Let $G = (V, E)$ be a graph. Consider the following set of constraints.

$$\begin{aligned} -x_v + \sum\nolimits_{u \in N(v)} x_u \leq -1 \quad & \forall v \in V \quad \text{(optional)} \\ x_v \in X \quad & \forall v \in V \end{aligned} \tag{3.8}$$

If $X \subseteq \mathbb{R}^+$, then the following holds:

if $\{v_1, v_2\} \in E$, then constraints v_1 and v_2 cannot be satisfied simultaneously.

Proof. Let $\{v_1, v_2\}$ be an edge in G and suppose for the sake of contradiction that we can satisfy both constraints v_1 and v_2. Then we have

$$\begin{aligned} -x_{v_1} + x_{v_2} + \textstyle\sum_{u \in N(v_1) \setminus \{v_2\}} x_u \leq -1 \\ -x_{v_2} + x_{v_1} + \textstyle\sum_{u \in N(v_2) \setminus \{v_1\}} x_u \leq -1. \end{aligned}$$

Since $x_v \geq 0$ for all $v \in V$, we have for the value of x_{v_1}

$$\begin{aligned} -x_{v_1} \leq -1 - x_{v_2} - \textstyle\sum_{u \in N(v_1) \setminus \{v_2\}} x_u \leq -1 - x_{v_2} \\ x_{v_1} \leq -1 + x_{v_2} - \textstyle\sum_{u \in N(v_2) \setminus \{v_1\}} x_u \leq -1 + x_{v_2}. \end{aligned}$$

Hence we must have $x_{v_2} + 1 \leq x_{v_1} \leq x_{v_2} - 1$, which can not happen for any $x_{v_1}, x_{v_2} \geq 0$. □

If one would now define a set $V' \subseteq V$ by $v \in V'$ if and only if optional constraint v is satisfied, then Lemma 3.19 especially implies that V' is an independent set. This argument will be used several times in subsequent proofs.

NP-completeness of the general problem

As mentioned, 0/1-C1 FLS $\mid \geq k, \geq c \mid \mathbb{N}, \boldsymbol{b}$ *uni*, $\boldsymbol{p} = \mathbf{1}$ is proven to be strongly NP-complete in [Jan13]. This was actually done by proving strong NP-hardness for the special case when $c = m$. We will see and use this in the next part of this section. For now, we prove that the unweighted problem in consideration is also strongly NP-complete. Therefore we use the equivalence of X-C1 FLS $\mid = k, = c \mid \mathbb{N}, \boldsymbol{b}$ *uni*, $\boldsymbol{p} = \mathbf{1}$ and X-C1 FLS $\mid = k, = c \mid \mathbb{Z}, \boldsymbol{b}$ *uni*, $\boldsymbol{p} = \mathbf{1}$, as proven in Proposition 3.10.

Theorem 3.20. X-C1 FLS $\mid = k, = c \mid \mathbb{Z}, \boldsymbol{b}$ *uni*, $\boldsymbol{p} = \mathbf{1}$ is strongly NP-complete.

Proof. For proving NP-hardness, we construct a polynomial-time reduction from Dec-MIS. Let us therefore be given an instance of Dec-MIS, which is a graph $G = (V, E)$ and a natural number K. We define an instance of X-C1 FLS $\mid = k, = c \mid \mathbb{Z}, \boldsymbol{b}$ *uni*, $\boldsymbol{p} = \mathbf{1}$ by $I = J = V$, $c = k = K$, $b = -1$ and

$$a_{ij} = \begin{cases} -1 & \text{if } i = j \\ 1 & \text{if } j \in N(i) \\ 0 & \text{otherwise.} \end{cases}$$

Then X-C1 FLS $\mid =k, =c \mid \mathbb{Z}, \boldsymbol{b}$ *uni*, $\boldsymbol{p} = \mathbf{1}$ is the problem of finding a feasible solution to the following binary program satisfying exactly K optional constraints.

$$\begin{aligned} \sum\nolimits_{u \in V} x_u &= K && \text{(binding)} \\ -x_v + \sum\nolimits_{u \in N(v)} x_u &\leq -1 \quad \forall v \in V && \text{(optional)} \\ x_v &\in X \quad \forall v \in V \end{aligned} \tag{3.9}$$

We will now prove that yes-instances correspond.

Suppose we were given a yes-instance for Dec-MIS, then there exists an independent set $V' \subseteq V$ of size K. A feasible solution $\boldsymbol{x}$ to (3.9) is then given by $x_v = 1$ if and only if $v \in V'$ and $x_v = 0$ otherwise. Moreover, all optional constraints v for $v \in V'$ are satisfied. For $v \notin V'$, we have $x_v = 0$ and therefore optional constraint v is not satisfied. Thus we have $|V'| = K$ satisfied optional constraints. Therefore, (3.9) is a yes-instance of X-C1 FLS $\mid =k, =c \mid \mathbb{Z}, \boldsymbol{b}$ *uni*, $\boldsymbol{p} = \mathbf{1}$.

Conversely, if (3.9) is a yes-instance of X-C1 FLS $\mid =k, =c \mid \mathbb{Z}, \boldsymbol{b}$ *uni*, $\boldsymbol{p} = \mathbf{1}$, then we are given a feasible solution $\boldsymbol{x} \in X^n$ satisfying exactly K optional constraints. We define the set $V' \subseteq V$ by $v \in V'$ if and only if optional constraint v is satisfied. By Lemma 3.19, V' does not contain a pair of nodes that constitute an edge in G. Therefore, V' is an independent set. Moreover, $|V'| = K$. Hence V' solves Dec-MIS. □

Proposition 3.10 now allows us to state the following corollary immediately.

Corollary 3.21. X-C1 FLS $\mid =k, =c \mid \mathbb{N}, \boldsymbol{b}$ *uni*, $\boldsymbol{p} = \mathbf{1}$ is strongly NP-complete. □

We continue our investigation by looking into special cases of X-C1 FLS $\mid =k, =c \mid \mathbb{N}, \boldsymbol{b}$ *uni*, $\boldsymbol{p} = \mathbf{1}$ by specifying values of k and c. For some of these values we have proven the problem to be in P. Now we will see that for other choices, the problem remains (strongly) NP-complete. We start by looking at certain values of c.

Specifying values of c

For the more general weighted variant 0/1-C1 FLS $\mid =k, =c \mid \mathbb{N}, \boldsymbol{b}$ *uni* we have obtained weak NP-completeness for $c > 0$. This result does not necessarily hold for the unweighted problem 0/1-C1 FLS $\mid =k, =c \mid \mathbb{N}, \boldsymbol{b}$ *uni*, $\boldsymbol{p} = \mathbf{1}$, since this is a special case and therefore, could be easier to solve.

It was proven in [Jan13] that satisfying all optional constraints is strongly NP-complete. Formally, this was proven for the problem 0/1-C1 FLS $\mid \geq k, \geq c \mid \mathbb{N}, \boldsymbol{b}$ *uni*, $\boldsymbol{p} = \mathbf{1}$.

Proposition 3.22 ([Jan13])**.** For $c = m$, 0/1-C1 FLS $\mid \geq k, \geq c \mid \mathbb{N}, \boldsymbol{b}$ *uni*, $\boldsymbol{p} = \mathbf{1}$ is strongly NP-compete.

Of course, satisfying at least all optional constraints is equivalent to satisfying exactly all optional constraints. Therefore Proposition 3.22 also holds for $\diamond_d$ being $=$. We shall repeat the argument given in [Jan13] here.

Proposition 3.23. For $c = m$, 0/1-C1 FLS $\mid \geq k, = c \mid \mathbb{N}, \boldsymbol{b}$ *uni*, $\boldsymbol{p} = \mathbf{1}$ is strongly NP-complete, even for $b = 1$ and $a_{ij} \in \{0, 1\}$.

Proof. For proving NP-hardness, we construct a polynomial-time reduction from the problem Dec-MIS to the problem 0/1-C1 FLS $\mid \geq k, = m \mid \mathbb{N}, \boldsymbol{b}$ *uni*, $\boldsymbol{p} = \mathbf{1}$. In the decision variant of MIS, an instance is a pair (G, K) consisting of a graph $G = (V, E)$ and a natural number K. We define an instance of 0/1-C1 FLS $\mid \geq k, = c \mid \mathbb{N}, \boldsymbol{b}$ *uni*, $\boldsymbol{p} = \mathbf{1}$ by $I = E$, $J = V$, $b = 1$, $k = K$ and

$$a_{ij} = \begin{cases} 1 & \text{if } j \in i \\ 0 & \text{otherwise.} \end{cases}$$

Then 0/1-C1 FLS $\mid \geq k, = m \mid \mathbb{N}, \boldsymbol{b}$ *uni*, $\boldsymbol{p} = \mathbf{1}$ is the problem of finding a feasible solution to

$$\begin{aligned} \textstyle\sum_{u \in V} x_u &\geq K \\ x_u + x_v &\leq 1 \qquad \forall \{u, v\} \in E \\ x_u &\in \{0, 1\} \quad \forall u \in V. \end{aligned} \tag{3.10}$$

We shall now prove that yes-instances correspond.
Suppose we have a yes-instance of MIS. Then we are given an independent set $V' \subseteq V$ such that $|V'| = K$. Define $\boldsymbol{x} \in \{0, 1\}^n$ by $x_u = 1$ if and only if $u \in V'$. Then $\sum_{u \in V} x_u = \sum_{u \in V'} 1 = |V'| = K$. Furthermore, if $\{u, v\} \in E$, then u and v are not both elements of V', implying that $x_u + x_v \leq 1$. Hence, all constraints of (3.10) are satisfied. In other words, (3.10) is a yes-instance of 0/1-C1 FLS $\mid \geq k, = m \mid \mathbb{N}, \boldsymbol{b}$ *uni*, $\boldsymbol{p} = \mathbf{1}$.
Suppose now we are given a yes-instance of 0/1-C1 FLS $\mid \geq k, = m \mid \mathbb{N}, \boldsymbol{b}$ *uni*, $\boldsymbol{p} = \mathbf{1}$, then we have $\boldsymbol{x} \in \{0, 1\}^n$ satisfying all constraints of (3.10). Define $V' \subseteq V$ by picking K elements $u \in V'$ such that $x_u = 1$. Then by all constraints $\{u, v\} \in E$, V' is an independent set. Furthermore, $|V'| = K$. Thus (G, K) is a yes-instance of MIS. □

Notice that this proof also works when we adapt to binding constraint of (3.10) to an equality. Therefore we must adapt the argument of obtaining a yes-instance of MIS, given a yes-instance of 0/1-C1 FLS $\mid = k, = m \mid \mathbb{N}, \boldsymbol{b}$ *uni*, $\boldsymbol{p} = \mathbf{1}$. Given a solution $\boldsymbol{x} \in \{0, 1\}^n$ satisfying the constraint $x_u + x_v \leq 1$ for all $\{u, v\} \in E$, we define an independent set $V' \subseteq V$ by taking *all* $u \in V$ such that $x_u = 1$ (which are K by the binding constraint $\sum_{u \in V} x_u = K$). Therefore we have proven the following result.

Proposition 3.24. For $c = m$, 0/1-C1 FLS $\mid = k, = c \mid \mathbb{N}, \boldsymbol{b}$ *uni*, $\boldsymbol{p} = \mathbf{1}$ is strongly NP-complete, even for $b = 1$ and $a_{ij} \in \{0, 1\}$. □

The arguments given in these proofs only hold for binary variables. It would be interesting to see whether similar results can be given for non-binary variables. Also it would be nice to consider more different values of c. We leave these as open problems.

Specifying values of k

We continue by specifying values of k. For $k \in O(1)$, we have seen that the problem X-C1 FLS $\mid = k, = c \mid \mathbb{N}, \boldsymbol{b}$ *uni*, $\boldsymbol{p} = \mathbf{1}$ is polynomial-time solvable. Here we will look at values of k for which we can conclude strong NP-completeness. These results hold for binary, natural and non-negative real variables.

The proofs in this part of the thesis are all very alike. Therefore, we shall write down the arguments very extensively in the proof of the next theorem. Thereafter we will refer to this theorem for details. The argument is based on the idea in [Jan13, Thm. 4.2.2].

First we have a look at values $k = \frac{r \cdot n}{p}$ for some constants $p, r \in \mathbb{N}$. Of course this expression only makes sense when it is a natural number. Taking $r = 1$, the following theorems state that for any k of the form $k = \frac{n}{p}$, the problem X-C1 FLS $\mid = k, = c \mid \mathbb{N}, \boldsymbol{b}$ *uni*, $\boldsymbol{p} = \mathbf{1}$ is strongly NP-complete. But the theorems also show NP-completeness when $k = \frac{2}{3}n$, $k = \frac{5}{6}n$ etc. as long as these are natural numbers.

Theorem 3.25. Let $p, r \in \mathbb{N}$ be constants such that $0 < r < p$. Then the problem 0/1-C1 FLS $\mid = k, = c \mid \mathbb{N}, \boldsymbol{b}$ *uni*, $\boldsymbol{p} = \mathbf{1}$ is strongly NP-complete for instances satisfying $k \cdot p = r \cdot n$. This holds even for $a_{ij} \in \{0, 1, 2\}$.

Proof. For proving NP-hardness, we construct a polynomial-time reduction from Dec-MIS to 0/1-C1 FLS $\mid = \frac{r \cdot n}{p}, = c \mid \mathbb{N}, \boldsymbol{b}$ *uni*, $\boldsymbol{p} = \mathbf{1}$, where we ensure that $\frac{r \cdot n}{p}$ is a natural number. Let therefore $G = (V, E)$ be a graph with $|V| = \eta$. Let $F(v)$ be the set of nodes with distance at least 2 from v, that is $F(v) = V \setminus (N(v) \cup \{v\})$. We define an instance of 0/1-C1 FLS $\mid = k, = c \mid \mathbb{N}, \boldsymbol{b}$ *uni*, $\boldsymbol{p} = \mathbf{1}$ by $I = V$, $J = V \times \{1, \ldots, p\}$ (J is the disjoint union of p copies of V), $k = r \cdot \eta$, $b = k - 1$ and

$$a_{ijl} = \begin{cases} 0 & \text{if } l = p \text{ and } j = i \\ 2 & \text{if } l = p \text{ and } j \in N(i) \\ 1 & \text{otherwise} \end{cases}$$

where $(j, l) \in J$. Notice that $n = |J| = \eta \cdot p$ and $k \cdot p = r \cdot n$, so that this is indeed an instance of the desired form. Furthermore, we define $c = K$, such that 0/1-C1 FLS $\mid = k, = c \mid \mathbb{N}, \boldsymbol{b}$ *uni*, $\boldsymbol{p} = \mathbf{1}$ is the problem of finding a solution satisfying K optional constraints in the following binary program.

$$\begin{aligned} \sum\nolimits_{(j,l) \in J} x_{jl} &= k && && \text{(binding)} \\ 0 \cdot x_{ip} + \sum_{j \in N(i)} 2x_{jp} + \sum_{j \in F(i)} x_{jp} + \sum_{j \in V} \sum_{l \neq p} x_{jl} &\leq b && \forall i \in I && \text{(optional)} \\ x_{jl} &\in \{0, 1\} && \forall (j, l) \in J \end{aligned} \tag{3.11}$$

The idea behind this construction is that if we would substitute the binding constraint

$$\sum\nolimits_{j \in V} \sum\nolimits_{l \neq p} x_{jl} = k - \sum\nolimits_{j \in V} x_{jp}$$

into all optional constraints, we have the following set of inequalities:

$$\begin{aligned} -x_{ip} + \sum_{j \in N(i)} x_{jp} \leq b - k = -1 \quad & \forall i \in I \\ x_{jl} \in \{0,1\} \quad & \forall (j,l) \in J \end{aligned}$$

which almost looks like the binary linear programming formulation (3.7) of MIS.

We shall now prove that there exists an independent set of size K in G if and only if there exists a solution to (3.11) satisfying exactly K optional constraints. Then we can conclude that yes-instances correspond.

Suppose that V' is an independent set in G of size K. Now consider the vector $\boldsymbol{x} \in \{0,1\}^n$ defined by

$$x_{jl} = \begin{cases} 1 & \text{if } l = p \text{ and } j \in V' \\ 1 & \text{for } k - K \text{ arbitrary elements } (j,l) \in V \times \{1, \ldots, p-1\} \\ 0 & \text{otherwise.} \end{cases} \tag{3.12}$$

Notice that $k - K = r \cdot \eta - K \geq (r-1) \cdot \eta \geq 0$ and $k - K \leq k = r \cdot \eta \leq (p-1) \cdot \eta = |V \times \{1, \ldots, p-1\}|$, so that $\boldsymbol{x}$ is well-defined. Since V' is isomorphic to the subset $V' \times \{p\} \subseteq V \times \{p\}$, we have $\sum_{(j,l) \in J} x_{jl} = k$. Let us now consider an optional constraint i of (3.11).
If $i \in V'$, then $x_{ip} = 1$ and $x_{jp} = 0$ for all $j \in N(i)$. Hence we have

$$\begin{array}{lccccccc} & 0 \cdot x_{ip} & + & \sum_{j \in N(i)} 2x_{jp} & + & \sum_{j \in F(i)} x_{jp} & + & \sum_{j \in V} \sum_{l \neq p} x_{jl} \\ = & 0 \cdot 1 & + & 0 & + & (K-1) & + & (k-K) \\ = & k-1 & & & & & & \\ = & b & & & & & & \end{array}$$

so that constraint i is satisfied.
If $i \notin V'$, then $x_{ip} = 0$ and $x_{jp} = 1$ for s elements $j \in N(i) \cup F(i)$. Then we have

$$\begin{array}{lccccccc} & 0 \cdot x_{ip} & + & \sum_{j \in N(i)} 2x_{jp} & + & \sum_{j \in F(i)} x_{jp} & + & \sum_{j \in V} \sum_{l \neq p} x_{jl} \\ \geq & 0 \cdot 0 & + & & K & & + & (k-K) \\ = & k & & & & & & \\ > & b & & & & & & \end{array}$$

so that constraint i is violated.
Hence, allocation (3.12) is a solution to (3.11) satisfying K optional constraints.

Now suppose that $\boldsymbol{x} \in \{0,1\}^n$ is a solution to (3.11) satisfying exactly K optional constraints. Let us define a set $V' \subseteq V$ by

$$v \in V' \quad \Leftrightarrow \quad \text{optional constraint } v \text{ is satisfied by the solution } \boldsymbol{x}. \tag{3.13}$$

Given the binding constraint $\sum_{(j,l) \in J} x_{jl} = k$, we have for every $i \in I$

$$0 \cdot x_{ip} + \sum_{j \in N(i)} 2x_{jp} + \sum_{j \in F(i)} x_{jp} + \sum_{j \in V} \sum_{l \neq p} x_{jl} \leq b$$

if and only if

$$\left(0 \cdot x_{ip} + \sum\nolimits_{j \in N(i)} 2x_{jp} + \sum\nolimits_{j \in F(i)} x_{jp} + \sum\nolimits_{j \in V} \sum\nolimits_{l \neq p} x_{jl}\right) - k \leq b - k$$

if and only if

$$-x_{ip} + \sum\nolimits_{j \in N(i)} x_{jp} \leq b - k = -1. \tag{3.14}$$

We thus have $i \in V'$ if and only if (3.14) holds. As we argued before, this is if and only if $x_{ip} = 1$ and $x_{jp} = 0$ for all $j \in N(i)$. Since $\{u, v\} \in E$ and $u, v \in V'$ would then imply $0 = x_{up} = 1$, we can conclude that V' is an independent set. Furthermore, $|V'| = K$. □

When having non-binary variables, we need to argue a little bit differently.

Theorem 3.26. Let $X \in \{\mathbb{N}, \mathbb{R}^+\}$ and let $p, r \in \mathbb{N}$ be constants such that $0 < r < p$. Then the problem X-C1 FLS $\mid =k, =c \mid \mathbb{N}, \boldsymbol{b}$ *uni*, $\boldsymbol{p} = \mathbf{1}$ is strongly NP-complete for instances satisfying $k \cdot p = r \cdot n$. This holds even for $a_{ij} \in \{0, 1, 2\}$.

Proof. We redo the proof of the previous theorem. Given an instance of sc Dec-MIS, we define the linear program (3.11), but now having $x_{jl} \in X$, as an instance for the problem X-C1 FLS $\mid =k, =c \mid \mathbb{N}, \boldsymbol{b}$ *uni*, $\boldsymbol{p} = \mathbf{1}$. Let us prove the correspondence of having an independent set of size K if and only if we have a solution $\boldsymbol{x} \in X^n$ satisfying K optional constraints of (3.11).

Given an independent set V' of size s, we can still do allocation (3.12) when using variables $x_{jl} \in X$. By the discussion below this allocation, we obtain a solution to the instance of X-C1 FLS $\mid =k, =c \mid \mathbb{N}, \boldsymbol{b}$ *uni*, $\boldsymbol{p} = \mathbf{1}$ satisfying K optional constraints. Conversely, suppose that we have a solution $\boldsymbol{x} \in X^n$ to the instance of X-C1 FLS $\mid =k, =c \mid \mathbb{N}, \boldsymbol{b}$ *uni*, $\boldsymbol{p} = \mathbf{1}$ satisfying K optional constraints. We follow definition (3.13) to define $V' \subseteq V$. Using the binding constraint, we similarly obtain that constraint i is satisfied if and only if (3.7) holds. Because the variables $x_{jl} \in X$ are all non-negative, we apply Lemma 3.19 to see that V' is an independent set. Hence, we obtain an independent set of size K.

We conclude that yes-instances correspond and that X-C1 FLS $\mid =k, =c \mid \mathbb{N}, \boldsymbol{b}$ *uni*, $\boldsymbol{p} = \mathbf{1}$ remains NP-complete, even when allowing non-negative integer or real variables. □

A similar trick can be used when k is of the form $\sqrt[r]{n}$ (whenever this is a natural number) for any integer $r \geq 1$. We shall give the argument when having binary variables and conclude that, using Lemma 3.19, a similar proof is valid when having non-binary variables.

Theorem 3.27. Let $r \in \mathbb{N}^{\geq 2}$ be a constant. Then the problem 0/1-C1 FLS $\mid =k, =c \mid \mathbb{N}, \boldsymbol{b}$ *uni*, $\boldsymbol{p} = \mathbf{1}$ is MIS-hard for instances satisfying $k^r = n$. This holds even for $a_{ij} \in \{0, 1, 2\}$.

Proof. We constructing a polynomial-time reduction from Dec-MIS. Let therefore $G = (V, E)$ be a graph and $|V| = \eta \geq 2$. We define an instance of X-C1 FLS $\mid =k, =c \mid$

$\mathbb{N}, \boldsymbol{b}\ uni, \boldsymbol{p} = \mathbf{1}$ by $I = V$, $J = V^r$, $k = \eta$ and $b = k - 1$. Furthermore, fix any $* \in V$ and define

$$a_{i,\boldsymbol{j}} = \begin{cases} 0 & \text{if } j_1 = \ldots = j_{r-1} = * \text{ and } j_r = i \\ 2 & \text{if } j_1 = \ldots = j_{r-1} = * \text{ and } j_r \in N(i) \\ 1 & \text{otherwise} \end{cases}$$

where $\boldsymbol{j} = (j_1, \ldots, j_r) \in V^r$. Notice that $n = |J| = \eta^r$ so that indeed $k^r = n$. Defining $c = K$, X-C1 FLS $| = k, = c \mid \mathbb{N}, \boldsymbol{b}\ uni, \boldsymbol{p} = \mathbf{1}$ is the problem of satisfying K optional constraints in the following binary program.

$$\begin{aligned} \sum\nolimits_{\boldsymbol{j} \in J} x_{\boldsymbol{j}} &= k && \text{(binding)} \\ 0 \cdot x_{(*,\ldots,*,i)} + \sum_{j \in N(i)} 2x_{(*,\ldots,*,j)} + \sum_{j \in F(i)} x_{(*,\ldots,*,j)} & \\ + \sum_{j_r \in V} \sum_{\substack{(j_1,\ldots,j_{r-1}) \\ \neq (*,\ldots,*)}} x_{(j_1,\ldots,j_r)} &\leq b \qquad \forall i \in I && \text{(optional)} \\ x_{\boldsymbol{j}} &\in \{0,1\} \quad \forall \boldsymbol{j} \in J \end{aligned} \tag{3.15}$$

For yes-instances to correspond, we prove that G has an independent set of size K if and only if (3.15) has a solution satisfying K optional constraints.

Given an independent set V' of size K in G, consider the vector $\boldsymbol{x} \in \{0,1\}^n$ defined by

$$x_{\boldsymbol{j}} = \begin{cases} 1 & \text{if } j_1 = \ldots = j_{r-1} = * \text{ and } j \in V' \\ 1 & \text{for } k - K \text{ arbitrary elements } \boldsymbol{j} \in V^r \setminus (\{*\}^{r-1} \times V) \\ 0 & \text{otherwise.} \end{cases}$$

Again, $\boldsymbol{x}$ is well-defined because $k - K \geq 0$ and $k - K \leq k = \eta \leq \eta^r - \eta = |V^r \setminus (\{*\}^{r-1} \times V)|$. Furthermore, the binding constraint is satisfied. Arguing similarly as in the proof of Theorem 3.25, we see that optional constraint i is satisfied if and only if $i \in V'$. Thus $\boldsymbol{x}$ is a solution to (3.15) satisfying K optional constraints.

Conversely, given a solution $\boldsymbol{x}$ to (3.15) satisfying K optional constraints, we define $V' \subseteq V$ by (3.13) in the proof of Theorem 3.25 and argue similarly that V' is an independent set of size K. □

Theorem 3.28. Let $X \in \{\mathbb{N}, \mathbb{R}^+\}$ and let $r \in \mathbb{N}^{\geq 2}$ be a constant. Then the problem X-C1 FLS $| = k, = c \mid \mathbb{N}, \boldsymbol{b}\ uni, \boldsymbol{p} = \mathbf{1}$ is MIS-hard for instances satisfying $k^r = n$. This holds even for $a_{ij} \in \{0,1,2\}$. □

So far we have only considered values of k that were at most n. This is because when having binary variables, X-C1 FLS $| = k, = c \mid \mathbb{N}, \boldsymbol{b}\ uni, \boldsymbol{p} = \mathbf{1}$ is infeasible for $k > n$. But taking $\mathbb{N}$ or $\mathbb{R}^+$ as our variable range, it does make sense to consider larger values of k. The following theorem gives a hardness result for this case, even for $k \geq n - 1$.

Theorem 3.29. Let $X \in \{\mathbb{N}, \mathbb{R}^+\}$. For any $k \geq n - 1$, the problem X-C1 FLS $| = k, = c \mid \mathbb{N}, \boldsymbol{b}\ uni, \boldsymbol{p} = \mathbf{1}$ is strongly NP-complete, even for $a_{ij} \in \{0,1,2\}$.

Proof. For any $k \geq n-1$, we construct a polynomial-time reduction from Dec-MIS to X-C1 FLS $\mid =k, =c \mid \mathbb{N}, \boldsymbol{b}\ uni, \boldsymbol{p} = \mathbf{1}$. Let us therefore be given a graph $G = (V, E)$ with $|V| = \eta$ and a natural number K. Let $F(v)$ be the set $F(v) = V \setminus (N(v) \cup \{v\})$. We define an instance of X-C1 FLS $\mid =k, =c \mid \mathbb{N}, \boldsymbol{b}\ uni, \boldsymbol{p} = \mathbf{1}$ by $I = V$, $J = V \cup \{*\}$, $k \geq \eta$, $b = k-1$ and for every $i \in I$:

$$a_{ij} = \begin{cases} 0 & \text{if } j = i \\ 2 & \text{if } j \in N(i) \\ 1 & \text{otherwise.} \end{cases}$$

Notice that $n = |J| = \eta + 1$ and $k \geq \eta$, so that $k \geq n-1$. For $c = K$, X-C1 FLS $\mid =k, =c \mid \mathbb{N}, \boldsymbol{b}\ uni, \boldsymbol{p} = \mathbf{1}$ is the problem of satisfying K optional constraints in the following linear program.

$$\begin{aligned} \sum\nolimits_{j \in J} x_j &= k && \text{(binding)} \\ 0 \cdot x_i + \sum\nolimits_{j \in N(i)} 2x_j + \sum\nolimits_{j \in F(i)} x_j + x_* &\leq b \quad \forall i \in I && \text{(optional)} \\ x_j &\in X \quad \forall j \in J \end{aligned} \tag{3.16}$$

We shall prove that there exists an independent set of size K in G if and only if (3.16) has a solution satisfying K optional constraints. This means that yes-instances correspond.

Suppose that V' is an independent set of size K in G. As before, let $N(V')$ again denote the nodes adjacent to V', i.e. $N(V') = \bigcup_{v \in V'} N(v)$. Now consider the vector $\boldsymbol{x} \in X^n$ defined by

$$x_j = \begin{cases} 1 & \text{if } j \in V' \\ k-K & \text{if } j = * \\ 0 & \text{otherwise.} \end{cases} \tag{3.17}$$

Notice that since $k \geq \eta$ and $K \leq \eta$, we have $k - K \geq 0$. Furthermore, $k - K$ is a natural number, so that $\boldsymbol{x}$ is well-defined for $X \in \{\mathbb{N}, \mathbb{R}^+\}$. Let us now consider an optional constraint i of (3.16).

If $i \in V'$, then $x_i = 1$ and $x_j = 0$ for all $j \in N(i)$. Hence we have

$$\begin{array}{lccccccc} & 0 \cdot x_i & + & \sum_{j \in N(i)} 2x_j & + & \sum_{j \in F(i)} x_j & + & x_* \\ = & 0 \cdot 1 & + & 0 & + & (K-1) & + & (k-K) \\ = & k-1 & & & & & & \\ = & b & & & & & & \end{array}$$

so that constraint i is satisfied.

If $i \notin V'$, then $x_i = 0$ and $x_j = 1$ for K elements $j \in N(i) \cup F(i)$. Then we have

$$\begin{array}{lccccccc} & 0 \cdot x_i & + & \sum_{j \in N(i)} 2x_j & + & \sum_{j \in F(i)} x_j & + & x_* \\ \geq & 0 \cdot 0 & + & & K & & + & (k-K) \\ = & k & & & & & & \\ > & b & & & & & & \end{array}$$

so that constraint i is violated.
Hence, allocation (3.17) is a solution to the instance of X-C1 FLS $| =k, =c | \mathbb{N}, \boldsymbol{b}\ uni, \boldsymbol{p} = \mathbf{1}$ satisfying K optional constraints.

Now suppose that $\boldsymbol{x} \in X^n$ is a solution to the instance (3.16) of X-C1 FLS $| =k, =c | \mathbb{N}, \boldsymbol{b}\ uni, \boldsymbol{p} = \mathbf{1}$ satisfying K optional constraints. Let us define a set $V' \subseteq V$ by (3.13) in the proof of Theorem 3.25. Using the binding constraint $\sum_{j \in J} x_j = k$, we argue as before and obtain that for every $i \in I$: $i \in V'$ if and only if

$$-x_i + \sum\nolimits_{j \in N(i)} x_j \leq -1.$$

Since for $X \in \{\mathbb{N}, \mathbb{R}^+\}$ holds that all variables are non-negative, we can conclude from Lemma 3.19 that V' is an independent set. Furthermore, $|V'| = K$. □

Next we will discuss what the obtained complexity results mean for the maximization problem Max-X-C1 FLS $| \diamond_b k | Y, \boldsymbol{b}\ uni$.

3.5 Hardness of the optimization problem

In this section we shall summarize the consequences of the obtained results for the optimization problems Max-X-C1 FLS $| \diamond k | \mathbb{N}, \boldsymbol{b}\ uni, \boldsymbol{p} = \mathbf{1}$. For the four decision variants, we have found some equivalences, polynomial-time cases and many hardness results.

Some polynomial-time cases, like deciding whether a solution exists such that none of the optional constraints are satisfied, do not provide tools for solving the optimization problem. On the other hand, polynomial-time solvable decision problems where we did not specify the value of c do.

We will see that the polynomial-time reductions from Dec-MIS can actually be turned into cost-preserving polynomial-time reductions between the corresponding optimization problems, yielding a bad inapproximability result. On top of these we provide a Turing reduction, which gives a somewhat weaker conclusion.

3.5.1 Equivalence of optimization problems

In this chapter we observed the problems X-C1 FLS $| \geq k, \geq c | \mathbb{N}, \boldsymbol{b}\ uni, \boldsymbol{p} = \mathbf{1}$ and X-C1 FLS $| =k, \geq c | \mathbb{N}, \boldsymbol{b}\ uni, \boldsymbol{p} = \mathbf{1}$ to be polynomial-time equivalent (Theorem 3.4). In this section we combine this with a result in [Jan13] to obtain equivalence of the corresponding optimization problems.

Lemma 3.30 ([Jan13])**.** Max-X-C1 FLS $| \geq k | \mathbb{N}, \boldsymbol{b}\ uni, \boldsymbol{p} = \mathbf{1}$ has an optimal solution $\boldsymbol{x}$ satisfying $\sum_{j \in J} x_j = k$.

In [Jan13], this result is only proven for the case of binary variables. However, the argument given in Lemma 3.3 easily shows that it also holds for non-binary variables.

We will now prove that optimization problems MAX-X-C1 FLS $\mid \geq k \mid \mathbb{N}, \boldsymbol{b}\ uni, \boldsymbol{p} = \mathbf{1}$ and MAX-X-C1 FLS $\mid = k \mid \mathbb{N}, \boldsymbol{b}\ uni, \boldsymbol{p} = \mathbf{1}$ are equivalent. Recall that these problems involve the following set of linear constraints, where $\diamond_b \in \{=, \geq\}$.

$$\begin{aligned} \sum\nolimits_{j\in J} x_j \diamond_b k & \qquad && \text{(binding)} \\ \sum\nolimits_{j\in J} a_{ij} x_j \leq b & \quad \forall i \in I && \text{(optional)} \\ x_j \in X & \quad \forall j \in J \end{aligned}$$

Let us denote $U_\diamond$ for this instance of MAX-X-C1 FLS $\mid \diamond k \mid \mathbb{N}, \boldsymbol{b}\ uni, \boldsymbol{p} = \mathbf{1}$.

Lemma 3.31. If $\boldsymbol{x} \in X^n$ is an optimal solution to $U_=$, then $\boldsymbol{x}$ is an optimal solution to $U_\geq$.

Proof. Let $\boldsymbol{x} \in X^n$ be an optimal solution to $U_=$ satisfying OPT optional constraints and suppose it is not optimal for $U_\geq$. Then $U_\geq$ has a optimal solution $\boldsymbol{y} \in X^n$ such that $|S(\boldsymbol{y})| >$ OPT. But then from Lemma 3.30, there exists an optimal solution $\boldsymbol{z} \in X^n$ to $U_\geq$ satisfying $\sum_{j\in J} x_j = k$ and $|S(\boldsymbol{z})| = |S(\boldsymbol{y})| >$ OPT. Since $\boldsymbol{z}$ is a feasible solution to $U_=$, this contradicts the optimality of $\boldsymbol{x}$. □

Lemma 3.32. If $\boldsymbol{x} \in X^n$ is an optimal solution to $U_\geq$, then we can find an optimal solution $\boldsymbol{y} \in X^n$ to $U_=$.

Proof. This is straightforward from our previous results. Let $\boldsymbol{x} \in X^n$ be an optimal solution to $U_\geq$ with $|S(\boldsymbol{x})| =$ OPT. Construct $\boldsymbol{y} \in X^n$ as in the proofs of Lemmas 3.2 and 3.3. Then especially we have $|S(\boldsymbol{y})| \geq$ OPT. Suppose $\boldsymbol{y}$ is not optimal for $U_=$, then there exists a feasible solution $\boldsymbol{z} \in \{0,1\}^n$ to $U_=$ such that $|S(\boldsymbol{z})| > |S(\boldsymbol{y})|$ and $\boldsymbol{z}$ is optimal. By Lemma 3.31, $\boldsymbol{z}$ is then an optimal solution to $U_\geq$. But since we have $|S(\boldsymbol{z})| >$ OPT, this contradicts the optimality of $\boldsymbol{x}$. □

Combining these two results we obtain equivalence of the optimization problems.

Theorem 3.33. The optimization problems MAX-X-C1 FLS $\mid \geq k \mid \mathbb{N}, \boldsymbol{b}\ uni, \boldsymbol{p} = \mathbf{1}$ and MAX-X-C1 FLS $\mid = k \mid \mathbb{N}, \boldsymbol{b}\ uni, \boldsymbol{p} = \mathbf{1}$ are equivalent. □

Furthermore, we have seen in Proposition 3.10 that instances of X-C1 FLS $\mid = k, \diamond c \mid \mathbb{N}, \boldsymbol{b}\ uni, \boldsymbol{p} = \mathbf{1}$ and X-C1 FLS $\mid = k, \diamond c \mid \mathbb{Z}, \boldsymbol{b}\ uni, \boldsymbol{p} = \mathbf{1}$ can be transformed into each other such that solutions coincide. Then certainly optimal solutions of the corresponding optimization problems coincide. Therefore, we obtain another equivalence.

Theorem 3.34. The optimization problems MAX-X-C1 FLS $\mid = k \mid \mathbb{Z}, \boldsymbol{b}\ uni, \boldsymbol{p} = \mathbf{1}$ and MAX-X-C1 FLS $\mid = k \mid \mathbb{N}, \boldsymbol{b}\ uni, \boldsymbol{p} = \mathbf{1}$ are equivalent. □

3.5.2 Turing reduction from MIS

We have seen that the decision problem X-C1 FLS $\mid \diamond_b k, \diamond_d c \mid \mathbb{N}, \boldsymbol{b}\ uni, \boldsymbol{p} = \mathbf{1}$ is sometimes polynomial-time solvable. This was the case for $X = \{0,1\}$ and k being either in $O(1)$ or in $n - O(1)$. For $X = \mathbb{N}$, the problem is polynomial-time solvable for $k \in O(1)$.

In these cases, one could also solve the optimization problem MAX-X-C1 FLS | $\diamond_b k$ | $\mathbb{N}, \boldsymbol{b}$ *uni*, $\boldsymbol{p} = \mathbf{1}$ by subsequently asking whether there exists a feasible solution satisfying (at least) $m-p$ optional constraints, for $p = 0, \ldots, m$. Since this procedure involves at most m calls to an polynomial-time algorithm, the whole procedure runs in polynomial time.

For other cases, we have seen that a decision problem is NP-complete. This means that the decision problem is polynomial-time solvable if and only if P=NP. As the latter is widely believed to be false, this result usually means that one could not hope to solve the corresponding optimization problem efficiently.

There is yet another way to see that optimization problems are not expected to be efficiently solvable. Therefore we construct a Turing reduction (Definition 1.12 in Chapter 1) from another optimization problem that is not believed to be efficiently solvable. Next we provide a Turing reduction from MIS to MAX-X-C1 FLS | $=k$ | $\mathbb{Z}, \boldsymbol{b}$ *uni*, $\boldsymbol{p} = \mathbf{1}$. The reduction uses the idea of the above describes procedure.

Theorem 3.35. There exists a Turing reduction from MIS to MAX-X-C1 FLS | $=k$ | $\mathbb{Z}, \boldsymbol{b}$ *uni*, $\boldsymbol{p} = \mathbf{1}$.

Proof. Suppose we are given an instance of MIS, which is a graph $G = (V, E)$, and let $|V| = \eta$. Then Algorithm 1 solves MIS, given an oracle to MAX-X-C1 FLS | $=k$ | $\mathbb{Z}, \boldsymbol{b}$ *uni*, $\boldsymbol{p} = \mathbf{1}$.

Algorithm 1. Solving MIS given an oracle to MAX-X-C1 FLS | $=k$ | $\mathbb{Z}, \boldsymbol{b}$ *uni*, $\boldsymbol{p} = \mathbf{1}$.

input : Graph $G = (V, E)$ with $|V| = \eta$.
output: Cardinality of a maximum independent set in G.

1 *initialize* boolean bool = false, int size = η
2 **while** bool = false *and* size ≥ 1 **do**
3 Use the oracle to solve the following instance of $\mathrm{MFS}^{=}_{\{0,1\},\mathbb{Z}}(k)$:
4 $\sum_{u \in V} x_u$ = size
5 $-x_v + \sum_{u \in N(v)} x_u \leq -1 \quad \forall v \in V$ (optional)
6 $x_v \in X \quad \forall v \in V$
7 **if** $|S(\boldsymbol{x})|$ = size **then** bool = true
8 **else** size = size -1
9 **end**
10 **return** size

Notice that one optional constraint can always be satisfied by setting $x_v = 1$ for some $v \in V$ and x_u for $u \neq v$. Therefore, Algorithm 1 terminates after at most η iterations of the while loop. In order to check the correctness of the algorithm, we must show that whenever G contains an independent set of size size, Algorithm 1 must find it. That is: if there exists a vector in X^η such that size of the optional constraints in line 5 can be satisfied, then there must be some $\boldsymbol{x}$ such that $\boldsymbol{x}$ satisfies size optional constraints and $\sum_{u \in V} x_u$ = size. Let us prove that such a solution indeed exists.

Lemma 3.36. If OPT is the maximum number of optional constraints that can be satisfied in line 5 of Algorithm 1, then there exists $\boldsymbol{y} \in X^\eta$ such that $\boldsymbol{y}$ satisfies these OPT optional constraints and $\sum_{v \in V} y_v = \mathsf{OPT}$.

Proof. Let $\boldsymbol{x} \in X^\eta$ satisfy optional constraints $v_1, \ldots, v_{\mathsf{OPT}}$. Then $\boldsymbol{y} \in X^\eta$, defined by $y_v = 1$ for $v = v_1, \ldots, v_{\mathsf{OPT}}$ and $y_v = 0$ otherwise, satisfies optional constraints $v_1, \ldots, v_{\mathsf{OPT}}$ and we have $\sum_{v \in V} y_v = \mathsf{OPT}$. ■

As long as needed, the algorithm searches for solutions of Max-X-C1 FLS | = size | $\mathbb{Z}, \boldsymbol{b}$ *uni*, $\boldsymbol{p} = \mathbf{1}$ satisfying exactly size optional constraints. Earlier we saw that such a solution indeed corresponds to a independent set of size vertices. Since we initialize by size = η, the solution found by the algorithm corresponds to a maximum independent set.

The algorithm calls at most η times for the oracle to solve Max-X-C1 FLS | = size | $\mathbb{Z}, \boldsymbol{b}$ *uni*, $\boldsymbol{p} = \mathbf{1}$. Therefore, Algorithm 1 runs in polynomial time $O(\eta)$. □

One could extend Algorithm 1 by keeping track of the solution $\boldsymbol{x}$ and also return this vector. Then the extended algorithm would return the largest possible independent set in the graph G.

For binary variables, a stronger version of Lemma 3.36 can be stated. In this case, every solution $\boldsymbol{x} \in \{0,1\}^n$ satisfying size optional constraints has the property $\sum_{j \in J} x_j = k$.

Lemma 3.37. Suppose OPT is the maximum number of optional constraints that can be satisfied in line 5 of Algorithm 1. If $\boldsymbol{x}$ satisfies OPT optional constraints in line 5, then we have $\sum_u x_u = \mathsf{OPT}$.

Proof. Suppose $\boldsymbol{x} \in \{0,1\}^\eta$ satisfies OPT of the optional constraints in line 5 and let us denote the satisfied optional constraints by $v_1, \ldots, v_{\mathsf{OPT}}$. Then we know $x_v = 1$ for $v = v_1, \ldots, v_{\mathsf{OPT}}$ and $x_v = 0$ for $v \in N(\{v_1, \ldots, v_{\mathsf{OPT}}\}) := N(v_1) \cup \ldots \cup N(v_{\mathsf{OPT}})$. Thus $\sum_u x_u \geq \mathsf{OPT}$. Suppose $\sum_u x_u > \mathsf{OPT}$. Then there exists $v \neq v_1, \ldots, v_{\mathsf{OPT}}$, $v \notin N(\{v_1, \ldots, v_{\mathsf{OPT}}\})$ such that $x_v = 1$. But then $\boldsymbol{y} \in \{0,1\}^\eta$, defined by

$$y_u = \begin{cases} 1 & \text{if } u = v \\ x_u & \text{if } u \in \{v_1, \ldots, v_{\mathsf{OPT}}\} \cup N(\{v_1, \ldots, v_{\mathsf{OPT}}\}) \\ 0 & \text{otherwise} \end{cases}$$

is a solution satisfying OPT + 1 optional constraints, which is a contradiction. □

The Turing reduction given by Algorithm 1 tells us that if we would be able to solve Max-X-C1 FLS | $= k$ | $\mathbb{Z}, \boldsymbol{b}$ *uni*, $\boldsymbol{p} = \mathbf{1}$ efficiently, then we can also solve the optimization problem MIS. The latter cannot be done unless P=NP. We saw before that the problems Max-X-C1 FLS | $= k$ | $\mathbb{Z}, \boldsymbol{b}$ *uni*, $\boldsymbol{p} = \mathbf{1}$ and Max-X-C1 FLS | $= k$ | $\mathbb{N}, \boldsymbol{b}$ *uni*, $\boldsymbol{p} = \mathbf{1}$ are equivalent. Therefore we obtain the following corollary immediately.

Corollary 3.38. Unless P=NP, there does not exist a polynomial-time algorithm for solving Max-X-C1 FLS | $= k$ | $\mathbb{N}, \boldsymbol{b}$ *uni*, $\boldsymbol{p} = \mathbf{1}$. □

3.5.3 Inapproximability results

We close our investigation by taking a closer look at the conclusions of strong NP-completeness for different values of k (Theorems 3.25 to 3.29). All these proofs had similar approaches: given a natural number K as part of the instance of Dec-MIS, we constructed an instance of X-C1 FLS $\mid =k, =c \mid \mathbb{N}, \boldsymbol{b}\ uni, \boldsymbol{p} = \mathbf{1}$, where in particular we defined c to be equal to K.

Considering the correspondence of yes-instances, we could therefore prove that there exists an independent set of size K if and only if a solution satisfying the binding constraint and exactly K of the optional constraints. When we take this to the level of optimization problems, the arguments in Theorems 3.25 to 3.29 show that we can solve MIS with an objective value of K if and only if we can solve Max-X-C1 FLS $\mid =k \mid \mathbb{N}, \boldsymbol{b}\ uni, \boldsymbol{p} = \mathbf{1}$ with an objective value of K.

This means that the given polynomial-time reductions between the decision problems actually provides us with cost-preserving polynomial-time reductions between the optimization problems MIS and Max-X-C1 FLS $\mid =k \mid \mathbb{N}, \boldsymbol{b}\ uni, \boldsymbol{p} = \mathbf{1}$. In Chapter 1, we have explained that these reductions are approximation preserving, meaning in particular that any inapproximability result for MIS also holds for Max-X-C1 FLS $\mid =k \mid \mathbb{N}, \boldsymbol{b}\ uni, \boldsymbol{p} = \mathbf{1}$. Combining this observation with Håstad's result (Theorem 1.15), we obtain an inapproximability result for Max-X-C1 FLS $\mid =k \mid \mathbb{N}, \boldsymbol{b}\ uni, \boldsymbol{p} = \mathbf{1}$. Therefore we notice that in the proofs of Theorems 3.25 to 3.29, the vertex set V of a graph $G = (V, E)$ corresponds to the index set of the optional constraints I.

Theorem 3.39. Unless any problem in NP can be solved in probabilistic polynomial time, Max-X-C1 FLS $\mid =k \mid \mathbb{N}, \boldsymbol{b}\ uni, \boldsymbol{p} = \mathbf{1}$ is not approximable within $m^{1-\varepsilon}$ for all $\varepsilon > 0$ in the following special cases:

- instances satisfying $k \cdot p = r \cdot n$ for constants $p, r \in \mathbb{N}$ such that $0 < r < p$;
- instances satisfying $k^r = n$ for a constant $r \in \mathbb{N}^{\geq 2}$;
- instances satisfying $X \in \{\mathbb{N}, \mathbb{R}^+\}$ and $k \geq n - 1$. □

This inapproximability result also holds for Max-X-C1 FLS $\mid =k \mid \mathbb{N}, \boldsymbol{b}\ uni, \boldsymbol{p} = \mathbf{1}$ (when we do not specify the value of k).

We have seen in Propositions 3.11 and 3.12 that satisfying at least one optional constraint is polynomial-time solvable. This argument provides us either with an m-approximation for Max-X-C1 FLS $\mid =k \mid \mathbb{N}, \boldsymbol{b}\ uni, \boldsymbol{p} = \mathbf{1}$ or with the conclusion that the optimal objective value is zero. The above theorem shows that we cannot hope for a better approximation algorithm.

Table 3.1 gives an overview of the obtained complexity results in this chapter. Because of our results for the case of having binary variables, we can draw a 'solvability diagram' (Figure 3.1). In this figure, an arrow $A \Longrightarrow B$ means: if A is polynomial-time solvable, then so is B.

Problem	**Value** k	**Value** c	**Complexity**	**Even for**
0/1-C1 FLS $\mid = k, \diamond c \mid \mathbb{N}, \boldsymbol{b}$ *uni*, $\boldsymbol{p} = \mathbf{1}$	$O(1)$		P	
0/1-C1 FLS $\mid = k, \diamond c \mid \mathbb{N}, \boldsymbol{b}$ *uni*, $\boldsymbol{p} = \mathbf{1}$	$n - O(1)$		P	
$\mathbb{N}$-C1 FLS $\mid = k, \diamond c \mid \mathbb{N}, \boldsymbol{b}$ *uni*, $\boldsymbol{p} = \mathbf{1}$	$O(1)$		P	
X-C1 FLS $\mid \diamond k, \geq c \mid \mathbb{N}, \boldsymbol{b}$ *uni*, $\boldsymbol{p} = \mathbf{1}$		1	P	
X-C1 FLS $\mid \geq k, = c \mid \mathbb{N}, \boldsymbol{b}$ *uni*, $\boldsymbol{p} = \mathbf{1}$		0	P	
0/1-C1 FLS $\mid = k, = c \mid \mathbb{N}, \boldsymbol{b}$ *uni*		$\frac{m}{2}$	weakly NP-complete	$a_{ij} \in \{0, 1\}$, $b = 0$
0/1-C1 FLS $\mid = k, = c \mid \mathbb{N}, \boldsymbol{b}$ *uni*		> 0	weakly NP-complete	$a_{ij} \in \{0, 1, 2\}$, $b = 0$
0/1-C1 FLS $\mid \diamond_b k, \diamond_d c \mid \mathbb{N}, \boldsymbol{b}$ *uni*, $\boldsymbol{p} = \mathbf{1}$		m	strongly NP-complete	$a_{ij} \in \{0, 1\}$, $b = 1$
$\mathbb{N}$-C1 FLS $\mid = k, = c \mid \mathbb{N}, \boldsymbol{b}$ *uni*, $\boldsymbol{p} = \mathbf{1}$	$\geq n - 1$		strongly NP-complete	$a_{ij} \in \{0, 1, 2\}$
$\mathbb{R}^+$-C1 FLS $\mid = k, = c \mid \mathbb{N}, \boldsymbol{b}$ *uni*, $\boldsymbol{p} = \mathbf{1}$	$\geq n - 1$		strongly NP-complete	$a_{ij} \in \{0, 1, 2\}$
X-C1 FLS $\mid = k, = c \mid \mathbb{N}, \boldsymbol{b}$ *uni*, $\boldsymbol{p} = \mathbf{1}$	$\frac{r \cdot n}{p}$		strongly NP-complete	$a_{ij} \in \{0, 1, 2\}$
X-C1 FLS $\mid = k, = c \mid \mathbb{N}, \boldsymbol{b}$ *uni*, $\boldsymbol{p} = \mathbf{1}$	$\sqrt[r]{n}$		strongly NP-complete	$a_{ij} \in \{0, 1, 2\}$
X-C1 FLS $\mid = k, = c \mid \mathbb{Z}, \boldsymbol{b}$ *uni*, $\boldsymbol{p} = \mathbf{1}$			strongly NP-complete	$a_{ij} \in \{0, 1, -1\}$, $b = -1$
X-C1 FLS $\mid = k, = c \mid \mathbb{N}, \boldsymbol{b}$ *uni*, $\boldsymbol{p} = \mathbf{1}$			strongly NP-complete	

Table 3.1: Summary of theoretical results.

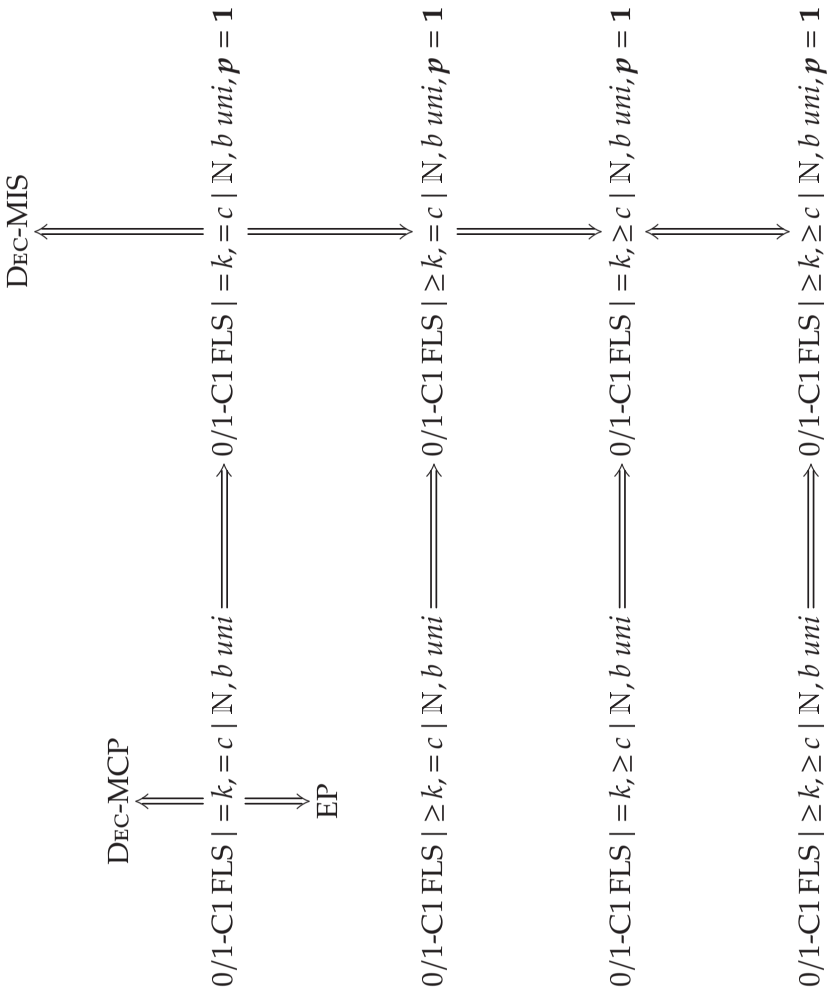

Figure 3.1: Diagram of solvability.

Part II

SIMULATIONS

Chapter 4

Modeling design

In this chapter we shall introduce a simulation platform in order to solve instances of the problem that was introduced in Chapter 2. We will create an imaginary airport with several flight routines and housing locations. How such an environment looks like will be clarified in Section 4.2. The written source code for the program that generates instances for this special kind of airport is described in Section 4.3. Knowing all the data of such an instance, we feed the problem to a solver called SCIP, of which more information can be found in Section 4.1.

4.1 Modeling in SCIP

SCIP (http://scip.zib.de/) is a framework for both mixed integer linear programming and constraint programming. It is being developed since 2001 at the Zuse Institute Berlin (ZIB) and at the moment, it is the fastest non-commercial solver for *constraint integer programs*. These are (mixed) integer linear programs that also support general constraints from constraint programming. A detailed description of the features of SCIP can be found in [Ach09a].

For solving constraint integer programs, SCIP uses a combination of mixed integer linear programming techniques and constraint programming techniques. Solving techniques from mixed integer linear programming that are used are, amongst others, branch and bound, relaxation methods, cutting planes and column generation. We shall not explore these techniques here, but refer to [BT97] for detailed information. Another important feature that SCIP uses are constraint handlers, which provide extra information or algorithms to analyze certain types of constraints. Constraint handlers could for example be used to find redundant optional constraints. Moreover, heuristics are applied to search for solutions.

In order to check whether SCIP indeed finds an optimal solution, the framework keeps track of both the dual problem and the primal problem. As we are considering a maximization problem, a feasible solution to the primal problem gives us a lower bound on the optimal objective value (primal bound) and similarly, a feasible solution

to the dual problem gives an upper bound (dual bound). SCIP terminates whenever the primal bound and the dual bound are equal. By the Strong Duality Theorem (1.5) we are guaranteed that an optimal solution is then found.

We shall use SCIP in order to solve instances of (mixed) integer linear programs. The problem of which instances will be tested is the following. We use the notation of Chapter 2.

$$\begin{array}{rlll} \max & \sum_{h\in H} \omega_h z_h & & \\ \text{subject to} & \sum_{r\in R} \pi_r x_r & \geq \kappa & \\ & \sum_{r\in R} \alpha_{hr} x_r & \leq \beta + M\cdot(1-z_h) & \forall h \in H \\ & x_r & \in \mathbb{N} & \forall r \in R \\ & z_h & \in \{0,1\} & \forall h \in H \end{array} \tag{4.1}$$

One must notice that we are considering a weighted version of the binding constraint and we therefore have introduced *capacities* π_r for all flight routines $r \in R$. Capacities represent the number of passengers that the aircraft type corresponding to a flight routine can carry. The motivation for introducing these parameters is as follows.

Some little tests of instances without the capacities seemed to have 'boring' solutions: only the aircraft type with lowest noise level was used. This is probably due to lack of real data. In this way there are no advantages of using noisier aircraft types, while in reality one would like to use larger aircrafts (with therefore probably larger noise levels). In order to let the noisier aircraft types be more attractive (profitable), we introduce the capacities, where a larger capacity corresponds to a larger noise level. The natural number κ in problem (4.1) then represents the *minimum number of passengers* that should be carried, instead of the minimum number of flights that should be executed.

We model problem (4.1) in the programming language ZIMPL (`http://zimpl.zib.de/`). This is an AMPL-like language, which looks a lot like plain mathematics. For example, it is possible to define sets in ZIMPL. Therefore it is very easy to implement a mathematical problem. Furthermore, the advantage of modeling in ZIMPL is that SCIP can directly read these models.

We use the ability of reading data from files in order to initialize all parameters. When generating an instance of problem (4.1), we also create a file `scip.dat` in which all data is stored in a format that is understood by the ZIMPL `read` command. When reading into files, empty lines and lines that start with a comment symbol (`comment`) are ignored. When calling the `read` command, one can also specify the number of readable lines that should be ignored (`skip`) and the number of lines that should be read (`use`).

An example of the file `scip.dat` can be found in Figure 4.1. This example consists of six flight routines and three housing locations with resp. 9, 4 and 13 houses. Furthermore, the threshold is 50.000, while 80.000 passengers must be carried.

```
#m
3

#n
6

#k
80000

#b
50000

#w[I]
1:    9         # location (7,10)
2:    4         # location (8,5)
3:    13        # location (8,8)

#p[J]
1:    400          # aircraft type 1
2:    400          # aircraft type 1
3:    400          # aircraft type 1
4:    300          # aircraft type 2
5:    300          # aircraft type 2
6:    300          # aircraft type 2

#a[I*J]
1,1:      192.913         # location (7,10)
1,2:      192.766         # location (7,10)
1,3:      192.463         # location (7,10)
1,4:      92.9126         # location (7,10)
1,5:      92.7664         # location (7,10)
1,6:      92.4626         # location (7,10)
2,1:      195.624         # location (8,5)
2,2:      195.219         # location (8,5)
2,3:      194.646         # location (8,5)
2,4:      95.6245         # location (8,5)
2,5:      95.2187         # location (8,5)
2,6:      94.6456         # location (8,5)
3,1:      194.037         # location (8,8)
3,2:      193.766         # location (8,8)
3,3:      193.336         # location (8,8)
3,4:      94.0368         # location (8,8)
3,5:      93.7659         # location (8,8)
3,6:      93.3356         # location (8,8)
```

Figure 4.1: Example file `scip.dat`.

ZIMPL only uses two data types: numbers and strings. They can be recognized by 'n' or 's'. Numbers can be all values such as binaries, integers and reals. For example, the weights of the housing locations are initialized by the following code.

```
param w[I] := read "scip.dat" as "<1n> 2n" comment "#" skip 4 use m ;
```

When executing the `read` command, every line in the input file `scip.dat` is interpreted as columns of text, separated by some whitespace or other punctuation marks. When reading a line like

```
3:    13       # location (8,8)
```

the first column `<1n>`, which is a number, is interpreted as the current argument of `w` that should be initialized. Arguments of array-like variables are denoted by `<>`. The second column `2n`, which is also a number, represents the value that is initialized. The result of the code above is then that `w[3]` is initialized by `13`.

The ZIMPL model of problem (4.1) is stored in the file `BigM_integer.zpl`, which can be found in Figure 4.2. Decision variables are by default non-negative reals. A detailed description of the ZIMPL programming language can be found in [Koc04].

4.2 Design of the environment

In this section, we shall introduce the environment of the airport and its flight routines. As explained in Chapter 2, a flight routine depends on four aspects: the type of aircraft, the runway, the procedure and the route. We shall explore most of these features, but will omit the aspect of procedures.

We will distinguish the types by two properties: noise level and capacity. By capacity we mean the number of passengers it can carry. Hereby we assume that the larger the capacity of an aircraft type, the larger the size (and weight) of the aircraft should be. Therefore it is reasonable to assume that an aircraft type with larger capacity has a larger noise level.

We will consider an airport with a single lane. This lane can be used in both directions, so that there are two runways, represented by the endpoints of the lane. Each aircraft type is assigned to one runway. Aircrafts start (or stop) at the corresponding endpoint of the lane. Although this might seem somewhat simple, there do exist many airports having this lay-out. This is even the case for commercial airports, for example Eindhoven Airport.

Once every aircraft type is assigned to a runway, there are three routes in which the aircraft can departure or arrive. For departures, the aircraft can either go straight on or bend to the left by 45° or bend to the right by 45°. Altogether, the number of flight routines is three times the number of aircraft types.

The situation is displayed in a grid, where the lane is positioned in the middle. We assume that distance is measured in kilometers. The number of points in the grid

BigM_integer.zpl

```
#--------------------------------------
#  MFS MODEL integer
#--------------------------------------

# PARAMETERS

param m := read "scip.dat" as "1n" comment "#" use 1 ;
param n := read "scip.dat" as "1n" comment "#" skip 1 use 1 ;

set I := { 1..m } ;
set J := { 1..n } ;

param k := read "scip.dat" as "1n" comment "#" skip 2 use 1 ;
param b := read "scip.dat" as "1n" comment "#" skip 3 use 1 ;
param w[I] := read "scip.dat" as "<1n> 2n" comment "#" skip 4 use m ;
param p[J] := read "scip.dat" as "<1n> 2n" comment "#" skip m+4 use n ;
param a[I*J] := read "scip.dat" as "<1n,2n> 3n" comment "#" skip m+n+4 use m*
    n ;

param M := 1.0e8 ;

# VARIABLES

var x[J] integer ;
var z[I] binary ;

# OBJECTIVE

maximize satisfied:
   sum <i> in I: w[i] * z[i] ;

# CONSTRAINTS

subto binding:
   sum <j> in J: p[j] * x[j] >= k ;
subto optionals:
   forall <i> in I:
      sum <j> in J: a[i,j] * x[j] <= b + M * (1 - z[i]) ;
```

Figure 4.2: ZIMPL model `BigM_integer.zpl`.

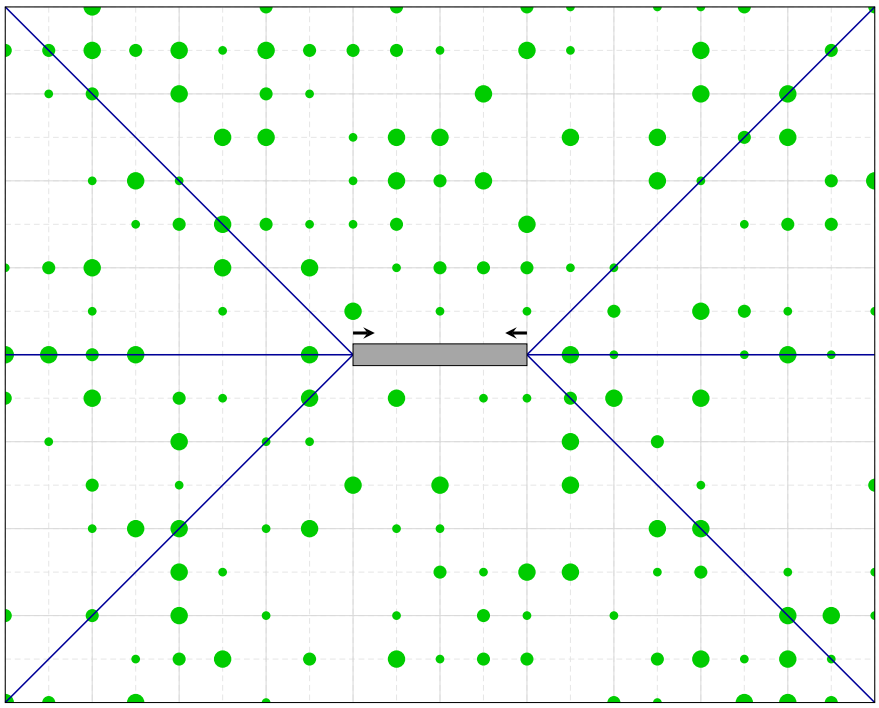

Figure 4.3: Airport environment (example): top view.

depends on the number of measure points per kilometer. Each grid point is represented by its coordinate, where the origin $(0, 0)$ is the bottom left corner of the grid.

Figure 4.3 shows an example of an environment for a grid with a width of 10 km, a height of 8 km, a lane of 2 km and two measure points per kilometer. This means that the grid consists of 21 points in horizontal direction and 17 points in vertical direction. The center of the grid (which is the center of the lane) has coordinate $(10, 8)$.

The green dots represent housing locations. The example containts 170 of them. The size of a dot depends on the number of houses (the weight) that are located there. The larger the dot, the larger the weight of the location. We have distinguished three dot sizes in the example.

The dimensions of the grid (in kilometers), the number of measure points per kilometer and the length of the runway will be parametrized. This also holds for the number of aircraft types and their properties. On the other hand, the housing locations will be created randomly.

The noise pollution of a flight routine depends on the type of aircraft (especially its noise level) and the route the aircraft uses. For any point in the grid, we also take into account the distance of the point from the used route. The exact calculation of noise pollution will be explored in the next section.

4.3 Source code design

In this section we shall describe the source code `generate_instance.cpp` that creates an instance for the model described in Section 4.1. This means that we have to create the data file `scip.dat` that contains all parameters defined in the model `BigM_integer.zpl`. We therefore have to implement the grid environment and the flight routines. The complete source code is downloadable via dropbox (no need to register). Here, we only mention the most important components. The programming language that we used is C++.

4.3.1 Defining classes

In order to implement the environment and the simulation of flight routines, we first define some classes.

```
struct GridData
{
   int steps ;
   int width ;
   int height ;
   int length ;
   int grid_width ;
   int grid_height ;
   int grid_mid_width ;
   int grid_mid_height ;
   int runway_ending_left ;
   int runway_ending_right ;
   Route routes[6] ;
} ;
```

```
struct AircraftData
{
   int nr_of_aircrafts ;
   double max_noise_level ;
   double min_noise_level ;
   double threshold ;
   int nr_of_locations ;
   int nr_of_houses ;
   int max_nr_houses ;
   int max_cap ;
   int min_cap ;
   int min_capacity ;
} ;
```

The class `GridData` consists of all data needed to draw the grid. These are the width `width` and height `height` of the grid (in km), the length `length` of the lane (in km) and the number `steps` of measure points per kilometer. In order to be able to point out the center of the grid (where the lane is placed), both numbers of grid points in horizontal (`grid_width`) and vertical (`grid_height`) must be odd. Therefore, we assure that `width`, `height` and `length` are (positive) even integers.

Furthermore, we store the center of the grid (`grid_mid_height`, `grid_mid_width`) and the lane, which ranges horizontally from `runway_ending_left` to `runway_ending_right`. Once the data elements `width`, `height`, `length` and `steps` have been initialized by the user, the other integer variables in `GridData` are easily computed by the following code (class name omitted).

```
grid_width = (width*steps) + 1 ;
grid_height = (height*steps) + 1 ;
grid_mid_width = 0.5*width*steps ;
grid_mid_height = 0.5*height*steps ;
```

```
runway_ending_left = grid_mid_width - 0.5*length*steps ;
runway_ending_right = grid_mid_width + 0.5*length*steps ;
```

The last element of the class `GridData` is an array of routes, where the type `Route` is defined as `vector<pair<int,int>>`, a vector of pairs of integers. In C++, a vector can be understood as an array of undefined length. Before initializing, any vector has length 0 and this length changes automatically whenever elements are added to or removed from the vector.

As we have two runways and for each runway three possible directions, there are six possible routes. These can be determined once all other data elements of `GridData` have been initialized. The pairs of integers that are stored in such a vector `Route` are the coordinates of the grid points that belong to that route. The lane itself is always part of a route. For example in Figure 4.3, an aircraft starting from the left endpoint of the lane and bending 45° to the left has route

$$\{(8,8),(8,9),(8,10),(8,11),(8,12),(7,13),(6,14),(5,15),(4,16),(3,17),(2,18),(1,19),(0,20)\},$$

in terms of C++ vectors, where the five first coordinates denote the runway.

In order to define the aircraft types, we store a number of data elements in the class `AircraftData`. The integer `nr_of_aircrafts` represents the number of aircraft types. The largest type has noise level `max_noise_level` and capacity `max_cap` and similarly, the smallest type has noise level `min_noise_level` and capacity `min_cap`. Also the threshold `threshold` of noise pollution allowed on each housing location (b in the model) and the minimum number of passengers `min_capacity` (k in the model) are stored in this class.

Furthermore, we store the number of housing locations `nr_of_locations`, the total number of houses `nr_of_houses`, which is the sum of the weights of the housing locations, and the maximal number of houses at a location `max_houses`. The latter number is used to determine the dot sizes in the pictures.

Although the classes `GridData` and `AircraftData` are used only once, we chose to collect the data into these classes. The reason for this is to reduce the number of arguments in the functions. A drawback is that the implementation of the functions themselves needs some space, as for each data element `data`, we have to type `class_name.data` in the source code everytime `data` is needed.

As mentioned before, for defining flight routines we explore the aspects of aircraft type, runway and route to define a flight routine. This easily constitutes a class named `FlightRoutine`. Since any aircraft type is determined by its noise level and capacity, we also introduce the class `AircraftType` storing these data.

```
struct FlightRoutine
{
   int number ;
   AircraftType type ;
   int runway ;
   Route route ;
} ;
```

```
struct AircraftType
{
   int number ;
   double noise_level ;
   int capacity ;
} ;
```

The last class we need for our implementation is the class `GridPoint`. As the name indicates, this class represents a point in the grid. For every grid point we store the number of houses `nr_of_houses` at that point. This is an integer greater than 0 whenever the grid point represents a housing location and is equal to 0 when it does not.

Although the threshold value `threshold` is already stored in the class `AircraftData`, it is convenient to do this again at every grid point (where a threshold of 0 means that the point does not represent a housing location). In this way, one could easily switch to different thresholds at every location.

Moreover, for every grid point we make a list of all noise pollution values produced by the different flight routines. As the number of flight routines is not fixed, we store these values in a vector `noise_pollution`.

```
struct GridPoint
{
   vector<double> noise_pollution ;
   int nr_of_houses ;
   double threshold ;
} ;
```

At this point we are ready to introduce the `main` function.

4.3.2 Exploring the `main` funtion

As already mentioned, we shall need one variable (`grid_data`) of type `GridData` and one variable (`aircraft_data`) of type `AircraftData`. Furthermore, we introduce a vector `aircraft_type` that contains all aircraft types, a vector `flight_routine` containing all flight routines and we store all grid points in a two-dimensional vector named `grid_point`.

After initilizing, `aircraft_type` will have size `aircraft_data.nr_of_aircrafts`. As each type can go into three directions, `flight_routine` will be a vector of length

$$3 \cdot \texttt{aircraft\_data.nr\_of\_aircrafts}.$$

The two-dimensional vector `grid_point` will have size

$$\texttt{grid\_data.grid\_height} \cdot \texttt{grid\_data.grid\_width}$$

and `grid_point[i][j]` denotes the grid point in row `i` and column `j`.

```
int main()
{
   GridData grid_data ;
   AircraftData aircraft_data ;

   vector<AircraftType> aircraft_type ;
   vector<FlightRoutine> flight_routine ;
   vector<vector<GridPoint>> grid_point ;

   make_grid_data (grid_data) ;
   make_aircrafts (aircraft_data, aircraft_type) ;
   make_flight_routines (grid_data, aircraft_type, flight_routine) ;
   calculate_noise_pollution (grid_data, flight_routine, grid_point) ;
   put_houses (grid_data, aircraft_data, grid_point) ;
   complete_data (grid_data, aircraft_data, grid_point) ;
   write_grid_file (grid_data, aircraft_data, aircraft_type, flight_routine,
       grid_point) ;
   write_scip_file (grid_data, aircraft_data, flight_routine, grid_point) ;
   write_tikz_source (grid_data, aircraft_data, grid_point) ;
   write_locations_file (grid_data, aircraft_data, flight_routine, grid_point
       ) ;
   write_rename_file (grid_data, aircraft_data, flight_routine) ;

 return 0 ;
}
```

We shall now go through the functions and highlight some important features. As there is only one variable of type `GridData` and one of type `AircraftData`, we will omit these class names when mentioning their elements.

The function

```
void make_grid_data (GridData& grid_data) ;
```

initializes all data elements of `grid_data`. This is done by asking the user to enter values for `width`, `height`, `length` and `steps`. The other integer elements of `grid_data` are initialized as explained before. For initializing the routes `routes[6]`, we add to every route the coordinates of the lane and then add to every route the other coordinates belonging to that route, untill we reach a boundary of the grid.

The first three routes belong to the runway corresponding to the right endpoint of the lane, the last three routes to the runway corresponding to the right endpoint of the lane. Furthermore, for each runway, the first of the three routes is the one bending left, the second is going straight on and the third is bending right.

The function

```
void make_aircrafts (AircraftData& aircraft_data, vector<AircraftType>&
    aircraft_type) ;
```

initializes some data elements of `aircraft_data`, namely those that define the aircraft types. Therefore, the function asks the user to enter the number of types and both the minimum and maximum noise level and capacity. The noise levels and capacities of all aircraft types are then computed by a linear function such that the first type has noise level `max_noise_level` and capacity `max_cap` and the last type has noise level `min_noise_level` and capacity `min_cap`. If there is only one type of aircraft, then the maximum capacity and noise level are assigned to this type. Initialization of the aircraft types is done by the following code.

```
if(aircraft_data.nr_of_aircrafts == 1)
{
   aircraft_type.push_back({0, aircraft_data.max_noise_level,
       aircraft_data.max_cap}) ;
}
else
{
   for(int i=0 ; i < aircraft_data.nr_of_aircrafts ; i++)
   {
      int nr = i ;
      double noise = aircraft_data.max_noise_level+i*(aircraft_data.
          min_noise_level-aircraft_data.max_noise_level)/(aircraft_data.
          nr_of_aircrafts-1) ;
      int pass = aircraft_data.max_cap + i * (aircraft_data.min_cap -
          aircraft_data.max_cap)/(aircraft_data.nr_of_aircrafts - 1) ;
      aircraft_type.push_back({nr,noise,pass}) ;
   }
}
```

Once the routes and aircraft types have been initialized, we can almost define the flight routines. The following function initializes the vector of flight routines `vector< FlightRoutine> flight_routine`.

```
void make_flight_routines (GridData grid_data, vector<AircraftType>
    aircraft_type, vector<FlightRoutine>& flight_routine) ;
```

The function first assigns every aircraft type to a runway. Every type thus can only use one runway to execute flight routines. The assignment is done randomly by using a *Mersenne Twister* pseudo-random number generator. The following code defines a uniform integer distribution on $\{0, 1\}$.

```
random_device rd ;
mt19937 generator(rd()) ;
uniform_int_distribution<int> uni(0,1) ;
```

The number 0 represents the runway corresponding to right endpoint of the lane and 1 the one corresponding to the left endpoint of the lane. When the aircraft types are assigned to the runways, the flight routines can be initialized. Three routines are defined for each aircraft type. This is done in such a way that the first of three has the route that is bending left, the second has the route that goes straight on and the third has the route that is bending right.

```
   for(int i = 0 ; i < aircraft_type.size() ; i++)
   {
      int runway_assignment = uni(generator) ;
      cout << "Type␣" << aircraft_type[i].number << "␣is␣assigned␣to␣runway␣"
           << runway_assignment << endl ;
      for(int dir = 0 ; dir < 3 ; dir++)
      {
         int nr = 3*i + dir ;
         flight_routine.push_back({nr, aircraft_type[i], runway_assignment,
             grid_data.routes[3*runway_assignment + dir]}) ;
      }
   }
```

Now that the flight routines are known, we can calculate the noise pollution of every routine on all grid points.

```
void calculate_noise_pollution (GridData grid_data, vector<FlightRoutine>
    flight_routine, vector<vector<GridPoint>>& grid_point) ;
```

This function consists of three **for**-loops: one for the rows of the grid, one for the columns of the grid (defining a coordinate of a grid point) and one for the flight routines.

Given a flight routine r, let N be the noise level of the aircraft type of routine r and let R be the route of routine r. Given furthermore a coordinate (i, j) in the grid and a coordinate (i', j') of R, point (i', j') contributes to the noise pollution on grid point (i, j) with $\max\left\{0, N - \|(i, j) - (i', j')\|\right\}$. The noise pollution of routine r on grid point (i, j) is then calculated by the average contribution of its route, i.e. the noise pollution on (i, j) is given by

$$\frac{\sum_{(i',j')\in R} \max\left\{0, N - \|(i, j) - (i', j')\|\right\}}{|R|}.$$

In the code, the noise pollution at a given grid point is calculated by the function `noise_pollution_at_point`. The function `dist(int point_1[2], int point_2[2], int steps)` calculates the Euclidean distance (in km) between `point_1` and `point_2`.

```
double noise_pollution_at_point(GridData grid_data, FlightRoutine routine,
    int point[2])
{
   double sum = 0 ;
   for(int index = 0 ; index < routine.route.size() ; index++)
   {
      int arr[2] = {routine.route[index].first, routine.route[index].second}
          ;
      sum += max(0.0, routine.type.noise_level - dist(arr, point, grid_data.
          steps)) ;
   }
   double pollution = sum/routine.route.size() ;

 return pollution ;
}
```

Notice that the function discussed above calculates noise pollution at *every* grid point, instead of only at housing locations. The function

```
void put_houses (GridData grid_data, AircraftData& aircraft_data, vector<
    vector<GridPoint>>& grid_point) ;
```

determines the housing locations and the weights of the locations, i.e. the number of houses on that location. This is both be done randomly by using uniform integer distributions, where again Mersenne Twisters are used as pseudo-random number generators. The function assures that there are no housing locations on the runway. Furthermore, the function stores the values of `nr_of_locations`, `nr_of_houses` and `max_nr_houses` in the variable `aircraft_data`.

The interval on which the uniform distributions are defined on is determined by the programmer. In Figure 4.3, for example, the probability of having a housing location is $\frac{1}{2}$ and the weights are uniformly taken from the set $\underline{30}$. For larger grids it might be convenient to have a smaller probability of having a housing location, because using this one might constitute into tenthousands of housing locations.

At this point, almost all parameters from the model presented in Section 4.1 are initialized. The function

```
void complete_data (GridData grid_data, AircraftData& aircraft_data, vector<
    vector<GridPoint>>& grid_point) ;
```

asks the user to enter values for the threshold `threshold` (the value of b) and for the minimum capacity that should be carried `min_capacity` (the value of k). If one would like to work with different thresholds at each housing locations, this function must be adapted.

The C++ program `generate_instance` terminates after creating several text files, which is done by the following functions.

```
void write_grid_file (GridData grid_data, AircraftData aircraft_data, vector<
    AircraftType> aircraft_type, vector<FlightRoutine> flight_routine, vector
    <vector<GridPoint>> grid_point) ;
void write_scip_file (GridData grid_data, AircraftData aircraft_data, vector<
    FlightRoutine> flight_routine, vector<vector<GridPoint>> grid_point) ;
void write_tikz_source (GridData grid_data, AircraftData aircraft_data,
    vector<vector<GridPoint>> grid_point) ;
void write_locations_file (GridData grid_data, AircraftData aircraft_data,
    vector<FlightRoutine> flight_routine, vector<vector<GridPoint>>
    grid_point) ;
void write_rename_file (GridData grid_data, AircraftData aircraft_data,
    vector<FlightRoutine> flight_routine) ;
```

The function `write_grid_file` writes all data to a 'human-readable' textfile `grid.dat`. The housing locations and noise pollution are shown in a matrix-like environment. The function `write_scip_file` creates the file `scip.dat` that is readable for the model `BigM_integer.zpl`, as explained in Section 4.1. The function `write_tikz_source` writes a `.tex` file that displays the problem environment, as in Figure 4.3.

The function `write_locations_file` creates a file called `locations.dat`, in which all data that are needed to draw a picture of the solution are stored. Running another program `generate_tikz_sol`, which reads into the file `locations.dat`, results in a `.tex` file that draws a picture as in Figure 4.3, using different colors for violated or satisfied housing locations, and moreover draws routines of the solution given by SCIP, using different colors for every aircraft type.

Remark 4.1. Because of differences in the modeling techniques in C++ (using vectors) and latex (using the tikz package), one must beware that all images created by the tikz package have to be vertically mirrored in order to obtain the correct view.

Finally, the function `write_rename_file` writes a file with terminal commands for renaming the data, solution and statistic files belonging to one instance of the problem. The new names of these files contain certain data of the problem, for example the number of flight routines and the number of housing locations.

Both the entire source codes `generate_instance.cpp` and `generate_tikz_sol.cpp` can be found online.

Chapter 5

Test results

In this chapter we present the results of several tests with SCIP. We generate an instance using the program `generate_instance` and feed this to SCIP. We focus on two parameters, namely the minimum capacity that should be carried (κ) and the threshold (β).

The main questions that we would like to answer are:

1. How do the solutions evolve when fixing all parameters except for κ and β?
2. How does SCIP perform in terms of running time?
3. How large (in terms of number of variables and number of constraints) could the instance be when requiring an 'acceptable' running time?
4. Could we reduce running time when relaxing on the integrality of some variables and still obtain 'good' solutions?
5. What happens with the solution when all housing locations are violated?

The last question arises from similar tests done in [Jan13]. There it seemed that whenever some housing locations were violated, all additional flights were taken over these locations (since they are violated anyway). In other words, the noise was not spread over the area. We would like to see whether SCIP treats such situations differently.

Furthermore, we will look close into the results to see whether other observations can be made.

In our tests, we shall use five different aircraft types for which Table 5.1 summarizes their properties. The runway of an aircraft type is given by a binary number. A 0 means that the type is belonging to the right endpoint of the lane, meaning that this type will be moving over the area on the West of the airport. The number 1 corresponds to the left endpoint of the lane, so that such a type moves over the East of the airport.

Every type is given a color, which corresponds to its color in all solution pictures later on in this chapter. Since every type can choose between three directions, the instances will have 15 flight routines.

Type	1	2	3	4	5
Noise level	300	250	200	150	100
Capacity	450	350	250	150	50
Runway	0	1	1	0	1

Table 5.1: Test instance: aircraft type properties.

We will consider different dimensions of the grid, but always have a runway of 4 km and we will put 2 measure points per kilometer. The number of houses at a housing location is a number between 1 and 30. The images in this chapter will distinguish three types of dot sizes to picture the weights of the housing locations. These are defined in Table 5.2.

All figures in this chapter represent top views of the surroundings of the airport. We shall only show a few images depicting solutions calculated by SCIP. We have too many test results to put all pictures in this document. A complete set of test results can be found online.

5.1 40x30km grid

For the first instance we consider a grid with a height of 30 km and a width of 40 km. The probability of having a housing location on a grid point is $\frac{1}{2}$. This instance has 2469 housing locations and a total of 38371 houses. The situation is depicted in Figure 5.1.

For this grid, we have executed 38 test routines. Tables 5.3 and 5.4 summarize the results. In Table 5.3, a blue number gives the current objective value and a gray number represents the running time of SCIP (in seconds). Cells are colored red when all housing locations are violated and green if all housing locations are satisfied. Some cases yielded long running times, in which we chose to terminate SCIP manually when the solving time exceeded one hour. These cells are given a yellow background.

The content of Table 5.4 needs some explanation: for every case (β, κ), the solution, as computed by SCIP, is written down. This means that every number in the table represents the number of times a certain flight routine is used in that case. Every cell contains a new table, where the numbers are shown corresponding to the solution figures. The column on the left represents flight routines that fly over the left-hand side of the airport environment. Since aircraft types 1 and 4 are assigned to runway 0, these types occur in this column. Furthermore, numbers are placed in rows that correspond

Dot size	Small	Medium	Big
Number of houses	1–10	11–20	21–30

Table 5.2: Test instance: dot sizes at housing locations in images.

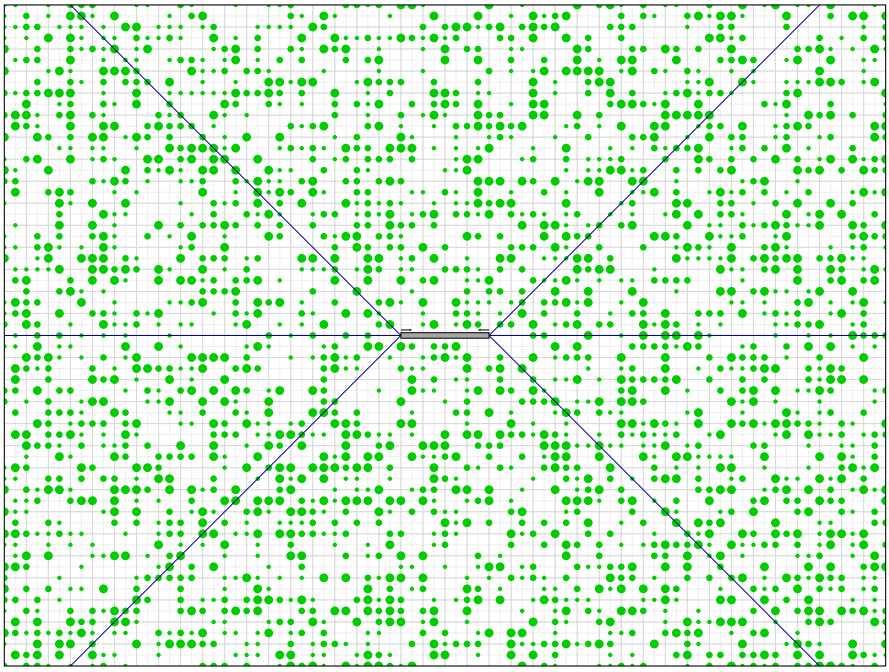

Figure 5.1: Airport environment 40x30km grid.

to the route of the flight routines. Flight routines that are heading to the right (from runway 0) are shown in the upper left cell of the table, flight routines that are going straight on appear in the middle left cell and flight routines that are heading to the left (from runway 0) are written down in the lower left cell of the table. Colors represent the types of aircraft, as defined in Table 5.1. A similar explanation holds for the column on the right.

When we look into Table 5.3, the first thing we observe is that there are 'twin columns', where the case of $\kappa = 1400$ is an exception. That is, there are subsequent columns that yield equal objective values. When looking into the solutions in Table 5.4, we see that in the most non-easy cases (cases where not all locations are either satisfied or violated), also the same solutions occur in two consecutive columns.

Considering the running time in non-easy cases, we see that when $\kappa \geq 1450$, the running time in the left element of a twin column is much larger than the running time in the right element of that twin column. Since the solutions coincide, it seems that, for example in the case $\kappa = 1450$, SCIP computes a solution that yields a total capacity of 1500, but then keeps trying to find an optimal solution with lower capacity. Either it eventually concludes that such solution does not exist, or we have stopped SCIP from running manually (resulting in the current solution).

In this case we see that a larger variation of results would have been obtained when we varied κ in bigger steps than 50.

β \ κ	1300	1350	1400	1450	1500	1550	1600	1650	1700
750	0 73.54	0 94.58							
800	93 72.81	93 122.35	0 58.08						
850	19117 666.92	19117 152.76	0 69.91						
900	38371 43.30	38371 1.35	5686 765.51	0 91.66	0 72.80				
950		38371 51.19	33460 703.90	5938 3764.62	5938 297.46	0 195.62	0 75.71		
1000			38371 0.56	33460 3261.06	33460 970.16	3099 3767.42	3099 182.37	0 75.64	0 130.86
1050				38371 1.03	38371 0.70	33460 3074.11	33460 503.92	3099 292.41	3099 125.44
1100						38371 7.06	38371 1.01	32156 2877.45	32156 311.09
1150								38371 16.30	38371 21.01

Table 5.3: Test results for 40x30km grid.

β \ κ	1300	1350	1400	1450	1500	1550	1600	1650	1700
750	10 9	3							
800	3	3	3 1						
850	3	3	2,1 1						
900	3	3	4	3 1	2 1,1				
950		3	2 2	3,1	3,1	3 1	3 1		
1000			4	2 1,1	2 1,1	3 1	3 1	3,2 1	3 3 2
1050				3 1	3 1	2 2	2 2	3 1	3 1
1100						1 3	2 2	2 1 1	2 1 1
1150								1 1 2	1 3

Table 5.4: Solutions calculated by SCIP for 40x30km grid

When looking into the running time even more, we see that the extreme cases are quickly determined. In the case of satisfying all locations (the green cells), SCIP terminated in most cases within one minute (or even within one second). In the case of violating all locations (the red cells), SCIP terminated within several minutes. This is not very surprising, since discovering infeasibility or feasibility results is not very hard for a (mixed) integer programming solver.

Furthermore, we see that when fixing some value of κ, there are only four different objective values possible (using the current thresholds). Figures 5.2 to 5.5 show the solutions for the case $\kappa = 1500$. Red dots represent violated housing locations, while the houses at green dots satisfy their threshold. The thickness of the solution lines shows the number of flight routines corresponding to these lines: a thicker line corresponds to a larger amount of its corresponding flight routine.

When searching for an optimal solution to our problem, it should be a good property for a feasible solution to 'spread the noise pollution' as much as possible. If this is done properly, every housing location (may) receive some noise pollution, hoping that the threshold is not exceeded. In the solution pictures this would mean that solution lines are drawn in as many different directions as possible.

In Figures 5.2 and 5.3, we do not see this behavior. All routines are going over the upper half of the airport environment in the first figure, while in the second case all routines are situated on the left-hand side of the area. In Figures 5.4 and 5.5, the routines are a bit more spread over the area, but not as much as one would expect in an optimal solution. One explanation for this behavior could be that the area to be considered is too small. Since the noise pollution of a flight routine depends on distances in the grid (in km) and the route of the routine, the noise pollution functions for different routines of a fixed type are all similar. This could be solved by considering a larger grid. In that case, distances are larger (reducing the contribution to the noise pollution) and routes consist of more elements (so that points close to the airport do not contribute that much to the noise pollution).

When looking at the solutions in Table 5.4 again, we conclude moreover that aircraft type 1 is used in almost every case, while type 5 is used only once. This could also be due to the small grid: since all points are relatively close to the airport, the pollution of type 5 (with noise level 100) to every grid point is very large compared to its capacity (50). When considering a larger grid, the noise pollution of every type reduces and type 5 will not produce as much pollution, especially on grid points that are somewhat distant from the airport.

We close this section by observing that in every solution (with exception of case $(750, 1300)$), every type is used at most 4 times and that the total number of flight routines is 8 at the most. We would like to see more varied and larger solutions. We also hope to solve this problem by looking at a larger grid.

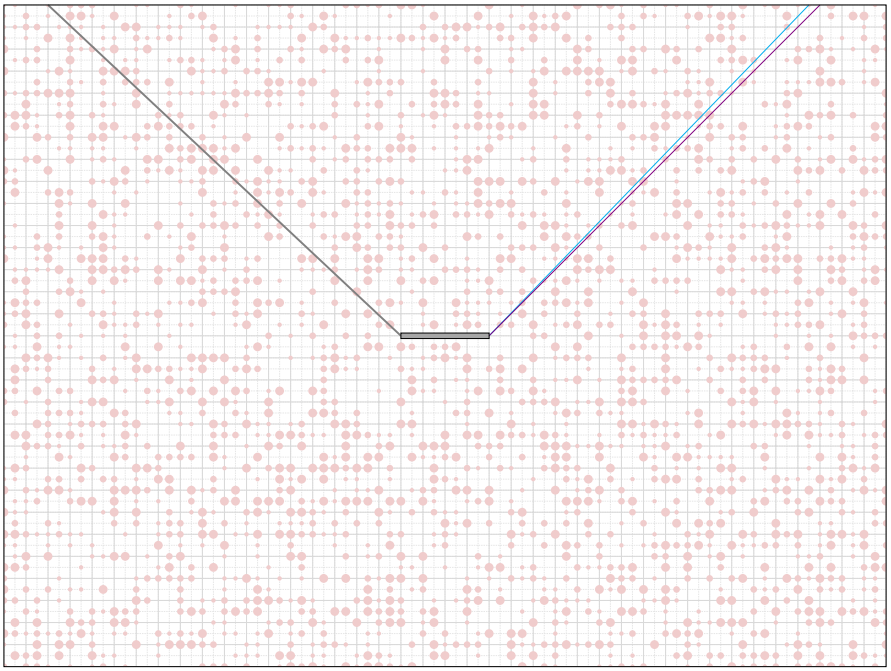

Figure 5.2: Solution: 40x30km grid, case (900, 1500).

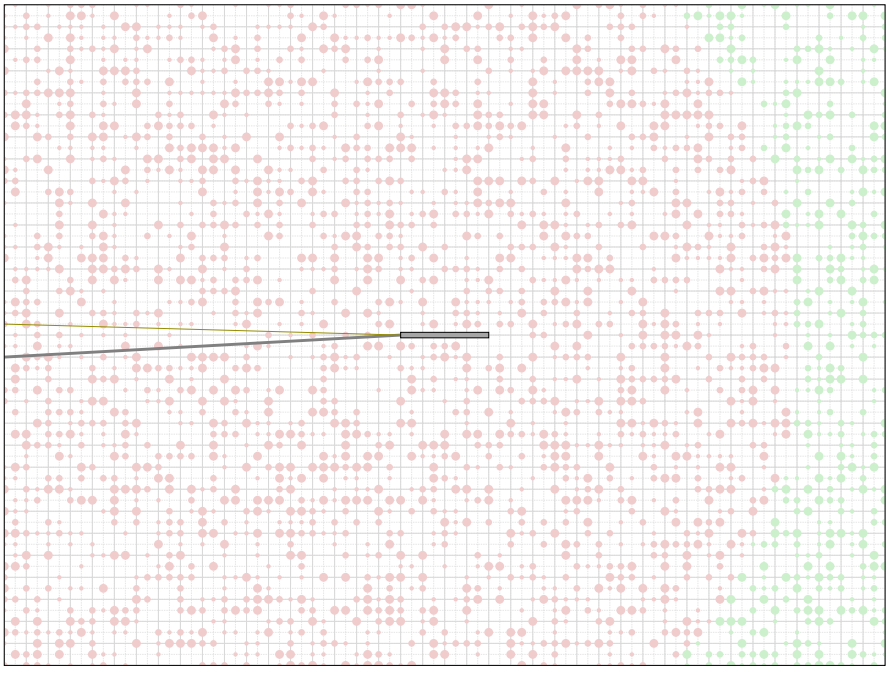

Figure 5.3: Solution: 40x30km grid, case (950, 1500).

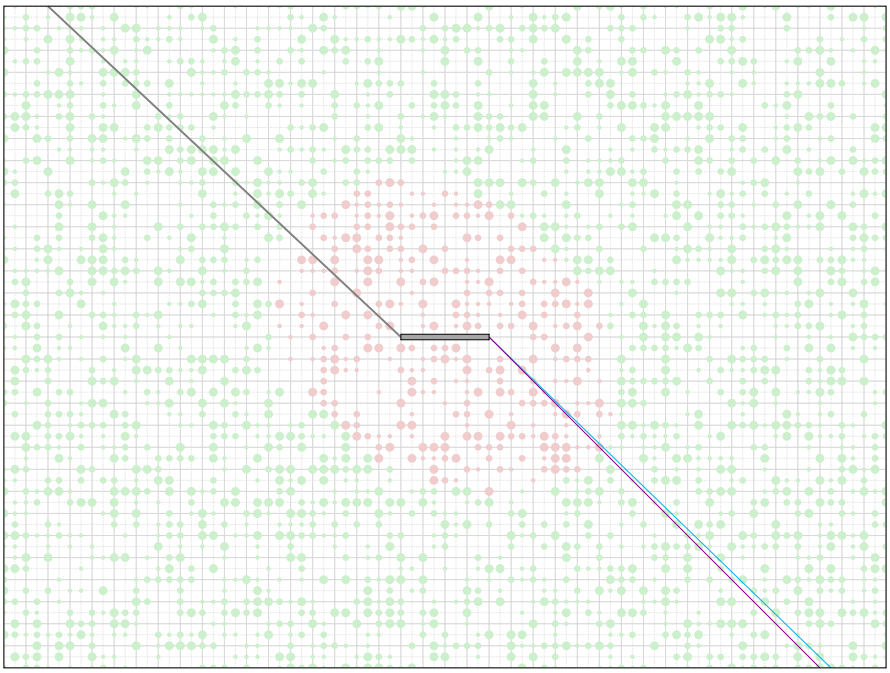

Figure 5.4: Solution: 40x30km grid, case $(1000, 1500)$.

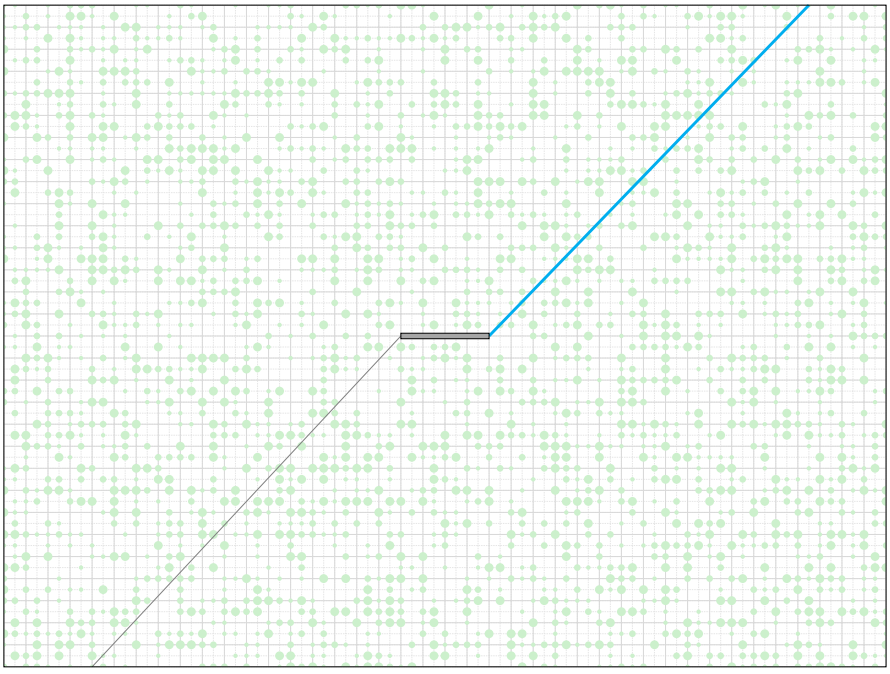

Figure 5.5: Solution: 40x30km grid, case $(1050, 1500)$.

5.2 100x80km grid

Having observed that a larger grid could yield more interesting results, we shall now consider a grid with a height of 80 km and a width of 100 km. Furthermore, we shall only consider values of κ that are multiples of 100.

In order to avoid having a huge amount of housing locations (optional constraints), we adapt the way of creating housing locations to a somewhat more realistic case. In a circle with a radius of 20 km around the midpoint of the grid, there are only a few housing locations: the probability of having a housing location is $\frac{1}{24}$. Outside this circle, the probability of having a housing location is $\frac{1}{8}$.

This instance has 3606 housing locations and a total of 56884 houses. The situation is given in Figure 5.6. We have executed 63 test routines for this situation.

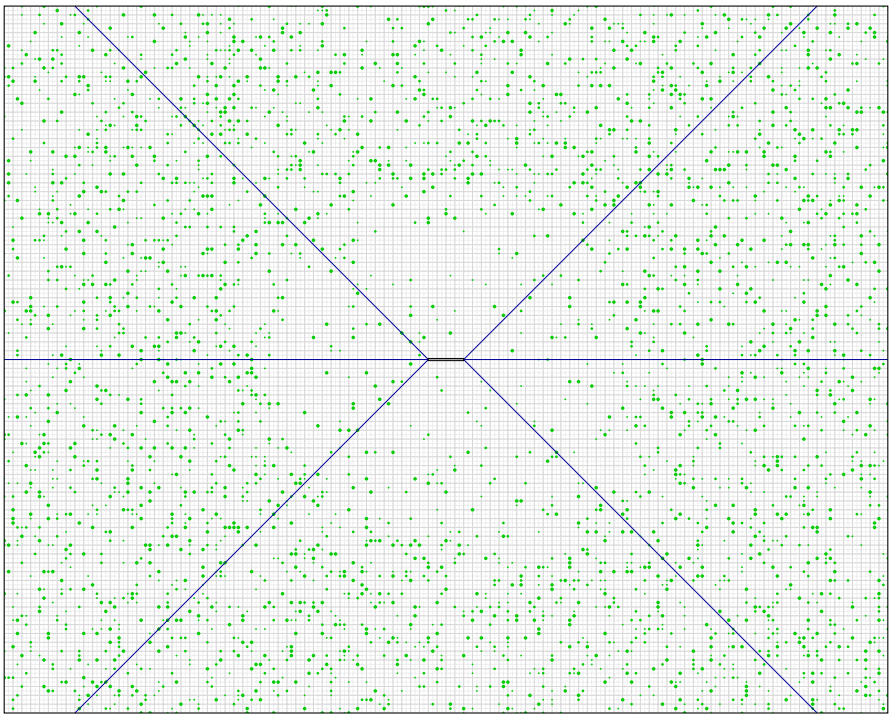

Figure 5.6: Airport environment 100x80km grid.

In this situation, the routes have much more elements and the noise pollution at a grid point is therefore different from the pollution calculated in the previous instance. Because of increased distances in the picture and size of the routes, the noise pollution at a grid point close to the lane will now probably be small compared to the previous instance. Therefore we expect that satisfaction and violation of all housing locations are achieved when having a smaller threshold.

Table 5.5 summarizes the test results for this instance.

β \ κ	1300	1400	1500	1600	1700
350	0 134.54	0 183.47			
400	121 129.79	30 150.80	0 145.39	0 113.96	
450	311 234.46	133 116.80	83 184.34	30 131.83	0 149.53
500	696 309.35	397 157.81	255 208.90	121 211.86	66 167.95
550	1113 315.95	720 183.10	482 184.13	311 326.93	155 193.44
600	1817 304.54	1263 203.40	778 190.43	630 169.27	311 134.11
650	2482 8652.23	1818 1422.02	1281 265.49	856 200.18	696 147.09
700	10790 15719.74	2482 3540.72	1853 3859.89	1413 191.70	901 195.09
750	25007 53001.50	10790 43235.47	2614 2964.02	1918 381.08	1413 307.34
800	47040 9731.00	24951 40848.09	11278 39408.87	2482 5763.27	1941 1517.33
850	56884 0.95	47040 12843.46	25007 62220.40	7205 119269.04	2482 28200.38
900		56884 1.21	47040 12667.30	23188 51026.72	7205 43739.77
950			56884 2.43	47040 6644.11	23188 34426.80
1000				56884 0.77	43498 19225.66
1050					56884 1.79

Table 5.5: Test results for 100x80km grid.

Our first observation is that the table yields some large running times (over ten hours). Sometimes we did terminate SCIP manually when it was running for more than about eleven hours. Though we find cases where we let SCIP run for about fifteen or even thirty-three hours, e.g. when it was running during the night. In practice these large running times are of course not desirable. The next section will therefore investigate how SCIP performs when relaxing on the integrality of some variables.

As with our previous instance, we observe that there are cases which are solved quickly: these are the green and red cells. When all locations can be satisfied, SCIP finds an optimal solution within seconds. When all locations must be violated, SCIP finds an optimal solution within three minutes.

Considering an area of bigger size, we indeed see that a larger variety of objective values exists for some fixed capacity (κ). The solution pictures in the case $\kappa = 1500$ can be found in Appendix C.

Table 5.6 gives the solutions calculated by SCIP. One of our goals by enlarging the size of the area (and therefore, refining the noise pollution functions) was to find more variated solutions. That is, scheduling a larger number of total flight routines and more variations on used aircraft types. Especially in every upper half column, we see that the total number of scheduled flight routines has become bigger (26–35), but they all have the same type (aircraft type 5). In every lower half column we again see that in most cases, aircraft type 1 is used a lot. We do also see a few solutions that have more variation in terms of aircraft types, for example in the cases $(700, 1400)$, $(800, 1600)$ and $(850, 1700)$.

Looking at the large numbers appearing in every upper half column of Table 5.6, we especially see that no effort was put in spreading the noise pollution over the area. When looking into the pictures for $\kappa = 1500$, we see that the flight routines are only spread over the area in the last two cases $\beta = 900$ and $\beta = 950$. Looking closer into Table 5.6, we conclude that there are only a few cases in which the noise pollution is spread over the area in the way one would expect. These are the cases where four different flight routines are all scheduled once and the corresponding lines are pointing into different directions. We see this happening in the cases $(800, 1300)$, $(850, 1400)$, $(900, 1500)$ and $(950, 1600)$. Of course, the optimal solutions in other cases do not have to be unique.

The sudden change in solutions in every column is interesting: at one moment, the solution exists of a large amount of flight routines with aircraft type 5, while the next moment (enlarging the threshold with 50), the capacity is reached by using only a few flight routines, but with larger aircraft types. We also observe that in every upper half column, there exists one solution that appears to be an optimal solution for several thresholds. Again, an optimal solution does not have to be unique.

Considering the large running times in quite some cases in Table 5.5, we seem to have reached an instance size that SCIP cannot handle anymore within 'acceptable' time,

β \ κ	1300	1400	1500	1600	1700
350	6,26 6,26 9 6	1 3 1,2			
400	25 1	27 1	2,3 3	2 1 2	
450	 1 25	28 	30 	32 	 3 1
500	 26	 28	 30	32 	33 1
550	26 	 1 27	 30	1 31	34
600	26 	28 	30 	 32	 1 34
650	5, 1 	28 	30 	32 	 34
700	 3,1	1,4, 1 	30 	32 	34
750	 3	 4	3,1 	32 	34
800	1 1 1 1	 4	3,1 	3,2, 1 	34
850	2 1,1	1 1 1 1	 3,1	 3 1	1,5, 2
900	3,1 	1 1,1 1	1 1 1 1	3 1 	 3 1
950			1 2 1	1 1 1 1	3 1
1000				2 2	2 1 1
1050					1 1 2

Table 5.6: Solutions calculated by SCIP for 100x80km grid.

which answers one of the questions we stated at the beginning of this chapter. Though it would be interesting to see how SCIP will perform when having a larger number of flight routines (we now only have fifteen of them). As we observe SCIP running into problems with such a large amount of optional constraints (which are the 3606 housing locations), one should probably test this with a small(er) amount of optional constraints.

The next section focuses on the huge running times. Two heuristics are applied to see how SCIP performs compared to the results obtained in the current section.

5.3 Solving LP-relaxations

As we experienced having large solving times (10–15 hours) in the previous section, we shall now relax on the integrality of the flight routines. Therefore we shall solve some cases where real-valued variables x_r are calculated. We use these as heuristics and see how they perform compared to the solutions of the integer program. Notice that we still have 3606 binary variables z_h, so that we have no guarantee of the problem becoming easy when only relaxing on the integrality of the flight routines (which are only 15 variables).

By rounding down these variables, we preserve satisfaction of optional constraints: if the noise pollution at a housing location satisfies the threshold with real-valued variables, then it still satisfies the threshold when rounding down the variables x_r to an integer. The only thing we have to take care of is the binding constraint. When rounding down variables, it is possible that the binding constraint $\sum_{r\in R} \pi_r x_r \geq \kappa$ is violated.

5.3.1 Adapting the binding constraint

By rounding down variables x_r to $\overline{x_r} := \lfloor x_r \rfloor \in \mathbb{N}$, we have $x_r - \overline{x_r} < 1$ for every r. Therefore, rounding down some variable x_{r^*} to $\overline{x_{r^*}}$ reduces the sum $\sum_{r\in R} \pi_r x_r$ by less then π_{r^*}. To assure satisfaction of the binding constraint when rounding down variables, we replace the binding constraint by the constraint

$$\sum\nolimits_{r\in R} \pi_r x_r \geq \kappa + \sum\nolimits_{r\in R} \pi_r.$$

In our instance, $\sum_{r\in R} \pi_r$ is equal to $3 \cdot (450 + 350 + 250 + 150 + 50) = 3750$, which is three times the sum of the capacities. If $\boldsymbol{x}$ satisfies the constraint above, then we have

$$\sum\nolimits_{r\in R} \pi_r \overline{x_r} = \sum\nolimits_{r\in R} \pi_r x_r + \sum\nolimits_{r\in R} \pi_r (\overline{x_r} - x_r) > \kappa + \sum\nolimits_{r\in R} \pi_r - \sum\nolimits_{r\in R} \pi_r = \kappa,$$

so that $\bar{\boldsymbol{x}}$ satisfies the original binding constraint, even with strict inequality. Notice that this method yields a somewhat rough estimate, but it is a necessary one to assure satisfaction of the original binding constraint.

Summarizing, when relaxing the integrality of the variables that represent the flight routines and adapting the binding constraint, we shall solve the following instance.

$$\begin{array}{rlll} \max & \sum_{h\in H} \omega_h z_h & & \\ \text{subject to} & \sum_{r\in R} \pi_r x_r & \geq \kappa + \sum_{r\in R} \pi_r & \\ & \sum_{r\in R} \alpha_{hr} x_r & \leq \beta + M \cdot (1 - z_h) & \forall h \in H \\ & x_r & \geq 0 & \forall r \in R \\ & z_h & \in \{0, 1\} & \forall h \in H \end{array} \tag{5.1}$$

Therefore we create a ZIMPL model `BigM_real.zpl` which is the model `BigM_integer.zpl` but removing `integer` for the variables `x[J]` and replacing `k` by `k + sum <j> in J: p[j]`.

We start this experiment by considering the case (800, 1700). Though this originally did not have a large running time (about 25 minutes), we are curious to see how the adapted model (5.1) performs anyway. The optimal solution calculated by SCIP of problem (5.1) in this case is the following.

3.9322	
3.9094,17.9868	

Since these values have a large fractional part, this is a good example of why the binding constraint should be adapted to the one in problem (5.1). SCIP found this solution within 237 seconds, which is much faster than the original running time of 25 minutes. The rounded solution

3	
3,17	

has a total capacity of $6 \cdot 450 + 17 \cdot 150 = 5250$, which is much more than the required 1700. Therefore, it results in an objective value of 0. Since the original integer solution yields an objective value of 1941, we can conclude that model (5.1) performs very badly for quickly finding good solutions for the original problem (4.1). We shall therefore not investigate the performance of this model any further.

5.3.2 Keeping the original binding constraint

Having observed that adapting the binding constraint in such a manner such that a feasible solution of the original problem is guaranteed does not perform well, we shall now allow the original binding constraint to be violated. Therefore we let SCIP calculate an optimal solution to the following instance.

$$\begin{aligned} \max \quad & \sum_{h\in H} \omega_h z_h & & \\ \text{subject to} \quad & \sum_{r\in R} \pi_r x_r & \geq \kappa & \\ & \sum_{r\in R} \alpha_{hr} x_r & \leq \beta + M\cdot(1-z_h) & \forall h \in H \\ & x_r & \geq 0 & \forall r \in R \\ & z_h & \in \{0,1\} & \forall h \in H \end{aligned} \tag{5.2}$$

We therefore create a ZIMPL model `BigM_relax.zpl` which is the model `BigM_integer.zpl` but only remove `integer` for the variables `x[J]`.

Let $\boldsymbol{x}$ be the optimal solution calculated by SCIP and let $\bar{\boldsymbol{x}}$ be the vector obtained by rounding down $\boldsymbol{x}$. Furthermore, let $\underline{\mathsf{OPT}}$ denote the optimal objective value of the original problem (4.1), OPT the optimal objective value of problem (5.2), induced by $\boldsymbol{x}$, and $\overline{\mathsf{OPT}}$ the objective value induced by $\bar{\boldsymbol{x}}$. Because of enlarging the solution range in problem (5.2) compared to problem (4.1), we know that OPT is at least $\underline{\mathsf{OPT}}$. Since rounding down a solution preserves satisfaction of optional constraints, we have the following relations:

$$\overline{\mathsf{OPT}} \geq \mathsf{OPT} \geq \underline{\mathsf{OPT}}.$$

Let $\bar{\kappa}$ be the total capacity of of the rounded solution $\bar{\boldsymbol{x}}$, i.e.

$$\bar{\kappa} = \sum_{r\in R} \pi_r \overline{x_r}.$$

Then we have $\bar{\kappa} \leq \kappa$. Moreover, let $\underline{\tau}$ be the original running time for problem (4.1) and τ the corresponding running time for the LP-relaxation (5.2).

When comparing a rounded solution $\bar{\boldsymbol{x}}$ to the original results in Table 5.5, we think of $\bar{\boldsymbol{x}}$ as a 'good' solution when:

(i) the running time of the LP-relaxation (5.2) is much smaller than the running time for (4.1);

(ii) there is only a little bit lost in terms of capacity (the original binding constraint is 'almost' satisfied);

(iii) having a little bit less capacity, much more housing locations satisfy their threshold.

In terms of ratios, we would like to see the following happen:

(i) $\tau/\underline{\tau}$ is at most 1 and in the best case it is much smaller than 1;

(ii) $\bar{\kappa}/\kappa$ is a number very close to 1;

(iii) $\overline{\mathsf{OPT}}/\underline{\mathsf{OPT}}$ is much larger than 1.

β \ κ	1400	1500	1600	1700
700	2482 3540.72	1853 3859.89	1413 191.70	901 195.09
750	10790 43235.47	2614 2964.02	1918 381.08	1413 307.34
800	24951 40848.09	11278 39408.87	2482 5763.27	1941 1517.33
850	47040 12843.46	25007 62220.40	7205 119269.04	2482 28200.38

Table 5.7: Recall some test results for 100x80km grid.

Let us recall some of the test results we obtained for the 100x80 grid from Table 5.5. These are given in Table 5.7. The red cells now correspond to cases (β, κ) of which the running time was at least three hours. The yellow cells still represent two cases in which we manually terminated SCIP after about eleven hours.

Most interesting to see is if the LP-relaxation performs better for the red and yellow cells in Table 5.7. However, to determine whether the LP-relaxation could be used for solving the originial integer program, we must also ensure that it remains to behave well in the easy solvable cases (the white cells).

We summarize the test results for the LP-relaxation in Table 5.8. In this table, the green cells show cases in which the running time has improved compared to the running time in Table 5.7. The yellow cells represent cases in which SCIP did not terminate within twelve hours and still yielded a large gap between the primal and dual bound.

β \ κ	1400	1500	1600	1700
700	2614 7271.23	1853 660.98	1413 275.05	901 374.98
750		2614 6296.89	1918 635.22	1413 251.00
800			2614 6721.15	1941 778.34
850	55649 2071.08			2614 5151.45

Table 5.8: Test results for the 100x80km grid LP-relaxation.

For all cases where a cell is colored yellow, we observe that using the LP-relaxation still does not perform well. SCIP did not terminate within twelve hours and therefore, it is not efficient in practice. Let us calculate the ratios $\tau/\underline{\tau}$, $\bar{\kappa}/\kappa$ and $\overline{\mathsf{OPT}}/\underline{\mathsf{OPT}}$ for the non-yellow cells. These calculations are summarized in Table 5.9.

The green cells of Table 5.8 occur in Table 5.9 when there is a number smaller than 1 in column $\tau/\underline{\tau}$. In three of these cases, we observe that the running time of the LP-relaxation is more than five times as fast as the running time of the integer program. These are very good results. However, in one of these cases, $(\beta, \kappa) = (850, 1400)$, almost half of the capacity is lost: the total capacity of the rounded solution $\bar{x}$ is 800, while the original binding constraint required a total capacity of 1400.

Considering the column $\tau/\underline{\tau}$ a bit further, we find two cases in which the running time of the LP-relaxation is more than two times the running time of the original integer program. This are the cases $(\beta, \kappa) = (700, 1400)$ and $(\beta, \kappa) = (750, 1500)$. These are bad results, since the original running times $\underline{\tau}$ already were almost one hour. For two other cases we find the running time τ to be at least 50% larger than the original running time $\underline{\tau}$. These are not very bad results, since the LP-relaxation was solved within eleven minutes anyway.

Looking into the column $\overline{\mathsf{OPT}}/\underline{\mathsf{OPT}}$, we identify a consistent trend in the results, exept for three cases. In two of the cases, the number of satisfied houses in the LP-relaxation is about three times as large as the original objective value $\underline{\mathsf{OPT}}$. However, these cases are exactly the two yielding a bad ratio $\tau/\underline{\tau}$ in running time. For the case $(\beta, \kappa) = (800, 1600)$, the new number of satisfied houses is even sixteen times the old objective value. In this case, the running time and original capacity are preserved quite well too.

κ	β	$\bar{\kappa}$	$\bar{\kappa}/\kappa$	$\underline{\tau}$	τ	$\tau/\underline{\tau}$	$\underline{\mathsf{OPT}}$	$\overline{\mathsf{OPT}}$	$\overline{\mathsf{OPT}}/\underline{\mathsf{OPT}}$
1400	700	1350	0.964	3540.72	7271.23	2.054	2482	7643	3.079
	850	800	0.571	12843.46	2071.08	0.161	47040	56884	1.209
1500	700	1450	0.967	3859.89	660.98	0.171	1853	2093	1.130
	750	1350	0.900	2964.02	6296.89	2.124	2614	7643	2.924
1600	700	1550	0.969	191.70	275.05	1.435	1413	1650	1.168
	750	1500	0.938	381.08	635.22	1.667	1918	2442	1.273
	800	1350	0.844	5763.27	6721.15	1.166	2482	39906	16.080
1600	700	1650	0.971	195.09	374.98	1.922	901	1113	1.235
	750	1650	0.971	307.34	251.00	0.817	1413	1673	1.184
	800	1700	1.000	1517.33	778.34	0.513	1941	1941	1.000
	850	1650	0.971	28200.38	5151.45	0.183	2482	3536	1.425

Table 5.9: Ratio calculations for comparison of LP-relaxation and integer program.

The overall performance of the ratio $\bar{\kappa}/\kappa$ is quite good, exept for the loss of about half of the capacity in the case $(\beta, \kappa) = (850, 1400)$. We highlight the case $(\beta, \kappa) = (800, 1700)$, where both the total capacity and the original objective value are preserved exactly. Looking into the solution, we conclude that SCIP calculated almost the original integer solution.

	34
	0.010

Summarizing, we have seen both some good results and some bad results in terms of running time, objective value and total capacity. But taking into account the cases where SCIP still did not terminate within twelve hours, we cannot advise to use the LP-relaxation (5.2) in order to solve the original integer problem (4.1).

5.4 All locations violated

We close this chapter by testing instances of which the objective value is zero, i.e. none of the optional constraints is satisfied. In this case, we are interested in the behavior of the calculated optimal solutions. We compare this behavior to test results in [Jan13], where it seemed that whenever some housing location was violated, all additional flights were taken over this location.

We investigated several instances by fixing one threshold, $\beta = 400$, while increasing the least number of passengers that should be carried. From previous results we know that when $\beta = 400$, an objective value of 0 is obtained for $\kappa \geq 1500$. We have therefore tested instances for $\kappa = 1500, 1600, \ldots, 2800$. Let us recall the capacities of the aircraft types (Table 5.10).

Type	1	2	3	4	5
Capacity	450	350	250	150	50

Table 5.10: Test instance: aircraft type capacities.

The optimal solutions that SCIP calculated for these instances are given in Table 5.11. One should mention once more that these optimal solutions are not unique. Actually, any soluting meeting the binding constraint

$$450 \cdot (x_1 + x_2 + x_3) + 350 \cdot (x_4 + x_5 + x_6) + 250 \cdot (x_7 + x_8 + x_9) + 150 \cdot (x_{10} + x_{11} + x_{12}) + 50 \cdot (x_{13} + x_{14} + x_{15}) \geq \kappa$$

is an optimal solution (since all optional constraints are violated anyway in this case).

The first thing that we observe is that it contains a varied set of solutions. This answers our question whether once all locations have been violated, additional flights are taken into the same direction: this is not the case.

κ	1500	1600	1700	1800	1900
	2,3 3	 2 1 2	 6 2,1, 4	4	8,38 13 8,38 8
κ	2000	2100	2200	2300	2400
	1,3 3 3 1	2 11	 20 5,44	 6 7 6,16 7,10	4 2, 5 1
κ	2500	2600	2700	2800	
	1 10 1	1, 5 1 1, 5 2	1,3 8,1 8,9,54 8 53	 2, 6 6 1	

Table 5.11: Solutions calculated by SCIP for $\beta = 400$.

What we do see is that SCIP does not at all ensure to spread the noise pollution over the area. Actually, there are cases where all flight routines are going into the same direction:

- northwards: $\kappa = 1800$;
- eastwards: $\kappa = 2200$;
- southwards: $\kappa = 1700$;
- westwards: $\kappa = 2400$.

Another thing that we observe is that the calculated solutions are arbitrary in some sense. Sometimes, the total capacity of a solution does not exceed the required κ (much). This happens for example when $\kappa = 1800$ and $\kappa = 2100$, where the total capacity of the solutions are resp. $4 \cdot 450 = 1800$ and $2 \cdot 250 + 11 \cdot 150 = 2150$. For most instances, the total capacity of the solution is much more than the needed capacity. This especially occurs for the cases $\kappa = 1900$, $\kappa = 2200$ and $\kappa = 2700$, in which only a number of resp. 38, 44 and 53 flight routines of type 5 would have been sufficient.

Conclusion and Discussion

In this thesis, we have considered a maximum feasible linear subsystem problem in order to model noise pollution around airports. Within this model, we were given some government measure points and a number of measure points at housing locations, all accompanied with a threshold. Furthermore, we were given all flight routines and for every routine, we knew the noise pollution of that routine at all measure points. We had to schedule flight routines such that the demand of either passenger volume or number of flight movements was achieved and such that the number of houses suffering from more noise pollution than the acceptable amount was minimized.

On the theoretical side, we have looked into some specific versions of this problem. For obtaining complexity results, we have considered four different decision variants, in which we could specify the demand k and the number of satsified optional constraints c. We have drawn some connections and equivalences between these decision problems.

For some constant values of k and c, we saw the problem was polynomial-time solvable. Thereafter, we specified values of k and c for which the problem was NP-complete. For strong NP-completeness, we have constructed polynomial-time reductions from the Maximum Independent Set Problem. After noticing that these reductions could be turned into cost-preserving polynomial-time reductions, we obtained an inapproximability result for the problem having special values of k.

On the practical side of this thesis, we have created a C++ program that generates instances for our maximum feasible linear subsystem problem. These instances model environments of an imaginary airport with one lane that can be used in both directions. The calculation of the noise pollution was done by a linear function depending on the noise level of the aircraft type and the route of the flight routine. We have modeled these instances according to the Big-M method in order to solve them by the (mixed) integer linear programming solver SCIP. The main goal was to see which instances could be optimally solved within an acceptable running time.

Considering a small grid with 2469 optional constraints, most instances were solved by SCIP within one hour. Looking into the solutions, we observed that noise was not spread over the environment and only one aircraft type was used a lot. Furthermore, there were a few different possible objective values for a fixed demand of passenger volume. Looking into a larger grid with 3606 optional constraints, a lot more different objective values were obtained, but in almost half of the cases similar solutions appeared. These

solutions also involved a single type of aircraft and again the noise was not spread over the area. On top of this, we observed that there were quite some cases for which the running time of SCIP was very large: ten to fifteen hours.

In order to solve the problem of huge running times, we relaxed on the integrality of the integer variables (still yielding 3606 binary variables coming from the Big-M method). We have looked into two relaxations: one on which we adapted the binding constraint (such that solutions to the relaxation would satisfy the binding constraint of the original problem) and one in which we kept the original binding constraint (allowing to violate this by a rounded integer solution). In the first case, we did not obtain meaningful results. In the second experiment, we did find some cases in which the relaxation performed quite well, but we also did find cases in which SCIP still did not find an optimal solution within twelve hours.

The instances that we used for testing in SCIP dit not model reality, because we did not have the knowledge to do so. It is therefore interesting for further research to consider instances that are closer to reality. This could be done when having real existing data for flight routines and noise pollution functions. In our model, we were forced to impose profits on each flight routine in order to make the noisier aircraft types more attractive for usage. When considering a better model, this problem might be solved, for example by having constraints of minimal usage and a distiction between arrivals and departures. A feature of our model that was not used at all in our tests, is the granuality of the grid (i.e. the number of measure points per kilometer). We did not exploit this feature because houses were located at random. When considering a real airport environment, the effect of different grid granualities could be investigated.

Considering the model that was given to SCIP, we have used one (very large) value for M in the Big-M method for all optional constraints. One might want to find out whether the runnig time of SCIP is affected by choosing smaller (but sufficiently large) different values M_i for each optional constraint i. Looking into our test results on relaxations, it seems that the binary variables produced by the Big-M method are the bottleneck for the running time of SCIP. Therefore, it would be interesting to study whether a different method can be used to model the optional constraints, without involving this many binary variables.

On the theoretical side, we have looked into the unweighted problem (in which the binding constraint states a demand for the number of flight movements instead of passenger volume) in particular. We have found some polynomial-time cases for this problem only when we assumed either k or c to be a constant. It would be interesting to see whether there exist polynomial-time solvable instances in which both k and c are not constants. Furthermore, we did not determine any polynomial-time cases for the weighted problem, so therefore one might try to find these. Finally, we left some open questions for determining complexity results for special values of c when having non-binary variables.

Bibliography

[ABC08] E. Amaldi, M. Bruglieri, and G. Casale. A two-phase relaxation-based heuristic for the maximum feasible subsystem problem. *Comput. Oper. Res.*, 35(5):1465–1482, 2008.

[Ach09a] T. Achterberg. *Constraint Integer Programming*. PhD thesis, Technische Universität Berlin, 2009.

[Ach09b] T. Achterberg. Scip: Solving constraint integer programs. *Mathematical Programming Computation*, 1(1):1–41, 2009. http://mpc.zib.de/index.php/MPC/article/view/4.

[AK95a] E. Amaldi and V. Kann. The complexity and approximability of finding maximum feasible subsystems of linear relations. *Theoretical Computer Science*, 147(1–2):181–210, 1995.

[AK95b] E. Amaldi and V. Kann. On the approximability of removing the smallest number of relations from linear systems to achieve feasibility. Technical report, Department of Mathematics, Swiss Federal Institute of Technology, Lausanne and, 1995.

[BT97] D. Bertsimas and J.N. Tsitsiklis. *Introduction to Linear Optimization*. Athena Scientific, 5th edition, 1997.

[Cha94] N. Chakravarti. Some results concerning post-infeasibility analysis. *European Journal of Operational Research*, 73(1):139–143, 1994.

[Chi96] J.W. Chinneck. An effective polynomial-time heuristic for the minimum-cardinality iis set-covering problem. *Annals of Mathematics and Artificial Intelligence*, 17(1):127–144, 1996.

[Chi01] J.W. Chinneck. Fast heuristics for the maximum feasible subsystem problem. *INFORMS J. on Computing*, 13(3):210–223, 2001.

[ERRS09] K. Elbassioni, R. Raman, S. Ray, and R. Sitters. On the approximability of the maximum feasible subsystem problem with 0/1-coefficients. In *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA

'09, pages 1210–1219, Philadelphia, PA, USA, 2009. Society for Industrial and Applied Mathematics.

[EUR14] EUROCONTROL. Seven-Year Forecast, Flight Movements and Service Units 2014 – 2020, February 2014.

[GC99] O. Guieu and J.W. Chinneck. Analyzing infeasible mixed-integer and integer linear programs. *INFORMS J. on Computing*, 11(1):63–77, 1999.

[GJ79] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1979.

[Gro13] Schiphol Group. Feiten en Cijfers 2013, 2013.

[Hås96] J. Håstad. Clique is hard to approximate within $n^{1-\varepsilon}$. In *Acta Mathematica*, pages 627–636, 1996.

[Jan13] T.M.L. Janssen. Noise minimization on houses around airports. Master's thesis, TU Delft, 2013.

[Koc04] T. Koch. *Rapid Mathematical Prototyping*. PhD thesis, Technische Universität Berlin, 2004.

[LN93] M. Luby and N. Nisan. A parallel approximation algorithm for positive linear programming. In *Proceedings of the Twenty-fifth Annual ACM Symposium on Theory of Computing*, STOC '93, pages 448–457, New York, NY, USA, 1993. ACM.

[MA04] R.T. Marler and J.S. Arora. Survey of multi-objective optimization methods for engineering. *Structural and Multidisciplinary Optimization*, 26(6):369–395, 2004.

[PR96] M. Parker and J. 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.

[PS82] C.H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1982.

[Sch14] G. Schäfer. Lecture Notes of the Master Course 'Discrete Optimization', Utrecht University, academic year 13/14.

[ST08] J.C. Smith and Z.C. Taşkın. A tutorial guide to mixed-integer programming models and solution techniques. In *Optimization in Medicine and Biology*. Taylor and Francis, Auerbach Publications, 2008.

[Vaz01] V.V. Vazirani. *Approximation Algorithms*. Springer-Verlag New York, Inc., New York, NY, USA, 2001.

[vD04] F.W.J. van Deventer. Basiskennis Geluidzonering Luchtvaart, 2003/2004.

List of Tables

List of Figures

Appendix A

Notation

Since some mathematical symbols can be understood differently, depending on the background of the reader, we establish the meaning of a few symbols here. This notation is used throughout the thesis. Below, we have $X \subseteq \mathbb{R}$, $n, m \in \mathbb{N}$, $r \in \mathbb{R}$ and $a \in X$.

$\mathbb{R}$, $\mathbb{Q}$, $\mathbb{Z}$ and $\mathbb{N}$	The sets of resp. real, rational, integer and natural numbers. The set $\mathbb{N}$ is the set of natural numbers *including* 0.
$\underline{n}$	The set $\{1, \ldots, n\}$.
$X^{\geq r}$	The set of numbers greater than or equal to r, i.e. $X^{\geq r} = \{x \in X \mid x \geq r\}$.
$\mathbb{R}^+$	The set of non-negative reals, i.e. $\mathbb{R}^+ = \mathbb{R}^{\geq 0}$.
$\mathbb{Q}^+$	The set of non-negative rationals, i.e. $\mathbb{Q}^+ = \mathbb{Q}^{\geq 0}$.
X^n	The Cartesian product of X (n copies).
$\boldsymbol{x} = (x_1, \ldots, x_n)^\mathsf{T}$	Element of the set X^n, also named *vector*. Notice the difference between the use of bold font for vectors and normal font for components.
$\boldsymbol{a}$	The vector $(a, \ldots, a)^\mathsf{T} \in X^n$ with constant value a.
$\boldsymbol{A} \in X^{m \times n}$	$\boldsymbol{A}$ is a matrix consisting of m rows and n columns. Its entries are elements of X.
$\lvert X \rvert$	The cardinality of X.
$X \subseteq Y$	The set X is a subset of Y and $X = Y$ is a possibility.
$X \subset Y$	The set X is a *strict* subset of Y.

Appendix B

NP-completeness of EP and Dec-MCP

In Chapter 3, we rely on the NP-completeness of some decision problems. For famous problems like MIS, we assume that the reader already has seen an argument proving the problem to be NP-complete. In this appendix, we provide proofs of NP-completeness for the *Equipartition problem* (EP) and the decision variant of the *Maximum constraint partition problem* (MPC). Let us recall their definitions.

Equipartition problem (EP)

Instance:	Set $A \subset \mathbb{N}^{\geq 1}$ of even cardinality, such that $\sum_{a \in A} a = 2B$.
Question:	Does there exist a subset $A' \subseteq A$ such that $\lvert A' \rvert = \frac{1}{2}\lvert A \rvert$ and $\sum_{a \in A'} a = \sum_{a \notin A'} a = B$?

Garey and Johnson, [GJ79], state that this problem is (weakly) NP-complete, but their book does not give a proof.

Maximum constraint partition problem (MCP)

Instance:	Finite set A, subset $S \subseteq A$, sizes $s(a) \in \mathbb{N}^{\geq 1}$ $(\forall a \in A)$ such that $\sum_{a \in A} s(a) = 2B$, special element $a_0 \in A$.
Goal:	Find a subset $A' \subseteq A$ such that $a_0 \in A'$, $\sum_{a \in A'} s(a) = \sum_{a \notin A'} s(a) = B$ and such that $\lvert S \cap A' \rvert$ is maximized.

We turn MCP into a decision problem by fixing a number K and replacing the goal in the above definition the question: does there exist a subset $A' \subseteq A$ such that $a_0 \in A'$, $\sum_{a \in A'} s(a) = \sum_{a \notin A'} s(a) = B$ and such that $\lvert S \cap A' \rvert = K$?

We prove EP and Dec-MCP to be NP-complete by giving a polynomial-time reduction from the *Partition problem* (PART).

Partition problem (PART)

Instance: Set $A = \{a_1, \ldots, a_r\} \subset \mathbb{N}^{\geq 1}$ such that $\sum_{i=1}^{r} a_i = 2B$.
Question: Does there exist $I \subseteq \underline{r}$ such that $\sum_{i \in I} a_i = \sum_{i \notin I} a_i = B$?

PART is a well-known weakly NP-complete problem.

Proposition B.1. The Equipartition Problem is weakly NP-complete.

Proof. Any certificate to a yes-instance of EP is just a set $A' \subseteq A$. We can in polynomial time compute the sums $\sum_{a \in A'} a$ and $\sum_{a \notin A'} a$ and check if they are equal. Also we can check if $|A'| = \frac{1}{2}|A|$ in polynomial time. Therefore, EP $\in$ NP.

Now we reduce PART to EP. Given an instance $A = \{a_1, \ldots, a_r\}$ of PART, we define $a_{r+i} = 2B \cdot a_i$ for every $i \in \underline{r}$. We then construct the set $E(A) = A \cup \{a_{r+i} \mid i \in \underline{r}\}$. Clearly $|E(A)| = 2r$ and we have

$$\sum_{a \in E(A)} a = \sum_{a \in A} a + 2B \sum_{a \in A} a = 2B + 4B^2 = 2(B + 2B^2).$$

Hence $E(A)$ is a well-defined instance for EP. We prove that we have a yes-instance for PART if and only if the constructed instance for EP is a yes-instance.

Suppose we have a yes-instance for PART, i.e. a set $I \subseteq \underline{r}$ such that $\sum_{i \in I} a_i = \sum_{i \notin I} a_i = B$. Then consider

$$E(I) = I \cup \{r + i \mid i \notin I\}.$$

Then clearly $|E(I)| = r$ and we have $\sum_{i \in E(I)} a_i = \sum_{i \in I} a_i + \sum_{i \notin I} a_{r+i} = B + 2B \sum_{i \notin I} a_i = B + 2B^2$. Hence $E(I)$ solves the given EP.

Now suppose the given EP is a yes-instance. Then we have a set $E = I \cup J$ with $|E| = r$, $I \subseteq \underline{r}$ and $J \subseteq \underline{2r} \setminus \underline{r}$ and such that $\sum_{i \in E} a_i = \sum_{i \notin E} a_i = B + 2B^2$. Let us first see why I can not be equal to $\underline{r}$. Suppose it is, then we have

$$\sum_{i \in E} a_i = \sum_{i \in I} a_i + 2B \sum_{i \in J} a_{i-r} = 2B + 2B \sum_{i \in J} a_{i-r}$$

and

$$\sum_{i \notin E} a_i = \sum_{i \in \underline{r} \setminus I} a_i + 2B \sum_{i \in \underline{2r} \setminus J} a_{i-r} = 2B \sum_{i \in \underline{2r} \setminus J} a_{i-r},$$

from which we can conclude $\sum_{i \in J} a_{i-r} + 1 = \sum_{i \in \underline{2r} \setminus J} a_{i-r}$. But then we have

$$2B = \sum_{a \in A} a = \sum_{i \in \underline{r}} a_i = \sum_{i \in J} a_{i-r} + \sum_{i \in \underline{2r} \setminus J} a_{i-r} = 2 \sum_{i \in J} a_{i-r} + 1,$$

which is a contradiction.

Consider the set I and suppose that it isn't a solution for PART. Without loss of generality we assume $\sum_{i \in I} a_i > \sum_{i \in \underline{r} \setminus I} a_i$. Then

$$0 < \sum_{i \in I} a_i - \sum_{i \in \underline{r} \setminus I} a_i < 2B,$$

where the last inequality holds since I cannot contain all of $\underline{r}$. But from $\sum_{i\in E} a_i = \sum_{i\notin E} a_i$ we also have

$$\sum\nolimits_{i\in I} a_i - \sum\nolimits_{i\in \underline{r}\setminus I} a_i = \sum\nolimits_{i\in \underline{2r}\setminus J} a_i - \sum\nolimits_{i\in J} a_i$$

and then the above inequality contradicts the construction of $E(A)$: for every $i \in \underline{2r} \setminus \underline{r}$ the number a_i is a multiple of $2B$, thus also the difference $\sum_{i\in \underline{2r}\setminus J} a_i - \sum_{i\in J} a_i$ should be a multiple of $2B$. Hence I must solve PART.

Since PART is weakly NP-complete, the result follows. □

Proposition B.2. Dec-MCP is weakly NP-complete.

Proof. Given some certificate $A' \subseteq A$, we can check in polynomial-time whether A' meets the requirements. Therefore, Dec-MCP $\in$ NP. For proving NP-hardness, we again consider the weakly NP-complete problem PART. Suppose we are given an instance of PART, which is a set $W = \{a_1, \ldots, a_r\} \subset \mathbb{N}^{\geq 1}$ such that $\sum_{i=1}^{r} a_i = 2B$. Then we construct an instance for MCP by defining the set $A = \underline{r}$ and sizes $s(i) = a_i$, pick $a_0 \in A$ arbitrarily and define $S = \{a_0\}$.

We shall now prove that yes-instances correspond. That is, we must prove that we have a yes-instance for PART if and only if the above defined instance for MCP is a yes-instance satisfying $|S \cap A'| = 1$.

Suppose we have a solution A' to MCP such that $|S \cap A'| = 1$. Then by assumption we have $\sum_{i\in A'} a_i = \sum_{i\in A'} s(i) = B$, hence also $\sum_{i\notin A'} s(i) = B$. Therefore A' solves PART.

So suppose now we have a solution to PART. That is, we have a set $I \subseteq \underline{r}$ such that $\sum_{i\in I} a_i = \sum_{i\in \underline{r}\setminus I} a_i = B$. Pick $A' \in \{I, \underline{r} \setminus I\}$ such that $a_0 \in A'$. Then A' is subset of A and we have $\sum_{i\in A'} s(i) = \sum_{i\in I} a_i = B$ or $\sum_{i\in A'} s(i) = \sum_{i\in \underline{r}\setminus I} a_i = B$. Moreover, $S \cap A' = S$ and thus $|S \cap W| = 1$. Hence, A' solves Dec-MCP. □

Appendix C

Solutions 100x80km grid (k=1500)

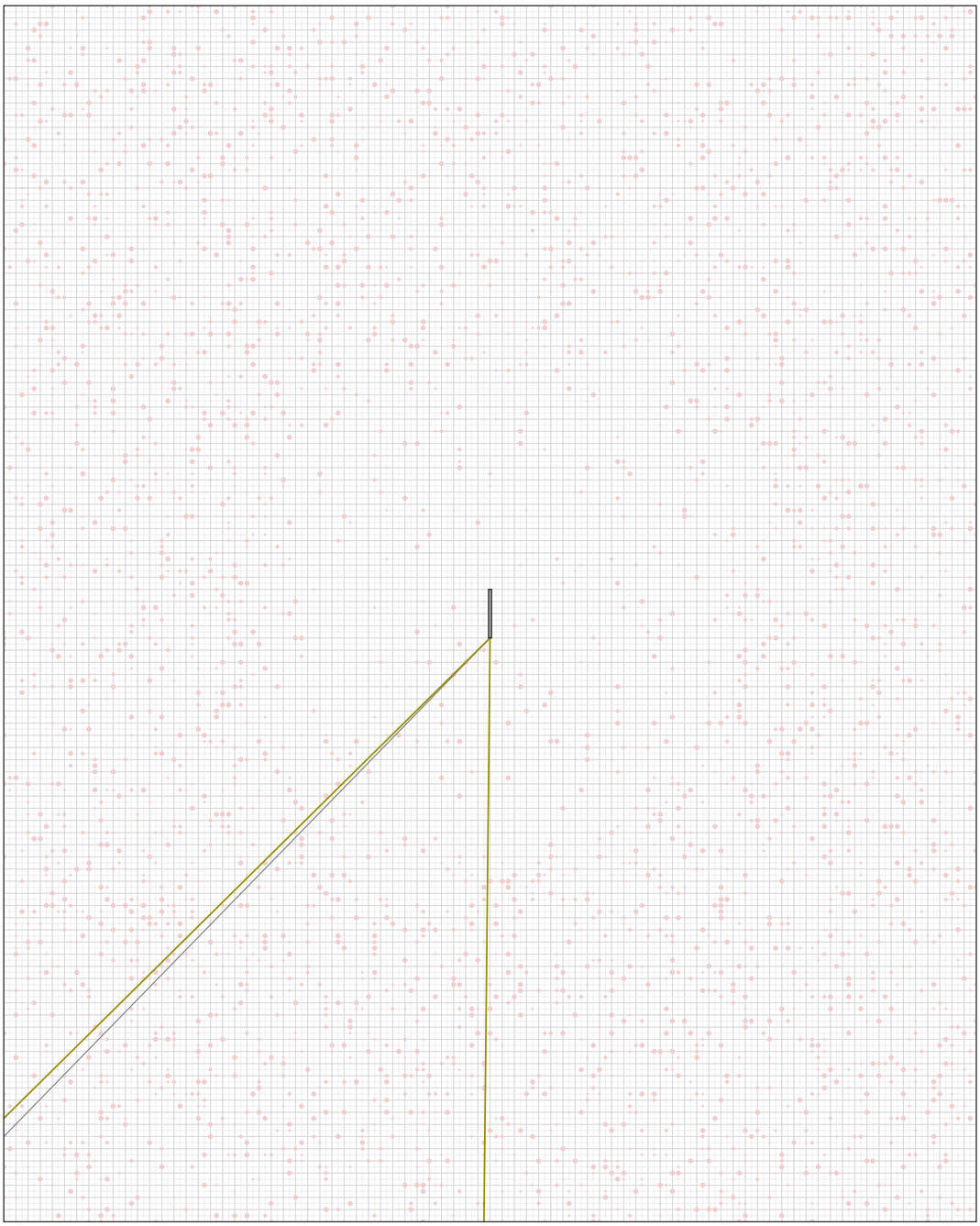

Figure C.1: Solution: 100x80km grid, case (400, 1500).

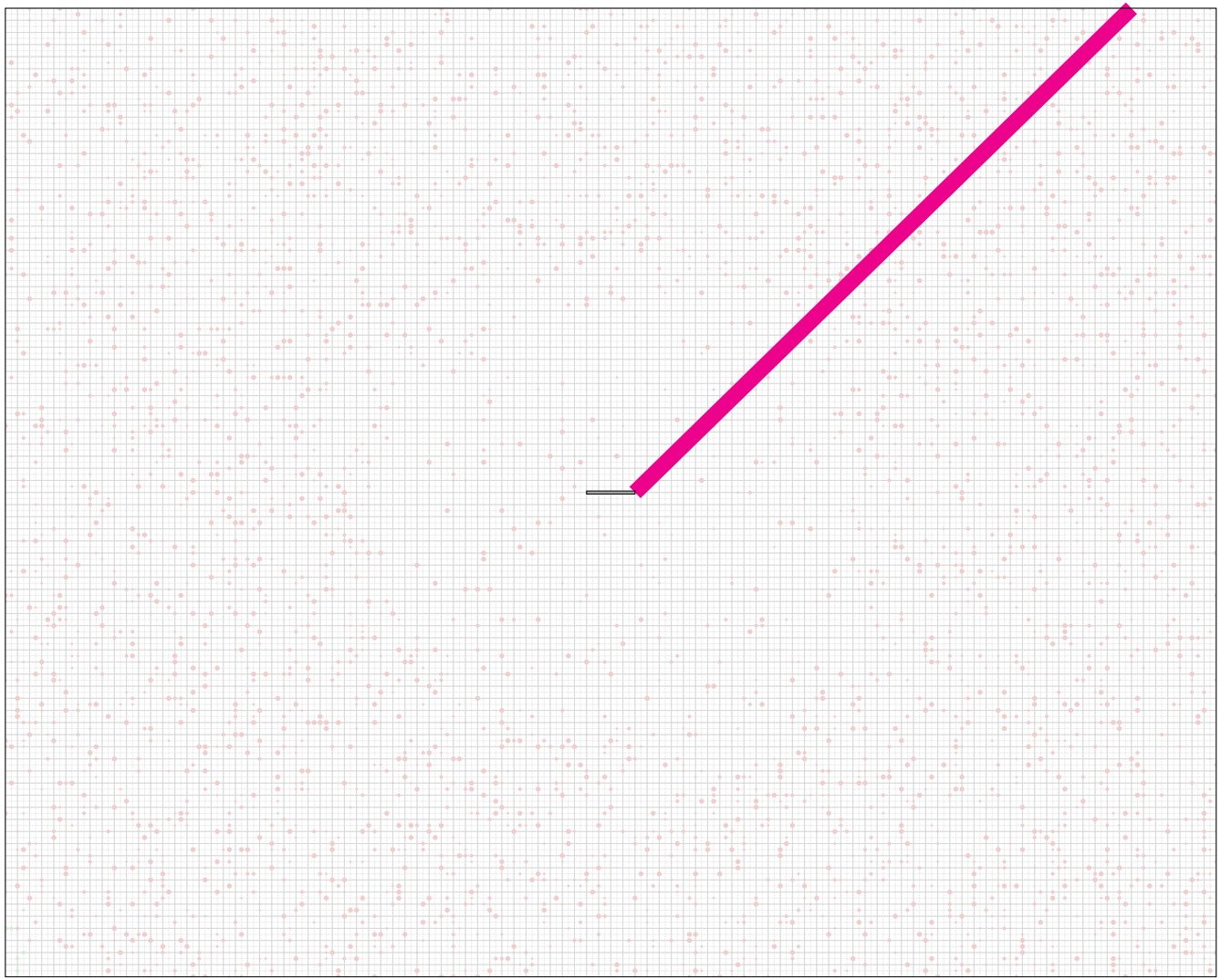

Figure C.2: Solution: 100x80km grid, case $(450, 1500)$.

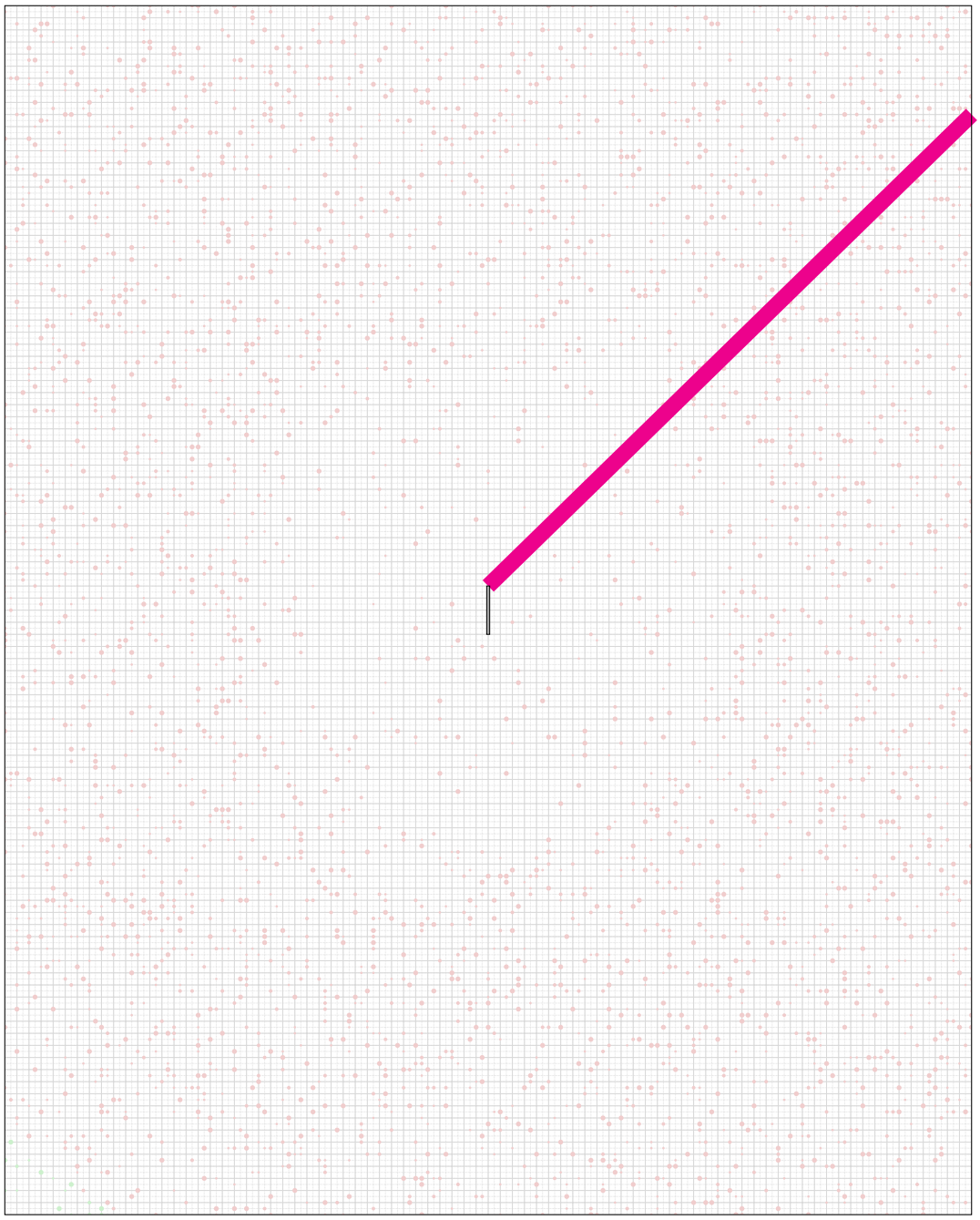

Figure C.3: Solution: 100x80km grid, case (500, 1500).

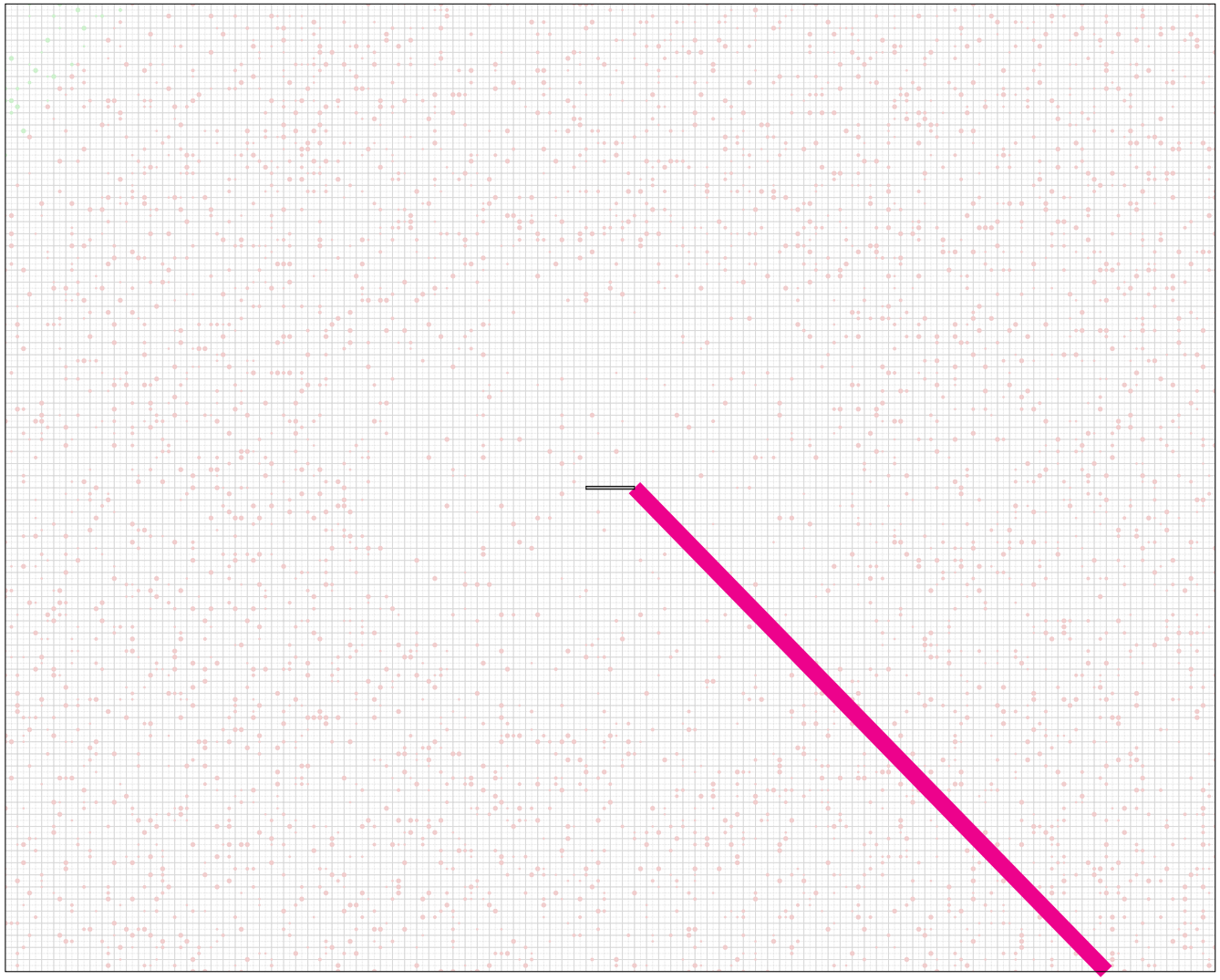

Figure C.4: Solution: 100x80km grid, case (550, 1500).

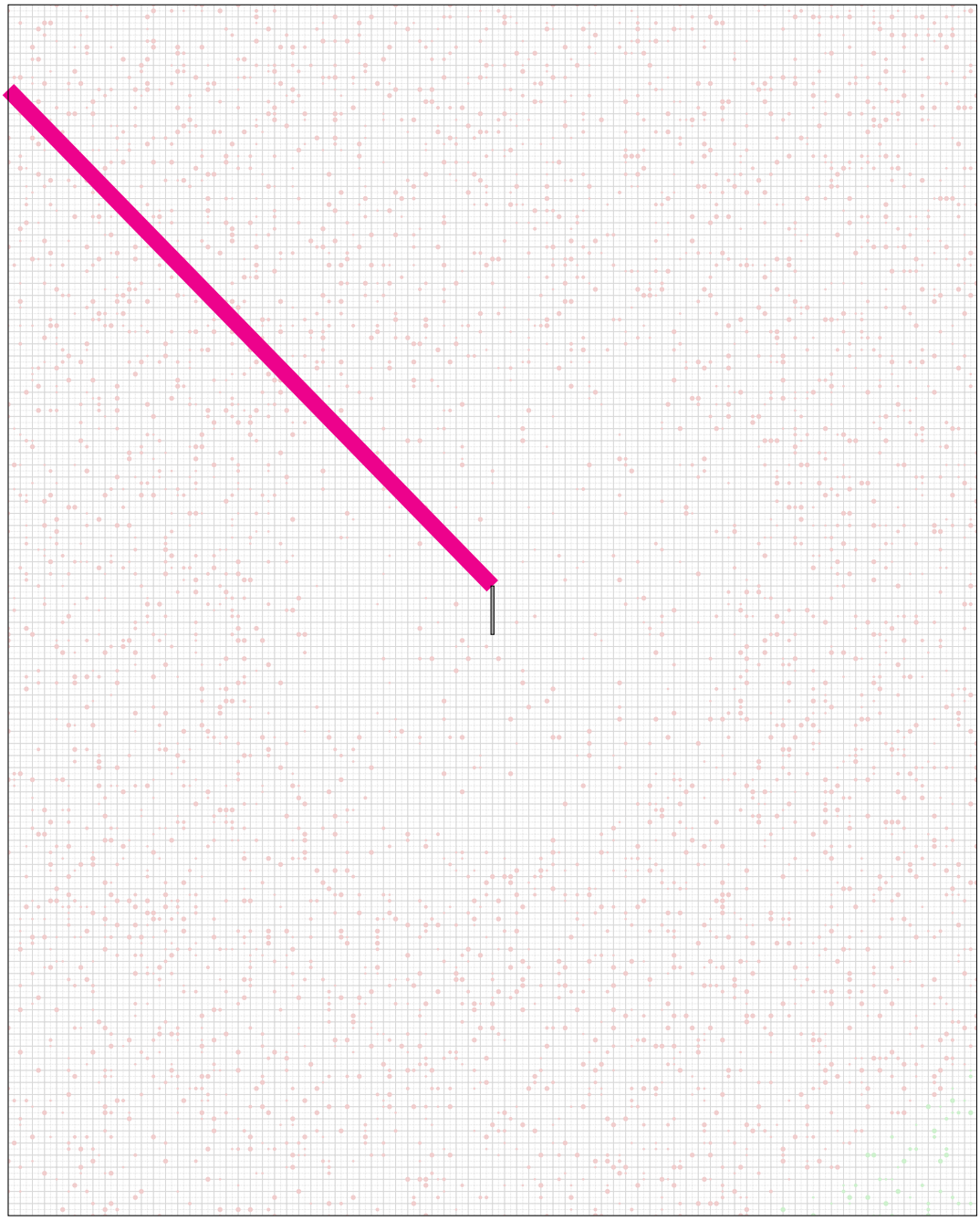

Figure C.5: Solution: 100x80km grid, case (600, 1500).

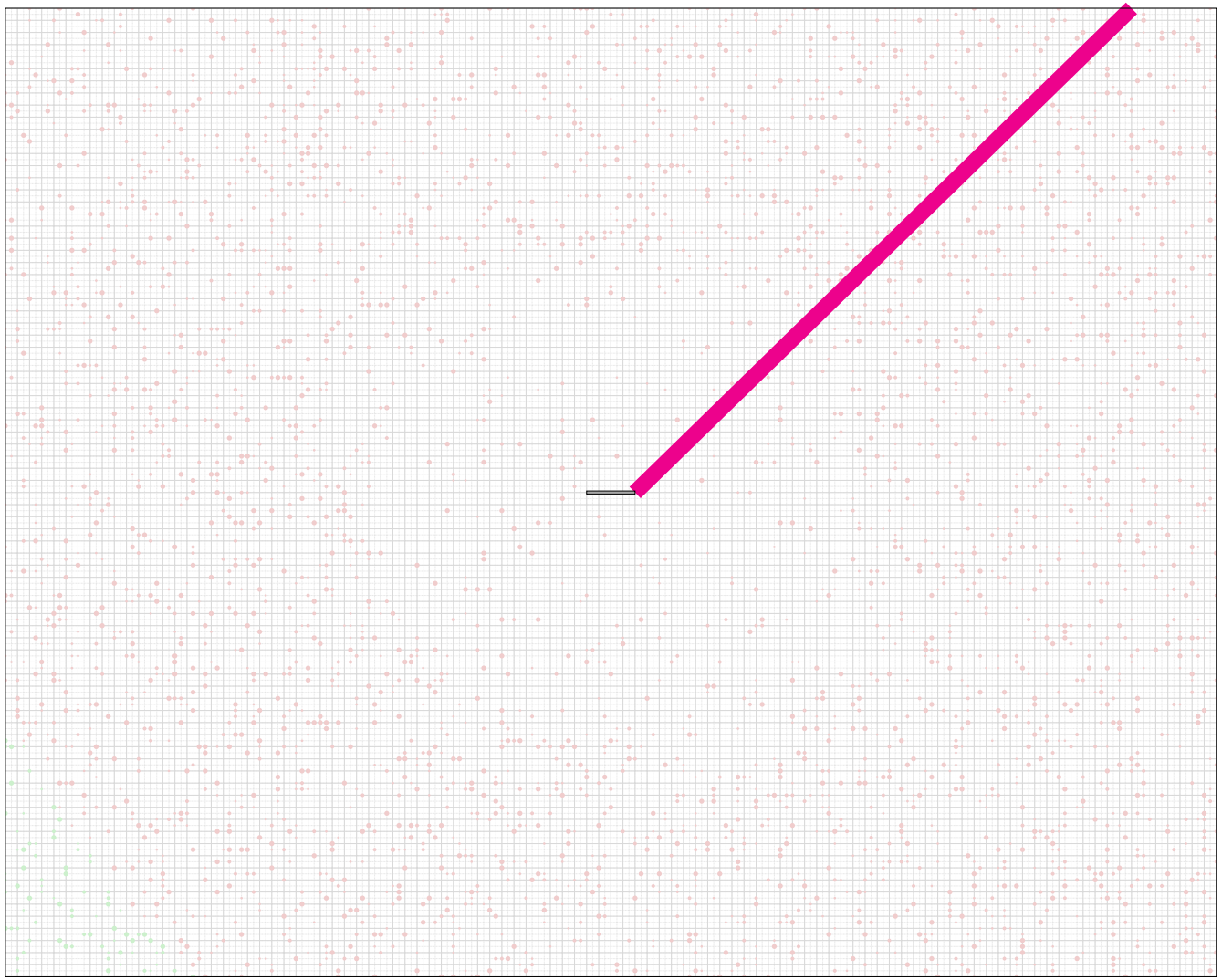

Figure C.6: Solution: 100x80km grid, case (650, 1500).

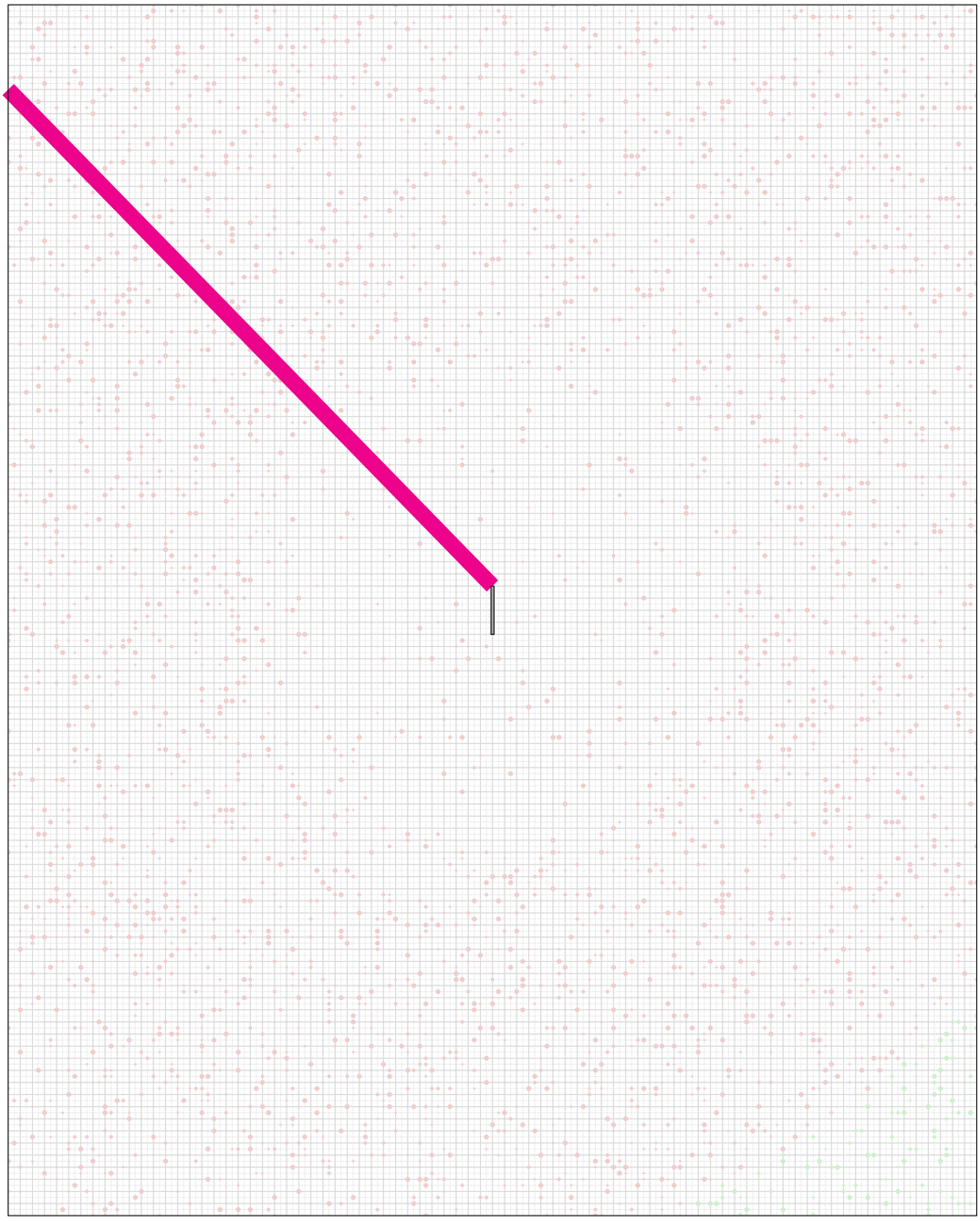

Figure C.7: Solution: 100x80km grid, case (700, 1500).

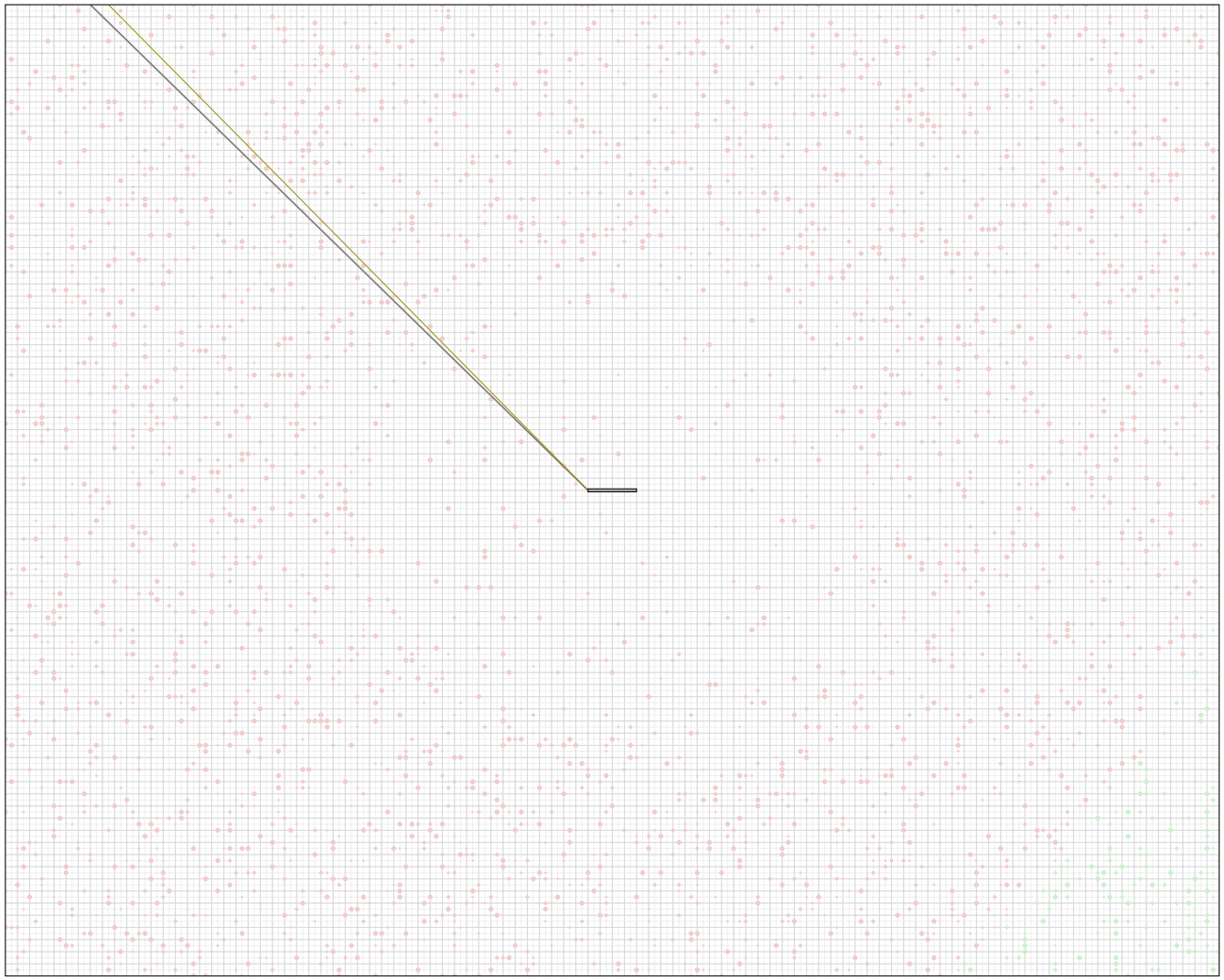

Figure C.8: Solution: 100x80km grid, case (750, 1500).

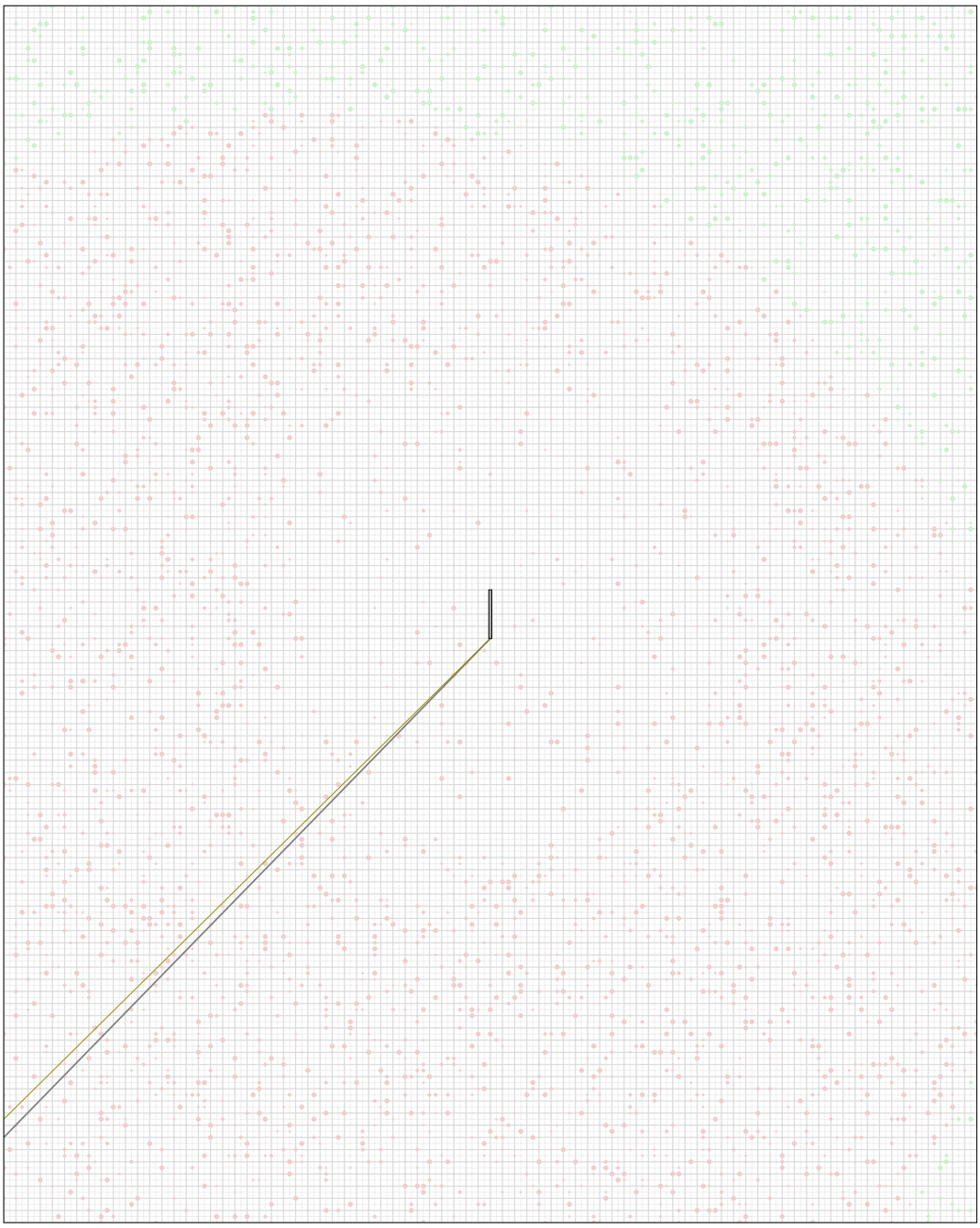

Figure C.9: Solution: 100x80km grid, case (800, 1500).

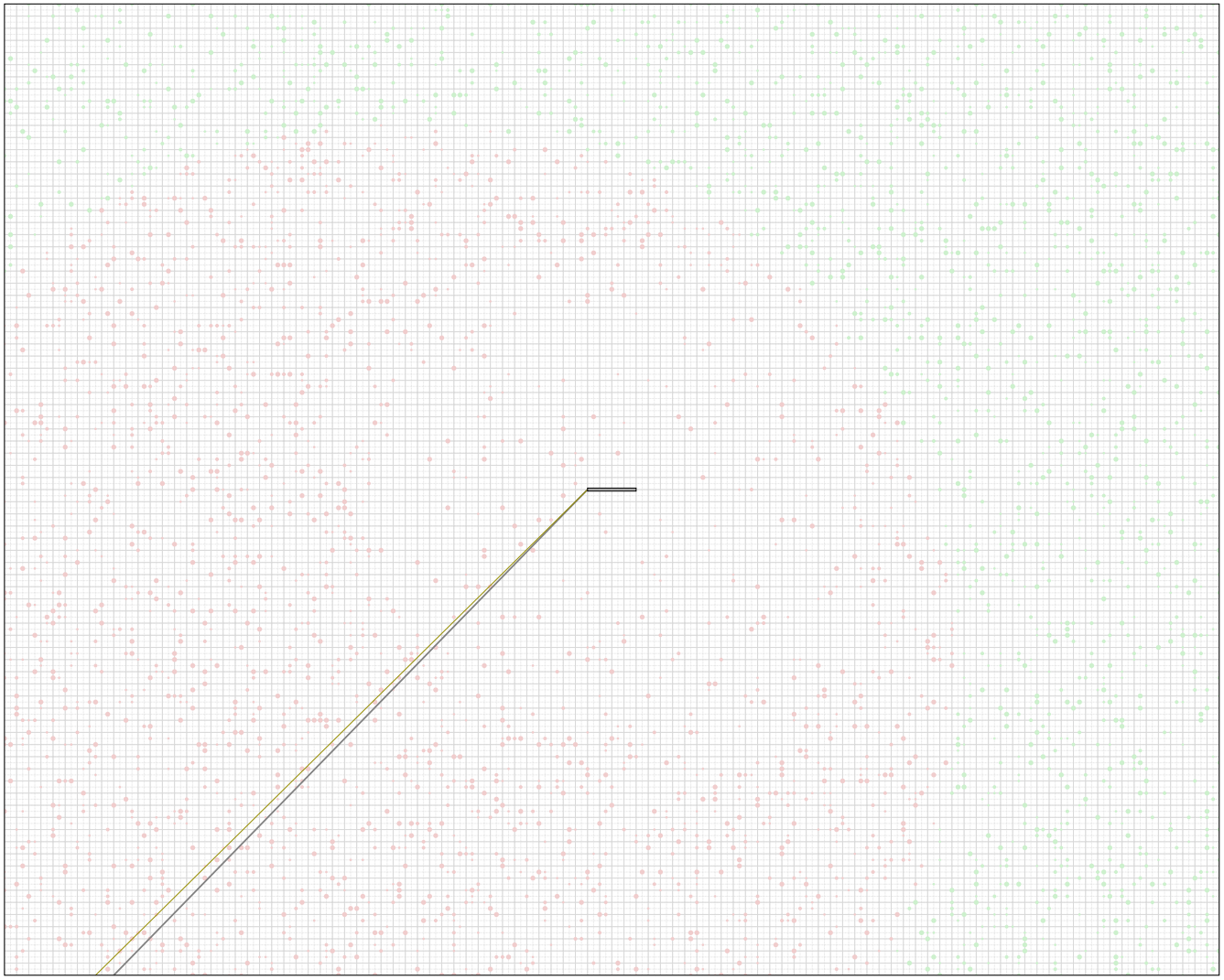

Figure C.10: Solution: 100x80km grid, case (850, 1500).

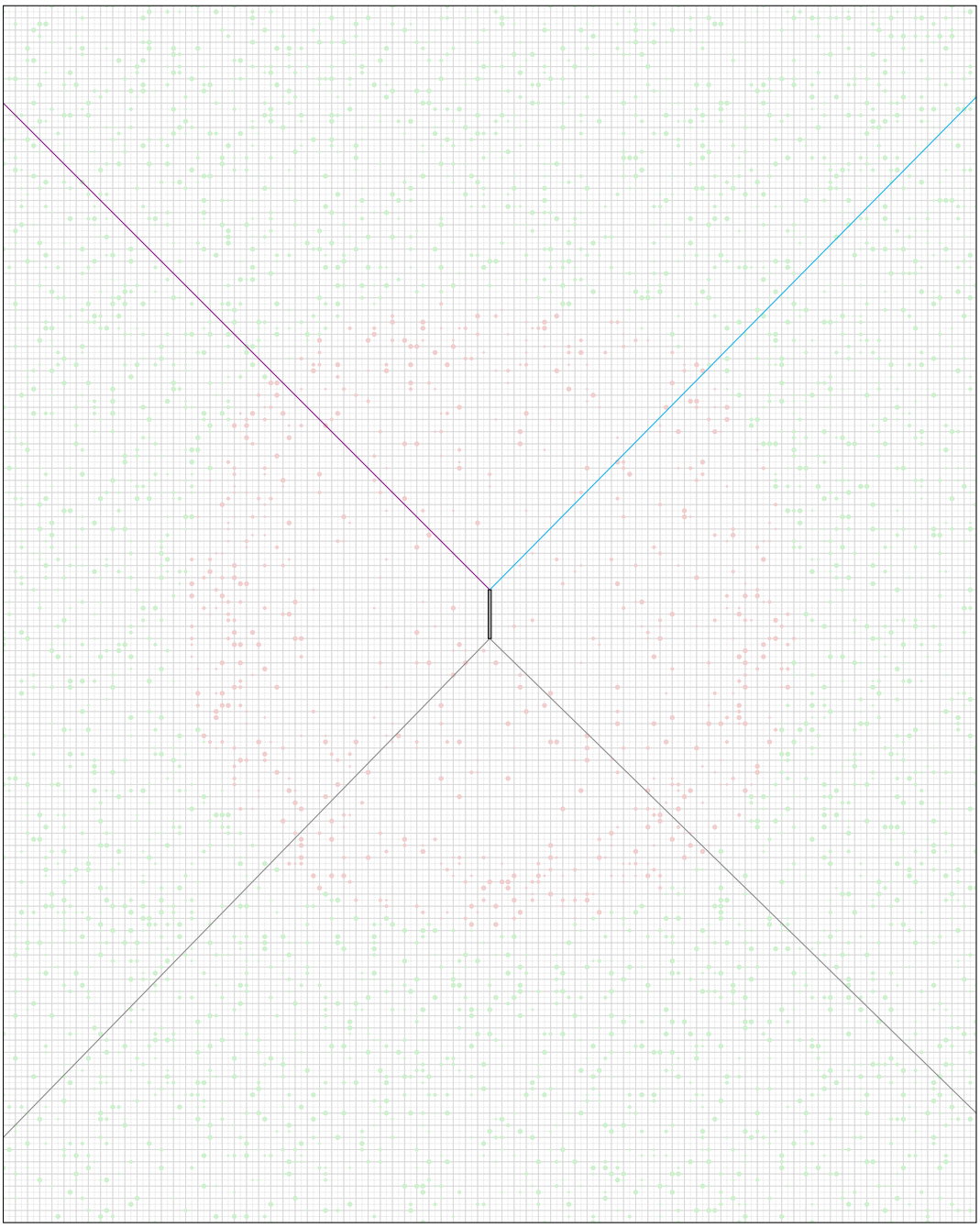

Figure C.11: Solution: 100x80km grid, case (900, 1500).

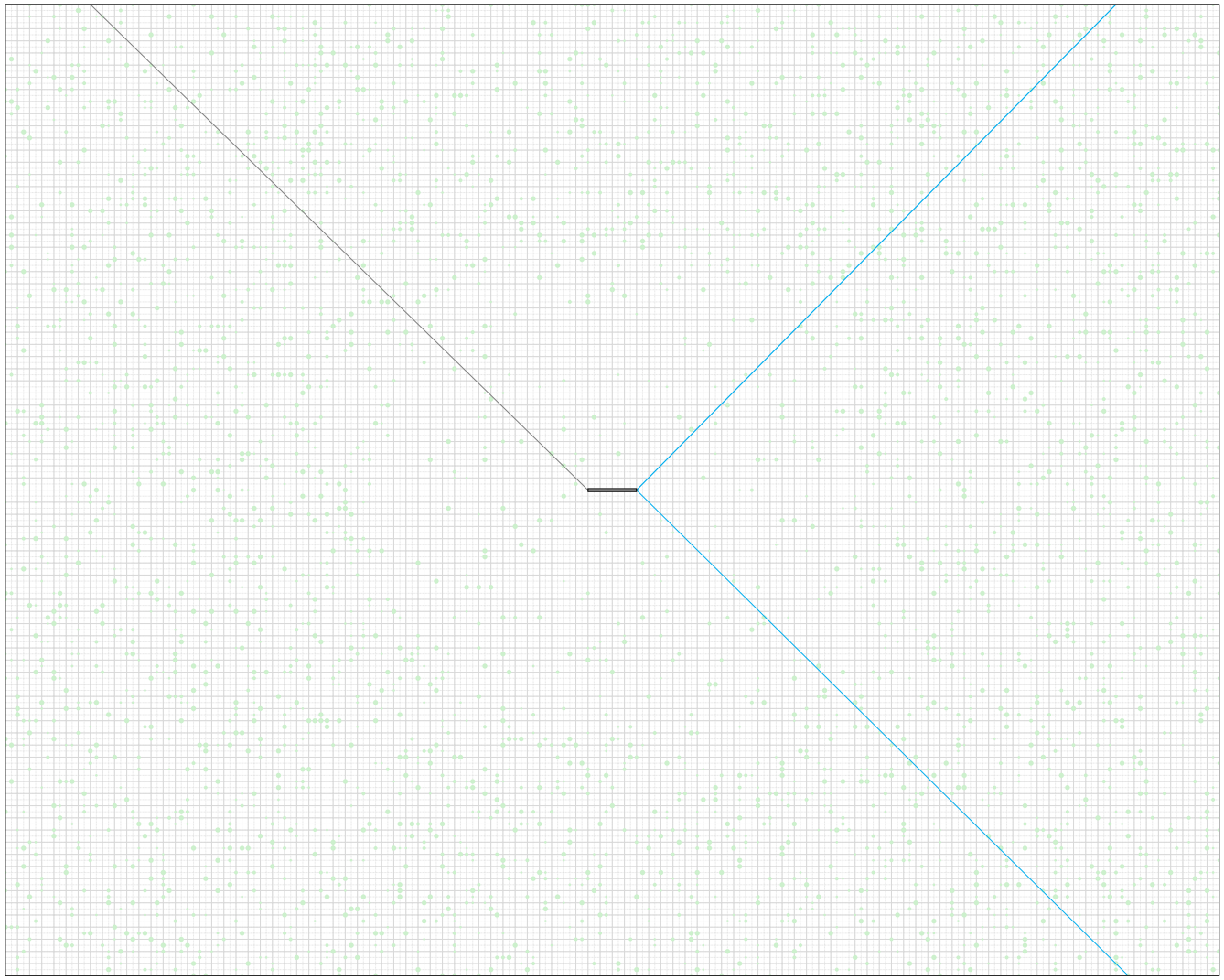

Figure C.12: Solution: 100x80km grid, case (950, 1500).