Technische Universität München
Zentrum Mathematik

Packing under Balancing Constraints Applications in Semiconductor Design and Flight Scheduling

Michael Ritter

Vollständiger Abdruck der von der Fakultät für Mathematik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Die Dissertation wurde am 02. Juli 2008 bei der Technischen Universität München eingereicht und durch die Fakultät für Mathematik am 12. November 2008 angenommen.

Für meine Eltern
und für Elisabeth

Abstract

We study two practical applications of linear and integer mathematical programming that are both instances of general packing problems under balancing constraints. In the first part, we investigate the causes for power loss in semiconductor circuits and develop a mathematical optimization model encompassing all relevant parameters. We then characterize the optimal solutions and present an efficient algorithm for the solution of the problem. In the second part, an application in airport flight scheduling is considered, wich can also be modeled as a packing problem subject to balancing constraints. We study the structure of optimal and non-optimal flight schedules and devise a comprehensive optimization model, that is then successfully evaluated on a real-world instance.

Zusammenfassung

Ausgehend von zwei praktischen Problemstellungen werden in dieser Arbeit lineare und ganzzahlige mathematische Optimierungsmodelle entwickelt, analysiert und gelöst, die sich allgemein als Packungsprobleme unter Ausgleichs-Nebenbedingungen beschreiben lassen. Im ersten Teil wird die Entstehung von Verlustleistung in modernen Halbleiter-Schaltkreisen untersucht; ein mathematisches Optimierungsmodell, das alle relevanten Problemparameter enthält, wird entwickelt. Eine Charakterisierung der optimalen Lösungen ermöglicht die Entwicklung eines effizienten Algorithmus für die Lösung des Problems. Im zweiten Teil wird ein Modell für die Erstellung von Flugplänen vorgestellt, das ebenfalls auf Packungsproblemen mit Ausgleichs-Nebenbedingungen beruht. Die Struktur optimaler und nicht-optimaler Lösungen wird vergleichend untersucht und ein umfassendes Optimierungsmodell entwickelt, das abschließend erfolgreich an Problemdaten aus der Praxis getestet wird.

Contents

Chapter 1

Introduction

1.1 Packing under Balancing Constraints

Among the most important and most widely applied problems in combinatorial optimization are the many variants of packing problems. In its most simple form, a packing problem is specified by a set of items and some sort of container (which may be defined by a set of abstract constraints), and the task is to choose a maximum subset of the items that fits into the container. The Knapsack problem is a prototypical incarnation of this simple packing problem: Given a set of items, each of a specified weight and value, and a container of a certain maximum weight capacity, which items should be packed into the container so as to maximize the sum of their values? A somehow "dual" viewpoint is expressed by the Bin Packing problem: Given a number of items, each of a specified weight, and a container with a certain maximum capacity, how many containers of that same type are needed in order to pack all the items?

In the Knapsack and the Bin Packing problem neither the form nor the order of the items in the container is of any interest. However, for practical applications these aspects are often of considerable importance. A prominent example is the problem of loading a freight container with heterogeneous goods: If only the weight or volume of the items that are to be loaded is considered in a mathematical model, the optimal solutions produced by this model will virtually never be realizable in practice. This leads to higher-dimensional and more complex variants of Bin Packing or Knapsack problems, where the geometry of both the items and the container(s) is taken into account. Examples of such problems include Pallet Loading, Container Loading, and Multi-Processor Scheduling; for more information on these topics see, e. g., the articles [GMM90], [CGJ97] and [LMV02].

In practical applications, there is also a variety of loading problems where the arrangement and distribution of the load is restricted in some way. Constraints of this "balancing type" arise for instance in aircraft loading, where it is important to balance the load such that the center of gravity falls within a specified region around the geometrical center of the aircraft. Thus it is not only the value and the weight and/or volume of the items loaded that need to be considered, but also their arrangement within the container, which defines the load's center of gravity. For an aircraft, the question of finding an optimal distribution of the goods can be reduced to a one-dimensional problem, namely along the longitudinal axis of the airplane, because there is only little movement along the vertical and lateral axes while an aircraft is airborne. A more detailed treatment of that problem can be found in [ABIVZ92] and in [CY85]. Of course, similar problems exist in higher dimensions. A two-dimensional version would be natural for a freight container, where it is desirable to avoid tilt while lifting the container upon transshipping; a three-dimensional version arises if the aim is to position the center of gravity not only close to the center of a container's floor, but also not too high above the floor for reasons of stability. More on such problems is reported in the articles [BR95] and [DB99].

We would like to mention one more example, which will later exhibit a different (and somewhat surprising) connection to one of the problems we are dealing with in this thesis, namely the Turbine Runner Balancing problem. A hydraulic turbine runner consists of a cylindrical center piece and a number of blades welded to the center piece at regular distances. Such a blade may weigh up to 16 tons and due to manufacturing imperfections, not all the blades are identical with respect to their mass distribution. As a consequence, when the blades are welded to the center piece, the center of gravity might not coincide with the axis of the turbine runner, which leads to unbalance. During operations, such unbalance causes increased mechanical load and may ultimately lead to the destruction of the turbine runner. To minimize this unwanted effect, one may change the arrangement of the blades on the center piece and thus try to move the center of gravity as close as possible towards the axis; again a problem where the order of the items is crucial to the solution. For details on the Turbine Runner Balancing problem see [LM88; Mos86] and [Woe03].

All those problems have some characteristics in common: There is a collection of items (boxes, turbine runners) that have to be positioned within a given container (in many applications, the container is a cube or a parallelepiped, where the dimension is often between one and three, sometimes higher). To evaluate the quality of a packing, there is an imbalance measure, i. e., a function that assigns a real value to every packing. That function can be the distance of the center of gravity of the items to some given point (e. g., the geometrical center of the container) or it can be a little more abstract. The goal is then either to minimize the imbalance itself, or to optimize some other objective subject to an upper bound on the imbalance.

For the classical problem of loading k items of masses $m_1, \dots, m_k$ into a container, the imbalance of a packing can be computed by a function like

$$\mathit{imbalance}(x_1, \dots, x_k) := c - \frac{1}{\sum_{i=1}^{k} m_i} \sum_{i=1}^{k} m_i x_i,$$

where $x_1, \dots, x_k$ are the positions of the items and c is the geometrical center point of the container. The objective here is to minimize $\mathit{imbalance}$ subject to constraints on the packing (e. g., space or weight constraints). Obviously, the imbalance function here decomposes into a sum of univariate functions, so each item's contribution to the total imbalance is determined by its own position alone (and, of course, by its mass and the center point). Similar imbalance functions can be given for the other problems we mentioned above.

In this thesis, we will study two practical applications of packing problems under more complex balancing constraints. The first problem is formally a two-stage problem, where we ask not only for an optimal packing of the given items, but also for an arrangement that maximizes the potential for optimization in the positioning step. The second problem is an example for a maximum packing problem subject to complex local and global balancing constraints. In the rest of this chapter we will briefly describe the applications leading to and motivating these problems and introduce their most important aspects. In Section 1.4 we will also collect some mathematical preliminaries and clarify notations used throughout this thesis. Chapters 2–4 will then provide an in-depth treatment of various aspects of the application problems.

1.2 Optimal Wire Ordering and Spacing

In today's technological world, semiconductor circuits serve an abundance of different purposes. Key issues in the area of semiconductor design include mobility of electronic devices and miniaturization of the relevant technology. Not only notebook computers and increasingly popular mobile multimedia devices need to combine high computational power with a slim form factor and long battery lifetime; low energy consumption is also of major importance for medical applications or electronic devices used in spacecraft technology. Besides the desire to maximize the operating time before a battery recharge becomes necessary, there are other advantages that make a good case for the use of low energy circuits also for non-mobile devices. High power consumption usually results in an increased emission of heat, which can lead to erratic behavior of the circuit and even cause permanent damage of the silicon substrate. To avoid this, costly cooling measures, often combined with special housing material for the circuits, have to be implemented. Also, electronic circuits account for a significant part of the CO_2 emissions in industrialized societies, clearly exhibiting an increasing trend. For these reasons, the design of low power circuits has become an important and active area of research.

Power consumption in electric circuits depends heavily on the capacitances between adjacent wires. Recent technology requires the typical on-chip wire's thickness (see Figure 1.1) to become larger than both its width and the wire distance, so one can think of on-chip wires as parallel metal plates, see Figure 1.1 for an illustration. Whenever an activity change occurs on one of the wires, the electric field between this wire and its neighbors changes; thereby dissipating energy stored in the field in the form of heat.

The amount of *power loss* depends mainly on two factors, namely the amount of signal transitions that occur on each particular wire (referred to as *switching frequency*) and the distance between two neighboring wires. The higher the distance, the less energy is consumed when a switch occurs on one of the wires. More precisely, the energy E stored in an *electric field* within a capacitor is determined by the formula

$$E = C \cdot U^2,$$

where U is the operating voltage of the circuit and C is the capacitance. For a parallel-plate

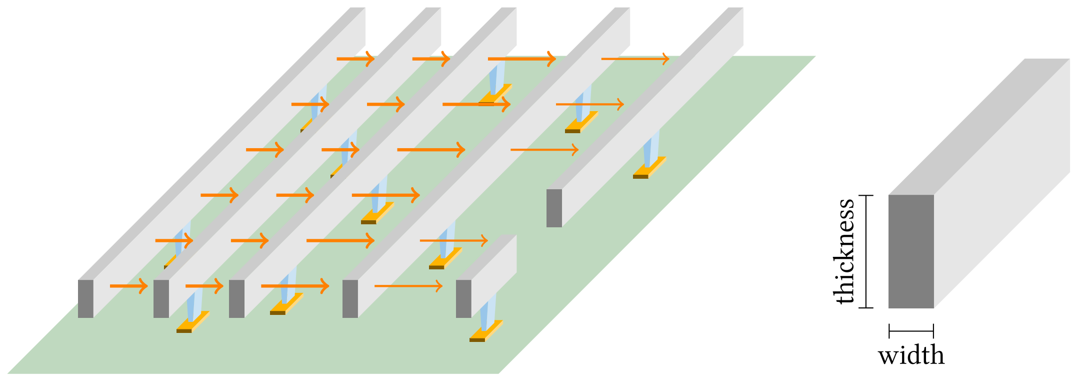


Figure 1.1: Schematic drawing of a one-layered semiconductor circuit. The substrate (green) with the transistors (yellow) contains the semiconductor elements, which are connected to one another by wires (gray). The wires within one layer are all routed in the same direction and are connected to the transistors (and possibly to other layers above and below) by so called vias. Whenever an activity change on one of the wires occurs, the electric fields between neighboring wires (symbolized by orange arrows) change, thereby causing energy loss in the form of heat.

capacitor, the capacitance C is

$$C = \varepsilon \cdot \frac{A}{d},$$

where A is the area of the plates, d is the distance between them and ε is the so called *dielectric coefficient*, a constant that depends on the material between the two parallel plates. As all wires have the same dimensions (at least on local patches of a complex circuit, see Section 2.1 for a thorough discussion of the model assumptions), and the operating voltage is considered fixed, the power loss depends only on the distance of neighboring wires and on their switching frequencies $\alpha(w)$. Thus (up to a constant term) the total power loss can be expressed as

$$\sum_{w \in W} \left(\frac{\alpha(w)}{x_{\text{left}}(w)} + \frac{\alpha(w)}{x_{\text{right}}(w)} \right), \tag{1.1}$$

where W is the set of all wires.

Naturally, space is limited in a semiconductor circuit, thus the natural question is “How should one distribute the available space between the wires in order to minimize power loss?”, or, in other words,

> “How should one arrange the wires on the chip such as to comply with the packing constraints (i. e., the available space) and such as to minimize power loss?”

This is clearly a packing problem with an imbalance objective, very similar to the ones we have encountered in Section 1.1. The goal is to position a collection of items (the wires) within a given container (the space available on the semiconductor) such as to minimize the imbalance measure defined by (1.1). However, there is an important difference to the examples given above, namely in the nature of the imbalance measure. Notice that $x_{\text{left}}(w)$ and $x_{\text{right}}(w)$ do not depend on the position of w alone, but also on the position of its left and right neighbor, and hence the objective function (1.1) does not decompose into univariate functions, but instead into functions of position pairs. Further, not all possible pairs are considered, but only those of neighboring wires are relevant for the imbalance measure, giving the objective function both an analytical (the distance terms) and a combinatorial (the arrangement of the wires) aspect.

The fact that we have to consider pairs should make the problem more difficult at first sight. On the positive side, there is not a large number of hard packing constraint here. Basically, we just have to observe the space restriction, and we may not place wires too close together for technological reasons. In Chapter 2 we show how these constraints can be used to characterize the optimal distances of the wires analytically.

What really makes the problem challenging is the fact that only neighboring wires are considered in the objective. By this requirement, the order of the wires acquires an influence on the objective. Indeed, as we will see in Chapter 2 later, the quest for an optimal order will lead to a Traveling Salesman problem (which is known to be $\mathcal{NP}$-hard, cf. [GJ79]). Fortunately, by making use of the analytical characterization of the optimal distances, we will be able to reduce the general problem to an instance that will turn out to be easily solvable. Thus we will be able to derive an algorithm that completely solves the Optimal Wire Placement problem in running time polynomial in the number of wires.

1.3 Flight Scheduling for Airport Operations

Over the last decades, civil air traffic has seen a tremendous increase both in terms of air transport movements and in passenger numbers. For an illustration see Figure 1.2, where the increase of both passenger numbers and flight movements from 1987 to 2007 is depicted for München airport. In the year 2007, München airport (cf. [MUC07]) has handled a total of 431 815 flight movements and 33 959 422 passengers (the numbers for Germany's largest airport Frankfurt/Main were 492 569 flight movements and 54 167 817 passengers, respectively, cf. [FRA07]). For München, this is an increase in flight movements by almost 145% over the last twenty years. To give an impression of the recent rapid growth of air traffic: For the old München airport at Riem the mark of 30 million passengers was reached in 1972, counting every passenger from the opening of the airport in 1949 (cf. [MUC06]). Today, the same number of passengers is handled at München airport within less than one year. The corresponding numbers for Frankfurt exhibit the same trend, but are a little less impressive in their growth rates, because München had (and still has) some spare capacity, while Frankfurt has been operating at its capacity limit for years.

With these developments, airports around the world are increasingly facing the problem of insufficient capacity to serve all requests from airlines. Building new runways, parking positions and terminal buildings is impossible in many cases due to space restrictions, and even if additional capacities can be built, this is often a lengthy process. Additionally, the development of new

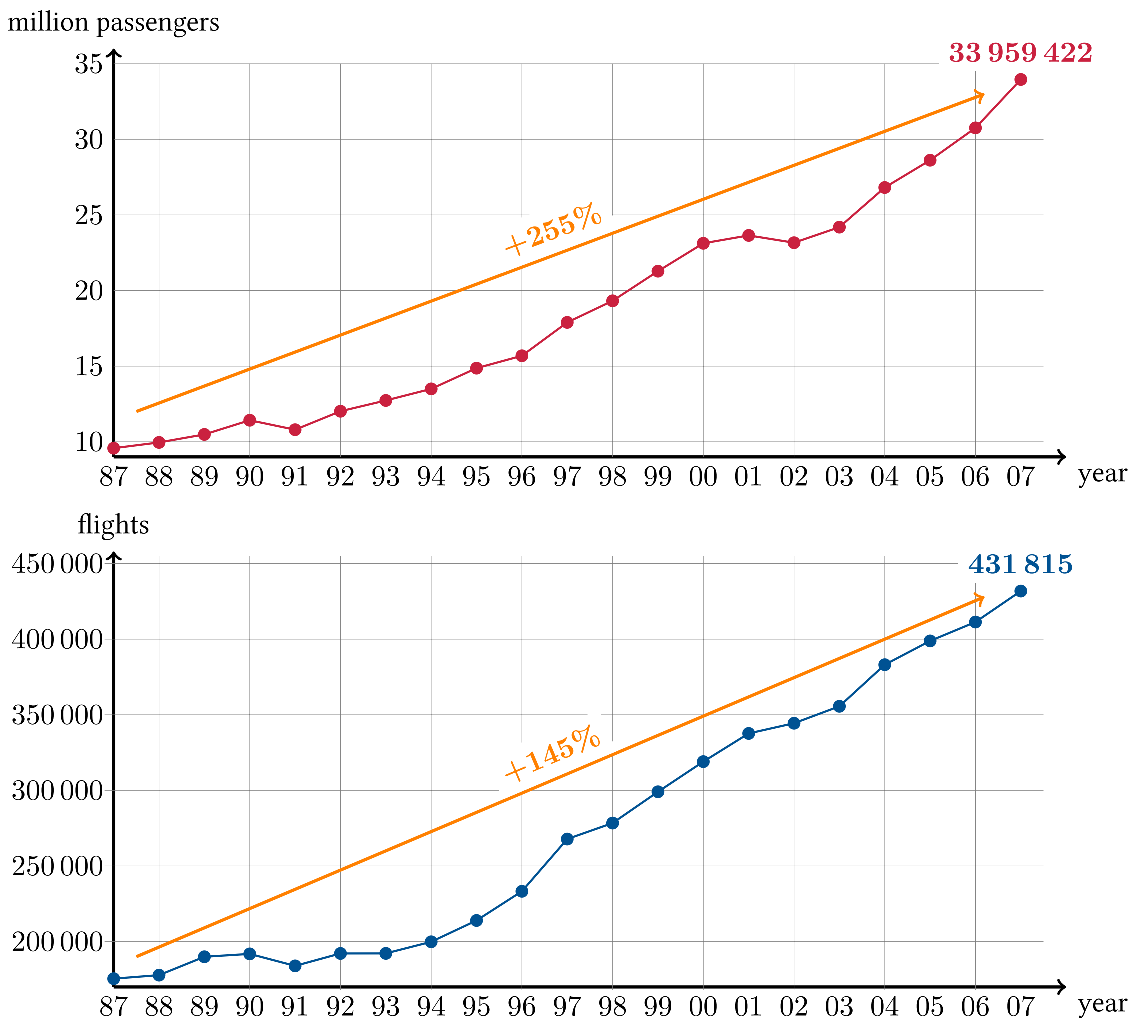


Figure 1.2: Passenger numbers (above, red) and commercial flight traffic (below, blue) at München airport for the years 1987 to 2007. (Numbers according to [MUC07].)

aircraft types that can accommodate a growing number of passengers requires complex handling operations for the airports' ground personnel and also gives rise to new air and ground safety concerns. The location of cities as well as of flora/fauna habitats near an airport naturally poses another problem: The noise emitted by the aircrafts requires a complex noise regulation policy in order to protect the environment and residents of nearby cities. As a result, most major airports around the world become congested at certain times because the demand exceeds one or more of the factors limiting air side capacity.

To resolve this situation, the so called *slot system* has been developed as a means of appropriately allocating the available capacity to the airlines. The slot system addresses such questions as fairness issues, property rights, customary law and ensuring competition by providing for sufficiently low market entrance barriers. In air traffic, there is a summer schedule ranging from March to October and a winter schedule ranging from October to March, and *flight scheduling* is applied for allocating airport capacities in the creation of these schedules. In contrast to that long-term planning problem is the short-term rescheduling process employed in daily airport operations, which is necessary due to unforeseeable circumstances such as delays enforced by bad weather or early arrival due to beneficial winds. In this thesis, we will solely be concerned with the long-term scheduling task, although some of our methods can also be applied to short-term rescheduling. However, there is often an on-line component involved with short-term decisions, which will not be considered here.

Under the slot system, an airline wishing to offer an air connection to and from some airport first needs to acquire the right to land and take off at that airport at a specified time. Such rights are referred to as *slots* (more precisely, *arrival* or *departure slots*), hence the name slot system. To avoid governments giving privileges to national carriers, the European Union has passed authoritative legislation on the allocation of slots to airlines, thereby introducing the slot system for all of its member states. The system which is in effect today was described in 1993 in the council regulation 95/1993 ([EU93]) and has since been amended in 2002 ([EU02]), 2003 ([EU03]) and 2004 ([EU04]). These documents provide the legal basis for the allocation of airport capacity to airlines at all European airports. Similar legislation or procedures are in effect for all major airports, often regulated by scheduling guidelines [IATA07] of the *IATA* (International Air Traffic Organization). Thus the slot system is used to regulate air traffic in all regions of the world, though often with minor modifications in the side constraints.

Slot allocation is subject to a number of restrictions (which will be detailed in Chapter 4), but the most important class of constraints is specified by giving a “time window” and an associated bound on the number of flights within. More precisely, such a constraint consists of a class of time intervals of the same length together with three bound values limiting the number of arrivals and departures as well as the number of total flight movements within each of the intervals (naturally, the latter value will usually be smaller than the sum of arrivals and departures bound). For example, such a restriction could allow for a maximum of 25 arrivals, 25 departures and 40 total flight movements within each 60 minute interval, starting on the hour throughout the whole planning horizon.

These “time window bounds” come in two slightly different flavors (see Figure 1.3 for an illustration): In most cases, they are applied as so called *shifting bounds*, which means the bound is applied for *every* time window of the respective length. Imagine taking the time window and slowly shifting it over the entire time scale, counting the number of arrivals and departures that lie within the time window as you move along. Then the three given bounds must not be exceeded anywhere along the time scale. As small delays are practically unavoidable in air traffic, flight schedules are not precise to the minute, but usually use ten minute time steps (for more details, see Section 4.1 and *time notions* in the glossary), thus it suffices to shift the time window in steps of ten minutes (providing for a finite number of constraints). In contrast to the shifting bounds, the time windows may also be applied consecutively to one another, i. e., the respective intervals do not overlap. We will refer to that variant as *non-shifting bounds* or *consecutive bounds*.

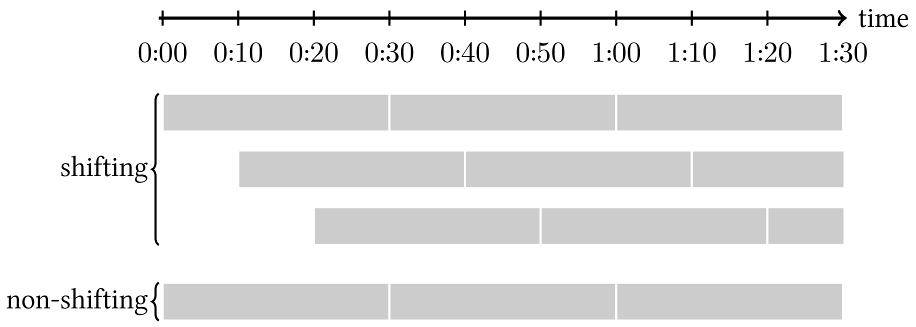


Figure 1.3: Example of shifting bounds and non-shifting bounds. A time window of 30 minutes is moved over the time scale in a shifting and non-shifting way.

The task encountered in flight scheduling is typical for a packing problem:

> "Find a flight schedule subject to one or more time window bounds, shifting or non-shifting, that consists of a maximum number of flights."

While the non-shifting bounds are very similar to classical knapsack constraints, the shifting variant is of a more complex type. Here, in addition to the local restrictions on the flights within each time window, the overlap of the time windows provides for a "spread-out" of the effect of allocating a specific slot to one flight, so the local constraints are to some extent "globalized". The imbalance measure of a flight schedule is thus determined by a number of time intervals, where each flight influences not just one, but several of these intervals, so the imbalance function is not easily decomposable anymore. In contrast to the Wire Placement problem described in Section 1.2, the imbalance is not part of the objective function here, but is solely bounded above resulting in a constraint on the distribution of the flights.

In Chapter 3, we will present an abstract model of flight scheduling that allows us to investigate complexity issues as well as the structure of an optimal distribution of flights subject to different time window constraints. We will discuss properties of "good" and "bad" (we will make these notions precise in Chapter 3) slot allocations and present some ideas of how to avoid the latter.

Following the theoretical treatment, Chapter 4 presents a concise mathematical model encompassing numerous constraints that are relevant for slot allocation in an application-specific context. The model is then tested extensively on real-world and realistic simulated data. We will see that the results of these tests exhibit many of the structural properties devised by the theoretical reflections of Chapter 3. Finally, alternative constraint structures motivated by Chapter 3 are considered and tested in a realistic setting.

1.4 Notational Conventions

The purpose of this section is to introduce some of the notation that will be used frequently throughout this thesis and to clarify the terminology. This section may be skipped without any loss and used for reference in case the reader should encounter some unfamiliar notation later. A short overview is also given in the list of symbols on page 169.

1.4.1 Vectors, Matrices and Inequalities

For the vector space $\mathbb{R}^n$, we denote the i-th unit vector by u_i, the all-ones vector by $\mathbb{1}$ (or $\mathbb{1}_n$, if the length n is not clear from the context), and for any subset $I \subset \{1, \dots, n\}$ the vector $\sum_{i \in I} u_i$ by $\mathbb{1}_I$. Similarly, for a finite set $G = \{g_1, \dots, g_m\}$, we write $u_{g_i} = u_i$ for the incidence vector of $\{g_i\}$, and, more generally,

$$\mathbb{1}_S := \sum_{g \in S} u_g$$

for any subset $S \subset G$.

For two vectors $x, y \in \mathbb{R}^n$ the inequality $x \leq y$ is meant to hold component-wise, and $x < y$ if $x \leq y$ and $x_i < y_i$ for at least one $i \in \{1, \dots, n\}$. The notations $x \geq y$ and $x > y$ are analogous.

For a matrix $A \in \mathbb{R}^{m \times n}$ the matrix entries will commonly be written as $A = (a_{ij})$. The i-th row of A is denoted by a_i^T, while $a^{(j)}$ means the j-th column.

1.4.2 Special Sets

For an arbitrary set S we denote by $\mathcal{P}(S)$ the power set of S, i. e., the set of all subsets (including the empty set), and by $S' \subset S$ we mean that S' is a (not necessarily strict) subset of S. Furthermore, $(S)^\star$ denotes the set S amended by the element "∞" (which will be used as a symbol to denote a special situation in various contexts), i. e.,

$$(S)^\star := S \cup \{\infty\}\,.$$

For $a, b \in \mathbb{Z}$, $a \leq b$, the integer interval $\{a, a+1, \dots, b\}$ is denoted by $[a,\ b] := \{a, a+1, \dots, b\}$. For an integer $n \in \mathbb{N}$ the intervals $[1,\ n]$ and $[0,\ n-1]$ will be abbreviated by $[n]$ and $[n]_\circ$ (to denote an interval of *length* n), respectively.

For a finite ground set G, a subset $S \subset G$ and some function $f : G \to \mathbb{R}$ we denote by $f(S)$ the sum

$$f(S) := \sum_{s \in S} f(s).$$

To simplify notation, we define $f(s') := 0$ for any $s' \notin G$, and consequently

$$f(S') := f(S' \cap G)$$

for an arbitrary set S'. The same notation will be used for a vector $c \in \mathbb{R}^n$ and a subset $S \subset \{1, \ldots, n\}$:

$$c(S) := \sum_{s \in S} c_s,$$

where we mean to imply $c_{s'} = 0$ for $s' \notin \{1, \ldots, n\}$.

1.4.3 Numbers

For our models, we will frequently have to restrict numbers like indices to some specified discrete interval. In more complex expressions, however, this often gives rise to somewhat clobbered notation, because several special cases need to be handled separately. In order to avoid this unnecessary complication, we introduce a special notation.

Definition 1.1
Let $n \in \mathbb{N}$ and $m \in \mathbb{Z}$ be two integers. Then we use the following notation:

1. $[\![m]\!]_n$ is the unique integer $k \in \{0, \ldots, n-1\}$ such that $k \equiv m \mod n$.
2. $[\![m]\!]_{[n]}$ is the unique integer $k' \in \{1, \ldots, n\}$ such that $k' \equiv m \mod n$.

Let $S \subset [n]$ and let $t \in [n]$, then the *circular Minkowski sum* with respect to $[n]$ is defined as

$$[\![t + S]\!]_{[n]} := \left\{ [\![t + s]\!]_{[n]} : s \in S \right\}.$$

In Chapter 4, we will sometimes measure the distance of two numbers with respect to a directed circle of given length n (rather than along the line). For $n \in \mathbb{N}$ and $a, d \in \mathbb{N}$ let

$$\operatorname{dist}_n(a, d) := [\![d - a]\!]_n$$

be the *circular distance* of a and d with respect to n.

Example 1.2
Let us illustrate the above definitions by a small example. For $n = 5$, we have

$$[\![8]\!]_5 = 3 \quad \text{and} \quad [\![8]\!]_{[5]} = 3;$$
$$[\![10]\!]_5 = 0 \quad \text{and} \quad [\![10]\!]_{[5]} = 5.$$

For $S = \{2, 4\}$, we get $[\![2 + S]\!]_{[5]} = \{4, 1\}$. The circular distance from 2 to 4 is $\operatorname{dist}_5(2, 4) = [\![2]\!]_5 = 2$. Notice that the circular distance depends on the order of the operands, $\operatorname{dist}_5(4, 2) = [\![2 - 4]\!]_5 = [\![-2]\!]_5 = 3$; see Figure 1.4 for an illustration. $\diamondsuit$

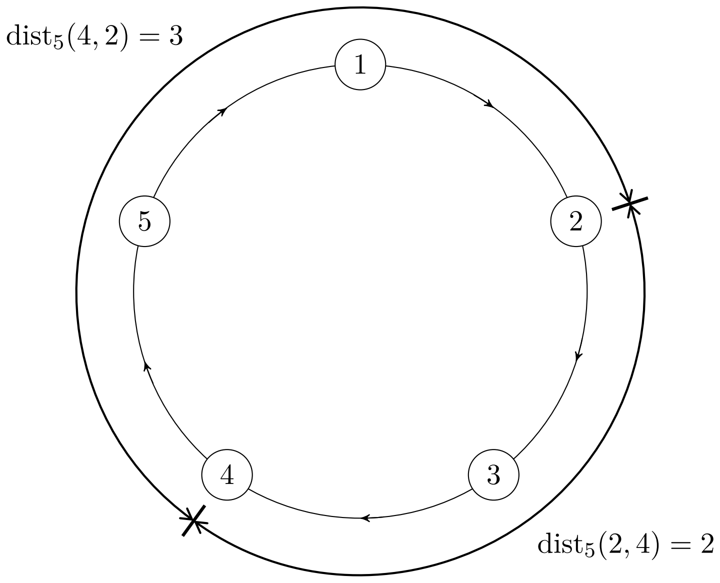


Figure 1.4: Illustration of the circular distances $\mathrm{dist}_5(2,4)$ and $\mathrm{dist}_5(4,2)$, respectively.

1.4.4 Matroids

In Chapter 3, matroid techniques will be used to provide some complexity results for flight scheduling problems. We collect the necessary terminology and some important results from matroid theory here as a reference. Details on matroid theory, including the proofs of the results cited here, can be found in [Wel95; Bix82; NW99].

Matroids were first defined by Whitney in 1935, who also gave a number of characterizations in his paper [Whi35]. We mainly follow [Bix82] for our notation.

Definition 1.3
An *independence system* $(M, \mathcal{M})$ consists of a finite ground set M and a nonempty collection $\mathcal{M} \subset \mathcal{P}(M)$ of subsets of M called *independent sets* such that the following conditions hold:

1. $\emptyset \in \mathcal{M}$

2. $S \in \mathcal{M}$ and $S' \subset S \Rightarrow S' \in \mathcal{M}$

For any subset $S \subset M$, a set $B \subset S$ is called *maximal independent set in* S if $B \cup \{i\} \notin \mathcal{M}$ for all $i \in S \backslash B$. If for all $S \subset M$ all maximal independent subsets of S have the same cardinality, then $(M, \mathcal{M})$ is called a *matroid.* For a matroid $(M, \mathcal{M})$, the maximal independent subsets of M are called the *bases* of the matroid.

One prominent class of matroids is constituted by the *partition matroids*, which will naturally arise from our applications in flight scheduling in Chapter 3. To unify the statements there, we now give a brief description of partition matroids.

Theorem 1.4
For a finite ground set M, let $M_1, \dots, M_k$ be a partition of M into disjoint subsets and let $m_1, \dots, m_k \in \mathbb{N}_0$. Then the definition

$$\mathcal{M} := \{N \subset M : |N \cap M_i| \leq m_i \quad \textit{for all } i \in \{1, \dots, k\}\}$$

yields a matroid $(M, \mathcal{M})$ known as partition matroid.

Proof. Clearly, $|\emptyset \cap M_i| = 0 \leq m_i$ for all $i \in \{1, \dots, k\}$, and for every subset $N' \subset N$ of a set $N \in \mathcal{M}$ the inequalities $|N' \cap M_i| \leq |N \cap M_i| \leq m_i$ hold for all $i \in \{1, \dots, k\}$, thus $(M, \mathcal{M})$ is an independence system. For the matroid property, let $B, B' \in \mathcal{M}$ be two maximal independent sets and suppose $|B| > |B'|$. Then there is at least one $j \in \{1, \dots, k\}$ where $|B \cap M_j| > |B' \cap M_j|$, hence we can choose an element $b \in (B \cap M_j) \setminus (B' \cap M_j)$. As $M_1, \dots, M_k$ is a partition of M, we know $b \in B$ and $b \notin B'$, thus $B^* := B' \cup \{b\}$ is a strict superset of B'. As $|B^* \cap M_j| \leq |B \cap M_j| \leq m_j$ and $|B^* \cap M_i| = |B' \cap M_i| \leq m_i$ for all $i \in \{1, \dots, k\} \setminus \{j\}$, the set B^* is an element of $\mathcal{M}$, contradicting maximality of B'. □

We will later use an important characterization of matroids, which is also due to Whitney, cf. [Whi35]. A proof formulated in the terminology used here can be found in [Bix82].

Theorem 1.5 (Matroid Exchange Property, [Whi35])
An independence system $(M, \mathcal{M})$ over the ground set M is a matroid if and only if for any two independent sets $U, V \in \mathcal{M}$ with $|U| = |V| + 1$ there exists $u \in U \backslash V$ such that $V \cup \{u\} \in \mathcal{M}$.

The primary interest in matroids in the field of combinatorial optimization stems mainly from two algorithmic results.

Theorem 1.6 (Greedy Algorithm and Two Matroid Intersection)

1. *Let $(M, \mathcal{M})$ be a matroid and $w : M \to \mathbb{Q}$ a weight function on the elements of M. Then the problem of identifying a set $M' \in \mathcal{M}$ of maximum weight $w(M')$ can be solved by a greedy algorithm (for details on the formulation of the algorithm see [Bix82]).*
2. *Let $(M, \mathcal{M}_1)$ and $(M, \mathcal{M}_2)$ be two matroids on the same ground set M and let $w : M \to \mathbb{Q}$ be a weight function on the elements of M. Then a maximum-weight set $M' \in \mathcal{M}_1 \cap \mathcal{M}_2$ (WEIGHTED MATROID INTERSECTION problem) can be found in polynomial time, provided membership in $\mathcal{M}_1$ and $\mathcal{M}_2$ can be tested in polynomial time.*

The algorithm proving the latter result is generally referred to as *two matroid intersection* algorithm. More details on the greedy property of matroids and a rigorous formulation of the greedy algorithm can be found in [Bix82]. The matroid intersection algorithm uses results by Lawler and Edmonds (see [Law75] and [Edm79], a textbook treatment can be found, e. g., in [NW99]).

1.5 Acknowledgements

1.5.1 Co-workers

The results of Chapter 2 were obtained in joint work with Peter Gritzmann and Paul Zuber; the paper [GRZ08] summarizing many of the results has been accepted for publication. An extended abstract for some of the results has already been published in [ZGRS05]. Paul Zuber's dissertation [Zub07] and [ZBIRS08] also contain some aspects from an engineering point of view.

It is due to Paul Zuber that we became aware of the Wire Spacing and Wire Ordering problems in the first place, and in the cooperation that lead to the results presented here, Paul, besides numerous discussions, provided valuable input and the "engineer's point of view" that is necessary for applied problems such as this. We are also grateful to Walter Stechele for helpful discussions on the practical applicability of our method.

Many of the results of Chapter 4 were obtained in joint work with Andreas Brieden and Peter Gritzmann. We gratefully acknowledge the support of our partners at *Fraport AG*[1] for an introduction to flight scheduling and numerous discussions on scheduling practice, applicable constraints, and the validity and quality of our results. Furthermore, Thomas Müller helped with the re-implementation of our model in a modularized and more concise way so that integrating new aspects became much more convenient. Finally, a sincere thanks goes to Oliver Bastert for helpful advice in applying Xpress-MP Optimizer to the problems of Chapter 4.

1.5.2 Thanks

My first and foremost thanks goes to my advisor Peter Gritzmann for his supervision, for sharing his ideas and visions, and for helpful discussions on the topics of this thesis and beyond.

This thesis would probably never have been completed without the support of my friends and colleagues at Technische Universität München. Special thanks goes to Barbara Langfeld, who was always ready to listen to my problems and discuss possible solutions, who often encouraged me and who also helped to push me forward sometimes. René Brandenberg also provided help, encouragement, and valuable advice, I am grateful for this. Thanks also goes to the other current and former members of the research group *Applied Geometry and Discrete Mathematics*, namely Andreas Alpers, Oliver Bastert, Franziska Berger, Steffen Borgwardt (for help with my teaching obligations during the final preparation of this thesis), Julia Böttcher (for spreading happiness and sometimes a little confusion in the group), David Bremner (for insights into the Canadian humor and the quirks of the English language), who was a guest of the group for almost a year, Andreas Brieden (for introducing me to flight scheduling problems and for keeping the ice cream supply up), Markus Brill, Nico Düvelmeyer, Tobias Gerken (for teaching me the value of coffee for mathematical research), Tanja Gernhard (for help with sliding contests when she happened to be at Munich), Elias Janjal, Markus Jörg (for his somewhat sarcastic sense of humor), Heidi Karpat, Katja Lord (for helpful discussions and for being ready to help whenever I needed

[1]For further information on Fraport, see the company website `http://www.fraport.de` and [FRA08].

someone for exam supervision and other obligations), Christoph Metzger, Lucia Roth (for help with various obligations like sliding contests and exam supervision), Anusch Taraz, Thorsten Theobald, Gottfried Tinhofer and Sven de Vries (for help with our computer systems whenever my knowledge about Solaris was too limited). Additionally, I would like to thank all of the colleagues I had and still have the pleasure to work with here at Technische Universität München. Silvan Rutzke helped me with IT related obligations, and Thomas Stolte provided helpful comments and advice, and — together with Barbara Langfeld — often made me feel better when I felt a little lost. I am very grateful for that.

My deep and heartfelt thanks goes to my family, friends, and especially to Elisabeth, for their help and support during my studies.

Chapter 2

Optimal Wire Placement for Low Power Semiconductor Circuit Design

In this chapter, we will analyze the main cause for power loss in recent semiconductor circuits, namely capacitances emerging between adjacent circuit wires. As outlined in Chapter 1, the power consumption of a semiconductor circuit depends on the distances of adjacent wires and the frequency of power switches on these wires. Determining an optimal spacing and possibly reordering of parallel wires is a key issue in the design of low power semiconductor circuits. We begin by presenting and analyzing a mathematical model that encompasses all major aspects of the Wire Placement problem (which combines an optimal spacing and ordering of the wires) in integrated circuit design. As the first step towards a concise algorithm we then consider the Optimal Wire Spacing (OWS) problem separately. As it turns out, the underlying convex optimization problem can essentially be solved analytically. This can be utilized to reduce the combined wire ordering and spacing problem to a specific kind of Minimum Hamilton Path (MHP) problem (or Traveling Salesman problem, TSP). While the general MHP is notoriously $\mathcal{NP}$-hard, our algorithm for the Optimal Wire Placement (OWP) problem on N parallel wires relies on strong new structural results and will be shown to run in total worst-case $\mathcal{O}(N \log N)$ time.

Both *wire ordering* and *wire spacing* according to various objectives have a long history in electronic *design automation*, see [YK99; MMP01; MPS03; MMK06]. Loosely related to wire spacing is a technique called *wire spreading* that was presented in 1997 (cf. [SD97]) and is implemented in most commercial software packages for semiconductor circuit design. Here the spacing of the wires is changed locally in order to decrease the probability of a short circuit caused by small material defects on the silicon wafers used in chip production, so the main objective of wire spreading is an increased chip yield, a decrease in power consumption is merely a byproduct. Wire spacing as a means of decreasing power consumption was first investigated in [MMP02], where a heuristic was suggested to decrease power dissipation. Wire ordering first appears in the literature in [Gro89], although in a very different context. Around the year 2000, the idea of wire ordering as a means of reducing power consumption is taken up by various authors. In [SS01] a simulated annealing approach for reordering chip wires is suggested and experimental results showing significant power savings are reported. The authors of [SM06] investigate a model of Wire Ordering as a Traveling Salesman problem and use the TSP solver *Concorde* (cf. [ABCC03]) to provide an optimal wire ordering. However, their model and methodology differ significantly from the one presented in this work; and no wire spacing component is employed. Besides power, different objectives are used in the literature, e. g., timing, area, crosstalk avoidance, yield and combinations of these. For a detailed overview of the literature concerning wire ordering and spacing we refer the reader to [Zub07, Chapter 3].

In this Chapter, we will derive a model of Wire Placement as a Packing problem with power loss as a measure of imbalance in the objective function and distance requirements as packing type side constraints. Subsequently, an efficient rigorous algorithm for power optimal Wire Placement is presented. The following Section 2.1 gives relevant background information on low power semiconductor design to motivate and justify the mathematical model for the Wire Placement problem that will be presented afterwards, together with the main result of this chapter. As mentioned earlier, we will first investigate the Wire Spacing problem, solve the underlying convex optimization problem and derive an efficient algorithm for Optimal Wire Spacing in Section 2.2. Based on these results, Section 2.3 will then add the optimal wire ordering task and reduce the combined placement problem to the solution of a certain class of Minimum Hamilton Path problems. We will study the structure of this class of problems, relate it to a certain class of Traveling Salesman problems and derive an efficient algorithm which will subsequently lead to the asserted $\mathcal{O}(N \log N)$ algorithm for the combined wire ordering and spacing problem presented in Section 2.4. Finally, Section 2.5 contains some remarks regarding applicability, practical results and directions for future research.

2.1 Low Power Semiconductor Design

We will now give some background information on power issues in semiconductor circuits and introduce wire spacing and wire ordering as a means of decreasing power loss. A first mathematical description of the optimal placement problem, still formulated from a natural application specific perspective, will subsequently be refined and abstracted to allow us to take up a more mathematical position for the problem statements.

Naturally, we do not aim at a comprehensive treatment of all relevant aspects from electrical engineering here but will concentrate on those that motivate and justify our concise mathematical model given in Section 2.1.2. For further information and more technical background we refer the reader to [Vyg04; CKP01; MWK06], the *International Technology Roadmap for Semiconductors* [SIA07] and Paul Zuber's doctoral thesis in electrical engineering [Zub07] as well as the articles and references quoted therein.

2.1.1 Power Loss in Semiconductor Circuits

Typically, a semiconductor circuit is designed in several consecutive steps. After the circuit diagram has been finalized, placement, routing and simulation are used to reach the final design for use in production of the semiconductor. A manufactured semiconductor circuit is assembled in several *layers* that are stacked on top of each other, see Figure 2.1 and Figure 1.1 for an illustration. The bottom layer (*substrate*) is made of silicon and contains the transistors and possibly other electronic components. Above that layer, one or more *metal layers* stacked on top of each other follow. The metal layers contain wires (called *interconnects*) that connect the transistors to each other and to input/output pins of the circuit. Within a single metal layer all wires are of the same thickness and the wires are all parallel (possibly with the exception of some very small wire segments); the direction of the wires within one layer is called that layer's *preferred routing direction*. The preferred routing directions alter perpendicularly between two adjacent metal layers. To connect the wires within one layer to the bottom and to neighboring layers small contacts (called *vias*) exist between the layers. To give the reader an idea of the dimensions, a modern microprocessor (as of 2008) has a die size of 100 mm^2 to 200 mm^2 and consists of about 400 to 800 million transistors,[1] ten to twenty metal layers and several kilometers of total wire length.

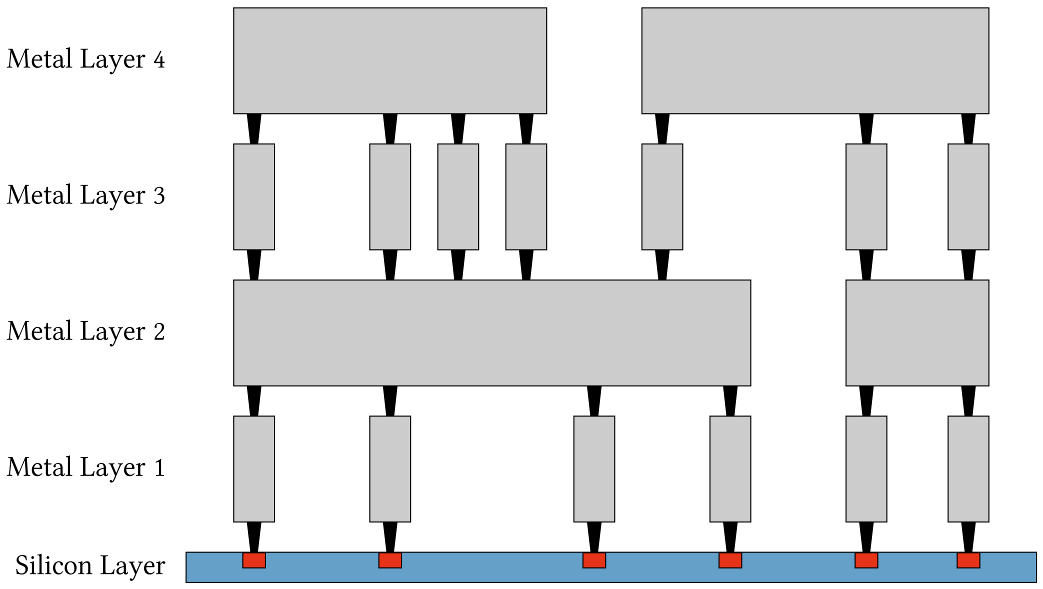


Figure 2.1: Schematic illustration of a typical semiconductor circuit composed of a bottom silicon layer and four metal layers. The wires in the metal layers are connected to the silicon and to each other by contacts, so called vias. Wires within one metal layer are routed in the same direction.

In digital semiconductor circuits, there are only two possible states for a signal, zero or one. A high voltage level (today in the order of magnitude of 1 V) represents a one, a low voltage level (0 V) represents a zero. So whenever the state of a circuit changes, the voltage level on one or more wires changes from 0 to 1 or vice versa; we will say that a signal transition occurs or simply a *switch* occurs on a wire. When this happens, the electric fields between a switching wire and any

[1]For instance, an Intel Xeon quad core processor ("Harpertown") comprises 820 million transistors on a die size of 214 mm^2; thermal design power (which roughly means "heat emission") currently rises up to 130 W. For more data on current microprocessors, see the survey [Ben08]. By the end of the year 2008, processors with more than two billion transistors could be launched (currently in development, see [Win08]).

adjacent metal surface change. To build up the field, energy is required, half of which is dissipated as heat, and the other half is stored in the field. When the field diminishes, the energy stored in the field is dissipated as heat as well.

To be precise, the power consumption of a semiconductor circuit is usually decomposed into two parts, a *static component* attributed to leakage at transistor level and a *dynamic component* caused by switching capacitances between adjacent wires and short circuit currents.[2] As of today, capacitances between adjacent wires account for the major part of total power consumption, and although leakage has increased over the past years, simultaneously a relatively increasing fraction of the capacitances has moved from transistors to wires as the following quote from [Per06] illustrates (for more details see [SIA07]):

> "Engineers today recognize interconnects as a major impediment to the performance trajectory that microprocessors have been on for the past 35 years. It is the wires, not the transistors themselves, that are sucking up power, threatening chip performance, and dragging out design cycles. In today's billion-plus transistor chips, which have multiple layers of wires connecting transistors and many kilometers of interconnects per square centimeter, the wires cost more than the transistors."

This aspect is of special importance at low ambient temperature (where leakage is lower due to physical reasons) or in low leakage circuits frequently found in mobile devices, where in non-idle phases the dynamic component exceeds the static component by several orders of magnitude. Even if circuit design is not focused on low leakage, static and dynamic power loss may be balanced against each other: By changing the ratio of threshold voltage and operating voltage of a circuit, the leakage may be decreased at the cost of higher dynamic power loss. For these reasons we concentrate on the dynamic aspects of power loss here.

The amount of energy stored in an electric field between two adjacent metal surfaces is directly proportional to the *capacitance*, which can be expressed as the quotient of the adjacent surface area and the distance of the two wires, as outlined in Section 1.2. Power loss caused by interconnects in semiconductor circuits is mainly due to capacitances emerging between neighboring wires whenever their relative voltage changes. Over the last years, the demand for ever higher integration densities has substantially increased, and for technical reasons this requires the typical on-chip wire's thickness[3] to become larger than both its width and the wire distances. This technological change results in a change of relevance of the different kinds of occurring capacitances. While in the past the highest fraction of the sum of capacitances was caused by the coupling between different layers and the bottom areas of the substrate, now the edge-to-edge capacitances within one layer dominate; see [WZS02].

[2]These short circuit currents are due to transistor behavior: A transistor "opens" when a certain voltage is available on one of its gates. On the other hand, the transistor "closes" when that voltage drops below a certain threshold, which is usually lower than the "opening threshold". As a consequence, whenever a signal transition occurs, some transistors may be open at the same time as power rises or falls, which may lead to short-time short circuits.

[3]Note that by thickness we mean here the elevation of a wire above the ground level of a metal layer, see Figure 1.1 for an illustration. While a bit unusual in geometry, this terminology is standard in electrical engineering.

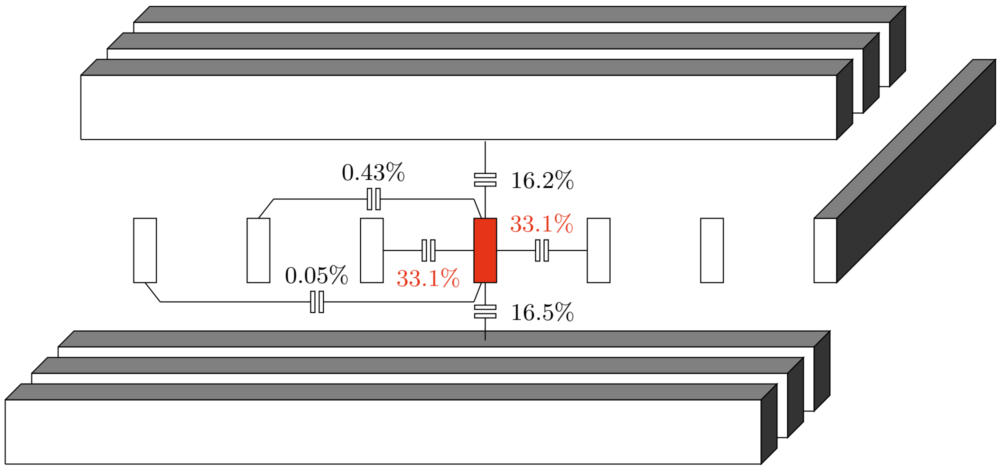


Figure 2.2: Capacitances in a $0.13\,\mu$m process. Not shown are two minor capacitances on the right hand side, which are equal to those on the left.

Figure 2.2 shows the capacitances in a layout simulated with the software package QuickCap (see [CI92], now distributed by Magma Design Automation). The test setup comprises a wire (red) on the third metal layer of a typical $0.13\,\mu$m process embedded into a fully crowded proximity. The capacitances between the wire and its immediate neighbors clearly dominate all other capacitance components. The fraction may reach up to 40% on each side in case of less population in the layers above and below.

In summary, the power loss caused by capacitances between adjacent wires depends on two different factors. As the electric field between two wires remains constant as long as the voltages of the wires do not change, electric power is lost only when a signal transition occurs on one of the wires. The frequency of such a toggle is called the *switching frequency* of a wire w and is modeled as a positive number $\alpha(w)$. If the switches lead to changes between zero voltage and the operating voltage (that is constant throughout the whole part of the chip), the (suitably normalized) number $\alpha(w)$ can be interpreted as the probability of a signal transition on the wire w at any of the given periods of time. For an existing layout, this value can be derived by a simulation; it then represents the actual number of toggles of a wire.

We will assume that the physical dimensions of the wires are fixed. For the thickness and width of the wires this is due to technological constraints, for the length of the wires we assume that we are working on a local patch of the complete circuit which consists of parallel wires of the same length (see below for comments on this assumption). Thus only the distances between wires matter for the determination of the capacitances between a wire and its respective neighbors. More specifically, the power loss for a wire w depends on the distances x_{left} and x_{right} to the two neighboring wires on each side, and (up to a constant) can be expressed as

$$\alpha(w) \cdot \left(\frac{1}{x_{\text{left}}} + \frac{1}{x_{\text{right}}} \right),$$

where two neighbors at distances x_{left} and x_{right} are present. (Here we tacitly assume that neighboring wires do not switch simultaneously, see below and also Section 2.5 for comments on that assumption.) Hence by optimizing the distances of the wires one can expect a decrease in power loss. In addition to that, we will also see that reordering the wires can have an effect on the total power loss of a circuit. We can specify a wire placement by the relative positions of the wires, i. e., by associating with each wire a real number in a given interval $[0, r]$. We assume here that r represents the actual available space, i. e., the given space reduced by the wire widths, so that we can regard a wire as having zero width for the model. Naturally, wires must not be placed too close together, so a minimum distance d must be enforced.

Before we formalize the model, let us comment on our assumptions. First, we only consider parallel wires of equal length. As we already mentioned, wires within one layer generally run in the same direction, so parallel wires are a very natural assumption. On the other hand, the wires within one layer are certainly not all of equal length, but normally end in differently placed vias connecting the wires' ends to layers above and below. However, local patches where our assumption holds may easily be identified and real-world circuits contain a considerable amount of such areas. Application of Wire Spacing to such patches may require the addition of small "detours" (see Figure 2.3 for an illustration) to the wires, which is technologically feasible with very little impact on power consumption and possible other objectives of the routing process. More details on the identification and selection of suitable local patches can be found in [Zub07, Chapters 4 and 5]. Application of wire ordering to local patches requires a little more effort, as locally reordered wires have to be connected to their original positions so as not to influence the routing outside of the selected patch. To this end, a so called *permutation network* can be used. A permutation network is basically an extra layer containing the necessary connections between the original wires and the reordered wires at both ends of the selected patch (of course, one such extra layer may contain more than one permutation network). These connections are generally very short and hence do not add considerable overhead in terms of extra power loss, timing and area. The effects of permutation networks have been studied in detail in [MMP01] with the conclusion that the overhead is more than compensated for by the benefit of wire ordering.

It should also be mentioned that both Wire Placement and Wire Ordering can be applied straightforward to bus connections on a chip, where often the connections at both ends of a bus may be ordered arbitrarily in the routing process, thus enabling the use of wire ordering without resorting to permutation networks. Such bus connections are fairly common for modern microprocessor architectures, especially when many cores are integrated on a common chip, connected to each other by bus wires. A third aspect exploits the fact that today's circuit designs are often *IP-based* ("Intellectual Property based"), meaning that a number of pre-developed modules exist that are then combined to design new circuits. The connections of these modules to the rest of the circuit (called pads) is usually fixed arbitrarily in the design process of the modules. As the modules are relatively small, reordering the wires within the modules may already be possible by just reordering the pads (i. e., not using permutation networks), hence an optimal wire placement can be applied in the design of such modules. This aspect is all the more important as the modules are reused in many different designs thousands of times, so even small power savings within an IP module add up to considerable amounts in practice.

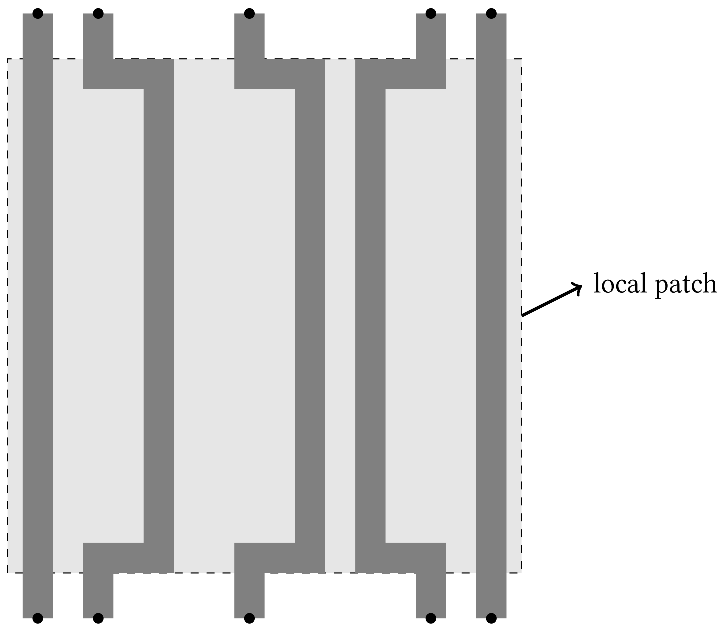


Figure 2.3: Local wire spacing using small detours. The circles mark the original connects of the wires, a different spacing is applied within the gray area limited by two border wires.

In our model, we also assume a common *minimum distance d* between any two wires. While this is true for most wires of a semiconductor circuit, there may be cases where a larger minimum distance is required. This situation may occur for power supply wires, which are sometimes wider than ordinary signal wires, thus requiring a larger minimum distance to be manufacturable. Often these power supply wires build a ring around the entire core of the chip and are not subject to optimization; in some larger circuits (and only on a fraction of the existing layers), power meshes may even be drawn through the chip. In the latter case, the power wires can be regarded as natural boundaries of the relevant problems, as these wires cannot be moved and no signal transitions occur on a power supply wire (hence such a wire partitions the wires considered into two completely independent parts). A second reason for larger minimum distances is signal integrity. In order to avoid *crosstalk* (i. e., signal transitions on one wire influencing the signal on a neighboring wire) wires with high signal levels should be placed at a somewhat larger distance to each other than to wires with low signal levels. Unfortunately, predicting crosstalk issues from switching activities and other parameters of the wires still presents a technological challenge to engineers, so integrating these issues into the model does not seem a viable approach. Generally, crosstalk issues are handled by a final “post polishing step” after the layout has been optimized, using various simulations and then slightly moving wires where crosstalk issues may occur until the problems are solved. For these reasons, it is common practice to largely ignore crosstalk in the modeling step. However, as we will see later, our approach places larger distances between wires with high switching activities, hence the solutions obtained from our model should be less susceptible to crosstalk anyway. It should be noted at this point that wire spacing with different, wire dependent minimal distances does not pose any problems to our wire spacing approach, this aspect can be integrated in a straightforward manner.

One last important assumption that we will make is that signal transitions on neighboring wires generally do not occur simultaneously. For a simultaneous switch the electric field between two neighboring wires may not change at all (if the same transition occurs on the two wires) or it may be reversed (if an opposite transition occurs). Hence the power loss is either zero (in the first case) or twice the usual value (in the second case). However, integration of simultaneous switching into the model would require knowledge about the correlated switching activities of all wires, which is not only harder to obtain (much more simulations are needed), but would also bloat the model to a large extent. In a complex semiconductor circuit, one can generally assume that the probabilities of a signal transition on any two neighboring wires are independent to a large extent, so one should expect that the effects of simultaneous transitions cancel out on average and this aspect may safely be ignored.

2.1.2 A Model for Optimal Wire Placement

We consider a scenario involving N parallel wires which are regarded as being enclosed between two static wires with switching frequencies 0. On a chip these boundary wires could be power or shield wires.

In the following let $N \in \mathbb{N}$, and let $w_1, \dots, w_N$ denote different *(proper) parallel wires*. Further, let w_0 and w_{N+1} be two additional *dummy wires*, and set $W = \{w_1, \dots, w_N\}$ and $\widehat{W} = W \cup \{w_0, w_{N+1}\}$.

Let $r \in \,]0, \infty[$ be the given *spacing range*, and let $d \in \,]0, r]$ be the *minimum accepted inter wire distance*. Both wire spacing and wire ordering can be determined by allocating to each wire a real number in the interval $[0, r]$, so we define a *wire placement* to be a map $\varphi : \widehat{W} \to [0, r]$ with the properties

$$\varphi(w_0) = 0 \quad \text{and} \quad \varphi(w_{N+1}) \geq \varphi(w) \quad \text{for } w \in W;$$
$$|\varphi(w) - \varphi(w')| \geq d \quad \text{for } w, w' \in \widehat{W} \text{ with } w \neq w'.$$

As it turns out, the underlying optimization problem can be described best in terms of the two separate tasks of wire ordering and wire spacing. A *wire ordering* is a bijection $\pi : \widehat{W} \to \{0, 1, \dots, N, N+1\}$ such that $\pi(w_0) = 0$ and $\pi(w_{N+1}) = N+1$. Let $\mathcal{P}_N$ denote the set of all wire orderings for a given number of wires N. An admissible *wire spacing* is a function

$\delta : \{0, 1, \dots, N, N+1\} \to [0,\, r]$ with

$$\delta(0) = 0 \quad \text{and} \quad \delta(j) + d \leq \delta(k) \quad \text{for any } j, k \in \{0, 1, \dots, N+1\} \text{ with } j < k.$$

Let $\mathcal{D}_N(r, d)$ denote the set of all admissible wire spacings. Note that the above constraints are already implied by the conditions on all pairs of adjacent positions.

Of course, any pair (π, δ) of a wire ordering and a wire spacing constitutes a wire placement φ via $\varphi = \delta \circ \pi$ and vice versa. Hence we will not distinguish between those and, in particular, also speak of (π, δ) as a wire placement.

Finally, let $\alpha : W \to [0,\, \infty[$ encode the *switching frequencies* of the proper wires. The set of all such functions will be denoted by $\mathcal{A}_N$.

Then, the *power loss* $L(\pi, \delta)$ (being the imbalance measure) of a wire placement (π, δ) is given by

$$L(\pi, \delta) = \sum_{w \in W} \alpha(w) \left(\frac{1}{\delta(\pi(w)) - \delta(\pi(w) - 1)} + \frac{1}{\delta(\pi(w) + 1) - \delta(\pi(w))} \right),$$

and the Optimal Wire Placement problem is the following task: Given $N \in \mathbb{N}$, $r, d \in]0,\, \infty[$ and $\alpha \in \mathcal{A}_N$, find $\pi^* \in \mathcal{P}_N$ and $\delta^* \in \mathcal{D}_N(r, d)$ such that

$$L(\pi^*, \delta^*) = \min \left\{ L(\pi, \delta) : \pi \in \mathcal{P}_N \ \wedge \ \delta \in \mathcal{D}_N(r, d) \right\},$$

or decide that no such minimum exists.

2.1.3 The Optimal Wire Placement Problem

Note that in the preceding subsection, the specific set W does not play any role; all that matters are the switching frequencies associated with the wires. Also the function π can be identified with a permutation on $\{1, \dots, N\}$. Using this abstraction, we can give a more concise mathematical formulation of Optimal Wire Placement: We will describe the task in terms of the variables x_i, denoting the distance of the i-th wire from its left neighbor for $i = 1, \dots, N+1$. These variables are related to the functions π and δ through

$$x_{\pi(w)} = \delta(\pi(w)) - \delta(\pi(w) - 1)$$

Furthermore, the switching frequencies will be encoded by a vector $(s_1, \dots, s_N)$ where $s_i = \alpha(w_i)$ for $i = 1, \dots, N$, and $\mathcal{S}_N$ shall denote the *symmetric group* on N elements. Then Optimal Wire Placement can be formalized as the following mathematical optimization problem:

Problem 2.1: Optimal Wire Placement (OWP)
Instance: $N \in \mathbb{N}$; $s_1, \dots, s_N \in [0,\, \infty[$; $d, r \in]0,\, \infty[$.
Question: Decide whether there exists a solution $(\pi, x) \in \mathcal{S}^N \times \mathbb{R}^{N+1}$ of

$$\begin{aligned}
\min \quad & \sum_{i=1}^{N} s_{\pi(i)} \left(\frac{1}{x_i} + \frac{1}{x_{i+1}} \right) & \\
\text{s.t.} \quad & \sum_{i=1}^{N+1} x_i \leq r & \\
& x_i \geq d & \text{for } i = 1, \dots, N+1 \\
& \pi \in \mathcal{S}_N &
\end{aligned}$$

and, if so, give one.

In addition, an instance of OWP will be called *all-distinct*, if the values $s_1, \dots, s_N$ are distinct, i. e., if $|\{s_1, \dots, s_N\}| = N$. When the permutation π is fixed, we are confronted with an instance of Optimal Wire Spacing (OWS); the input is the same but the objective is to just find optimal wire distances x_i; see Section 2.2. While Optimal Wire Spacing is a nonlinear programming problem under linear side constraints, the minimization over all permutations makes Optimal Wire Placement a combinatorial optimization problem with nonlinear objective function.

As we will see, in order to compute the distances that solve Optimal Wire Spacing we need to be able to compute the square roots of the switching frequencies, thus we formally have to handle real numbers of arbitrary length in a single elementary operation. This is why we employ a model of computation different from the Turing machine, namely the real *Random Access Machine (RAM)* for most of our results. For a detailed introduction into this concept see [PS85], or [Pap95] for a more formal treatment; in context of our results it will be sufficient to think of a real RAM as a computation device (similar to a Turing machine) that can handle arbitrarily long real numbers.

The main result of this chapter is to show that both Optimal Wire Spacing and Optimal Wire Placement can be solved efficiently.

Theorem 2.2
Both problems, Optimal Wire Spacing *and* Optimal Wire Placement *can be solved using at most* $\mathcal{O}(N \log N)$ *time in the real RAM model.*

Similar results can be obtained for the binary Turing machine model when the input is restricted to the rationals and the output is computed up to a precision given as part of the input; see Sections 2.2 and 2.3 for an analysis of the algorithms using the binary Turing machine model.

2.2 The Wire Spacing Problem

2.2.1 Characterization of Optimal Wire Distances

In this section, we will consider the Optimal Wire Spacing problem separately, so assume a permutation $\pi \in \mathcal{S}_N$ is given and is fixed throughout this section. Without loss of generality we may assume that π is the identity for more convenient notation. In the following we use a slight reformulation which uses $s_0 = s_{N+1} = 0$ and $q_i = s_{i-1} + s_i$ for $i = 1, \dots, N+1$, thus we are confronted with the following convex optimization problem:

Problem 2.3: Optimal Wire Spacing (OWS)
Instance: $N \in \mathbb{N}$; $q_1, \dots, q_{N+1} \in [0, \infty[$; $d, r \in]0, \infty[$.
Question: Decide whether there exists a solution $x \in \mathbb{R}^{N+1}$ of

$$\begin{aligned} \min \quad & F(x) = \sum_{i=1}^{N+1} \frac{q_i}{x_i} \\ \text{s.t.} \quad & \sum_{i=1}^{N+1} x_i \le r \\ & x_i \ge d \qquad\qquad \text{for } i = 1, \dots, N+1 \end{aligned}$$

and, if so, give one.

In a given instance of OWS the parameters $q_1, \dots, q_{N+1}$ do not explicitly rely on $s_1, \dots, s_{N+1}$ and can hence be ordered without loss of generality. Note, however, that a different order of the switching frequencies leads to a different set of $q_1, \dots, q_{N+1}$.

The feasible region P of an instance of OWS is compact, in fact a simplex. Since the objective function F is continuous on the feasible region ($x_i \ge d > 0$ for all i), the minimum is indeed attained unless P is empty. But $P = \emptyset$ if and only if $r < (N+1)d$, otherwise $x = d \cdot \mathbb{1}$ is always a feasible point. In fact, for $(N+1)d = r$, there exists only this trivial solution, hence we may subsequently assume that $r > (N+1)d$. Also, we may require (and will do so for convenience) that $q_1, \dots, q_{N+1} > 0$ rather than $q_1, \dots, q_{N+1} \ge 0$, since for $q_1 = \dots = q_{N+1} = 0$ any feasible $x \in P$ is optimal, and $q_{i_0} = 0$ for some $i_0 \in \{1, \dots, N+1\}$ implies $x^*_{i_0} = d$ for each optimal solution x^* of the given instance.

The following lemma characterizes optimal wire spacings.

Lemma 2.4
Let $(N, q_1, \dots, q_{N+1}, r, d)$ be an instance of OWS *with $r > (N+1)d$ and $q_1, \dots, q_{N+1} > 0$. Then the objective function F is strictly convex on the feasible region P, and the minimum of F over P is uniquely determined.*

For $x = (x_1, \dots, x_{N+1})^T \in \mathbb{R}^{N+1}$ define

$$D(x) = \{i \in \{1, \dots, N+1\} : x_i = d\} \quad \textit{and} \quad R(x) = \{1, \dots, N+1\} \setminus D(x).$$

Then a vector $x^* = (x^*_1, \dots, x^*_{N+1})^T$ is the optimal solution, if and only if

$$d < x^*_k = \frac{\sqrt{q_k}(r - |D(x^*)|\, d)}{\sum_{i \in R(x^*)} \sqrt{q_i}} \le \frac{\sqrt{q_k}}{\sqrt{q_j}} d \tag{2.1}$$

holds for all $j \in D(x^*)$ and all $k \in R(x^*)$.

The proof of this lemma employs the theorem of Karush, Kuhn and Tucker, which we reproduce here in a form suitable for our purposes. A proof of this theorem can be found in [SW70] or [Roc72].

Theorem 2.5 (Karush, Kuhn and Tucker)
Let $g : \mathbb{R}^n \to \mathbb{R}$, $g_1, \dots, g_m : \mathbb{R}^n \to \mathbb{R}$ be convex functions, and define $G : \mathbb{R}^n \to \mathbb{R}^m$, $G(x) := (g_1(x), \dots, g_m(x))$. Suppose the functions g, g_1, ..., g_m are differentiable in some $\mathbb{R}^n$-neighborhood of the feasible region $C := \{x \in \mathbb{R}^n : G(x) \le 0\}$, and there is some $x \in C$ with $G(x) < 0$. Then $x^* \in C$ is a minimum of the function g on the feasible set C if and only if there exist non negative Lagrangian multipliers $y = (\eta_1, \dots, \eta_m) \ge 0$ such that the following Karush-Kuhn-Tucker conditions hold:

$$\nabla g(x^*)^T + y^T \nabla G(x^*) = 0 \quad \text{and} \quad y^T G(x^*) = 0.$$

Proof (of Lemma 2.4). Let $F = F(x)$ denote the objective function of OWS. Of course, F is differentiable on the set of feasible points P and for $i, j = 1, \dots, N+1$ we have

$$\frac{\partial F}{\partial x_i} = -\frac{q_i}{x_i^2} \quad \text{and} \quad \frac{\partial^2 F}{\partial x_i \partial x_j} = \begin{cases} \frac{2q_i}{x_i^3} & \text{for } i = j, \\ 0 & \text{else.} \end{cases}$$

As $x_i \ge d > 0$ and $q_i > 0$ for each $i \in \{1, \dots, N+1\}$, the Hessian of F is a positive definite (diagonal) matrix, thus F is strictly convex on P, and its minimum over P is unique.

The constraints of OWS are all linear (thus differentiable) and by our assumptions the strict interior of the feasible region is not empty. Hence we can employ the Karush-Kuhn-Tucker conditions (see Theorem 2.5), stating that a feasible vector $x^* = (x_1^*, \dots, x_{N+1}^*)^T \in \mathbb{R}^{N+1}$ is optimal if and only if there exist non negative Lagrangian multipliers $\lambda_0, \lambda_1, \dots, \lambda_{N+1} \ge 0$ such that

$$\begin{aligned} \frac{q_i}{(x_i^*)^2} = -\nabla F(x^*)^T u_i &= \lambda_0 - \lambda_i && \text{for} \quad i = 1, \dots, N+1; \\ \lambda_0 \left(r - \mathbb{1}^T x^* \right) &= 0; \\ \lambda_i \left(x_i^* - d \right) &= 0 && \text{for} \quad i = 1, \dots, N+1. \end{aligned}$$

Let x^* be an optimal solution. Since all q_i's are positive, we have $\lambda_0 > 0$, and hence

$$\sum_{i=1}^{N+1} x_i^* = r.$$

As a consequence, $r > (N+1)d$ implies that $R(x^*) \neq \emptyset$.

Now, let $i, k \in R(x^*)$. Then $x_i^*, x_k^* > d$, hence $\lambda_i = \lambda_k = 0$ and therefore

$$0 = \lambda_i = \lambda_0 - \frac{q_i}{(x_i^*)^2} = \lambda_0 - \frac{q_k}{(x_k^*)^2} = \lambda_k, \quad \text{thus} \quad (x_i^*)^2 = \frac{q_i}{q_k} (x_k^*)^2,$$

which yields

$$r = \sum_{i=1}^{N+1} x_i^* = \sum_{i \in D(x^*)} x_i^* + \sum_{i \in R(x^*)} x_i^* = |D(x^*)|\, d + \frac{x_k^*}{\sqrt{q_k}} \sum_{i \in R(x^*)} \sqrt{q_i}\,,$$

proving the first part of (2.1).

On the other hand, for $j \in D(x^*)$ and $k \in R(x^*)$ we get $\lambda_k = 0$ and $\lambda_j \geq 0$, which yields

$$0 = \lambda_k = \lambda_0 - \frac{q_k}{(x_k^*)^2} \leq \lambda_j = \lambda_0 - \frac{q_j}{(x_j^*)^2}, \quad \text{thus} \quad \frac{q_j}{d^2} = \frac{q_j}{(x_j^*)^2} \leq \frac{q_k}{(x_k^*)^2},$$

completing the "only if" part of the proof.

Now, let $x^* \in \mathbb{R}^{N+1}$ satisfy (2.1). Then x^* is feasible since $x_i^* \geq d$ for all $i \in D(x^*) \cup R(x^*)$ and

$$\sum_{i=1}^{N+1} x_i^* = |D(x^*)|\, d + \sum_{j \in R(x^*)} \sqrt{q_j} \left(\sum_{i \in R(x^*)} \sqrt{q_i} \right)^{-1} (r - |D(x^*)|\, d) = r.$$

Further, we have

$$\frac{q_j}{(x_j^*)^2} = \frac{q_j}{d^2} \leq \frac{q_k}{(x_k^*)^2} \qquad \text{for all } j \in D(x^*) \text{ and } k \in R(x^*),$$

and

$$\frac{q_j}{(x_j^*)^2} = \frac{q_k}{(x_k^*)^2} \qquad \text{for all } j, k \in R(x^*).$$

Denoting this latter constant by λ_0, and setting

$$\lambda_i := \lambda_0 - \frac{q_i}{(x_i^*)^2} \qquad \text{for } i = 1, \ldots, N+1,$$

we see that $\lambda_0, \ldots, \lambda_{N+1}$ are non negative and satisfy the Karush-Kuhn-Tucker conditions for x^*, hence x^* is optimal. □

Note that for sufficiently large r, more precisely for

$$r > \max \left\{ \frac{d}{\sqrt{q_k}} \sum_{i=1}^{N+1} \sqrt{q_i} : k \in \{1, \ldots, N+1\} \right\},$$

the minimum distance constraint is not binding, i. e.,

$$d < \frac{r \cdot \sqrt{q_k}}{\sum_{i=1}^{N+1} \sqrt{q_i}}$$

for all $k \in \{1, \ldots, N+1\}$ and thus $D(x^*) = \emptyset$ for any optimal solution x^*. In that case, the problem is solved completely by Lemma 2.4. In general, Lemma 2.4 at least reduces the given instance of OWS to the determination of the *minimum distance set* D or, equivalently, its complement R.

2.2.2 Optimal Wire Spacing in the Real RAM Model

For an algorithmic implementation of the results of Lemma 2.4, we may of course assume that $q_1 \leq \ldots \leq q_{N+1}$, then $x_1^* \leq \ldots \leq x_{N+1}^*$ for the optimal solution x^*; hence all that matters is the cardinality of D. Therefore, to solve a given instance of OWS we may proceed in the following way:

- Order the q_i's so that $q_1 \leq \cdots \leq q_{N+1}$.
- Compute x_1' according to (2.1), starting with $D = \emptyset$.
- If $x_1' > d$, compute the other components of the solution vector x', and permute back x' to obtain the optimal solution x^*. Otherwise, replace x_1' by d and augment D by $\{1\}$.
- Use the same procedure to compute $x_2', \ldots, x_{N+1}'$ and permute back x' to obtain the optimal solution x^*.

Algorithm 2.1 presents a structured form of this sketch, formulated in real arithmetic.

Algorithm 2.1: Solving Optimal Wire Spacing on the real RAM.

```
Input: An instance $(N, q_1, \ldots, q_{N+1}, r, d)$ of OWS with $r > (N+1)d$
and $0 < q_1, \ldots, q_{N+1}$.
Output: An optimal solution $x^*$.
1 Sort $(q_1, \ldots, q_{N+1})$ to obtain $(q_1', \ldots, q_{N+1}')$ with $q_1' \leq \ldots \leq q_{N+1}'$.
2 Initialize: $S \leftarrow \sum_{i=1}^{N+1} \sqrt{q_i'}$ and $\Delta \leftarrow 0$.
3 for $i = 1, \ldots N+1$ do
4     Compute $x_i' \leftarrow \sqrt{q_i'} \cdot S^{-1} (r - \Delta \cdot d)$.
5     if $x_i' \leq d$ then
6         Set $x_i' \leftarrow d$, $S \leftarrow S - \sqrt{q_i'}$ and $\Delta \leftarrow \Delta + 1$.
7     end
8 end
9 Permute back the vector $x'$ according to the permutation that obtained $q'$ from $q$ to get the
  solution vector $x^*$.
```

Theorem 2.6
Algorithm 2.1 correctly solves Optimal Wire Spacing *and requires at most $\mathcal{O}(N \log N)$ arithmetic operations in the real RAM model of computation.*

Proof. For ease of notation we assume that the q_i are already sorted, i. e., $q_1' = q_1, \ldots, q_{N+1}' = q_{N+1}$ and $x^* = x'$ in the algorithm. Let $S^{(i)}$ and $\Delta^{(i)}$ denote the values of S and Δ *after* the i-th pass through the "for" loop. Denote by x^* the solution produced by the algorithm and let

$S^* = S^{(N+1)}$, $\Delta^* = \Delta^{(N+1)}$ be the final values of S and Δ, respectively. We show that condition (2.1) of Lemma 2.4 holds for all $j \in D(x^*)$ and all $k \in R(x^*)$.

The first inequality of (2.1) is clear. Let $m = \max\{l : l \in D(x^*)\}$, then $S^* = S^{(m)}$ and $\Delta^* = \Delta^{(m)} = m$. Monotonicity of $q_1, \ldots, q_{N+1}$ implies that $x_1^* \leq \cdots \leq x_{N+1}^*$, thereby establishing the equality part of (2.1) for all $k \in R(x^*)$. To prove the second inequality of (2.1), first note that

$$x_j^* = d, \quad x_k^* = \frac{\sqrt{q_k}}{S^*}\left(r - \Delta^* \cdot d\right).$$

Hence the fact that $S^* = S^{(m-1)} - \sqrt{q_m}$ and $\Delta^* = (m-1) + 1 = \Delta^{(m-1)} + 1$ implies

$$\begin{aligned} x_k^* &= \frac{\sqrt{q_k}}{\sqrt{q_j}} \frac{\sqrt{q_j}}{S^*}(r - \Delta^* \cdot d) \leq \frac{\sqrt{q_k}}{\sqrt{q_j}} \frac{\sqrt{q_m}(r - \Delta^* \cdot d)}{S^*} \\ &= \frac{\sqrt{q_k}}{\sqrt{q_j}} \left(\frac{\sqrt{q_m}(r - \Delta^{(m-1)} \cdot d)}{S^{(m-1)} - \sqrt{q_m}} - \frac{d\sqrt{q_m}}{S^{(m-1)} - \sqrt{q_m}} \right) \\ &= \frac{\sqrt{q_k}}{\sqrt{q_j}} \left(\frac{\sqrt{q_m}(r - \Delta^{(m-1)} \cdot d)}{S^{(m-1)}} \cdot \frac{S^{(m-1)}}{S^{(m-1)} - \sqrt{q_m}} - \frac{d\sqrt{q_m}}{S^{(m-1)} - \sqrt{q_m}} \right). \end{aligned}$$

Since

$$\frac{\sqrt{q_m}(r - \Delta^{(m-1)} \cdot d)}{S^{(m-1)}} \leq d$$

we conclude

$$x_k^* \leq \frac{\sqrt{q_k}}{\sqrt{q_j}} d \left(\frac{S^{(m-1)}}{S^{(m-1)} - \sqrt{q_m}} - \frac{\sqrt{q_m}}{S^{(m-1)} - \sqrt{q_m}} \right) = \frac{\sqrt{q_k}}{\sqrt{q_j}} d.$$

Thus, by Lemma 2.4, x^* is the optimal solution to Problem 2.3.

For the stated running time, note that the sorting step requires at most $\mathcal{O}(N \log N)$ arithmetic operations. The "for" loop is executed $N + 1$ times and each passage requires a constant number of operations. In total, the algorithm can be implemented to run using at most $\mathcal{O}(N \log N)$ arithmetic operations in the real RAM model. □

One might wonder if it is indeed necessary to compute the optimal values for x_i one after the other instead of using a "one-shot" (or rather "two-shot") approach, where one would first compute some vector $\tilde{x}$ under the assumption of $D = \emptyset$, then set $D := \{i : \tilde{x}_i \leq d\}$ and compute x^* using this set D. Unfortunately, this procedure might lead to wrong results, as $\tilde{x}$ does not necessarily provide the correct set D. The following example illustrates this.

Example 2.7
Let $q_1 = 1$, $q_2 = \cdots = q_N = 4$ and $q_{N+1} = \rho^2$ for some $\rho \geq 2$. If we compute $\tilde{x}$ under the assumption of $D = \emptyset$, we obtain

$$\tilde{x}_1 = \frac{r}{2N - 1 + \rho}, \quad \tilde{x}_2 = \cdots = x_N = \frac{2r}{2N - 1 + \rho}, \quad \tilde{x}_{N+1} = \frac{\rho r}{2N - 1 + \rho}.$$

Suppose now that the value of d is strictly between $\tilde{x}_1$ and $\tilde{x}_2$, i. e., $\tilde{x}_1 < d < \tilde{x}_2$, then

$$\tilde{x}_1 < d < \tilde{x}_2 = \cdots = \tilde{x}_N \le \tilde{x}_{N+1}.$$

Hence $\tilde{x}$ is not a feasible solution, and we set $x'_1 = d$ and $D = \{1\}$. Recomputation of the values for the remaining indices yields the vector x' with

$$x'_1 = d, \quad x'_2 = \cdots = x'_N = \frac{2(r-d)}{2N-2+\rho}, \quad x'_{N+1} = \frac{\rho(r-d)}{2N-2+\rho}.$$

A straightforward calculation shows that the values of N, r and ρ may be chosen such that

$$\tilde{x}_1 < x'_2 = \frac{2(r-d)}{2N-2+\rho} < \frac{2r}{2N-1+\rho} = \tilde{x}_2.$$

So, we may assume without loss of generality that d was chosen such that $\tilde{x}_1 < x'_2 < d < \tilde{x}_2$, thus $x'_2, \ldots, x'_N < d$, and the vector x' is still not feasible. Therefore the set D produced by this approach cannot be correct for the optimal solution. This shows that one must indeed proceed iteratively to obtain the correct values for the optimal solution. ♢

2.2.3 Wire Spacing in the Turing Machine Model

Let us conclude this section by briefly analyzing the adaptations that are needed for working on rational input in the binary Turing machine model. Clearly, the optimum solution x^* produced by Algorithm 2.1 may be irrational (and so may the objective value), so we have to settle for an approximation of x^* on a computer working on rational input data only.

Problem 2.8: Optimal Rational Wire Spacing (ℚ-OWS)
Instance: $N \in \mathbb{N}$; $q_1, \ldots, q_{N+1} \in \mathbb{Q}_{\ge 0}$; $d, r \in \mathbb{R}_{>0}$, $\varepsilon \in \mathbb{Q}_{>0}$.
Question: Decide whether there exists a real solution $x^* \in \mathbb{R}^{N+1}$ of

$$\begin{aligned} \min \quad & F(x) = \sum_{i=1}^{N+1} \frac{q_i}{x_i} \\ \text{s.t.} \quad & \sum_{i=1}^{N+1} x_i \le r \\ & x_i \ge d \qquad\qquad \text{for } i = 1, \ldots, N+1 \end{aligned}$$

and, if so, find a rational feasible point $\tilde{x} \in \mathbb{Q}^{N+1}$ such that

$$|F(\tilde{x}) - F(x^*)| < \varepsilon,$$

or report that no such point exists.

Similar to the real OWS, the feasible region $P_{\mathbb{Q}}$ is again a simplex and thus compact. If $(N+1)d > r$, there is no feasible point (neither rational nor real), if $(N+1)d = r$, then $\tilde{x} = d \cdot \mathbb{1}$ is the only feasible solution, and as $d \in \mathbb{Q}$, it is of course rational. For $(N+1)d < r$, $P_{\mathbb{Q}}$ contains for every feasible point x^* a rational point $\tilde{x}$ that is arbitrarily close to x^* and by continuity of the objective function F on $P_{\mathbb{Q}}$, that point may be chosen such that $|F(\tilde{x}) - F(x^*)| < \varepsilon$. Again, if $q_1 = \cdots = q_{N+1} = 0$ any rational feasible point is optimal, and $q_{i_0} = 0$ for some $i_0 \in \{1, \ldots, N+1\}$ implies that $\tilde{x}_{i_0} = d \in \mathbb{Q}$ for an optimal solution, so the search can be restricted to the corresponding affine subspace.

While the existence of a rational feasible point with objective value arbitrarily close to the optimal value is simply a matter of continuity of the objective function, it is much harder to actually compute such an approximate rational solution $\tilde{x}$, and to analyze how the error bound ε influences the runtime of the algorithm. On a binary Turing machine, we will have to compute an approximation for the square roots appearing in Algorithm 2.1. There are many ways to do this, e. g., a general approach like bisection or Newton's algorithm (details on these algorithms can be found in standard textbooks on numerical mathematics, e. g., [HH94; DH93]) or a specialized algorithm relying on some standardized form of floating point arithmetic and binary representation of numbers. What is important for our purposes is that there are algorithms to approximate the square root of an arbitrary number $q \in \mathbb{Q}_{>0}$ up to any prescribed error bound $\varepsilon' \in \mathbb{Q}_{>0}$ using at most $\mathcal{O}\left(\log_2 \frac{q}{\varepsilon'}\right)$ operations on a binary Turing machine, which means the algorithm is polynomial in the size of the input data.[4] This runtime bound can easily be proven for a bisection algorithm, but other algorithms (with possibly better runtime bounds) are suitable here, too.

However, there is a severe problem with Algorithm 2.1 originating in a possible discontinuity introduced by the approximation of the square roots. More specifically, the problem may be ill posed in the sense that the error in the objective function is not necessarily continuously dependent on the error allowed for the square root approximation. This phenomenon is due to the discrete nature of the set $D(x^*)$. Suppose there are one or more distances x_i^* in an optimal distance vector $x^* \in \mathbb{R}^{N+1}$ that are very close to the minimal distance d. Then a slight error in the calculation of the values for these x_i^* can lead to slightly smaller values, which would result in the x_i^* being set to d; furthermore, the set $D(x^*)$ would be augmented by one or more of the corresponding indices. Alas, the size of D does not depend on the approximation error of the square root computations in a continuous way, so this could ultimately result in the wrong minimum distance set, while the subsequent computations rely on a correct size of D. Errors in the calculation of other components of x^* might occur, leading to a possibly grave error in the objective function. Even worse, the "monotonicity property" used in the algorithm (meaning that when $x_{i_0}^* > d$ for some index i_0, then $x_i^* > d$ for all $i \geq i_0$) is put in jeopardy and thus the correctness proof for the algorithm is no longer true for that rational computation. Therefore, we have to restrict our analysis to well posed instances of $\mathbb{Q}$-OWS.

[4]Notice that the numbers q and ε' can be represented using $\mathcal{O}\left(\log_2 q + \log_2 \varepsilon'\right)$ bits on the binary Turing machine, so the $\log_2$ is essential for polynomial complexity.

Definition 2.9
Let $\delta > 0$. An instance of Optimal Rational Wire Spacing is called δ-*well posed*, if

$$\left| d - \frac{\sqrt{q_k}(r-(t-1)d)}{\sum_{i=t}^{N+1}\sqrt{q_i}} \right| > \delta$$

for all $t \in \{1, \ldots, N+1\}$ and all $k \in \{t, \ldots, N+1\}$. If there is a $\delta > 0$ such that the instance is δ-well posed, then it is called *well posed*, otherwise we speak of an *ill posed* instance. For a well posed instance, the *wellness condition* is the maximum $\delta > 0$ such that the instance is δ-well posed.

Essentially, well-posedness means that all possible distance values that may occur in the computation of the optimal distances are sufficiently different from the minimum distance d (namely at least δ). Thus if we approximate the square roots in the computation close enough to guarantee an error bound of at most δ, where δ is the wellness condition of the given instance, the resulting minimum distance set is guaranteed to be correct, thus avoiding the problems discussed above. Thus Algorithm 2.1 is guaranteed to produce a feasible solution x^* with the correct minimum distance set $D(x^*)$ if we supply an approximation of the square root function that guarantees an error bound of at most δ for the values of x_i^* as they are computed in line 4 of Algorithm 2.1.

For the analysis of error propagation we thus have to analyze first how an approximation error in the square root computations affects the errors in the resulting distance vector. To that end, let $S^{(m)}$, $\Delta^{(m)}$ denote the values of S and Δ computed in Algorithm 2.1 by the end of the m-th pass through the "for" loop with S^* and Δ^* being the final values. Furthermore, for a given rational number $\varepsilon' > 0$ and for $i \in \{1, \ldots, N+1\}$ denote by $sqrt(q_i)$ a rational approximation of $\sqrt{q_i}$ with absolute error

$$|sqrt(q_i) - \sqrt{q_i}| < \varepsilon',$$

and by $\tilde{S}^{(m)}$ the approximation of $S^{(m)}$ computed by the algorithm as a result of substituting $sqrt(q_i)$ for $\sqrt{q_i}$. Also, let x_i and $\tilde{x}_i$ denote the values computed in line 4 of Algorithm 2.1 in the i-th pass of the "for" loop, where x_i is computed in real arithmetic[5] and $\tilde{x}_i$ is computed by substituting $sqrt(q_j)$ for the respective values of $\sqrt{q_j}$ in the algorithm. As noted before, $sqrt(q_i)$ can be computed in polynomial time on a binary Turing machine.

[5]Note that x_i and the optimum solution x_i^* might differ, because $x_i < d$ is possible and would be corrected to $x_i^* = d$ in the lines following line 4 of the algorithm.

Let $q_{\min} = \min\{q_1, \dots, q_{N+1}\}$ and $q_{\max} = \max\{q_1, \dots, q_{N+1}\}$, then for any index $j \in \{1, \dots, N+1\}$ we have

$$
\begin{aligned}
|x_j - \tilde{x}_j| &= \left|r - \Delta^{(j-1)} d\right| \cdot \left|\frac{sqrt(q_j)}{\tilde{S}^{(j-1)}} - \frac{\sqrt{q_j}}{S^{(j-1)}}\right| \\
&= \left|r - \Delta^{(j-1)} d\right| \cdot \left|\frac{sqrt(q_j) S^{(j-1)} - \sqrt{q_j} S^{(j-1)} + \sqrt{q_j} S^{(j-1)} - \sqrt{q_j} \tilde{S}^{(j-1)}}{\tilde{S}^{(j-1)} S^{(j-1)}}\right| \\
&\le \left|r - \Delta^{(j-1)} d\right| \cdot \frac{S^{(j-1)} \left|sqrt(q_j) - \sqrt{q_j}\right| + \sqrt{q_j} \left|S^{(j-1)} - \tilde{S}^{(j-1)}\right|}{\tilde{S}^{(j-1)} S^{(j-1)}} \\
&\le |r - Nd| \frac{S^{(j-1)} \varepsilon' + \sqrt{q_j}(N+1)\varepsilon'}{\tilde{S}^{(j-1)} S^{(j-1)}} \le |r - Nd| \frac{S^{(j-1)} \varepsilon' + \sqrt{q_j}(N+1)\varepsilon'}{(N+1)^2 q_{\min}(q_{\min} - \varepsilon')} \\
&\le |r - Nd| \frac{\varepsilon'(q_{\max} + \sqrt{q_{\max}})}{(N+1) q_{\min}(q_{\min} - \varepsilon')} \le |r - Nd| \frac{\varepsilon'(\frac{3q_{\max}+1}{2})}{(N+1) q_{\min}(q_{\min} - \varepsilon')}.
\end{aligned}
$$

Notice the last inequality is due to the inequality between geometric and arithmetic mean ($\sqrt{q_{\max} \cdot 1} \le \frac{q_{\max}+1}{2}$) and is used to avoid a square root term which would again have to be approximated when using the inequality for computation on a Turing machine model. The last right hand side term above is a function in ε' which tends to 0 for $\varepsilon' \to 0$ and is continuous in ε' (at least for $\varepsilon' < q_{\min}$). Therefore $|\tilde{x_j} - x_j|$ can be made arbitrarily small by setting ε' appropriately, more precisely for any given value of $\varepsilon_x > 0$ we get

$$
|x_j - \tilde{x}_j| < \varepsilon_x \quad \text{if} \quad \varepsilon' < \frac{\varepsilon_x (N+1) q_{\min}^2}{\frac{1}{2}(r - Nd)(3q_{\max} + 1) + \varepsilon_x (N+1) q_{\min}}. \tag{2.2}
$$

The value for ε' can certainly be computed in polynomial time on a binary Turing machine (essentially, this means computing $q_{\min}$ and $q_{\max}$) for a feasible instance of Optimal Rational Wire Spacing. Notice that for indices $j \in \{1, \dots, \Delta^*\}$ it suffices to perform the computations for ε_x being the wellness condition of the instance, because $x_j = d = x_j^*$ for these indices anyway, so the ex-post approximation error is 0. However, we need to make sure that the set $D(x^*)$ is computed correctly (which need of course not succeed for an ill posed instance), thus the approximation is still necessary. Once we have determined Δ^* and thus $D(x^*)$ (in course of the computation, we eventually arrive at the first index j where $x_j > d$, hence $D(x^*) = \{1, \dots, j-1\}$), we need to properly approximate the square roots to get a value for $\tilde{x}_j$ that is within the error bound ε_x from the true value.

Finally, let us see how the error in the distance vector propagates into the objective value. Suppose for $j \in \{1, \dots, N+1\}$ the values $\tilde{x}_j$ are an approximation of the optimal distances x_j^* with an error bound of ε_x, i. e.,

$$
\left|x_j^* - \tilde{x}_j\right| < \varepsilon_x.
$$

$$\text{This yields} \quad |F(x^*) - F(\tilde{x})| = \left| \sum_{i=\Delta^*+1}^{N+1} \frac{q_i}{x_i^*} - \frac{q_i}{\tilde{x}_i} \right|$$
$$\leq \sum_{i=\Delta^*+1}^{N+1} q_i \frac{|x_i^* - \tilde{x}_i|}{x_i^* \tilde{x}_i} \leq \varepsilon_x \sum_{i=1}^{N+1} q_i \frac{1}{d^2},$$

thus the absolute error in the objective value can be bounded above by choosing $\varepsilon_x > 0$ small enough. More specifically, if we allow for an absolute error of $\varepsilon > 0$ in the objective value, then

$$|F(x^*) - F(\tilde{x})| < \varepsilon$$

can be guaranteed for a well posed instance of Optimal Rational Wire Spacing with wellness condition δ by choosing

$$\varepsilon_x < \varepsilon \min \left\{ \delta, \frac{d^2}{\sum_{i=1}^{N+1} q_i} \right\}. \tag{2.3}$$

There is one minor issue we have not yet touched upon, and that is the constraint $\sum_{i=1}^{N+1} x_i \leq r$. In the real RAM algorithm, the optimal (real) solution x^* satisfies this constraint at equality. If, in the rational version of the algorithm, we compute an approximate solution $\tilde{x}$, the values for $\tilde{x}_j$ may be slightly greater than x_j^* for some or all $j \in \{1, \ldots, N+1\}$, thus violating this constraint. However, the problem can easily be solved by computing approximations $\tilde{x}_j$ that are close enough to x_j^*, but are guaranteed to be less or equal to the real solution values. As the square root approximations using the bisection algorithm (and many other algorithms) can easily be modified to yield an approximation that is less or equal (or greater or equal) to the exact value, the necessary modification is straightforward and just a matter of implementation; it does not change our estimates. Let us remark here that a more "technological approach" to that problem might be more appropriate in practice: If the approximations are accurate enough (which is possible for a well posed instance, as we have just seen), one can certainly guarantee that the spacing range r is never exceeded by more than a given parameter $\varepsilon_r > 0$. If, in turn, that parameter is kept sufficiently small (e. g., below the size of the smallest structures that can be manufactured in reality), a small violation of the spacing range constraint may be acceptable in applications, as minor errors are introduced through the manufacturing process anyway.

For the complexity of the algorithm on a binary Turing machine, let $SQRT(q_i, \varepsilon')$ denote the number of operations needed to compute a rational number $sqrt(q_i)$ such that $|sqrt(q_i) - \sqrt{q_i}| < \varepsilon'$. As we have argued above, this can be done, e. g., by bisection using at most $\mathcal{O}\left(\log_2 \frac{q_i}{\varepsilon'}\right)$ arithmetic operations. For a given maximum absolute error $\varepsilon > 0$ in the objective function of a well posed instance with wellness condition $\delta > 0$, we need to choose ε' according to (2.2) and (2.3), hence a value of

$$\varepsilon' = \frac{\varepsilon \min\left\{\delta, d^2 \cdot \left(\sum_{i=1}^{N+1} q_i\right)^{-1}\right\} (N+1) q_{\min}^2}{\frac{1}{2}(r - Nd)(3q_{\max} + 1) + \varepsilon \min\left\{\delta, d^2 \cdot \left(\sum_{i=1}^{N+1} q_i\right)^{-1}\right\} (N+1) q_{\min} + 1}$$

suffices (notice the "$+1$" at the end of the denominator to guarantee that ε' is strictly smaller than the original fraction). This results in the following corollary:

Corollary 2.10
A well posed instance $(N; q_1, \dots, q_{N+1}; d; r; \varepsilon)$ *of* Optimal Rational Wire Spacing *with wellness condition* $\delta > 0$ *can be solved on a binary Turing machine using at most*

$$\mathcal{O}\left(N \log N + N \cdot SQRT_{\max}\right)$$

operations, where

$$\begin{aligned} SQRT_{\max} = &\log_2(q_{\max}) - \log_2\left(\varepsilon \min\left\{\delta, \frac{d^2}{\sum_{i=1}^{N+1} q_i}\right\}(N+1)q_{\min}^2\right) \\ &+ \log_2\left(\frac{1}{2}(r - Nd)(3q_{\max}+1) + \varepsilon \min\left\{\delta, \frac{d^2}{\sum_{i=1}^{N+1} q_i}\right\}(N+1)q_{\min} + 1\right). \end{aligned}$$

(Here the second complexity term can be achieved using bisection for the approximation of the square roots.)

2.3 The Wire Ordering Problem

As Theorem 2.6 shows, an optimal wire spacing can be computed very efficiently. Furthermore, the characterization of the optimal wire spacing shows that a reordering of the wires adds an additional potential for optimization, but also mathematical difficulty. Let $\pi \in \mathcal{S}_N$ be a permutation that assigns the switching frequency $s_{\pi(i)}$ to the i-th position on the chip. Then by (2.1) the optimum of the objective function for the permutation π is

$$F(\pi, x^\pi) = \sum_{i \in D^\pi} \frac{s_{\pi(i-1)} + s_{\pi(i)}}{d} + \frac{1}{r - |D^\pi|\, d}\left(\sum_{i \in R^\pi} \sqrt{s_{\pi(i-1)} + s_{\pi(i)}}\right)^2, \tag{2.4}$$

where x^π is the optimal wire spacing and $D^\pi = D(x^\pi)$, and $R^\pi = R(x^\pi)$ according to Lemma 2.4. Here again, $s_0 = s_{N+1} = 0$ and $\pi(0) = 0$, $\pi(N+1) = N+1$ for a uniform notation, and we will identify $\mathcal{S}_N$ with the set of all permutations on $\{0, \dots, N+1\}$ with fixed points 0 and $N+1$.

Of course, we want to optimize over all such permutations $\pi \in \mathcal{S}_N$ now. Suppose for a moment, r was large enough to imply $D^\pi = \emptyset$ for each $\pi \in \mathcal{S}_N$. Then, in effect, we are asking for a permutation that minimizes

$$\sum_{i=1}^{N+1} \sqrt{s_{\pi(i-1)} + s_{\pi(i)}}. \tag{2.5}$$

This problem can be translated to a Minimum Hamilton Path problem on the complete graph $G = (V, E)$ with vertex set $V = \{0, 1, \dots, N+1\}$ and edge weights $d(\{j,k\}) := \sqrt{s_j + s_k}$

by asking for a minimum Hamilton Path with endpoints 0 and $N+1$. While the MHP problem is notoriously difficult (cf. [GJ79, problem GT39]), here we are dealing with a special class of efficiently solvable MHP (or equivalently TSP) problems. In this section, we will first show how to transform the problem to an easily solvable one in Section 2.3.1, mainly using a result due to Supnick [Sup57]. As an aside, we will also give an alternative derivation of the result which is less general (although sufficient for the problems considered in the context of this work), but uses only very elementary arguments in Section 2.3.2. Subsequently, Section 2.3.3 will provide some structural results on the set D to make the theory applicable to the Wire Ordering problem without the assumption of $D = \emptyset$. This will finally enable us to cast the results on the MHP problem into an efficient algorithm for Wire Placement in Section 2.4.

2.3.1 Utilizing the Supnick Property for Minimum Hamilton Path

Let $\omega_0, \omega_1, \dots, \omega_{N+1}$ denote the switching activities sorted in increasing order, i. e., $\omega_0 = s_0 = 0$, $\omega_1 = s_{N+1} = 0$ and $\{\omega_2, \dots, \omega_{N+1}\} = \{s_1, \dots, s_N\}$ with $\omega_0 \leq \omega_1 \leq \omega_2 \leq \cdots \leq \omega_{N+1}$. Then the problem of minimizing (2.5) is equal to finding a Minimum Hamilton Path in the complete graph on the vertices $\{0, 1, \dots, N+1\}$ with endpoints 0 and 1 whose edges $\{i, j\}$ carry the weights $\sqrt{\omega_i + \omega_j}$.

What makes this problem more tractable as opposed to the general MHP is the fact that the distance matrix

$$(d_{ij})_{i,j=0,\dots,N+1} \quad \text{with} \quad d_{ij} = \sqrt{\omega_i + \omega_j}, \tag{2.6}$$

has a structure known as *Monge property* in the context of Traveling Salesman problems.

Definition 2.11 (Hoffman [Hof63], see also [BDDVW98])
A matrix $(c_{ij})_{i,j=1,\dots,n}$ with nonnegative entries is called a *Monge matrix* (or said to have the *Monge property*), if

$$c_{ij} + c_{rs} \leq c_{is} + c_{rj} \quad \text{for all } 1 \leq i < r \leq n \text{ and } 1 \leq j < s \leq n.$$

A TSP with a Monge distance matrix can be solved to optimality in polynomial time using a dynamic programming approach based on the fact that there is an optimal tour that is pyramidal, i. e., it has the form $(i_1, i_2, \dots, i_{p-1}, n, i_{p+1}, \dots, i_{n-1}, i_n)$ with $i_1 \leq i_2 \leq \cdots \leq i_{p-1} \leq n$ and $n \geq i_{p+1} \geq \cdots \geq i_{n-1} \geq i_n$. Details on the algorithm and a proof can be found in [GLS85], an even more efficient refinement of that algorithm is presented in [Par91]. We will follow a similar approach in Section 2.3.2 and hence omit the details here.

A refinement of the Monge property was considered by Supnick.

Definition 2.12 (Supnick, [Sup57], see also [Bur90])
A matrix $(c_{ij})_{i,j=1,\dots,n}$ with nonnegative entries is called a *Supnick matrix* (or said to have the *Supnick property*), if it is symmetric and if

$$c_{ij} + c_{kl} \leq c_{ik} + c_{jl} \leq c_{il} + c_{jk} \quad \text{for all } 1 \leq i < j < k < l \leq n.$$

In his 1957 paper [Sup57] Supnick proved the following theorem (rephrased a little here):

Theorem 2.13 (Supnick [Sup57], see also [Bur90])
If the cost matrix of a Traveling Salesman *problem has the Supnick property, then*

$$(1, 3, 5, \dots, n, \dots, 6, 4, 2)$$

is an optimal tour.

Clearly, the distance matrix (d_{ij}) defined in (2.6) above is symmetric. The Monge property of (d_{ij}) is mainly due to the concavity of the square root function, we utilize a more general lemma to derive the inequality.

Lemma 2.14
Let $f : \mathbb{R} \to \mathbb{R}$ be concave; $x, y, \delta \in \mathbb{R}$ with $x \le y$ and $\delta \ge 0$. Then

$$f(x-\delta) + f(y) \le f(x) + f(y-\delta).$$

Furthermore, for strictly concave f, $x < y$ and $\delta > 0$ the above inequality is strict.

Proof. The case $\delta = 0$ is clear, so we assume $\delta > 0$. Since

$$\begin{aligned} x &= \frac{y-x}{y-x+\delta}(x-\delta) + \left(1 - \frac{y-x}{y-x+\delta}\right) y \\ \text{and} \quad y-\delta &= \frac{\delta}{y-x+\delta}(x-\delta) + \left(1 - \frac{\delta}{y-x+\delta}\right) y, \end{aligned}$$

concavity of f implies

$$\begin{aligned} f(x) &\ge \frac{y-x}{y-x+\delta} f(x-\delta) + \left(1 - \frac{y-x}{y-x+\delta}\right) f(y) \\ \text{and} \quad f(y-\delta) &\ge \frac{\delta}{y-x+\delta} f(x-\delta) + \left(1 - \frac{\delta}{y-x+\delta}\right) f(y). \end{aligned}$$

Addition of these two inequalities yields the asserted inequality. With strictly concave f and $x < y$, both above inequalities are strict, so their addition then yields the strict version of the assertion. □

Setting $x := \omega_i + \omega_l$, $y := \omega_k + \omega_l$ and $\delta := \omega_l - \omega_j$ in Lemma 2.14 immediately yields the Monge property for (d_{ij}).

It follows from a result of Burkard [Bur90] that (d_{ij}) is also a Supnick matrix. (Actually, Burkard not only proves that every symmetric Monge matrix has the Supnick property, which is essentially done by plugging symmetry into the Monge inequality to yield the Supnick inequalities, but also that every Supnick matrix can be transformed into a Monge matrix by possibly changing some of its diagonal elements.) The only "missing link" is now that between TSP and MHP: All the above results were stated for Traveling Salesman problems, while we are specifically interested in a Minimum Hamilton Path problem with given start and end nodes. But that can easily be resolved.

Lemma 2.15
Let $V = \{0, \dots, N+1\}$ be the node set of a complete undirected graph K_V with node weights $\{\omega_0, \dots, \omega_{N+1}\}$, such that $\omega_0 \leq \omega_1 < \omega_2 \cdots \leq \omega_{N+1}$ and edge lengths $d(\{i,j\}) = \sqrt{\omega_i + \omega_j}$. Then each minimum Traveling Salesman tour through K_V contains the edge $\{0,1\}$.

Proof. Suppose in the specified setting there was an optimal tour $\tau : \{0, \dots, N+1\} \to V$ that did not contain the edge $\{0,1\}$. We may assume that $\tau(0) = 0$ (the "start node" $\tau(0)$ of the tour may be chosen arbitrarily). Let $k := \tau^{-1}(1)$ and define a new tour $\tau' : \{0, \dots, N+1\} \to V$ as illustrated by Figure 2.4:

$$\tau'(i) := \begin{cases} \tau(k-1-i) & \text{for } 0 \leq i \leq k-2, \\ \tau(i+2) & \text{for } k-1 \leq i \leq N-1, \\ 0 = \tau(0) & \text{for } i = N, \\ 1 = \tau(k) & \text{for } i = N+1. \end{cases}$$

Obviously, τ' is a tour containing the edge $\{0,1\}$ with cost $\sqrt{\omega_0 + \omega_1}$; the cost of τ' is

$$\begin{aligned} C(\tau') = C(\tau) &- \sqrt{\omega_1 + \omega_{\tau(k+1)}} - \sqrt{\omega_0 + \omega_{\tau(1)}} \\ &+ \sqrt{\omega_0 + \omega_1} + \sqrt{\omega_{\tau(1)} + \omega_{\tau(k+1)}}. \end{aligned}$$

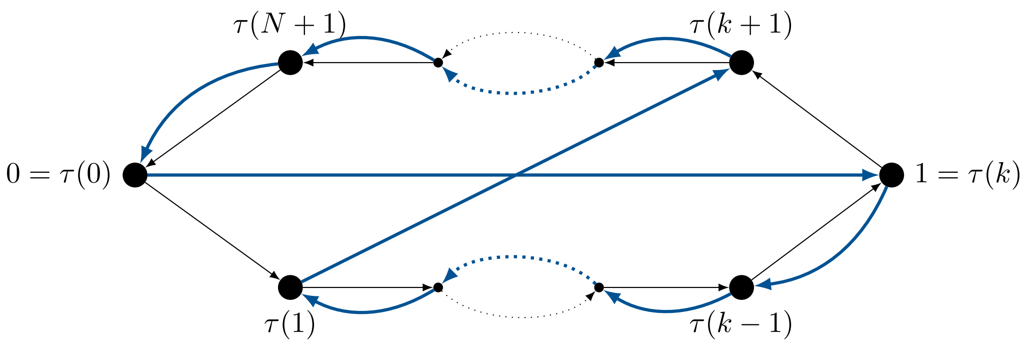


Figure 2.4: Definition of τ' (depicted in blue).

Application of Lemma 2.14 with

$$x := \omega_1 + \omega_{\tau(k+1)}, \quad y := \omega_{\tau(1)} + \omega_{\tau(k+1)} \quad \text{and} \quad \delta := \omega_{\tau(k+1)} - \omega_0$$

for the function $f(x) := \sqrt{x}$ (note $x < y$ and $\delta > 0$ due to $\omega_0 \leq \omega_1 < \cdots \leq \omega_{N+1}$) immediately yields $C(\tau') < C(\tau)$, contradicting optimality of τ. Thus τ must already contain the edge $\{0,1\}$. □

This shows that TSP and MHP on graphs that are relevant for the WIRE ORDERING problem are essentially the same: A minimum Hamilton Path can be found by simply solving a TSP and deleting the edge $\{0, 1\}$ that will always be contained, while on the other hand any minimum 0-1 Hamilton Path can be extended to an optimal TSP tour by adding $\{0, 1\}$.

A minimum Hamilton Path (and hence an optimal wire ordering) is therefore given by Theorem 2.13; reformulated using the notation of the present chapter we get the final result of this subsection.

Theorem 2.16
The WIRE ORDERING *problem for $D = \emptyset$ is solved to optimality by the permutation τ_N defined by*

$$\tau_N(i) = \begin{cases} 2i & \text{for } 0 \le i \le \frac{N+1}{2}, \\ 2(N-i)+3 & \text{for } \frac{N}{2}+1 \le i \le N+1 \end{cases}$$

and the corresponding distance vector x^{τ_N} determined by Algorithm 2.1.

At this point, let us briefly come back to the introduction in Section 1.2, where we shortly discussed the TURBINE RUNNER BALANCING problem (place turbine blades on the runner such as to minimize imbalance). It is a surprising fact that the very same permutation as in Theorem 2.16 also arises in the context of balancing turbine runners. Unfortunately, this happens only when one replaces the minimization of the imbalance by a maximization objective, where in contrast to that the minimization version is an $\mathcal{NP}$-hard problem. More details on this can be found in the article [Woe03].

2.3.2 An elementary approach to Concave Minimum Hamilton Path problems

As a side note, we will also give a proof of Theorem 2.16 that uses only elementary arguments. This approach is less general than that in [Sup57], but as Supnick uses a number of involved arguments, it may nevertheless be interesting on its own right. This subsection may be skipped without any loss for the rest of the text.

To simplify notation and make our treatment a little more general, we will first take up a more abstract position and define the problem of interest.

Problem 2.17: CONCAVE MINIMUM HAMILTON PATH (CMHP)
Instance: An integer $n \in \mathbb{N}$, nonnegative reals $(\omega_0, \omega_1, \dots, \omega_{n+1}) \in \mathbb{R}^{n+2}_{\ge 0}$ with $\omega_0 \le \omega_1 \le \dots \le \omega_{n+1}$, and edge weights $\phi : \{0, 1, \dots, n+1\}^2 \to \mathbb{R}_{\ge 0}$ such that there exists a concave function $f : \mathbb{R}_{\ge 0} \to \mathbb{R}_{\ge 0}$ with $\phi(i, j) = f(\omega_i + \omega_j)$ for all $i, j \in \{0, 1, \dots, n+1\}$.
Question: Find a permutation $\tau : \{0, 1, \dots, n+1\} \to \{0, 1, \dots, n+1\}$ with $\tau(0) = 0$ and $\tau(n+1) = 1$ that minimizes

$$C(\tau) = \sum_{i=1}^{n+1} \phi(\tau(i-1), \tau(i)).$$

The permutation τ is called *Hamilton Path from* 0 *to* 1 or short *tour*, $C(\tau)$ is referred to as the *cost* of τ and n will be called the *size* of the instance. Whenever it seems more appropriate we will express τ as the vector $(v_0, v_1, \ldots, v_{n+1})$ where $v_i = \tau(i)$.[6] Also, ω may be expressed in functional writing where convenient, i. e., $\omega(i) := \omega_i$ for $i \in V$.

If there exists even a strictly concave function $f : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ that induces ϕ, i. e., $\phi_{ij} := \phi(i,j) = f(\omega_i + \omega_j)$ for all $i, j \in V$, and if $\omega_0 \leq \omega_1 < \omega_2 < \cdots < \omega_{n+1}$ (i. e., the node weights are all distinct, with the possible exception of the two endpoints), we will speak of a Strictly Concave Minimum Hamilton Path problem or Strict-CMHP for short.

In the problem above, one may again associate a complete undirected graph, where the nodes will be $\{0, \ldots, n+1\}$ and the edges $\{i, j\}$ carry the weights ϕ_{ij}. Let us remark that the crucial requirement in CMHP is the existence of a *concave* function f that provides the edge weights. Of course, for $f(x) = \sqrt{x}$ this condition is fulfilled, so Wire Ordering for $D = \emptyset$ is a particular class of instances of CMHP.

We will now prove that τ_n, defined as in Theorem 2.16, is an optimal solution to CMHP.

Theorem 2.18
For a given instance of CMHP *of size* n *let* $\tau_n : \{0, 1, \ldots, n+1\} \to V$ *be defined by*

$$\tau_n(i) = \begin{cases} 2i & \textit{for } 0 \leq i \leq \frac{n+1}{2}, \\ 2(n-i)+3 & \textit{for } \frac{n}{2}+1 \leq i \leq n+1. \end{cases}$$

Then τ_n *is optimal. Furthermore, if the given instance of* CMHP *is strict,* τ_n *is the unique optimal tour.*

Proof. We proceed by induction on n. For $n = 1$ the tour τ_1 is the only feasible tour, hence it also is the unique optimal solution. So consider an instance on the node set $V = \{0, 1, \ldots, n+1\}$ with $n \geq 2$ and let σ_n be an optimal tour. Denote by τ_{n-1} and σ_{n-1} the corresponding tours on $\{0, 1, \ldots, n\}$ that are obtained by deleting the node $n+1$ from τ_n and σ_n, respectively. Then τ_{n-1} is just the tour defined in the statement of the theorem for $n-1$ instead of n, so it is optimal by the induction hypothesis, i. e.,

$$C(\tau_{n-1}) \leq C(\sigma_{n-1}). \tag{2.7}$$

Let σ_{n-1} be the sequence $(v_0, v_1, \ldots, v_{n-1}, v_n)$ with $v_0 = 0$ and $v_n = 1$ and let $k \in \{1, \ldots, n-1\}$ be chosen such that σ_n is the sequence $(v_0, v_1, \ldots, v_k, n+1, v_{k+1}, \ldots, v_n)$. By reversing σ_n if necessary (note that reversing does not change the cost of a tour), we may assume that

$$\omega(v_k) \leq \omega(v_{k+1}). \tag{2.8}$$

[6]Note that $v_0 = 0$ and $v_{n+1} = 1$, i. e., we think of v_0 as the first and v_{n+1} as the last node of the tour. Naturally, a reversal of this orientation does not change the problem.

Let $f : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ be a concave function that induces the edge weights ϕ, then we have

$$\begin{aligned} C(\tau_n) &= C(\tau_{n-1}) - f(\omega_{n-1} + \omega_n) + f(\omega_n + \omega_{n+1}) + f(\omega_{n-1} + \omega_{n+1}) \\ &\leq C(\sigma_{n-1}) - f(\omega_{n-1} + \omega_n) + f(\omega_n + \omega_{n+1}) + f(\omega_{n-1} + \omega_{n+1}) \\ &= C(\sigma_n) - f(\omega_{n-1} + \omega_n) + f(\omega_n + \omega_{n+1}) + f(\omega_{n-1} + \omega_{n+1}) \\ &\quad - f\big(\omega(v_k) + \omega_{n+1}\big) - f\big(\omega_{n+1} + \omega(v_{k+1})\big) + f\big(\omega(v_k) + \omega(v_{k+1})\big). \end{aligned} \tag{2.9}$$

Now, with

$$\begin{aligned} x = \omega_{n+1} + \omega(v_{k+1}), \quad y = \omega_n + \omega_{n+1}, \quad \delta = \omega_{n+1} - \omega_{n-1}, \\ \text{and} \quad \overline{x} = \omega(v_k) + \omega_{n+1}, \quad \overline{y} = \omega_{n-1} + \omega_{n+1}, \quad \overline{\delta} = \omega_{n+1} - \omega(v_{k+1}), \end{aligned} \tag{2.10}$$

we have

$$x \leq y, \quad \delta \geq 0, \qquad \overline{x} \leq \overline{y}, \quad \overline{\delta} \geq 0,$$

where $\overline{x} \leq \overline{y}$ is due to (2.8), as $\omega(v_k) \leq \omega(v_{k+1}) \leq \omega_n$ means $\omega(v_k) \leq \omega_{n-1}$. Thus application of Lemma 2.14 yields

$$f\big(\omega(v_{k+1}) + \omega_{n-1}\big) + f(\omega_n + \omega_{n+1}) \leq f\big(\omega_{n+1} + \omega(v_{k+1})\big) + f(\omega_{n-1} + \omega_n),$$

and

$$f\big(\omega(v_k) + \omega(v_{k+1})\big) + f(\omega_{n-1} + \omega_{n+1}) \leq f\big(\omega(v_k) + \omega_{n+1}\big) + f\big(\omega_{n-1} + \omega(v_{k+1})\big).$$

By adding these two inequalities we see that

$$C(\tau_n) \leq C(\sigma_n),$$

which proves optimality of τ_n.

For an instance of Strict-CMHP, first suppose $\tau_{n-1} \neq \sigma_{n-1}$. Then inequality (2.7) and hence also inequality (2.9) are strict, yielding $C(\tau_n) < C(\sigma_n)$, contradicting optimality of σ_n.

Thus $\tau_{n-1} = \sigma_{n-1}$, which means τ_n and σ_n can only differ by the position of node $(n+1)$. With $\sigma_n = (v_0, v_1, \ldots, v_k, n+1, v_{k+1}, \ldots, v_n)$ as above, we may again assume $\omega(v_k) \leq \omega(v_{k+1})$. Now if $\omega(v_{k+1}) \neq \omega_n$, then $x < y$ in (2.10), so by strict concavity of f, application of Lemma 2.14 yields

$$f\big(\omega(v_{k+1}) + \omega_{n-1}\big) + f(\omega_n + \omega_{n+1}) < f\big(\omega_{n+1} + \omega(v_{k+1})\big) + f(\omega_{n-1} + \omega_n),$$

leading to $C(\tau_n) < C(\sigma_n)$ as in the first part of the proof, again a contradiction. Along the same lines, if $\omega(v_k) \neq \omega_{n-1}$, then $\overline{x} < \overline{y}$ in (2.10), so

$$f\big(\omega(v_k) + \omega(v_{k+1})\big) + f(\omega_{n-1} + \omega_{n+1}) < f\big(\omega(v_k) + \omega_{n+1}\big) + f\big(\omega_{n-1} + \omega(v_{k+1})\big),$$

yielding the same contradiction as above. But then $\omega(v_{k+1}) = \omega_n$ and $\omega(v_k) = \omega_{n-1}$ must hold. As all node weights are distinct, this means $v_{k+1} = n$ and $v_k = n-1$, hence $\sigma_n = \tau_n$. So the optimal solution is indeed unique for Strict-CMHP. □

Of course, this result can be transferred to Concave Traveling Salesman problems (defined analogously to the CMHP problems) in the same way as outlined in the preceding subsection.

2.3.3 Characterization of the Minimum Distance Set D

We have now seen how to solve the Wire Placement problem provided that $D = \emptyset$. In practice, however, the set D is generally not empty. To make things worse, it is not even known in advance, and there are exponentially many possible candidates for D. In addition, recall that the general objective for an arbitrary wire ordering $\pi \in \mathcal{S}_N$ is

$$F(\pi, x^\pi) = \sum_{i\in D^\pi} \frac{s_{\pi(i-1)} + s_{\pi(i)}}{d} + \frac{1}{r - |D^\pi|\, d}\left(\sum_{i\in R^\pi} \sqrt{s_{\pi(i-1)} + s_{\pi(i)}}\right)^2,$$

cf. (2.4), which is not even a sum of edge weights and hence does not have the structure of an MHP or TSP, let alone the Monge property. So the naive approach of checking all different subsets of $\{1, \dots, N+1\}$ for the set D is pointless; not just due to complexity considerations, but also due to the fact that we do not yet have a way of solving the subproblems arising for fixed $D \neq \emptyset$. Actually, as the quadratic term indicates, the objective function value does not only depend on edge weights, but also on the selected Hamilton Path as a whole.

In the present section we derive some structural results to be able to handle the general wire ordering problem. We will show that there is always an optimal wire placement where π is *evenly separated*; see Definition 2.23. For such permutations the set D^π is already determined by its cardinality, which reduces the problem to the solution of N instances of the underlying MHP. In fact, as we will see later, we can even do better and determine the correct size of D in course of the algorithm without trying different possibilities. The following proofs utilize certain exchange techniques that will be introduced beforehand.

Definition 2.19
Let $(\pi, x) \in \mathcal{S}_N \times \mathbb{R}^{N+1}$ be a feasible wire placement, and let $j, k \in \{1, \dots, N\}$ with $j < k$. Define $S_{jk}(\pi), \overline{S}_{jk}(\pi) \in \mathcal{S}_N$ and $T_{jk}(x), \overline{T}_{jk}(x) \in \mathbb{R}^{N+1}$ by

$$S_{jk}(\pi)(i) = \begin{cases} \pi(k+j-i) & \text{for } j < i < k, \\ \pi(i) & \text{else;} \end{cases}$$

$$\overline{S}_{jk}(\pi)(i) = \begin{cases} \pi(k+j-i) & \text{for } j \le i \le k, \\ \pi(i) & \text{else;} \end{cases}$$

$$(T_{jk}(x))_i = (\overline{T}_{jk}(x))_i = \begin{cases} x_{k+j-i+1} & \text{for } j < i \le k, \\ x_i & \text{else.} \end{cases}$$

Then the operators

$$\begin{aligned}\mathscr{R}_{jk} &: \mathcal{S}_N \times \mathbb{R}^{N+1} \to \mathcal{S}_N \times \mathbb{R}^{N+1}, \quad \mathscr{R}_{jk}(\pi, x) := (S_{jk}(\pi), T_{jk}(x))\\ \overline{\mathscr{R}}_{jk} &: \mathcal{S}_N \times \mathbb{R}^{N+1} \to \mathcal{S}_N \times \mathbb{R}^{N+1}, \quad \overline{\mathscr{R}}_{jk}(\pi, x) := \left(\overline{S}_{jk}(\pi), \overline{T}_{jk}(x)\right)\end{aligned}$$

are called the *open* and *closed j-k reversal* respectively. Note that S_{jk}, T_{jk} and thus $\mathscr{R}_{jk}$ can likewise be defined for $k = N+1$.

Lemma 2.20
Let (π, x) be a feasible wire placement and let $j, k \in \{1, \dots, N\}$ with $j < k$. Then both, $\mathscr{R}_{jk}(\pi, x)$ and $\overline{\mathscr{R}}_{jk}(\pi, x)$, are feasible wire placements, and their objective values are

$$\begin{aligned}F(\mathscr{R}_{jk}(\pi, x)) &= F(\pi, x) + \left(s_{\pi(j)} - s_{\pi(k)}\right)\left(\frac{1}{x_k} - \frac{1}{x_{j+1}}\right),\\ F(\overline{\mathscr{R}}_{jk}(\pi, x)) &= F(\pi, x) + \left(s_{\pi(j)} - s_{\pi(k)}\right)\left(\frac{1}{x_{k+1}} - \frac{1}{x_j}\right).\end{aligned}$$

For the open j-k reversal, the result also holds for $k = N+1$.

Proof. Feasibility of both open and closed j-k reversal is clear, as the distance vectors $T(x) = \overline{T}(x)$ are just permuted versions of x, so the overall sum stays constant and no distance can fall below d. The new objective values are

$$\begin{aligned}F(\mathscr{R}_{jk}(\pi, x)) &= F(\pi, x) - \frac{s_{\pi(j)} + s_{\pi(j+1)}}{x_{j+1}} - \frac{s_{\pi(k-1)} + s_{\pi(k)}}{x_k}\\ &\quad + \frac{s_{\pi(j)} + s_{\pi(k-1)}}{x_k} + \frac{s_{\pi(j+1)} + s_{\pi(k)}}{x_{j+1}}\\ &= F(\pi, x) + \left(s_{\pi(j)} - s_{\pi(k)}\right)\left(\frac{1}{x_k} - \frac{1}{x_{j+1}}\right)\\ \text{and} \quad F(\overline{\mathscr{R}}_{jk}(\pi, x)) &= F(\pi, x) - \frac{s_{\pi(j-1)} + s_{\pi(j)}}{x_j} - \frac{s_{\pi(k)} + s_{\pi(k+1)}}{x_{k+1}}\\ &\quad + \frac{s_{\pi(j-1)} + s_{\pi(k)}}{x_j} + \frac{s_{\pi(j)} + s_{\pi(k+1)}}{x_{k+1}}\\ &= F(\pi, x) + \left(s_{\pi(j)} - s_{\pi(k)}\right)\left(\frac{1}{x_{k+1}} - \frac{1}{x_j}\right),\end{aligned}$$

respectively, completing the proof. □

Definition 2.21
Let (π, x) be a feasible wire placement, and set

$$\begin{aligned}l(x) &= \min\left\{i \in \{0, \dots, N\} : x_{i+1} > d\right\},\\ u(x) &= \max\left\{i \in \{1, \dots, N+1\} : x_i > d\right\}.\end{aligned}$$

Then $l(x)$ and $u(x)$ are called *lower* and *upper separation point*, respectively. For $x = x^\pi$ we use the abbreviations $l^\pi = l(x^\pi)$ and $u^\pi = u(x^\pi)$ (or, when there is no risk of confusion, simply l and u, respectively). An optimal wire placement (π, x) is called *separated*, if it has the following properties:

1. $x_{l+1}, x_{l+2}, \dots, x_u > d$
2. $\max\left\{s_{\pi(0)}, \dots, s_{\pi(l-1)}, s_{\pi(u+1)}, \dots, s_{\pi(N+1)}\right\} \le \min\left\{s_{\pi(l)}, s_{\pi(u)}\right\}$
3. $\max\left\{s_{\pi(l)}, s_{\pi(u)}\right\} \le \min\left\{s_{\pi(l+1)}, \dots, s_{\pi(u-1)}\right\}$

Lemma 2.22
For every feasible instance of Optimal Wire Placement *there is an optimal solution* (π, x^π) *that is separated. Furthermore, for an all-distinct instance, each optimal solution is separated.*

Proof. Suppose for some instance of OWP there is no optimal solution with Property 1. Let (π, x^π) be an optimal solution for which $l = l^\pi$ is maximal. Then there is some $k \in \{1, \dots, N\}$ with $l + 1 < k < u$ such that $x^\pi_k = d$, and we choose the maximal such k, i. e., $x^\pi_{k+1}, \dots, x^\pi_u > d$. Suppose first that $s_{\pi(l)} > s_{\pi(k)}$ (note this implies $s_{\pi(l)} > 0$ and thus $l > 0$). Then, by Lemma 2.20, the closed l-k reversal $\overline{\mathscr{R}}_{lk}(\pi, x^\pi)$ is feasible and has objective value

$$F(\overline{\mathscr{R}}_{lk}(\pi, x^\pi)) = F(\pi, x^\pi) + \underbrace{\left(s_{\pi(l)} - s_{\pi(k)}\right)}_{>0} \underbrace{\left(\frac{1}{x^\pi_{k+1}} - \frac{1}{x^\pi_l}\right)}_{<0} < F(\pi, x^\pi),$$

a contradiction to the optimality of (π, x^π). If, on the other hand, $s_{\pi(l)} \le s_{\pi(k)}$, then

$$F(\mathscr{R}_{lk}(\pi, x^\pi)) = F(\pi, x^\pi) + \underbrace{\left(s_{\pi(l)} - s_{\pi(k)}\right)}_{\le 0} \underbrace{\left(\frac{1}{x^\pi_k} - \frac{1}{x^\pi_{l+1}}\right)}_{>0} \le F(\pi, x^\pi).$$

If $s_{\pi(l)} < s_{\pi(k)}$, we even get strict inequality here, which again contradicts optimality of (π, x^π); this proves Property 1 holds at optimality for an all-distinct instance. For $s_{\pi(l)} = s_{\pi(k)}$, the open l-k reversal $\mathscr{R}_{lk}(\pi, x^\pi)$ is also an optimal solution, but its lower separation point is strictly greater than l, contradicting maximality of l. This shows that there is an optimal solution with Property 1 even for non-distinct instances.

In the following, let (π, x^π) be an optimal wire placement with Property 1, and let l and u be its lower and upper separation points. We prove by contradiction that (π, x^π) also has Properties 2 and 3. So, suppose there was some $m \in \{1, \dots, l-1\} \cup \{u+1, \dots, N\}$ such that $s_{\pi(m)} > \min\left\{s_{\pi(l)}, s_{\pi(u)}\right\}$. By reversing the wire placement if necessary, we may assume that $m \in \{1, \dots, l-1\}$. Let us first consider the case $u = N+1$, then $s_{\pi(m)} > s_{\pi(u)} = 0$.

Application of an open m-$(N+1)$ reversal yields

$$F(\mathscr{R}_{m,N+1}(\pi, x^\pi)) = F((\pi, x^\pi)) + \underbrace{\left(s_{\pi(m)} - s_{\pi(N+1)}\right)}_{>0} \underbrace{\left(\frac{1}{x_{N+1}} - \frac{1}{x_{m+1}}\right)}_{<0} < F((\pi, x^\pi))$$

contradicting optimality of (π, x^π).

For $u < N+1$, we may assume that $s_{\pi(m)} > s_{\pi(l)}$, else applying a closed l-u-reversal yields

$$F(\overline{\mathscr{R}}_{lu}(\pi, x^\pi)) = F(\pi, x^\pi),$$

since $x_l = x_{u+1} = d$. But then, as $x^\pi_m = d < x^\pi_{l+1}$, we have for the closed m-l reversal

$$F(\overline{\mathscr{R}}_{ml}(\pi, x^\pi)) = F(\pi, x^\pi) + \underbrace{\left(s_{\pi(m)} - s_{\pi(l)}\right)}_{>0} \underbrace{\left(\frac{1}{x^\pi_{l+1}} - \frac{1}{x^\pi_m}\right)}_{<0} < F(\pi, x^\pi),$$

contradicting optimality of (π, x^π).

For Property 3, suppose that there was some $m \in \{l+1, \dots, u-1\}$ such that $s_{\pi(m)} < \max\left\{s_{\pi(l)}, s_{\pi(u)}\right\}$. Note that $\pi(N+1) = N+1$ and $s_{N+1} = 0$, hence $u \leq N$.

We may again assume without loss of generality that $s_{\pi(l)} \leq s_{\pi(u)}$, using the same reasoning as for Property 2 above. Since $x^\pi_{u+1} = d < x^\pi_m$ by Property 1, we have

$$F(\overline{\mathscr{R}}_{mu}(\pi, x^\pi)) = F(\pi, x^\pi) + \underbrace{\left(s_{\pi(m)} - s_{\pi(u)}\right)}_{<0} \underbrace{\left(\frac{1}{x^\pi_{u+1}} - \frac{1}{x^\pi_m}\right)}_{>0} < F(\pi, x^\pi),$$

contradicting optimality of (π, x^π). This completes the proof. □

By Lemma 2.22 we can restrict our search for an optimal wire placement to separated solutions. Since the first part of the objective function $F(\pi, x^\pi)$ (cf. (2.4)) is just

$$\frac{1}{d} \cdot \sum_{i \in D^\pi} \left(s_{\pi(i-1)} + s_{\pi(i)}\right),$$

permuting the elements in positions $i \in \{1, \dots, l-1\} \cup \{u+1, \dots, N\}$ will not change the objective function value. Therefore, we can "normalize" the set D^π even further. We will do this in a way that is not only natural, but also most suitable for the subsequent application of the results obtained in the preceding subsections. For simplicity of exposition we will assume from now on that the wires are indexed by increasing switching frequency (with the exception of $s_{N+1} = 0$), i. e.,

$$0 = s_{N+1} = s_0 \leq s_1 \leq \cdots \leq s_N.$$

Definition 2.23
Let (π, x) be a separated wire placement with separation points l and u, and let $\Delta = |D(x)|$. Then (π, x) is called *evenly separated*, if the following conditions hold:

1. $D(x) = \left\{1, \dots, \left\lfloor \frac{\Delta}{2} \right\rfloor\right\} \cup \left\{N + 2 - \left\lceil \frac{\Delta}{2} \right\rceil, \dots, N + 1\right\}$;
2. $l = \left\lfloor \frac{\Delta}{2} \right\rfloor$ and $\pi(l) = 2\left\lfloor \frac{\Delta}{2} \right\rfloor$;
3. $u = N + 1 - \left\lceil \frac{\Delta}{2} \right\rceil$ and $\pi(u) = 2\left\lceil \frac{\Delta}{2} \right\rceil - 1$;
4. $\{\pi(i) : i \in \{0, \dots, l\}\} = \left\{0, 2, \dots, 2\left\lfloor \frac{\Delta}{2} \right\rfloor\right\}$;
5. $\{\pi(i) : i \in \{u, \dots, N+1\}\} = \left\{1, 3, \dots, 2\left\lceil \frac{\Delta}{2} \right\rceil - 1\right\} \cup \{N+1\}$.

Note that these conditions, in particular the last two requirements, aim at “compatibility” of the set $D(x^\pi)$ with the minimum Hamilton Path τ_N introduced in Theorem 2.16.

Theorem 2.24
Each feasible instance of OWP *admits an optimal solution that is evenly separated.*

Proof. Let (π, x^π) be an optimal wire placement and set $\Delta = |D^\pi|$. By Lemma 2.22, we can assume that (π, x^π) is separated. Hence by Definition 2.21, Properties 2 and 3, we may further assume that

$$\{\pi(l), \pi(u)\} = \left\{2\left\lfloor \frac{\Delta}{2} \right\rfloor, 2\left\lceil \frac{\Delta}{2} \right\rceil - 1\right\} = \{\Delta - 1, \Delta\}\,.$$

(Here we use that $0 = s_{N+1} = s_0 \le s_1 \le \dots \le s_N$.) By reversing the wire placement if necessary we obtain $\pi(l) = 2\left\lfloor \frac{\Delta}{2} \right\rfloor$. Similarly, we may assume

$$\{\pi(i) : i \in \{0, \dots, l\} \cup \{u, \dots, N+1\}\} = \{0, 1, \dots, \Delta\} \cup \{N+1\}\,.$$

The objective function then evaluates to

$$F(\pi, x^\pi) = \frac{2}{d}\sum_{i=1}^{\Delta-2} s_i + \frac{s_{\Delta-1} + s_\Delta}{d} + \frac{1}{r - d\cdot\Delta}\left(\sum_{i=l+1}^{u} \sqrt{s_{\pi(i-1)} + s_{\pi(i)}}\right)^2. \tag{2.11}$$

Let $L := \{0, \dots, l-1\} \cup \{u+1, \dots, N+1\}$, then $\pi|_L : L \to \{0, \dots, \Delta - 2\} \cup \{N+1\}$ is a bijection; and changing π to some permutation π' that differs from π only on the set L (i. e., $\pi|_{\{0,\dots,N+1\}\setminus L} = \pi'|_{\{0,\dots,N+1\}\setminus L}$) does not alter the objective value, since $\pi'(L) = \{0, \dots, \Delta - 2\} \cup \{N+1\}$. Doing so appropriately one can clearly alter π (thus reordering the positions in L) to obtain an optimal solution (π', x) that is evenly separated. $\square$

With the last theorem we can now restrict the search for an optimal wire placement to evenly separated solutions. The next section will combine this result with those of Section 2.3 to obtain a unified algorithm for Optimal Wire Placement.

2.4 An $\mathcal{O}(N \log N)$ Algorithm for the Wire Placement Problem

We have now reached the following principle procedure for finding optimal wire placements: If $r < (d+1)N$, the problem is infeasible; if $r = (d+1)N$, then for $x^* = d \cdot \mathbb{1}$ the complete set of optimal solutions is $\{x^*\} \times \mathcal{S}_N$. Otherwise, for each of the N possible values for Δ (recall there must be at least one non-minimal distance), we compute a set $D(\Delta)$ of cardinality Δ according to Theorem 2.24. Then, we are in effect confronted with the Minimum Hamilton Path problem on the node set $V(\Delta) = \{(\Delta-1), \ldots, N\}$ with edge weights $\sqrt{s_i + s_j}$ for $i, j \in V(\Delta)$ and $i \neq j$ and endpoints $(\Delta - 1)$ and Δ that is obtained by ignoring the square in the second component of (2.4). We solve this MHP problem by utilizing the underlying Supnick property, cf. Theorem 2.16. In the context of Optimal Wire Placement, the optimal permutation τ_N can be obtained by a simple algorithmic procedure: Start by placing the right and left border wires, order the wires by (weakly) increasing switching frequencies, and place them one after the other, always positioning one wire in between its two already placed predecessors.

The two parts can, however, be closely interwoven. In fact, the procedure for computing an optimal order for the wires between lower and upper separation point is in full accordance with the property of a wire placement to be evenly separated, cf. Definition 2.23. Hence, when the wires at minimum distance are added, the optimal tour τ_N coincides with the tour produced by adding to both sides of $\tau_{N-\Delta}$ the remaining wires at minimum distance in a way that yields an evenly separated wire placement. This means we obtain the same wire ordering for every value of Δ. So, rather than actually going through all different values for Δ, we can just compute an optimal wire ordering as if there was no minimum distance requirement and subsequently determine the optimal distances (and the correct set of minimum distance wires) along the lines of Algorithm 2.1 to obtain an optimal wire placement. The complete procedure is formalized in Algorithm 2.2

Algorithm 2.2: Solving Optimal Wire Placement

```
Input: A feasible instance of Optimal Wire Placement, i. e.,
       $N \in \mathbb{N}$; $s_1, \ldots, s_N \in [0, \infty[$ and $d, r \in ]0, \infty[$ with $r > (N+1)d$.
Output: An optimal wire placement $(\pi, x^\pi)$.
1 Set $s_0 = s_{N+1} = 0$ and sort $s_1, \ldots, s_N$ in increasing order, i. e., let $\rho \in \mathcal{S}_N$ be a permutation
  such that $0 = s_{\rho(0)} = s_{\rho(1)} \leq s_{\rho(2)} \leq \cdots \leq s_{\rho(N+1)}$.
2 Define the permutation $\pi \in \mathcal{S}_N$ by setting
      $$\pi(i) := \begin{cases} 2\rho(i) & \text{for } 0 \leq i \leq \frac{N+1}{2}, \\ 2(N - \rho(i)) + 3 & \text{for } \frac{N}{2} + 1 \leq i \leq N+1. \end{cases}$$
3 Set $q_i := s_{\pi(i-1)} + s_{\pi(i)}$ for $i = 1, \ldots, N+1$.
4 Compute optimal distances $x^\pi$ for the permutation $\pi$ and input parameters
  $(N, q_1, \ldots, q_{N+1}, r, d)$ using Algorithm 2.1.
```

Theorem 2.25
In the real RAM model of computation, Optimal Wire Placement *can be solved in time* $\mathcal{O}(N \log N)$*. In the binary Turing machine model, given* $\varepsilon > 0$ *and an instance of* Optimal Wire Placement *such that the associated instance of* Optimal Wire Spacing *for the optimal permutation is well posed with wellness condition* $\delta > 0$*, a feasible rational wire placement* $(\pi, \tilde{x}^{\pi})$ *with*

$$|F(\pi, \tilde{x}^{\pi}) - F(\pi, x^{\pi})| < \varepsilon$$

can be computed in time

$$\mathcal{O}(N \log N + N \cdot SQRT_{\max}), \text{ where}$$

$$SQRT_{\max} = \log_2(q_{\max}) - \log_2\left(\varepsilon \min\left\{\delta, d^2 \cdot \left(\sum_{i=1}^{N+1} q_i\right)^{-1}\right\}(N+1)q_{\min}^2\right)$$
$$+ \log_2\left(\frac{1}{2}(r - Nd)(3q_{\max}+1) + \varepsilon \min\left\{\delta, d^2 \cdot \left(\sum_{i=1}^{N+1} q_i\right)^{-1}\right\}(N+1)q_{\min} + 1\right).$$

Proof. The correctness and optimality of the solution (π, x^{π}) produced by Algorithm 2.2 is a direct consequence of Theorems 2.6, 2.24 and 2.16.

The sorting step in the algorithm can be implemented using no more than $\mathcal{O}(N \log N)$ arithmetic operations, an optimal wire ordering π can subsequently be computed in $\mathcal{O}(N)$ steps. Using Algorithm 2.1 to compute a corresponding optimal wire ordering requires at most $\mathcal{O}(N \log N)$ arithmetic operations in the real RAM model. For a well posed instance of Wire Spacing, a feasible rational wire spacing corresponding to the optimal permutation π of the wire placement can be computed using at most the number of operations given in the theorem on a binary Turing machine using Corollary 2.10 and a bisection algorithm for computing rational approximations to the square roots appearing in Algorithm 2.1 as described in the relevant parts of Section 2.2. □

2.5 Concluding Remarks

2.5.1 Electro-technical Significance

When dealing with a mathematical abstraction of a real world problem, one certainly has to be prepared to verify the results obtained for their significance to the problem that motivated the mathematical study. In this final section of the chapter, let us briefly comment on the issues raised in this context.

In comparison with measurements performed in experiments and simulations, the mathematical model given and justified in Section 2.1 turned out to be very realistic when dealing with parallel wires. Naturally, in order to fully exploit the potential of Wire Placement one would need to fully integrate spacing and ordering into the complete logical and physical design and layout process.

As was remarked in the derivation of the model in Section 2.1, our model only encompasses a layout made up of parallel wires. This setting is not uncommon in real world digital circuits, though. Straightforward application of our method can be performed for bus wires (usually up to 128 parallel wires connecting logical and/or memory units of a circuit) frequently found in microprocessors, embedded systems and (most prominently) in multi-core architectures. Of course, in a general-purpose semiconductor circuit, not all wires are bus wires. However, in a layered layout, the wires in every layer run in one common direction, so they are all parallel. These wires usually do not have the same length, so our results are not directly applicable to the layer as a whole, but rather to separate groups of parallel wires within one layer. In his doctoral thesis [Zub07], Paul Zuber is concerned with the question of how to find such areas to apply wire spacing (basically using a scan line algorithm).

One difficulty when dealing with wire placement is the integration of the results into the existing design process. To date, wire spacing and ordering aiming at reduced power loss is not incorporated in any of the major commercial software packages used for semiconductor circuit design. Applying wire spacing in an already finished design poses relatively little problems: Groups of parallel wires that admit wire spacing are easily identified, calculation of the optimal distances within these local patches is also straightforward, utilizing our algorithm for wire spacing. The actual application of these results may call for a displacement of some of the wires on the chip. Usually, these wires are connected to wires in other layers by vias, so when moving the wires, a new connection between wire ends and vias has to be introduced, cf. Figure 2.3. Luckily, these "detour connectors" are very short compared to the wires themselves, so their effect is negligible. This process is easily implemented as an ex-post optimization measure in commercial design tools, enabling a circuit designer to exploit the potential of existing routing tools before our results are utilized for further optimization. Experiments along these lines showed that optimal spacing within local patches of a large real-world semiconductor design (identified by a simple search procedure) already leads to an overall reduction in power consumption of $3 - 5\%$. The effect was even greater when optimal wire spacing was applied to a broad range of benchmark circuits produced by state-of-the-art commercial layout tools as a post layout optimization step. A comprehensive study can be found in [Zub07].

For the application of wire ordering in semiconductor design considerably higher effort is needed. Changing the positions of wires in an otherwise finished circuit design would destroy the connections to wires in neighboring layers, so new connectors are needed. However, these cannot normally be placed on the layer where wire ordering is applied, because unlike with pure wire spacing these connectors could cross each other leading to short circuits. Therefore, an additional layer (called a *permutation network*) has to be introduced that contains those so called *cross connectors*, adding the need for more material and production time. The effect of such permutation networks has been investigated in [MMP01]; it is concluded that in realistic circuits the overhead in terms of area and power is much more than compensated by the benefit of wire ordering (one should remark that the authors used a heuristic to determine a good wire ordering). Also, for some local buses frequently found in microprocessors (e. g., address buses or counters), the switching frequencies are known or can at least be accurately estimated before the design process is started. In that case, the connector pins of the units that are to be connected by bus wires can often be reordered without affecting the rest of the design too much, so an optimal (or at least a good) wire ordering can be implemented there. A similar approach exploits the fact that today's circuit designs are mostly IP-based (i. e., "Intellectual Property" based), hence pre-developed modules are frequently re-used and combined for new designs. The ordering of such a *module's* connectors (called *pads*) is generally fixed arbitrarily at the design of a module and is not subject to optimization during the design process of the chip. Thus the designers of the IP modules may apply optimal wire ordering as proposed in this thesis to the initial design of the modules. This methodology is all the more important as switching activities are usually module specific and one module might be reused in thousands of different designs, thereby multiply compensating for the additional effort. However, in the long run the full potential of wire ordering can only be exploited by completely integrating it into the design process, which calls for a joint treatment in an extended unified model encompassing all relevant factors.

2.5.2 Directions for Future Research

Of course, a unified model as discussed above will require a major research effort, because other than power issues, a lot of other factors have to be considered, e. g., timing issues or yield maximization, to name just a few that are loosely related to wire spacing and wire ordering.

Of greater relevance for our field of research is the issue of correlated switching frequencies. In the derivation of our model, we mentioned the assumption that no two adjacent wires switch at the same time. While this is not an unrealistic assumption in real world circuits, there are also many applications where the voltage levels on adjacent wires are related to some extent. Consider, for instance, a counter, where the wires represent the bits of a binary number that is increased in uniform intervals – every time a higher bit changes from 1 to 0 all the lower bits do the same, while the next highest bit changes from 0 to 1; so the switching frequencies of a counter exhibit a high correlation.

This behavior is problematic to the viability of our model, because we calculate the power loss of a single wire as being proportional to $1/x_{\text{left}}$ and $1/x_{\text{right}}$ whenever a switch on that wire occurs. However, in case one or both adjacent wire(s) switch at the same time, the effect changes: For two adjacent wires doing the same signal transition (either from 0 to 1 or vice versa), the electric field between these wires does not change at all, so no power loss is incurred. On the other hand, if two adjacent wires switch in adverse directions (one from 0 to 1, the other from 1 to 0), the effect doubles, so twice the energy is consumed by the electric field that builds up between the wires.

Mathematically speaking, we have to incorporate terms to measure the amount of synchronous switching in both directions, thus introducing correlations in place of the switching frequencies. Unfortunately, this leads to an objective function that is not so nicely decomposable, and hence not amenable to many of our arguments, particularly the interchange arguments frequently used in our proofs. In our above treatment, we assumed the switching probabilities of different wires to be independent, thus avoiding the problem completely. Although this approach models reality quite concisely in a rich variety of application, there are some cases occurring very frequently in common semiconductor circuits (e. g., the counter mentioned above) where the correlation between switching frequencies of different wires cannot be ignored; so a refined model and a different treatment will be needed to handle the additional complexity introduced into the model by paying regard to these inter-wire correlations.

Another point for further research comes from the extra wiring introduced into a finished design by displacing some of the wires in order to do wire spacing. Apart from the extra material needed for these "detour wires" also new parallel wires are introduced into the layout. Although these wires are usually very short (and experiments show they have little influence on power loss), at least an approximation of the additional costs incurred in routing these wires could be taken into the objective function. In [Zub07], a modification of our model is suggested that incorporates additional displacement costs that are proportional to the length of the extra wiring. Unfortunately, the model is non-differentiable and nonlinear, making it a little unpleasant to deal with from a mathematical viewpoint. Also, the algorithm suggested for its solution can produce non-optimal results (although it performs quite well in practice). Here, spending some thoughts on an adequate model incorporating wire displacement may be an interesting future challenge.

Chapter 3

Flight Scheduling Problems — Complexity, Structure and Algorithms

In this chapter we study several variants of a combinatorial packing problem under balancing constraints motivated by an applications in flight scheduling for airport operations. After a brief description of the problem, we will present a mathematical formulation capturing the core part of the problem in Section 3.1. This will lead to three combinatorial optimization problems, namely FLIGHT SCHEDULING (how to design an optimal flight schedule, given a set of flight requests), MAXIMUM SLOT PACKING (determine the maximum number of flights in a schedule) and MINIMUM SLOT COVER (how to"block" a schedule using a minimum number of flights), which will be investigated in detail in Sections 3.2–3.5. In Section 3.6, we will focus on the gap between a maximum slot packing and a minimum slot cover, and thus on the question of what separates a "good" from a "bad" flight schedule, and discuss the question of how to possibly avoid this gap. Section 3.7 contains some concluding remarks. A more detailed description of the practical background and all relevant constraints can be found in Chapter 4, where we will develop a concise mathematical model for the complete real-world problem, utilizing results from the present chapter.

3.1 Slot Allocation and Flight Scheduling

The scheduling of flights from an airport's perspective has both a long-term and a short-term aspect. Short-term planning is implemented as part of the normal operations of an airport, where frequently flights have to be slightly rescheduled due to delay, weather, and other environmental conditions. The rescheduling is, of course, based on a regular long-term schedule, which is devised by a thorough planning process. In this long-term planning, a new schedule is created twice a year, one for the summer season (roughly ranging from March to October, exact dates vary) and one for the winter season (ranging from October to March). Airport capacity is naturally limited, so the objective of long-term scheduling is to strike a balance between limited resources of an airport and the demands of airlines wishing to offer an air connection to and from that airport.

As a means of controlling the allocation of airport capacity the so-called *slot system* has been established by the *IATA (International Air Transport Association)* and by national and international legislation (see, e. g., [EU93; EU02; EU03; EU04] for the regulations in the European Union). Under the slot system, an airport's capacity is allocated in the form of *slots*, which designate the right for an airline to operate a landing or a take-off at some specified time at a specified airport. Any airline that wants to offer a service at an airport that implements the slot system needs to acquire a pair of arrival and corresponding departure slots. Of course, capacity is not a scarce resource everywhere, so not all airports implement the slot system. But at most major airports worldwide, there are at least certain peak times when demand widely exceeds the available capacity, thus calling for an allocation procedure. Airports where the slot system is implemented are designated *fully coordinated airports*. The allocation of slots to airlines (more precisely to airlines' flight requests) is implemented by an *airport coordinator* (often belonging or affiliated to some national authority), who is independent of both airports and airlines. The process of slot allocation will be explained in greater detail in Chapter 4; in the present chapter we will concentrate on the key aspects of the underlying planning problem to gain insight into the structures of optimal and also of non-optimal flight schedules. A more in-depth analysis of all practically relevant constraints and the development of a concise model leading to a solution of the real-world problem will also be the topic of Chapter 4.

We will look at the question of how to allocate the available slots to the airlines under two slightly different aspects in this chapter. First, a formal definition of slots, airport capacity, flight requests and flight schedule is in order. The problem of allocating slots to specific flight requests will then be investigated in Section 3.2. In Sections 3.3 and 3.4, we adopt a more abstract point of view and ask how flights should be distributed over the planning horizon in order to obtain a schedule with a maximum number of flight movements. Hence we do not rely on a set of flight requests any more, but instead devise results on the structure of optimal flight schedules. Of course, the counterpart to that question, namely how a "bad" flight schedule looks like (and how many flights it can accommodate), will also be an important subject, we will consider that question in Section 3.5. In Section 3.6 we will combine these two viewpoints and concern ourselves with the gap between good and bad flight schedules and means to decrease or avoid that gap in the flight scheduling process.

3.1.1 Flight Requests and Flight Schedules

An important class of constraints in flight scheduling is naturally imposed by the airlines' demands. In practice, these demands are communicated to the airport coordinator as *flight requests* or *series requests*, the coordinator then tries to match these with the airport's capacity restrictions to obtain a feasible flight schedule. Formally, this part of the problem is a specific kind of assignment problem. For the rest of this chapter let $\mathcal{S} = \{1, \ldots, n\}$ denote the *slot set* (by a slot set, we always mean a set of the form $\{1, \ldots, n\}$ for some $n \in \mathbb{N}$), which may be thought of as a discretization of the planning horizon. Usually, one slot marks a small time interval of ten (sometimes five) minutes, hence allocating a slot to an airline amounts to granting it the right to land or take off (depending on the type of slot allocated) within the respective time interval. The set $\mathcal{S}$ simply is a labeling of these time intervals in chronological order. Of course, as a slot denotes a whole time interval, one slot is not limited to one flight, but can accommodate several flight movements, possibly of different type (arrivals and departures).

In Chapter 4 we will take up a very problem-oriented position, enabling us to solve real-world instances of flight scheduling problems. In the present chapter, the foundation for this work will be laid by providing a more abstract and general perspective. However, in some cases it will be convenient (and intuitive) to use the same terminology in both chapters, often with a more specific meaning in Chapter 4. Where this might lead to confusion, we will prepend this chapter's definitions with the word *abstract*, as opposed to their meaning in Chapter 4. In most cases where the same term is used, the meaning in Chapter 4 will just denote a different way of specifying the same kind of data.

Airlines express their flight requests by specifying for each planned flight a number of alternative slots or slot pairs.

Definition 3.1 (Abstract Slot Request, Abstract Flight Request, Abstract Series Request)
Let $\mathcal{S} = \{1, \dots, n\}$ be a slot set.

1. An *abstract slot request* is a nonempty subset $G \subset \mathcal{S}$, specifying a number of alternative slots that may be allocated for a single requested flight movement.

2. A nonempty set of tuples

$$F \subset \mathcal{S} \times \mathcal{S} \quad \text{such that} \quad a < d \quad \text{for each } (a,d) \in F$$

   specifies alternative arrival/departure pairs for a flight and is called *(single) abstract flight request.* A tuple $(a,d) \in F$ of a flight request F is called *corresponding arrival/departure slot pair* or just *slot pair*, its components will be referred to as *requested arrival slot* and *requested departure slot*, respectively.

3. An *abstract series request* $(F, \mathcal{I})$ consists of a single abstract flight request F and a *starting point set* $\mathcal{I} \subset (\mathcal{S} \setminus \{n\}) \cup \{0\}$ where $0 \in \mathcal{I}$. The sets

$$S^{\mathcal{I}}_{(a,d)} := \{(a+t, d+t) : t \in \mathcal{I} \wedge a+t, d+t \in \mathcal{S}\}, \quad (a,d) \in F,$$

   are called *feasible slot series* for the series request $(F, \mathcal{I})$.

An abstract slot or flight request expresses an airline's desire to be granted one of the slots or slot pairs contained in the request for a planned flight. The slot request is the simplest form of request considered here and will not receive much attention in the following, as the theory for slot requests is often the same as for flight requests. However, slot requests do frequently arise in practice, namely in situations where an airline receives a major fraction of all slots available at an airport. In that case, the airline often prefers to just request single slots and connect arrival and departure slots on its own account after the slot allocation has been finalized. See Chapter 4 for more details.

While an abstract flight request is a set of feasible arrival/departure pairs for a flight, an abstract series request models the fact that an airline will normally not request just one slot pair (for a single flight), but a number of slot pairs on several days for the same service during a longer time. An initial slot pair allocated to such a series request implies that the flight is scheduled at the same time of day for each day where it is requested. The first slots of these days (i. e., usually the slot numbers corresponding to 0:00 at the respective days) are collected in the starting point set $\mathcal{I}$. This fact is reflected by the feasible slot series $S^{\mathcal{I}}_{(a,d)}$, that represent all arrival/departure pairs allocated to a series request when the initial slot pair $(a, d) \in F$ is chosen. Of course, a single flight request F can be identified with the series request $(F, \{0\})$, and we will sometimes use this fact to simplify notation.

Example 3.2
Let us illustrate the notion of a series request by means of an example. Assume an airline wants to submit the following request:

> "Arrival at 9:00 or 9:10, and a subsequent departure 40–50 minutes later for every Monday, Wednesday and Thursday within the planning horizon of two weeks, starting Monday."

Suppose the slot set discretizes time in steps of ten minutes per slot. The planning horizon is two weeks, starting on a Monday, 0:00 with slot number 1, corresponding to the time interval 0:00–0:09. Then one day is equivalent to 144 slots and $\mathcal{S} = \{1, \dots, 2020\}$.

The arrival times 9:00 and 9:10 on the first day of the planning horizon are equal to slot numbers $1 + 9 \cdot 6 = 55$ and $1 + 9 \cdot 6 + 1 = 56$, respectively, and the departure should take place either 4 or 5 slots after arrival. The single flight request for the first Monday thus translates to

$$F = \{(55, 59); (55, 60); (56, 60); (56, 61)\} .$$

Further, the time 0:00 for the two Mondays within the planning horizon corresponds to the slot numbers 0 and $144 \cdot 7 = 1008$, for the two Wednesdays the starting slot numbers are $2 \cdot 144 = 288$ and $9 \cdot 144 = 1296$, for the two Thursdays we obtain $3 \cdot 144 = 432$ and $10 \cdot 144 = 1440$, respectively. This yields the starting point set

$$\mathcal{I} = \{0, 288, 432, 1008, 1296, 1440\}$$

for the series request $(F, \mathcal{I})$.

Hence if we decided to allocate the initial slot pair $(55, 60)$ for that series request, we would end up with the slot series

$$S^{\mathcal{I}}_{(55,60)} = \{(55, 60); (343, 348); (487, 492); (1063, 1068); (1351, 1356); (1495, 1500)\}$$

corresponding to an arrival at 9:00 and a departure at 9:50 for all of the requested days, effectively allocating six arrival and six departure slots for that series request. $\diamondsuit$

Given a collection of abstract flight and series requests, the task of flight scheduling amounts to choosing one slot pair from each flight request or to mark the request as not scheduled, thus a flight schedule is basically a choice function.

Definition 3.3 (Abstract Flight Schedule)
Let $\mathcal{S}$ be a slot set and $\mathcal{F}$ a finite collection of abstract flight and series requests. An abstract flight schedule for $\mathcal{F}$ is a function

$$f : \mathcal{F} \to (\mathcal{S} \times \mathcal{S})^{\star}$$

with $f(F) \in (F)^{\star}$ for all abstract flight requests $F \in \mathcal{F}$ and $f((F, \mathcal{I})) \in (F)^{\star}$ for all abstract series requests $(F, \mathcal{I}) \in \mathcal{F}$. The image of an abstract series request under f is called *scheduled (initial) slot pair* in the abstract flight schedule f; an abstract flight or series request is termed *integrated*, if its image is not ∞. To simplify notation, we will leave out the double braces for abstract series requests and just write $f(F, \mathcal{I})$ for the scheduled slot pair of an abstract series request $(F, \mathcal{I})$.

The *size* or *cardinality* of a flight schedule f is defined as the number of flight movements that f represents, more precisely

$$|f| := |\{F \in \mathcal{F} : f(F) \neq \infty\}| \quad + \sum_{\substack{(F,\mathcal{I}) \in \mathcal{F} \\ f(F,\mathcal{I}) \neq \infty}} \left| S^{\mathcal{I}}_{f(F,\mathcal{I})} \right| .$$

For both single flight requests and series requests, a flight schedule selects one of the feasible arrival/departure pairs, where for a series request $(F, \mathcal{I})$ the interpretation is to schedule flights for *all* slot pairs in $S^{\mathcal{I}}_{f(F,\mathcal{I})}$, i. e., for the whole series corresponding to the initial slot pair $f(F, \mathcal{I})$. However, given that demand usually exceeds the available capacity of an airport, often not all requested flights can be scheduled. Hence, for both single and series requests, the “allocated slot” can also be ∞, meaning that a request is not scheduled at all. A natural measure for the quality of a flight schedule is the number of flight movements it corresponds to (because airports charge fees for every flight movement), so the length of a scheduled series has to be taken into account in the definition of $|f|$.

3.1.2 Time Window Bounds

The capacity of an airport is usually measured by the amount of flights that can be processed within certain time intervals. In this sense, many of the constraints mentioned above are captured by a construct known as *time window bounds*, providing upper bounds on the number of arrivals and/or departures that can take place within specified time windows. A time window bound can either be applied as a *shifting bound* or as a *consecutive* or *non-shifting bound*, and consequently we will usually refer to time window bounds by one of these two notions.

Definition 3.4 (Time Window Bound, Consecutive Bound, Shifting Bound)
Let $\mathcal{S}$ be a slot set. A *time window bound* $(L, b)^{(\sigma)}$ consists of a *length* $L \in \mathcal{S}$, a *bound value* (more precisely a triple of bound values)

$$b = \left(b^{\mathrm{A}}, b^{\mathrm{D}}, b^{\mathrm{M}}\right)^{T} \in \mathbb{N}_0^3 \quad \text{with} \quad \max\left\{b^{\mathrm{A}}, b^{\mathrm{D}}\right\} \leq b^{\mathrm{M}},$$

and a *shift specification* $\sigma \in \{1, L\}$. The values b^{A}, b^{D} and b^{M} are called *arrival*, *departure* and *movements* bound, respectively. The interval $[L]_\circ = [0,\, L-1]$ will be referred to as the *associated time window*. A time window bound is called *consecutive bound* or *non-shifting bound* if $\sigma = L$, and *shifting bound* if $\sigma = 1$. For a shifting bound $(L, b)^{(1)}$, we will normally just write (L, b) or $([L]_\circ, b)$, using its associated time window in place of L.

Furthermore, a time window bound $(L, b)^{(\sigma)}$ is called *symmetric*, if

$$\frac{b^{\mathrm{M}}}{2} \leq \min\left\{b^{\mathrm{A}}, b^{\mathrm{D}}\right\}.$$

A *time window bound* $(L, b)^{(\sigma)}$ constrains the number of flights to a maximum of b^{A} arrivals, b^{D} departures and b^{M} total flight movements within certain time intervals along the slot set. Specifically, for a *shifting bound each* time window of length L that is contained in $\mathcal{S}$ or at least starts in $\mathcal{S}$ (in which case the time window has to be truncated to its intersection with $\mathcal{S}$) is considered. Think of placing the time window along the planning horizon, aligning its left boundary with the first slot, and then shift the window along the whole time axis, moving one slot at a time. For a *non-shifting bound* $(L, b)^{(L)}$ only the time windows starting at slots $1, L+1, \ldots$ are considered (again, possibly truncated to their intersection with $\mathcal{S}$). So this time, the window is not shifted slot by slot, but instead it is shifted by its own length L, resulting in consecutive placement along the time axis (hence the notation $\sigma = L$).

The requirement $b^{\mathrm{A}}, b^{\mathrm{D}} \leq b^{\mathrm{M}}$ is just a technicality and can be assumed for all time window bounds without loss of generality. As the number of flight movements within a time window is constrained by its movements bound value b^{M}, it would certainly not make sense to allow for a higher number of arrivals and/or departures than b^{M}. The additional requirement $b^{\mathrm{A}}, b^{\mathrm{D}} \geq 1/2 \cdot b^{\mathrm{M}}$ for a symmetric bound reflects the fact that arrivals and departures are (at least in the long run) equal in number, so it should be possible for both arrivals and departures to make up for at least half of the flights within any time window. If this was not the case, arrivals could outweigh departures or vice versa in the long run, or the distribution of arrivals and departures would be forced to be non-symmetric by the shifting bounds alone — of course, local asymmetries are still possible, but in reality these are caused by an unbalanced distribution of demand for arrivals and departures (cf. Chapter 4), and not by asymmetries in the constraint system. Notice this constraint also implies $b^{\mathrm{M}} \leq b^{\mathrm{A}} + b^{\mathrm{D}}$. In practical applications, the time window bounds involved are usually symmetric, but there may also be situations where a non-symmetric bound is used. This could for instance pertain to an airport where many aircrafts stay over night, thus limiting the number of parking positions for arriving flights for the first few hours of the next day, until the overnight flights have departed and cleared their positions. In that situation, a very low value of b^{A} and a rather high value of b^{D} might be applicable during the first hours of traffic. As such situations do generally only apply to very short time periods, we will mostly assume time window bounds to be symmetric, but most of the results also hold for the non-symmetric situation.

Example 3.5
Consider a time window bound of length 30 minutes. Then the time intervals to which the bound values have to be applied are 0:00–0:29, 0:10–0:39, 0:20–0:49 and so on. Discretizing on a ten minute scale the shifting bound (30 minutes, b) translates to $(3, b)$ or $([3]_\circ, b)$, so the relevant time intervals are $(s + [3]_\circ) \cap \mathcal{S}$, $s \in \mathcal{S}$. For the non-shifting bound $([3]_\circ, b)^{(3)}$, the time intervals would be $(s + [3]_\circ) \cap \mathcal{S}$, $s \in \{1, L+1, \dots\} \cap \mathcal{S}$. ♢

In airport operations, the distinction between average capacity and peak capacity of an airport is important. While in the long run certain limits may not be exceeded, some variation is allowed in the short run leading to short-termed peaks. For example, air traffic control may be able to safely handle an average of 80 flight movements per hour, but no more than 15 flights at any given time (where "any given time" usually means "one slot", i. e., within ten minutes). So a higher load is temporarily allowed, provided a certain average load is not exceeded in the long run. Mathematically, this leads to multiple time window bounds (one for the peak capacity with a short length, at least one for the average capacity with a long time window) that are applied simultaneously. Such a system of time window bounds will be referred to as a *reference value system*.

Definition 3.6 (Reference Value System)
A *reference value system* is a nonempty, finite set $\mathcal{R}$ of time window bounds such that no two bounds in $\mathcal{R}$ have the same length. The reference value system $\mathcal{R}$ is called *symmetric*, if all time window bounds in $\mathcal{R}$ are symmetric, and *monotone*, if it contains only shifting or only non-shifting bounds and for any two time window bounds $(L, b)^{(\sigma)}, (L', b')^{(\sigma')} \in \mathcal{R}$ with $L < L'$ the inequalities

$$b < b' \quad \text{and} \quad \frac{b'}{L'} \leq \frac{b}{L} \quad \text{hold.}$$

The monotonicity of a reference value system states that longer time windows correspond to relatively stricter bound values. This condition is very natural, because for a non-monotone reference value system the bounds with longer time windows could simply be dropped if there were short time windows producing stronger bounds (for a suitably large slot set). Of course, for monotonicity one should only compare bounds of the same type. An example of a reference value system applied to a flight schedule will be given in the context of slot configurations below.

3.1.3 Slot Configurations and Feasible Flight Schedules

With the two foregoing subsections, the notion of a feasible flight schedule could simply be defined as a flight schedule observing all time window bounds of a given reference value system. However, it will be convenient to introduce a separate notation here (especially as this will prove valuable later).

Definition 3.7 (Slot Configuration)
Let $\mathcal{S}$ be a slot set. A *slot configuration* is a function

$$C : \mathcal{S} \to \mathbb{N}_0^3, \quad s \mapsto \Big(C^{\mathrm{A}}(s), C^{\mathrm{D}}(s), C^{\mathrm{M}}(s)\Big)$$

with $C^{\mathrm{M}}(s) = C^{\mathrm{A}}(s) + C^{\mathrm{D}}(s)$ for every $s \in \mathcal{S}$. The components $C^{\mathrm{A}}, C^{\mathrm{D}}, C^{\mathrm{M}} : \mathcal{S} \to \mathbb{N}_0$ are called *arrival*, *departure* and *movements configuration*, respectively; the number $|C| := C^{\mathrm{M}}(\mathcal{S})$ is called the *size* of the slot configuration C.

Furthermore, let $\mathcal{F}$ be a collection of abstract flight and series requests and $f : \mathcal{F} \to (\mathcal{S} \times \mathcal{S})^\star$ an abstract flight schedule. Then the *associated slot configuration* $C_f : \mathcal{S} \to \mathbb{N}_0^3$ is defined by

$$\begin{aligned}
C_f^{\mathrm{A}}(s) &:= |\{F \in \mathcal{F} : f(F) \in (\{s\} \times \mathcal{S})\}| + \Big|\Big\{(F,\mathcal{I}) \in \mathcal{F} : S^{\mathcal{I}}_{f(F,\mathcal{I})} \cap (\{s\} \times \mathcal{S}) \neq \emptyset\Big\}\Big|, \\
C_f^{\mathrm{D}}(s) &:= |\{F \in \mathcal{F} : f(F) \in (\mathcal{S} \times \{s\})\}| + \Big|\Big\{(F,\mathcal{I}) \in \mathcal{F} : S^{\mathcal{I}}_{f(F,\mathcal{I})} \cap (\mathcal{S} \times \{s\}) \neq \emptyset\Big\}\Big|, \\
C_f^{\mathrm{D}}(s) &:= C_f^{\mathrm{A}}(s) + C_f^{\mathrm{D}}(s).
\end{aligned}$$

Let us remark that technically a slot configuration C is defined by C^{A} and C^{D} alone, C^{M} is just a convenient shorthand notation for the sum $C^{\mathrm{A}} + C^{\mathrm{D}}$.

For the following definition recall that

$$C(s + [L]_\circ) \quad = \sum_{t \in (s+[L]_\circ) \cap \mathcal{S}} C(t), \qquad \text{see Section 1.4.2.}$$

Definition 3.8 (Feasible Slot Configuration, Feasible Flight Schedule)
Let $\mathcal{S}$ be a slot set and $\mathcal{R}$ a reference value system. A slot configuration $C : \mathcal{S} \to \mathbb{N}_0^3$ is *feasible with respect to* $\mathcal{R}$, if

$$\begin{aligned}
& C(s + [L]_\circ) \leq b, \\
\text{i. e.,} \quad & C^{\mathrm{A}}(s + [L]_\circ) \leq b^{\mathrm{A}}, \\
& C^{\mathrm{D}}(s + [L]_\circ) \leq b^{\mathrm{D}}, \\
\text{and} \quad & C^{\mathrm{M}}(s + [L]_\circ) \leq b^{\mathrm{M}}
\end{aligned}$$

for all shifting bounds $(L, b) \in \mathcal{R}$ and all $s \in \mathcal{S}$ such that $(s + [L]_\circ) \cap \mathcal{S} \neq \emptyset$, and for all non-shifting bounds $(L, b)^{(L)} \in \mathcal{R}$ and all $s \in \{1, L+1, \dots\}$ such that $(s + [L]_\circ) \cap \mathcal{S} \neq \emptyset$. If even

$$C(\llbracket s + [L]_\circ \rrbracket_{\mathcal{S}}) \leq b$$

holds, then C is called *circular feasible with respect to* $\mathcal{R}$. Furthermore, let $\mathcal{F}$ be a collection of abstract flight and series requests on $\mathcal{S}$ and let $f : \mathcal{F} \to (\mathcal{S} \times \mathcal{S})^\star$ be an abstract flight schedule. Then f is *(circular) feasible with respect to* $\mathcal{R}$, if its associated slot configuration C_f is (circular) feasible.

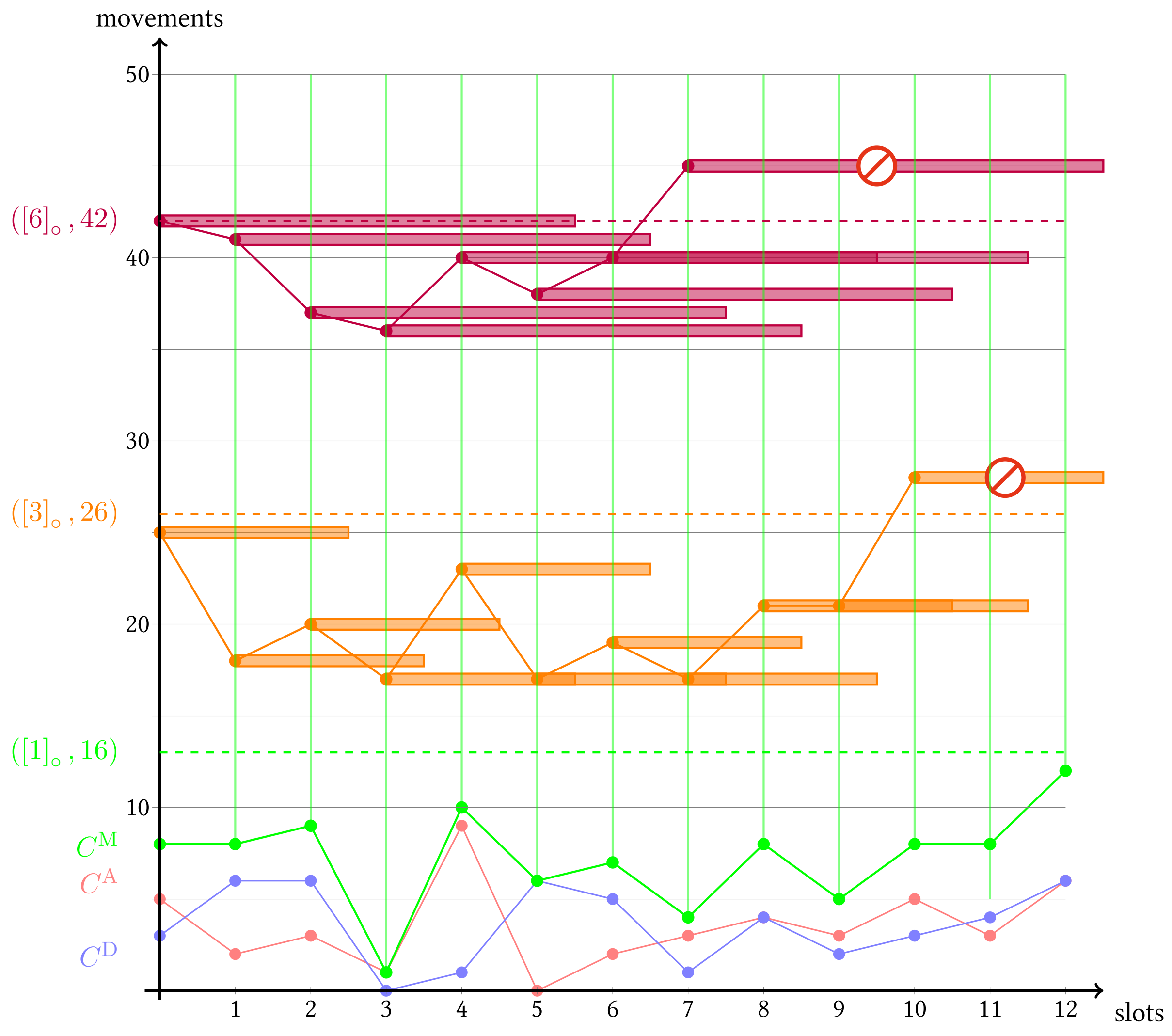


Figure 3.1: An example of a reference value system with shifting bounds "in action". The reference value system is $\mathcal{R} = \{([1]_\circ, 16), ([3]_\circ, 26), ([6]_\circ, 42)\}$. The values for the corresponding time windows (only movements values are shown) for the slot configuration $C = (C^\mathrm{A}, C^\mathrm{D}, C^\mathrm{M})$ are indicated on the axis of ordinates; the lengths of the time windows are visualized by suitably colored rectangles. Infeasibilities are marked by "$\oslash$".

Example 3.9
We illustrate the concept of a feasible slot configuration and a reference value system by means of an example: Consider a time period of two hours, discretized in units of ten minutes, and the reference value system $\mathcal{R} = \{([1]_\circ, 16), ([3]_\circ, 26), ([6]_\circ, 42)\}$. In Figure 3.1 the numbers of arrivals, departures and movements for every slot are indicated by the red, blue and green curves, respectively. Above, the number of movements for each thirty-minute time window is indicated by a point at the start of each of the time windows; the whole associated time window is sketched as a small colored box, whose ordinate value represents the number of flight movements within the respective time window. The same is done for sixty-minute intervals. Of course, similar graphs could be drawn for arrival and departure values for thirty and sixty minute time windows, but we chose to omit those for the sake of clarity. The respective bound values are drawn as dashed lines. Note that the slot configuration shown here is not feasible for the given reference value system, as both the last sixty-minute and the last thirty-minute time windows violate the respective bounds (indicated by a “forbidden sign”). $\diamondsuit$

Although a slot configuration does not tell us everything about a flight schedule, it nevertheless provides valuable information on the “pattern” generated by a flight schedule and thus allows us to analyze how a good flight schedule should “look like”, i. e., what structural properties are beneficial for obtaining a feasible flight schedule of maximum size. For a structural analysis of optimal flight scheduling it may be desirable to eliminate boundary effects from the model, which are due to the fact that slots at the beginning and at the end of the scheduling period are covered by less reference value time windows than those in the middle of the scheduling period. This is possible by considering the circular versions of the problems involved, where the shifting bounds “wrap around” both ends of the scheduling horizon. Geometrically speaking, we can picture the slots to be arranged along a circle rather than along a line, so the reference value time windows simply reach over from the last to the first slot in the slot set $\mathcal{S}$.

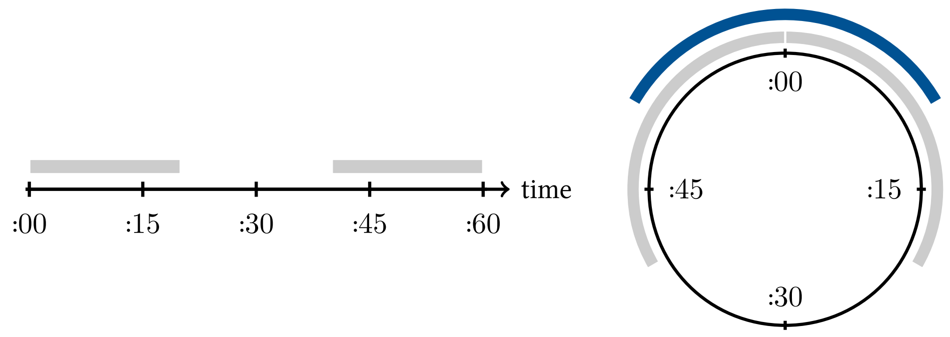


Figure 3.2: Normal view of the slot set $\mathcal{S}$ versus circular view for a planning horizon of one hour; some shifting bounds with a length of 20 minutes are indicated. The blue shifting bound is only present when considering circular shifting bounds.

For a non-shifting bound, we will not consider circular problems. The reason for this is that for a bound that does not naturally end at slot n, "wrapping around" the bound would completely break symmetry. As an example, consider a non-shifting bound of length 4 and $n = 9$. If we want to apply this bound in a circular way, we would have to consider the time windows $\{1, \dots, 4\}$, $\{5, \dots, 8\}$ and $\{9, \dots, 3\}$. But why stop here? In fact, for a shifting bound repetition would occur at this point — not so for a non-shifting bound. If we continue (and there is no good reason not to do so), we would have to add the time windows $\{4, \dots, 7\}$, $\{8, \dots, 2\}$, $\{3, \dots, 6\}$, $\{7, \dots, 1\}$, $\{2, \dots, 5\}$ and $\{6, \dots, 9\}$, in effect ending up with the same intervals as for a shifting bound of length 4. This is certainly not always the case (depending on the least common divisor of the time window's length and n). However, the only case that is relevant for applications is the one where n is a multiple of L for all non-shifting bounds $(L, b)^{(L)}$, and we will consider such cases where necessary.

3.1.4 Notation

In the following sections we will frequently use matrix-vector notation to provide for a concise representation of our problems. Let us introduce the necessary terminology beforehand.

Let $n \in \mathbb{N}$ and let $\mathcal{S} = \{1, \dots, n\}$ be a slot set. For a slot configuration $C : \mathcal{S} \to \mathbb{N}_0^3$ we identify the components $C^{\mathrm{A}}, C^{\mathrm{D}}, C^{\mathrm{M}} : \mathcal{S} \to \mathbb{N}_0$ with the vectors

$$\begin{aligned} C^{\mathrm{A}} &= \Big(C^{\mathrm{A}}(1), \dots, C^{\mathrm{A}}(n)\Big) \in \mathbb{N}_0^n, \\ C^{\mathrm{D}} &= \Big(C^{\mathrm{D}}(1), \dots, C^{\mathrm{D}}(n)\Big) \in \mathbb{N}_0^n, \\ C^{\mathrm{M}} &= \Big(C^{\mathrm{M}}(1), \dots, C^{\mathrm{M}}(n)\Big) \in \mathbb{N}_0^n. \end{aligned}$$

For a shifting bound (L, b), define the *incidence matrix* R_L and the bound vectors b_L^{A}, b_L^{D} and b_L^{M} by

$$R_L := \begin{pmatrix} \mathbb{1}^T_{1+[L]_\circ} \\ \mathbb{1}^T_{2+[L]_\circ} \\ \vdots \\ \mathbb{1}^T_{n-(L-1)+[L]_\circ} \end{pmatrix} \in \{0,1\}^{m(L)\times n} \quad \text{and} \quad \begin{aligned} b_L^{\mathrm{A}} &:= b^{\mathrm{A}} \cdot \mathbb{1}_{m(L)} \\ b_L^{\mathrm{D}} &:= b^{\mathrm{D}} \cdot \mathbb{1}_{m(L)} \\ b_L^{\mathrm{M}} &:= b^{\mathrm{M}} \cdot \mathbb{1}_{m(L)}, \end{aligned}$$

where $m(L) = n - (L - 1)$. For a non-shifting bound $(L, b)^{(L)}$, we analogously define

$$R'_L = \begin{pmatrix} \mathbb{1}^T_{1+[L]_\circ} \\ \mathbb{1}^T_{(L+1)+[L]_\circ} \\ \vdots \\ \mathbb{1}^T_{((m'(L)-1)L+1)+[L]_\circ} \\ \mathbb{1}^T_{[(m'(L)L+1),\, n]} \end{pmatrix} \in \{0,1\}^{m'(L)\times n} \quad \text{and} \quad \begin{aligned} (b')_L^{\mathrm{A}} &:= b^{\mathrm{A}} \cdot \mathbb{1}_{m'(L)} \\ (b')_L^{\mathrm{D}} &:= b^{\mathrm{D}} \cdot \mathbb{1}_{m'(L)} \\ (b')_L^{\mathrm{M}} &:= b^{\mathrm{M}} \cdot \mathbb{1}_{m'(L)}, \end{aligned}$$

where $m'(L) = \lfloor n/L \rfloor$.

For a reference value system $\mathcal{R} = \Big\{(L_1, b_1), \dots, (L_k, b_k), (L_{k+1}, b_{k+1})^{(L)}, \dots, (L_{k'}, b_{k'})^{(L)}\Big\}$, we define the *incidence matrix* R of $\mathcal{R}$ by

$$R = \begin{pmatrix} R_{L_1} \\ \vdots \\ R_{L_k} \\ R'_{L_{k+1}} \\ \vdots \\ R'_{L'_k} \end{pmatrix} \in \{0,1\}^{m(\mathcal{R}) \times n}$$

and

$$\begin{aligned} b^{\mathrm{A}} &:= \Big(b_1^{\mathrm{A}} \mathbb{1}^T_{m(L_1)}, \dots, b_k^{\mathrm{A}} \mathbb{1}^T_{m(L_k)}, b_{k+1}^{\mathrm{A}} \mathbb{1}^T_{m'(L_{k+1})}, \dots, b_{k'}^{\mathrm{A}} \mathbb{1}^T_{m(L_{k'})}\Big)^T, \\ b^{\mathrm{D}} &:= \Big(b_1^{\mathrm{D}} \mathbb{1}^T_{m(L_1)}, \dots, b_k^{\mathrm{D}} \mathbb{1}^T_{m(L_k)}, b_{k+1}^{\mathrm{D}} \mathbb{1}^T_{m'(L_{k+1})}, \dots, b_{k'}^{\mathrm{D}} \mathbb{1}^T_{m(L_{k'})}\Big)^T, \\ b^{\mathrm{M}} &:= \Big(b_1^{\mathrm{M}} \mathbb{1}^T_{m(L_1)}, \dots, b_k^{\mathrm{M}} \mathbb{1}^T_{m(L_k)}, b_{k+1}^{\mathrm{M}} \mathbb{1}^T_{m'(L_{k+1})}, \dots, b_{k'}^{\mathrm{M}} \mathbb{1}^T_{m'(L_{k'})}\Big)^T, \end{aligned}$$

where $m(\mathcal{R}) = m(L_1) + \cdots + m(L_k) + m'(L_{k+1}) + \cdots + m'(L_{k'})$. Thus, a slot configuration $(C^{\mathrm{A}}, C^{\mathrm{D}}, C^{\mathrm{M}}) \in \mathbb{N}_0^{3n}$ is feasible for $\mathcal{R}$ if and only if

$$RC^{\mathrm{A}} \leq b^{\mathrm{A}}, \quad RC^{\mathrm{D}} \leq b^{\mathrm{D}} \quad \text{and} \quad RC^{\mathrm{M}} = R(C^{\mathrm{A}} + C^{\mathrm{D}}) \leq b^{\mathrm{M}}.$$

Similarly, we define the *circular incidence matrix* $\mathring{R}_L$ of a shifting bound (L, b) as

$$\mathring{R}_L := \begin{pmatrix} \mathbb{1}^T_{[\![1+[L]_\circ]\!]_\mathcal{S}} \\ \mathbb{1}^T_{[\![2+[L]_\circ]\!]_\mathcal{S}} \\ \vdots \\ \mathbb{1}^T_{[\![n+[L]_\circ]\!]_\mathcal{S}} \end{pmatrix} \in \{0,1\}^{n \times n} \quad \text{and} \quad \begin{aligned} \mathring{b}_L^{\mathrm{A}} &:= b^{\mathrm{A}} \cdot \mathbb{1}_n \\ \mathring{b}_L^{\mathrm{D}} &:= b^{\mathrm{D}} \cdot \mathbb{1}_n \\ \mathring{b}_L^{\mathrm{M}} &:= b^{\mathrm{M}} \cdot \mathbb{1}_n. \end{aligned}$$

Finally, for a reference value system $\mathcal{R}$ that consists of only shifting bounds we define the incidence matrix $\mathring{R}$ of $\mathcal{R} = \{(L_1, b_1), \dots, (L_k, b_k)\}$ by

$$\mathring{R} = \begin{pmatrix} \mathring{R}_{L_1} \\ \mathring{R}_{L_2} \\ \vdots \\ \mathring{R}_{L_k} \end{pmatrix} \in \{0,1\}^{kn \times n}.$$

The bound vectors $\mathring{b}^{\mathrm{A}}, \mathring{b}^{\mathrm{D}}, \mathring{b}^{\mathrm{M}} \in \mathbb{N}_0^{kn}$ are defined accordingly as

$$\begin{aligned}\mathring{b}^{\mathrm{A}} &= \left(b_1^{\mathrm{A}}\mathbb{1}_n^T, \ldots, b_k^{\mathrm{A}}\mathbb{1}_n^T\right)^T,\\ \mathring{b}^{\mathrm{D}} &= \left(b_1^{\mathrm{D}}\mathbb{1}_n^T, \ldots, b_k^{\mathrm{D}}\mathbb{1}_n^T\right)^T,\\ \mathring{b}^{\mathrm{M}} &= \left(b_1^{\mathrm{M}}\mathbb{1}_n^T, \ldots, b_k^{\mathrm{M}}\mathbb{1}_n^T\right)^T.\end{aligned}$$

3.1.5 Graphics

In this chapter and the next we will frequently use graphics to visualize examples or test results. Most of these graphics will visualize slot configurations like the ones displayed in Figure 3.3.

Figure 3.3a shows a movements slot configuration (i. e., arrivals and departures are not distinguished, only flight movements are shown). The slots are depicted as stacks of blue rectangles, flight movements that are scheduled for one of the slots are indicated by the corresponding number of orange rectangles. If high numbers of movements are scheduled, this may also be indicated by writing the number of movements in the corresponding slot rectangle. Below the slots some of the time windows are shown as gray rectangles with the number of flight movements taking place within the respective time window written inside them. If a time window is at its movements limit, an orange frame is drawn around the rectangle. Here, nine slots are shown, where slots one and six contain three flight movements each.

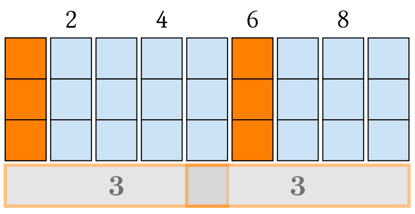
Plots like Figure 3.3b are used to display larger test results. The slots are plotted in the form of a "time line" along the x-axis, the number of arrivals, departures and flight movements is shown as a function plot. The values for all time window bounds are displayed in the same way, where the y-coordinate refers to the number of movements within the time window that starts at the respective x-coordinate. In this example, movements bounds for ten and thirty minutes are shown.

3.2 Maximum Flight Scheduling

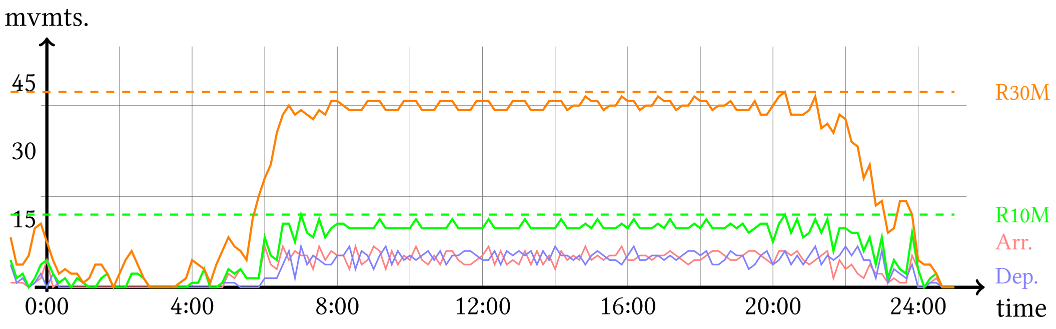
We can now formally state the Maximum Flight Scheduling problem.

Problem 3.10: Maximum Flight Scheduling
Instance: The slot count $n \in \mathbb{N}$, a collection $\mathcal{F}$ of abstract flight and series requests on $\mathcal{S} = \{1, \ldots, n\}$ and a reference value system $\mathcal{R}$.
Question: Find an abstract flight schedule for $\mathcal{F}$ that is feasible with respect to $\mathcal{R}$ and has maximum size (i. e., integrates a maximum number of flight movements).

This problem formalizes the main task of airport flight scheduling. Primarily, one is certainly interested in finding a flight schedule that accommodates as many flights as possible so that the airport and its facilities can be used to full capacity. Unfortunately, Maximum Flight Scheduling will turn out to be an algorithmically hard problem. In the following, we will present a complexity result supporting this claim. In contrast to this, we will analyze some exemplary special cases



(a) a small slot configuration



(b) visualization of test results

Figure 3.3: Graphics used in Chapters 3 and 4.

where the problem becomes algorithmically tractable. In addition, an integer programming model for Flight Scheduling will be devised that serves as the basis of our more complex model in Chapter 4.

3.2.1 Hardness of Maximum Flight Scheduling

Optimization problems are usually given in the form of a function problem (like Problem 3.10 above), calling for an output that specifies an optimal solution. In contrast to this, complexity results are generally stated for decision problems, which only demand “Yes” or “No” for an answer. However, for most function problems a corresponding decision version can easily be formulated; for Flight Scheduling, the decision version is this:

Problem 3.11: Flight Scheduling, Decision Version
Instance: The slot count $n \in \mathbb{N}$, a collection $\mathcal{F}$ of abstract flight and series requests on $\mathcal{S} = \{1, \ldots, n\}$, a reference value system $\mathcal{R}$ and a number $N \in \mathbb{N}$.
Question: Decide whether there is an abstract flight schedule for $\mathcal{F}$ that is feasible with respect to $\mathcal{R}$ and has size at least N.

For many optimization problems, the corresponding decision version is equivalent to the optimization problem (which asks for the objective value of an optimal solution) in terms of algorithmic tractability: If the optimization problem can be solved in polynomial time, one can of course decide upon the existence of a solution with a certain objective value, provided the value of the objective function can be computed in polynomial time. For the reverse direction, a binary search technique can be employed to solve the optimization problem using repeated calls to the decision problem. For details, we refer the reader to [GLS93] or [PS98]. For this reason, by referring to an optimization problem as $\mathcal{NP}$-hard, one usually means that the corresponding decision problem is $\mathcal{NP}$-hard.

Theorem 3.12
*Problem 3.11 (*Flight Scheduling, Decision Version*) is $\mathcal{NP}$-hard.*

We prove the theorem by giving a reduction to Flight Scheduling from 3D-Matching, which is known to be an $\mathcal{NP}$-hard optimization problem (cf. [GJ79]).

Problem 3.13: 3D-Matching
Instance: Three finite, disjoint sets X, Y, Z of equal cardinality, i. e., $|X| = |Y| = |Z|$, a set $M \subset X \times Y \times Z$ and an integer $N \in \mathbb{N}$.
Question: Decide whether there is a three dimensional matching of cardinality at least N in M, i. e., a subset $M' \subset M$ such no two triples in M' agree in any coordinate and such that $|M'| \geq N$.

Proof (of Theorem 3.12). Let X, Y, Z, an integer N and a set $M \subset X \times Y \times Z$ be an instance of 3D-Matching; we assume that $|X| = |Y| = |Z| = q$ and that $X = \{1, \ldots, q\}$, $Y = \{q+1, \ldots, 2q\}$ and $Z = \{2q+1, \ldots, 3q\}$. Taking $\mathcal{S} := \{1, \ldots, 2q\}$ as the slot set, define

$$\begin{aligned} F_z &:= \{(a,d) \subset \mathcal{S} \times \mathcal{S} : (a,d,z) \in M\}\,, \\ \mathcal{F} &:= \bigcup_{z \in Z} \{F_z\}\,, \\ \mathcal{R} &:= \{([1]_\circ\,;\,(1,1,1))\}\,; \end{aligned}$$

the above sets can be computed in time polynomial in the size of the input to 3D-Matching. We take $\mathcal{F}$ as the set of abstract flight requests and $\mathcal{R}$ as reference value system, i. e., we interpret a triple $(x,y,z) \in M$ as part of a flight request for the slot pair (x,y), and we take different values of z to denote different flight requests.

Now let $f : \mathcal{F} \to (\mathcal{S} \times \mathcal{S})^\star$ be a feasible abstract flight schedule for $\mathcal{R}$, then we can define a set

$$M'_f := \bigcup_{z \in Z : f(F_z) \neq \infty} \left\{ (f^{\mathrm{A}}(F_z), f^{\mathrm{D}}(F_z), z) \right\}.$$

Due to the restrictions imposed by $\mathcal{R}$, the set M'_f is a three dimensional matching in M of cardinality $|f|$. On the other hand, if $M' \subset M$ is a three dimensional matching in M, we can define a flight schedule $f_{M'} : \mathcal{F} \to (\mathcal{S} \times \mathcal{S})^\star$ that is feasible with respect to $\mathcal{R}$ by

$$f_{M'}(F_z) = \begin{cases} (x,y)\,, & \text{if there are } x \in X \text{ and } y \in Y \text{ such that } (x,y,z) \in M'; \\ \infty, & \text{else.} \end{cases}$$

Note that if there is some triple in M' with last coordinate z, then it is uniquely defined by the matching property of M', thus $f_{M'}$ is indeed well-defined. Also, $|M'| = |f_{M'}|$.

This shows that there is a solution of any given size $N \in \mathbb{N}$ to an instance of 3D-Matching if and only if the associated instance of Flight Scheduling has a solution of size N. Hence 3D-Matching reduces to Flight Scheduling, completing the proof of $\mathcal{NP}$-hardness. □

3.2.2 Integer Programming Formulation

To gain some insight into the inherent problem structure of Flight Scheduling, let us have a look at the concise representation provided by an integer programming formulation. The integer program consists of two "building blocks": One set of constraints that captures the assignment aspect, and a second set of constraints for the feasibility with respect to some reference value system $\mathcal{R}$. If $\mathcal{R}$ is given by its incidence matrix R on the slot set $\mathcal{S} = \{1, \dots, n\}$, we get the following integer program.

Problem 3.14: Flight Scheduling, IP Formulation

Instance: The number of slots $n \in \mathbb{N}$, a collection $\mathcal{F} = \{(F_1, \mathcal{I}_1), \dots, (F_k, \mathcal{I}_k)\}$ of abstract flight and series requests on $\mathcal{S} = \{1, \dots, n\}$ and the incidence matrix R of a reference value system.

Question: Find an optimal solution $x \in \{0,1\}^{|\mathcal{F}|\cdot n\cdot n}$, $C^{\mathrm{A}}, C^{\mathrm{D}}, C^{\mathrm{M}} \in \mathbb{N}_0^n$ for the following integer linear program:

$$\begin{aligned}
\max\quad & \mathbb{1}_n^T C^{\mathrm{M}} && && (3.1)\\
\text{s.t.}\quad & \sum_{a,d\in\mathcal{S}} x_{i,a,d} \le |\mathcal{I}_i| && \text{for all } (F_i,\mathcal{I}_i)\in\mathcal{F} && (3.2)\\
& \sum_{(F_i,\mathcal{I}_i)\in\mathcal{F}} \sum_{d\in\mathcal{S}} x_{i,a,d} = C^{\mathrm{A}}_a && \text{for all } a\in\mathcal{S} && (3.3)\\
& \sum_{(F_i,\mathcal{I}_i)\in\mathcal{F}} \sum_{a\in\mathcal{S}} x_{i,a,d} = C^{\mathrm{D}}_d && \text{for all } d\in\mathcal{S} && (3.4)\\
& x_{i,a,d} = x_{i,a+\Delta,d+\Delta} && \text{for all } \Delta\in\mathcal{I}_i \text{ s.t. } a+\Delta, d+\Delta\in\mathcal{S} && (3.5)\\
& RC^{\mathrm{A}} \le b^{\mathrm{A}} && && (3.6)\\
& RC^{\mathrm{D}} \le b^{\mathrm{D}} && && (3.7)\\
& RC^{\mathrm{M}} \le b^{\mathrm{M}} && && (3.8)\\
& C^{\mathrm{A}} + C^{\mathrm{D}} = C^{\mathrm{M}} && && (3.9)\\
& x_{i,a,d} = 0 && \text{for all } (F_i,\mathcal{I}_i)\in\mathcal{F} && (3.10)\\
& && \text{and all } (a,d)\in(\mathcal{S}\times\mathcal{S})\setminus \bigcup_{(s,t)\in F_i} S^{\mathcal{I}_i}_{(s,t)} && \\
& C^{\mathrm{A}}, C^{\mathrm{D}}, C^{\mathrm{M}} \in \mathbb{N}_0^n && && \\
& x\in\{0,1\}^{|\mathcal{F}|\cdot n\cdot n} && &&
\end{aligned}$$

Let us first remark that a feasible solution is always given by setting all variables to 0, so feasibility is not an issue. The meaning of the decision variables $x_{i,a,d}$ is

$$x_{i,a,d} = \begin{cases} 1, & \text{if series request } F_i \text{ is allocated arrival slot } a \text{ and departure slot } d; \\ 0, & \text{else,} \end{cases}$$

while C^{A}, C^{D} and C^{M} represent the slot configuration associated to the flight schedule that is determined by the values of x. The constraints (3.3) and (3.4) link the variables C^{A}, C^{D}, C^{M} to x (the sum simply counts the numbers of arrivals and departures for each slot, respectively), (3.9) conditions $\left(C^{\mathrm{A}}, C^{\mathrm{D}}, C^{\mathrm{M}}\right)$ to be a slot configuration. In (3.2), we see a variation on the classical assignment constraint ensuring that each series request $(F_i, \mathcal{I}_i)$ gets allocated at most one initial slot pair: Here, we need to allocate either at most $|\mathcal{I}_i|$ slot pairs to flight request F_i (usually exactly $|\mathcal{I}_i|$ or no slot pair at all, but if the slot set is "too short", it may be impossible to operate all flights). The series constraint (allocate slots for the same times relative to the starting points in $\mathcal{I}_i$ is expressed by (3.5), the time window bounds can be found in (3.6)–(3.8). In constraint (3.10), x is restricted to admissible arrival/departure pairs. Finally, the objective (3.1) is to maximize the number of flight movements.

We have only formulated the integer program for series requests, flight requests can simply be modeled by using the starting point set $\mathcal{I} = \{0\}$. Single slot requests may also be integrated into the model, for instance by introducing an additional binary variable $y_{i,s}$ for every slot request G_i and every slot $s \in \mathcal{S}$, which is 1 if and only if request i is allocated the slot s and inserting the variable in the appropriate inequalities, depending upon whether the request is for an arrival or for a departure slot.

Let us mention that, if $\mathcal{R}$ contains only shifting bounds, by replacing R by $\mathring{R}$ we obtain an integer programming formulation for flight scheduling subject to circular shifting bounds.

The value of the above integer program is apparently not due to a straightforward algorithmic applicability, but rather to clarify the interdependencies between the various optimization problems considered in this chapter. One can clearly see that flight scheduling can be considered as a composition of two different problems, namely a standard assignment problem and the balancing constraints imposed by the time window bounds. From this point of view, the connection to packing subject to balancing constraints becomes evident: Assignment is simply a very basic form of a standard packing problem, while the time window bounds constitute a balancing requirement. The time window bounds locally (namely within their respective time windows) set limits on the number of flights, thus requiring a certain distribution over the whole planning horizon. On the other hand, the "shifting component" of the shifting bounds propagates the effect of a flight assignment beyond a single slot. This way, a (albeit restricted) "globalization" of local assignments is obtained, which one should expect to lead to a more uniform distribution of flights, thus "balancing out" the flight schedule.

3.2.3 Matroid Representations of Flight Scheduling Problems

The "decomposition aspect" of Flight Scheduling into an assignment and a balancing part (constituted by the time window bounds) evident in Section 3.2.2 suggests an interpretation of the problem in terms of independence systems and matroids. In this subsection, we give an outline of these aspects to derive some complexity results for specialized instances of Flight Scheduling problems.

Lemma 3.15
For $n \in \mathbb{N}$, let $\mathcal{S} := \{1, \dots, n\}$ be a slot set and $\mathcal{G} = \{G_1, \dots, G_m\}$ a collection of slot requests, and let $(L, b)^{(\sigma)}$ be a time window bound with $b \geq \mathbb{1}_3$. Furthermore, define the set $\mathcal{B}_\mathcal{G}$ of possible slot assignments by

$$\mathcal{B}_\mathcal{G} := \bigcup_{i=1,\dots,m} (\{i\} \times G_i) .$$

1. *With $G'_i := \{i\} \times G_i$ for $i \in \{1, \dots, m\}$, the set*

   $$\mathcal{M}_\mathcal{G} := \{B \subset \mathcal{B}_\mathcal{G} : |G'_i \cap B| \leq 1 \text{ for all } i \in \{1, \dots, m\}\}$$

   is a matroid over the ground set $B_\mathcal{G}$.

2. *Let $S \subset \mathcal{S}$ be a nonempty subset of the slot set. The set of slot requests for the slot $t \in \mathcal{S}$ is defined as $H_t := \bigcup_{i=1 \dots, m} \{(i, t) : t \in G_i\}$ and the sets $H'_s := \bigcup_{t \in s + [L]_\circ} H_t$ collect all slot requests within a time window of length L starting at slot $s \in S$ (where S can be chosen suitably to accommodate for either a shifting or a non-shifting bound). Define*

   $$\begin{aligned}\mathcal{M}^{\mathrm{A}}_{(L,b)} &:= \left\{B \subset \mathcal{B}_\mathcal{G} : |H'_s \cap B| \leq b^{\mathrm{A}} \text{ for all } s \in S\right\} ,\\ \mathcal{M}^{\mathrm{D}}_{(L,b)} &:= \left\{B \subset \mathcal{B}_\mathcal{G} : |H'_s \cap B| \leq b^{\mathrm{D}} \text{ for all } s \in S\right\} ,\\ \mathcal{M}^{\mathrm{M}}_{(L,b)} &:= \left\{B \subset \mathcal{B}_\mathcal{G} : |H'_s \cap B| \leq b^{\mathrm{M}} \text{ for all } s \in S\right\} .\end{aligned}$$

   If $\{H'_s : s \in S\}$ is a collection of disjoint sets, then $\mathcal{M}^{\mathrm{A}}_{(L,b)}$, $\mathcal{M}^{\mathrm{D}}_{(L,b)}$ and $\mathcal{M}^{\mathrm{M}}_{(L,b)}$ are matroids over the ground set $\mathcal{B}_\mathcal{G}$.

Proof. The sets $G'_1, \dots, G'_m$ obviously constitute a partition of $\mathcal{B}_\mathcal{G}$ into disjoint subsets, thus $\mathcal{M}_\mathcal{G}$ is a partition matroid, cf. Theorem 1.4.

If $\{H'_s : s \in S\}$ is a collection of disjoint sets, then it can be turned into a partition of the ground set $\mathcal{B}_\mathcal{G}$ by complementing the sets H'_s with $H' := \mathcal{B}_\mathcal{G} \setminus \bigcup_{s \in S} H'_s$. Hence $\mathcal{M}^{\mathrm{A}}_{(L,b)}$, $\mathcal{M}^{\mathrm{D}}_{(L,b)}$ and $\mathcal{M}^{\mathrm{M}}_{(L,b)}$ are also partition matroids according to Theorem 1.4. □

The consequences for Flight Scheduling are now straightforward. While $\mathcal{M}_\mathcal{G}$ models the fact that each slot request G_i may be assigned at most one slot in G_i, the matroids $\mathcal{M}^{\mathrm{A}}_{(L,b)}$, $\mathcal{M}^{\mathrm{D}}_{(L,b)}$

and $\mathcal{M}^{\mathrm{M}}_{(L,b)}$ model time window bounds for arrivals, departures and movements, respectively. The condition for $\{H'_s : s \in S\}$ to be a collection of disjoint sets corresponds to the non-shifting case, i. e., $S = \{1, L+1, \ldots, \lfloor \frac{n}{L} \rfloor L + 1\}$ (for shifting bounds, we would have to take $S = \{1, \ldots, n - L + 1\}$, thus the sets H'_s would overlap for $L > 1$).

Corollary 3.16
Let $(n, \mathcal{G}, \mathcal{R})$ be an instance of Flight Scheduling *with the following restrictions:*

1. *$\mathcal{G}$ consists exclusively of slot requests;*
2. *the reference value system $\mathcal{R} = \left\{(L,b)^{(L)}\right\}$ consists of a single non-shifting bound, where two out of $\left\{b^{\mathrm{A}}, b^{\mathrm{D}}, b^{\mathrm{M}}\right\}$ have values greater than or equal to $|\mathcal{G}|$ (i. e., only one of the three bound values is actually relevant).*

Then the instance is solvable in polynomial time.

Proof. The feasible set of the above instance of Flight Scheduling can be represented as the intersection of the matroid $\mathcal{M}_{\mathcal{G}}$ with one of the independence systems $\mathcal{M}^{\mathrm{A}}_{(L,b)}$, $\mathcal{M}^{\mathrm{D}}_{(L,b)}$ or $\mathcal{M}^{\mathrm{M}}_{(L,b)}$, namely the one corresponding to the smallest value among b^{A}, b^{D} and b^{M}. As remarked above, the non-shifting bound $(L,b)^{(L)}$ corresponds to setting $S = \{1, L+1, \ldots, \lfloor \frac{n}{L} \rfloor L + 1\}$, so $\{H'_s : s \in S\}$ is indeed a collection of disjoint sets, meaning that $\mathcal{M}^{\mathrm{A}}_{(L,b)}$, $\mathcal{M}^{\mathrm{D}}_{(L,b)}$ and $\mathcal{M}^{\mathrm{M}}_{(L,b)}$ are all matroids. The membership of an arbitrary subset of $\mathcal{B}_{\mathcal{G}}$ in any of these matroids can be tested in polynomial time, thus by Theorem 1.6 the instance is solvable in polynomial time using the two matroid intersection algorithm. □

The proof of $\mathcal{NP}$-hardness for Flight Scheduling in Section 3.2.1 basically uses only one non-shifting bound on the number of flight movements (technically, arrival and departure bounds are also specified, but they have no effect). From the matroid representation, we conclude that this setting is easily solvable as long as we use only slot requests consisting of just a set of movement requests (instead of a set of slot *pairs*). Apparently, adding an arrival-departure coupling in the form of flight requests (and thus *slot pair* requests) makes the problem $\mathcal{NP}$-hard, even if only non-shifting bounds on the number of movements are applied. This clearly reflects the situation for matching problems: As is well known, classical Maximum Matching can be solved in polynomial time (e. g., by modeling it as two matroid intersection), while 3D-Matching is an $\mathcal{NP}$-hard problem.

3.3 Maximum Slot Packing

In Section 3.2.2 we have seen that Flight Scheduling can be regarded as a composition of an assignment problem with a packing problem under balancing constraints. In this section, we will focus on the “packing part” of the problem, more precisely on slot configurations that can arise from a flight schedule of maximum size. On the one hand, considering only the slot configurations provides upper bounds on the number of flights that may be scheduled and allows to evaluate an actual flight schedule by comparison. On the other hand, we will derive some results on the structure of optimal slot configurations that can then serve as guidelines in the search for optimal schedules. Additionally, by understanding how the structure and size of a maximum slot configuration depend upon the reference value system, one can gain insight into the mechanisms supporting or obstructing good flight schedules. These insights are of great advantage in practice once a change in the time window bounds (their lengths, values and shifting behavior or adding or deleting one or more time window bounds in the reference value system) must be evaluated.

Problem 3.17: Maximum (Circular) Slot Packing
Instance: A slot count $n \in \mathbb{N}$ and a reference value system $\mathcal{R}$.
Question: Find a slot configuration C on $\mathcal{S} = \{1, \ldots, n\}$ that is (circular) feasible with respect to $\mathcal{R}$ and has maximum size (i. e., maximizes the number of flight movements $C^{\mathrm{M}}(\mathcal{S})$).

Notice that the formal problem description does contain a somewhat malicious caveat: The problem is formulated as a function problem asking for an output in the form of a vector (in contrast to a decision problem, which only demands “Yes” or “No” as an answer, or an optimization problem, which asks for the objective value of an optimal solution, but not necessarily for the solution itself). This vector consists of $3n$ integers (where n of these — namely the movements values C^{M} — could be computed on the fly, reducing the size to $2n$ integers), thus $\Omega(n)$ bits are needed to encode the output on a binary Turing machine. In contrast to that, the input consists of the number n and $4\,|\mathcal{R}|$ numbers which encode the time window bounds (one length and three bound values for each time window bound), plus $|\mathcal{R}|$ indicator bits (denoting shifting or non-shifting bound), hence the input may be encoded in $\mathcal{O}\left(\log n + |\mathcal{R}|\right)$ bits. In order to classify complexity of a function problem (which would usually be given in a form calling for some elaborated output beyond “Yes” or “No”), one generally resorts to a “natural” decision version of the problem (as we have done in Section 3.2.1):

Problem 3.18: Maximum Slot Packing, Decision Version
Instance: A slot count $n \in \mathbb{N}$, a reference value system $\mathcal{R}$, and a number $N \in \mathbb{N}_0$.
Question: Decide whether there exists a slot configuration C on $\mathcal{S} = \{1, \ldots, n\}$ of size at least N that is (circular) feasible with respect to $\mathcal{R}$.

Of course, in practice one is not only interested in the objective value of an optimal solution, but also in the solution itself, which would be the slot configuration C in this case. The problem is now obvious: If the formulation calls for a slot configuration as an output, the size of the output is not polynomially bounded in the size of the input. Thus the natural decision version is not polynomially equivalent to the function problem here, because the output of the latter cannot possibly be computed in polynomial time (even writing it down requires exponential time); thus the function problem cannot be solved in polynomial time, even if a polynomial time algorithm for the decision version exists.

We propose two possible solutions to this dilemma.

1. For certain instances a classification of the natural decision version of Slot Packing is indeed possible, see Section 3.4.2, in particular Corollary 3.36. This is due to the fact that in some cases it is possible to compute the optimal objective value without actually computing a solution.

2. The fact that the wanted output is exponential in the size of the input certainly renders futile all attempts at polynomial time algorithms for the function problem (that actually produces a solution). The best we can hope for would then be an algorithm that is polynomial in the size of the input and in n, and we will investigate such algorithms. Formally, one can cast this into a more advanced decision problem.

   Problem 3.19: Maximum Slot Packing, Verification Version
   Instance: A slot count $n \in \mathbb{N}$, a reference value system $\mathcal{R}$, and a feasible slot configuration C on $\mathcal{S} = \{1, \dots, n\}$.
   Question: Decide whether there is a slot configuration on $\mathcal{S} = \{1, \dots, n\}$ that is feasible with respect to $\mathcal{R}$ and that has size greater than $|C|$ and, if so, find such a slot configuration.

   We call this the *verification version* of Slot Packing, because it can be used to verify whether a given slot packing is maximum. The verification version of Slot Packing is solvable in polynomial time if and only if the function problem is solvable in polynomial time (using the same binary search techniques that are normally applied to establish polynomial equivalence between an optimization problem and its natural decision version, see [PS98; Pap95] for details).

3.3.1 Integer Programming Formulation

As we have already noted in Section 3.2.2, one of two parts of the integer programming formulation for Flight Scheduling consists of the time window constraints. Basically, the integer program for (Circular) Slot Packing can be thought of as the time window bounds part of the Flight Scheduling integer program:

Problem 3.20: Maximum Slot Packing, IP Formulation
Instance: The number of slots $n \in \mathbb{N}$ and the incidence matrix R of a reference value system.
Question: Find slot configuration vectors $C^{\mathrm{A}}, C^{\mathrm{D}}, C^{\mathrm{M}} \in \mathbb{N}_0^n$ that are optimal for the following integer linear program:

$$\begin{aligned} \max \quad & \mathbb{1}_n^T C^{\mathrm{M}} \\ \text{s.t.} \quad & R C^{\mathrm{A}} \leq b^{\mathrm{A}} \\ & R C^{\mathrm{D}} \leq b^{\mathrm{D}} \\ & R C^{\mathrm{M}} \leq b^{\mathrm{M}} \\ & C^{\mathrm{A}} + C^{\mathrm{D}} = C^{\mathrm{M}} \\ & C^{\mathrm{A}}, C^{\mathrm{D}}, C^{\mathrm{M}} \in \mathbb{N}_0^n \end{aligned}$$

For the circular version, simply replace all occurrences of R and b^{A}, b^{D}, b^{M} by $\mathring{R}$ and $\mathring{b}^{\mathrm{A}}$, $\mathring{b}^{\mathrm{D}}$, $\mathring{b}^{\mathrm{M}}$, respectively.

3.3.2 Complexity Considerations

Where the integer programming formulation for Flight Scheduling exhibited information on the internal composition of the problem, the integer program for Slot Packing is of even greater value, as it enables us to derive a positive complexity result due to the special structure of the constraint matrix.

Definition 3.21
A matrix $A \in \{0,1\}^{p\times q}$ with rows $a_1^T, \dots, a_p^T \in \{0,1\}^q$ is called *consecutive ones matrix* if for every row $a_i^T \neq 0$ there are $j, k \in \{1, \dots, q\}$ with $j \leq k$ such that $a_i^T = \mathbb{1}_{[j,k]}^T$.

As the rows of an incidence matrix R corresponding to a reference value system $\mathcal{R}$ are actually incidence vectors of time windows (i. e., intervals), such a matrix is a consecutive ones matrix. The importance of this observation is due to the following result (a similar result can be found in [Sch86] under the term “interval matrices”):

***Lemma* 3.22**
A consecutive ones matrix is totally unimodular.

For the proof, we will use a characterization of total unimodularity due to Ghouila-Houri (see [GH62]):

***Lemma* 3.23**
Let $A \in \{-1,0,+1\}^{p\times q}$ and denote the columns of A by $a^{(1)}, \dots, a^{(q)}$. Then A is totally unimodular if and only if for each $J \subset \{1,\dots,q\}$ there is a partition $J = J^+ \cup J^-$ such that

$$\sum_{j\in J^+} a^{(j)} - \sum_{j \in J^-} a^{(j)} \in \{-1,0,+1\}^p \,. \tag{3.11}$$

Proof (of Lemma 3.22). Let $A \in \{0,1\}^{p\times q}$ be a consecutive ones matrix. Deleting rows and/or columns from A does not destroy this property, thus each sub-matrix of A is also a consecutive ones matrix. Let $J = \{j_1, \dots, j_k\} \subset \{1,\dots,q\}$ be a collection of column indices of A and set $J^+ := \{j_1, j_3, \dots\}$ and $J^- := \{j_2, j_4, \dots\}$. Then by the consecutive ones property the sum (3.11) is in $\{-1,0,+1\}$ for every row of the resulting vector, therefore A is totally unimodular. □

Remark 3.24
A concept similar to consecutive ones matrices are *circular ones matrices*. In addition to zero rows and rows of the form $\mathbb{1}_{[j,k]}^T$, a circular ones matrix may also contain rows of the form $\mathbb{1}^T - \mathbb{1}_{[j,k]}^T$ for some $j \leq k$. The matrix $\mathring{R}$ defined in Section 3.1.4 is such a circular ones matrix. Unfortunately, a circular ones matrix is not totally unimodular in general, as is illustrated by the example

$$\mathring{R} := \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{pmatrix},$$

which corresponds to the slot set $\mathcal{S} = \{1, 2, 3\}$ and a reference value system $\{([2]_\circ, b)\}$ for arbitrary $b \in \mathbb{N}_0^3$. The determinant of $\mathring{R}$ is then $\det(\mathring{R}) = 2$.

As linear programming problems with a totally unimodular constraint matrix always have integral optimal solutions (all vertices of the feasible polyhedron are integral, see for instance [Sch86]), this property is very valuable for integer programming problems. The "integer" part of such problems need not be treated explicitly if one employs a linear programming algorithm that is guaranteed to yield a vertex of the feasible polyhedron (provided the problem is feasible and bounded). Such an algorithm can be implemented in polynomial time, e. g., using the ellipsoid method, see [GLS93]), thus integer programming problems with a totally unimodular constraint matrix are also polynomially solvable.

Alas, in the case of Slot Packing, the coupling of arrivals and departures produces a constraint matrix that is not totally unimodular anymore, although it is composed of totally unimodular matrices. This is due to the fact that we consider two different kinds of flights (arrivals and departures), coupled via a common movements constraint. However, a closer look reveals that for symmetric reference value systems we can completely disregard the distinction between arrivals and departures, as the following lemma shows (the proof is inspired by a characterization of total unimodularity by Baum and Trotter, see [BT77] or [Sch86]).

Lemma 3.25
Let $n \in \mathbb{N}$, $\mathcal{S} = \{1, \ldots, n\}$ be a slot set and $\mathcal{R}$ a symmetric reference value system. Then any function $\tilde{C} : \mathcal{S} \to \mathbb{N}_0$ with

$$\begin{aligned} &\tilde{C}(s + [L]_\circ) \leq b^{\mathrm{M}} && \text{for all } ([L]_\circ, b) \in \mathcal{R} \text{ and all } s \in \mathcal{S}, \\ \text{and}\quad &\tilde{C}(s + [L]_\circ) \leq b^{\mathrm{M}} && \text{for all } ([L]_\circ, b)^{(L)} \in \mathcal{R} \text{ and all } s \in \left\{1, L+1, \ldots, \left\lfloor \frac{n}{L} \right\rfloor L + 1\right\} \end{aligned}$$

can be extended to a feasible slot configuration, i. e., there is a slot configuration $C : \mathcal{S} \to \mathbb{N}_0^3$ that is feasible for $\mathcal{R}$ and satisfies $C^{\mathrm{M}}(s) = \tilde{C}(s)$ for all $s \in \mathcal{S}$.

Proof. Let $R \in \{0, 1\}^{m \times n}$ be the incidence matrix of the reference value system $\mathcal{R}$, let $z \in \mathbb{N}_0^n$ be the vector $z = \left(\tilde{C}(1)^T, \ldots, \tilde{C}(n)^T\right)^T$. We need to establish existence of a vector $x \in \mathbb{N}_0^n$ such that x is a feasible arrivals slot configuration (i. e., represents the C^{A} component) and $(z - x)$ is a feasible departures slot configuration (i. e., represents the C^{D} component). Hence consider the polytope

$$P := \left\{x \in \mathbb{R}^n : 0 \leq x \leq z \ \wedge \ Rz - b^{\mathrm{D}} \leq Rx \leq b^{\mathrm{A}}\right\}.$$

Let $q := \min\left\{\frac{b^{\mathrm{A}}}{b^{\mathrm{M}}}, \frac{b^{\mathrm{D}}}{b^{\mathrm{M}}} : ([L]_\circ, b)^{(\sigma)} \in \mathcal{R}\right\}$ (notice $\frac{1}{2} \leq q \leq 1$ due to symmetry of $\mathcal{R}$, and thus $1 - q \leq q$) and define $x := q \cdot z$.

Clearly, $0 \leq x \leq z$ by definition of x. Also, for every row r_i^T of the matrix R we have

$$r_i^T x = q \cdot r_i^T z \leq \frac{b_i^{\mathrm{A}}}{b_i^{\mathrm{M}}} b_i^{\mathrm{M}} = b_i^{\mathrm{A}}$$

$$\text{and} \quad r_i^T(z-x) = (1-q)r_i^T z \le (1-q)b_i^{\mathrm{M}} \le q b_i^{\mathrm{M}} \le \frac{b_i^{\mathrm{D}}}{b_i^{\mathrm{M}}} b_i^{\mathrm{M}} = b_i^{\mathrm{D}}.$$

Therefore $x \in P \neq \emptyset$, and by total unimodularity of R, the polytope P has only integral vertices. Define a slot configuration $C : \mathcal{S} \to \mathbb{N}_0^3$ by letting C^{A} be some (integral) vertex of P and $C^{\mathrm{D}} := z - C^{\mathrm{A}}$, then clearly C is an extension of $\tilde{C}$ and is feasible by definition of P. □

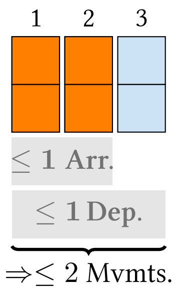


Figure 3.4: Illustration of Remark 3.26. The first two slots contain two movements each, which is feasible with respect to the movements bounds. However, only one arrival and one departure are allowed for these slots, yielding an implied movements bound of two.

Remark 3.26
Note that the symmetry condition $b^{\mathrm{A}}, b^{\mathrm{D}} \ge \frac{b^{\mathrm{M}}}{2}$ for every time window bound $(L,b)^{(\sigma)} \in \mathcal{R}$ cannot be dropped for the proposition of Lemma 3.25. To see this, consider the reference value system

$$\mathcal{R} := \left\{ ([2]_\circ, (1,3,4)^T);\ ([3]_\circ, (4,1,5)^T) \right\}$$

on the slot set $\mathcal{S} = \{1,2,3\}$. Obviously, the b^{M}-values allow for a slot configuration with $C^{\mathrm{M}} = (2,2,0)^T$, cf. Figure 3.4. In contrast, the shifting bound of length 3 allows for at most one departure within the first three slots, while the shifting bound of length 2 restricts the number of arrivals within the first two slots to one, so at most two movements (one arrival and one departure) can be scheduled in slots 1 and 2. This is clearly in conflict with the four movements prescribed by $C^{\mathrm{M}} = (2,2,0)^T$. Notice that each of the two time window bounds alone does not imply such a contradictory behavior, this is only encountered when both are applied simultaneously.

Lemma 3.25 allows us to consider *movements slot configurations* of the form $C^{\mathrm{M}} : \mathcal{S} \to \mathbb{N}_0$ interpreted as flight movements value of some extension of C. A (full) slot configuration $C : \mathcal{S} \to \mathbb{N}_0^3$ can then be constructed from C^{M} as in the proof of the foregoing lemma. Due to total unimodularity of the incidence matrix R, the construction is also algorithmically efficient. The corresponding problem will be referred to as Maximum Movements Slot Packing (the same term will be used for the circular case). As a consequence of this and total unimodularity of the incidence matrix of a reference value system, we immediately obtain the following complexity result.

***Theorem** 3.27*
*For a symmetric reference value system, Problem 3.19 (*Maximum Slot Packing, verification version*) can be solved in polynomial time. Equivalently, Problem 3.17 (*Maximum Slot Packing*) can be solved in time polynomial in n.*

Proof. The proof is a direct consequence of Lemma 3.22 and Lemma 3.25: Given a slot set $\mathcal{S} = \{1, \ldots, n\}$ and a reference value system $\mathcal{R}$ with incidence matrix R, a maximum movements slot configuration $C^{\mathrm{M}} \in \mathbb{N}_0^n$ can be found in polynomial time by solving the linear programming problem

$$\begin{aligned} \max \quad & \mathbb{1}_n^T C^{\mathrm{M}} \\ \text{s.t.} \quad & R C^{\mathrm{M}} \le b^{\mathrm{M}} \\ & C^{\mathrm{M}} \ge 0 \end{aligned}$$

and finding an optimal vertex of the feasible set, which is also integral. Then, C^{M} can be extended to a maximum slot packing by determination of a feasible vertex x of the polytope

$$P := \left\{ x \in \mathbb{R}^n : 0 \le x \le C^{\mathrm{M}} \ \wedge \ R C^{\mathrm{M}} - b^{\mathrm{D}} \le Rx \le b^{\mathrm{A}} \right\}.$$

By total unimodularity, x is integral and $C : s \mapsto (x_s, C_s^{\mathrm{M}} - x_s, C_s^{\mathrm{M}})^T$ is a maximum slot packing which is feasible with respect to $\mathcal{R}$. □

3.4 Algorithms for Maximum Slot Packing

In this section we will be looking into the Slot Packing problem from an algorithmic point of view. Although Section 3.3 in principle established a polynomial algorithm to solve the verification version of Maximum Slot Packing (and also the function problem, polynomial in the input and in n), there are still good reasons to explore alternative algorithmic approaches. On the one hand, the special structure of the constraints can be exploited to devise an easily implementable algorithm for Maximum Slot Packing and its circular version (note that the latter cannot be tackled by the approach of Section 3.3). On the other hand, albeit polynomial in theory, the practical solution of linear programming problems is often carried out by the simplex method, which can have exponential running time in the worst case (but is usually much faster than approaches based on the ellipsoid algorithm in practice). Thus alternative polynomial algorithms may not only be faster than a linear programming based technique, but also reveal some structural results about relevant classes of solutions for the Maximum Slot Packing problem. In this section, we will focus on Maximum Movements Slot Packing problems (cf. Section 3.3.2, in particular Lemma 3.25) in order to provide for a concise presentation of the algorithmic results. Of course, this implies that for a time window bound $(L, b)^{(\sigma)}$ only the movements bound value b^{M} is relevant. For this reason, we will sometimes omit the values b^{A} and b^{D} and simply write $(L, b^{\mathrm{M}})^{(\sigma)}$ instead of $(L, (b^{\mathrm{A}}, b^{\mathrm{D}}, b^{\mathrm{M}}))$, especially in the examples and illustrations.

Problem 3.28: Maximum Movements Slot Packing
Instance: A slot count $n \in \mathbb{N}$ and a reference value system $\mathcal{R}$.
Question: Find a movements slot configuration $C^{\mathrm{M}} : \mathcal{S} \to \mathbb{N}_0$ on $\mathcal{S} = \{1, \dots, n\}$ that is (circular) feasible with respect to $\mathcal{R}$ and has maximum size (i. e., maximizes the number of flight movements $C^{\mathrm{M}}(\mathcal{S})$).

3.4.1 Algorithms for Maximum Movements Slot Packing

We will first present an approach to Maximum Movements Slot Packing that is based on a reformulation for consecutive ones problems (cf. Definition 3.21) proposed by Veinott and Wagner (cf. [VW62a; VW62b]). The idea closely resembles a treatment of staff scheduling problems proposed in [BIOR80]. A more recent overview of related work on consecutive and circular ones constraints (cf. Remark 3.24) can be found in [HL06], that work also contains an outline of shortest path approaches to linear programming problems with consecutive and circular ones matrices.

A Shortest Path Formulation

In order to establish a connection between Maximum Movements Slot Packing and Shortest Path, we will look at a reformulation based on a linear programming formulation of Maximum Movements Slot Packing that is straightforwardly derived from the linear program presented in Section 3.3.1. For a slot set $\mathcal{S} = \{1, \dots, n\}$, and a reference value system $\mathcal{R} = \left\{(L_1, b_1)^{(\sigma_1)}, \dots, (L_k, b_k)^{(\sigma_k)}\right\}$ with incidence matrix R, find a vector (indicating the number of flight movements per slot) $C^{\mathrm{M}} \in \mathbb{N}_0^n$ (or $\mathbb{R}_{\geq 0}^n$ in the LP relaxation) that solves the following linear program:

$$\begin{aligned} \max \quad & \mathbb{1}^T C^{\mathrm{M}} \\ \text{s. t.} \quad & R C^{\mathrm{M}} \leq b^{\mathrm{M}} \\ & C^{\mathrm{M}} \geq 0 \end{aligned}$$

We start by defining decision variables $w_0, \dots, w_n$, where w_j counts the number of flight movements up to slot j, i. e.,

$$w_0 := 0, \qquad w_j := \sum_{t=1}^{j} C_t^{\mathrm{M}} \quad \text{for } j \in \{1, \dots, n\}. \tag{3.12}$$

Thus, the Slot Packing linear program is transformed into

$$
\begin{aligned}
\max \quad & w_n && && (3.13)\\
\text{s.t.} \quad & w_{s+L_i} - w_s \le b_i^{\mathrm{M}} && \text{for all } (L_i, b_i) \in \mathcal{R} \\
& && \quad \text{and all } s \in \{0, \dots, n - L_i\} && (3.14)\\
& w_{s+L_i} - w_s \le b_i^{\mathrm{M}} && \text{for all } (L_i, b_i)^{(L_i)} \in \mathcal{R} \\
& && \quad \text{and all } s \in \{0, L_i, \dots, (\lfloor n/L_i \rfloor - 1)\, L_i\} && (3.15)\\
& w_n - w_{\lfloor n/L_i \rfloor L_i} \le b_i^{\mathrm{M}} && \text{for all } (L_i, b_i)^{(L_i)} \in \mathcal{R}, \\
& && \quad \text{where } n \text{ is not an integer multiple of } L_i && (3.16)\\
& w_{s-1} - w_s \le 0 && \text{for all } s \in \{1, \dots, n\} && (3.17)\\
& w_0 = 0 && && (3.18)
\end{aligned}
$$

The inequalities (3.14) correspond to the shifting bounds, whereas (3.15) and (3.16) reflect the non-shifting bounds of the original problem. The constraints (3.17) model the non-negativity conditions $z \ge 0$.

We will now show that (3.13)–(3.18) is in fact the dual of a Shortest Path problem. To see this, define a digraph $G = (V, A)$ on the node set $V := \{0, \dots, n\}$ as illustrated in Figure 3.5:

- For every shifting bound $(L_i, b_i) \in \mathcal{R}$ and every time window $[s+1,\ s+L_i]$ ($s = 0, 1, \dots, n - L_i$) define a forward arc $(s, s+L_i)$, yielding the arc sets
$$A_i := \{(s, s+L_i) : s = 0, 1, \dots, n - L_i\} \quad \text{for all } (L_i, b_i) \in \mathcal{R}.$$

- For every non-shifting bound $(L_i, b_i)^{(L_i)} \in \mathcal{R}$ and every time window $[s+1,\ s+L_i]$ (for suitably chosen $s \in \mathcal{S}$) and, if necessary, for the "truncated" time windows ending in slot n, define a forward arc $(s, s+L_i)$, yielding the arc sets
$$
\begin{aligned}
A_i &:= \{(s, s+L_i) : s = 0, L_i, \dots, (\lfloor n/L_i \rfloor - 1)\, L_i\} \quad \text{for all } (L_i, b_i)^{(L_i)} \in \mathcal{R},\\
A' &:= \left\{ \left(\lfloor n/L_i \rfloor L_i, n\right) : (L_i, b_i)^{(L_i)} \in \mathcal{R} \ \wedge\ n \text{ is not an integer multiple of } L_i \right\},
\end{aligned}
$$

- For each pair of consecutive nodes $u-1, u$ define a backward arc $(u, u-1)$, yielding the arc set
$$B := \{(n, n-1), (n-1, n-2), \dots, (1, 0)\}.$$

Thus the graph $G = (V, A)$ has the arc set
$$A := \left(\bigcup_{i=1,\dots,k} A_i \right) \cup A' \cup B,$$

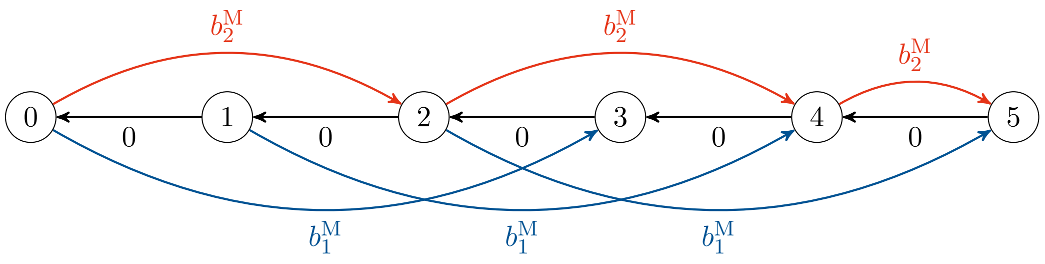


Figure 3.5: Slot Packing as Shortest Path problem. Shown is an example for $n = 5$ slots and the reference value system $\mathcal{R} = \{([3]_\circ, b_1), ([2]_\circ, b_2)^{(2)}\}$, containing one shifting and one non-shifting bound.

containing one forward arc for every time window of a time window bound plus n backward arcs.

Furthermore, define an arc length function $l : A \to \mathbb{N}_0$,

$$l((u,v)) := l_{uv} := \begin{cases} b_i^{\mathrm{M}}, & \text{if } (u,v) \in A_i; \\ b_i^{\mathrm{M}}, & \text{if } (u,v) = (\lfloor n/L_i \rfloor L_i, n) \in A' \\ 0, & \text{if } (u,v) \in B. \end{cases}$$

In terms of the digraph G the linear program (3.13)–(3.18) now reads

$$\begin{aligned} \max \quad & w_n \\ \text{s. t.} \quad & w_v - w_u \le l_{uv} && \text{for every arc } (u,v) \in A \\ & w_0 = 0. \end{aligned}$$

With Kronecker's delta function $\delta_{un} \in \{0,1\}$, defined as $\delta_{un} = 1 \Leftrightarrow u = n$, the dual of that linear program is

$$\min \quad \sum_{(u,v)\in A} l_{uv} \cdot f_{uv} \tag{3.19}$$

$$\text{s. t.} \quad \sum_{v\in V:(v,u)\in A} f_{vu} - \sum_{v\in V:(u,v)\in A} f_{uv} = \delta_{un} \qquad \text{for } u = 1,\dots,n \tag{3.20}$$

$$\sum_{v\in V:(v,0)\in A} f_{v0} - \sum_{v\in V:(0,v)\in A} f_{0v} = g \tag{3.21}$$

$$f \ge 0 \tag{3.22}$$

$$g \in \mathbb{R}. \tag{3.23}$$

This linear program is easily recognized as a network flow problem with flow variables f_{uv}, $(u,v) \in A$, where (3.20) and (3.21) constitute the flow conservation requirements. Notice that (3.21) is redundant and can be dropped together with the decision variable g (both originate from considering w_0 as a variable and $w_0 = 0$ as a constraint in the primal). We then arrive at the problem of sending one unit of flow from the node 0 to the node n at minimum cost, where the cost of sending one unit of flow across arc $(u, v) \in A$ is l_{uv}. This special network flow problem is, of course, equal to the problem of finding a shortest path from 0 to n in the graph G with arc length function l. Notice that for a shortest path problem we are also guaranteed that an integral solution w to the dual shortest path program (3.13)–(3.18) can be found efficiently.

Multiple algorithms exist for the solution of Shortest Path problems. For the setting at hand (featuring a digraph with nonnegative arc lengths) one can, e. g., employ Dijkstra's algorithm (cf. [Dij59]) requiring at most $\mathcal{O}\left(n^2\right)$ operations. Using clever data structures (like a Fibonacci heap), the algorithm can also be implemented with running time $\mathcal{O}\left(|A| + n \log n\right)$, see [AMO93]. Dijkstra's algorithm has one particular advantage for the setting encountered in the context of Slot Packing: It is fundamentally a primal-dual algorithm[1], which means it does not only solve the Shortest Path problem, but also produces a solution to the original (dual shortest path problem) along with the shortest path. Thus we arrive at Algorithm 3.1, solving the verification version of Maximum Movements Slot Packing in polynomial time.

Algorithm 3.1: Dijkstra's algorithm for Maximum Movements Slot Packing

```
Input: A slot count $n \in \mathbb{N}$ and reference value system $\mathcal{R} = \left\{(L_1, b_1)^{(\sigma_1)}, \dots, (L_k, b_k)^{\sigma_k}\right\}$
       on the slot set $\mathcal{S} = \{1, \dots, n\}$.
Output: A movements slot configuration $C^{\mathrm{M}} : \mathcal{S} \to \mathbb{N}_0$, which is feasible with respect to $\mathcal{R}$
        and has maximum size.
1  Construct a graph $G = (V, A)$ with arc length function $l : A \to \mathbb{N}_0$ as in Section 3.4.1.
2  Set $U \leftarrow \{0\}$, $w_0 \leftarrow 0$.
3  Set $w_v \leftarrow l_{0v}$ for all $v \in V$ with $(0, v) \in A$.
4  Set $w_v \leftarrow \infty$ for all $v \in V$ with $(0, v) \notin A$.
5  while $U \neq V$ do
6      Find $u \in V \setminus U$ such that $w_u = \min\{w_v : v \in U \setminus V\}$.
7      Set $U \leftarrow U \cup \{u\}$.
8      Set $w_v \leftarrow \min\{w_v, w_u + l_{uv}\}$ for all $v \in V$ with $(u, v) \in A$.
9  end
10 Set $C^{\mathrm{M}}_u \leftarrow w_u - w_{u-1}$ for all $u \in \mathcal{S} = \{1, \dots, n\}$.
```

Theorem 3.29
An instance of Maximum Movements Slot Packing *on n slots and with a reference value system consisting of k time window bounds can be solved in running time $\mathcal{O}\left(kn + n \log n\right)$.*

[1]A primal-dual algorithm basically considers a primal problem and its dual simultaneously. For a primal feasible solution it tries to produce a corresponding dual solution such that the primal-dual pair fulfills the complementary slackness conditions, thus providing for a certificate of optimality. If that is not possible (due to non-optimality of the primal feasible solution), an improvement to the primal solution can be computed. Repeating this process yields an optimal primal-dual pair after finitely many steps. A detailed exposition on primal-dual algorithms and Dijkstra's algorithm in particular can be found in [PS98].

Proof. Using the transformation described above, Maximum Movements Slot Packing can be transformed into a Shortest Path problem on a digraph with $(n+1)$ nodes and at most $(n+nk)$ arcs (at most n arcs are required for each time window bound). A Fibonacci heap implementation of Dijkstra's algorithm (like the one described in detail in [AMO93]) thus produces an optimal solution of both the Shortest Path problem and its dual (3.13)–(3.18) in running time $\mathcal{O}\left((k+1)n + n\log n\right) = \mathcal{O}\left(kn + n\log n\right)$. By reversing the transformation (3.12) (which can be done using n elementary operations), we obtain an optimal solution within the running time claimed in the statement of the theorem. □

A Greedy Algorithm

We will now show that close analysis of the Shortest Path approach yields the insight that the algorithm can even be implemented as a pure greedy strategy for Maximum Movements Slot Packing. As stated above, Dijkstra's algorithm can be used on a suitably defined graph to obtain a solution of the slot packing problem, cf. Algorithm 3.1.

However, the particular graph $G = (V, A)$ with arc length function $l : A \to \mathbb{N}_0$ defined in Section 3.4.1 has a very special property: For every node $u \in V \setminus \{0\} = \{1, \ldots, n\}$, there is exactly one backward arc $(u, u-1)$, all other arcs are forward arcs, meaning they have the form (u, v) for some $v > u$. Moreover, the backward arcs all have length 0, while the forward arcs all have positive length (assuming all time window bounds are strictly greater than 0). Dijkstra's algorithm basically works like this (cf. Algorithm 3.1):

- Choose and "tag" (i. e., add to the set U) a node u with minimum label w_u among all nodes in $V \setminus U$.
- "Propagate" the node label w_u to all neighbors $v \in V \setminus U$ of node u, setting w_v to $w_u + l_{uv}$ if this decreases v's label.

Hence whenever a node u gets tagged and $(u-1)$ has not yet been tagged, we can simply set w_{u-1} to w_u (as there is an edge of length 0 from u to $(u-1)$) and tag $u-1$ as well. We can therefore assume without loss of generality that the nodes are tagged in order of their node number. After the algorithm has finished, the labels meet the condition

$$w_{u-1} < w_u \Rightarrow w_v + l_{vu} = w_u \quad \text{for some } v < u \text{ such that } (v,u) \in A \text{ is a forward arc,}^{2}$$

thus at the optimal solution for every node $u \in V$ one of the following cases applies:

1. There is some node $v < u$ such that $(v, u) \in A$ and $w_v + l_{vu} = w_u$: This situation corresponds to some time window bound $(L_i, b_i)^{(\sigma_i)} \in \mathcal{R}$, such that $[v+1,\, u]$ is one of its corresponding time windows and $w_u - w_v = b_i^{\mathrm{M}}$. (This means that the flight movements added at slot u make that time window "active".)

[2]These conditions are actually optimality conditions and are equal to complementary slackness for the shortest path problem.

2. $w_{u-1} = w_u$: This situation corresponds to some time window bound $(L_i, b_i)^{(\sigma_i)} \in \mathcal{R}$ and a slot $s \in \mathcal{S}$, such that
    - $(s+1) + [L_i]_\circ$ is a time window corresponding to the time window bound,
    - $(u-1), u \in (s+1) + [L_i]_\circ$ and
    - $w_{u-1} - w_s = b_i^{\mathrm{M}}$.

   (This means no more movements can be added to slot u, as it already contained in some "active" time window.)

With this analysis in mind, we can interpret Dijkstra's algorithm for the Maximum Movements Slot Packing problem as the following greedy algorithm:

- Start at slot 1 and set c_1^{M} to the maximum number of movements that is possible without violating a time window bound.
- After slots 1 to u have been processed, move on to the next slot $u+1$ and set C_{u+1}^{M} to the maximum number that is possible without violating a time window bound.

For a concise presentation of that algorithm, let us define the following notation:

Definition 3.30
Let $\mathcal{S} = \{1, \dots, n\}$ be a slot set, $s \in \mathcal{S}$ and $(L, b)^{(\sigma)}$ a time window bound. The *covering time windows set* $\mathcal{W}((L, b)^{(\sigma)}, s)$ is then defined as the set of all time windows corresponding to $(L, b)^{(\sigma)}$ on the slot set $\mathcal{S}$ that contain the slot s, precisely:

- For a shifting bound ($\sigma = 1$)
$$\mathcal{W}((L, b)^{(\sigma)}, s) := \{[t+1,\, t+L] : t \in \{s-L, \dots, s-1\} \;\wedge\; [t+1,\, t+L] \subset \mathcal{S}\}\,;$$
- for a non-shifting bound ($\sigma = L$)
$$\mathcal{W}((L, b)^{(\sigma)}, s) := \begin{cases} [(\lceil s/L \rceil - 1)\, L + 1,\, \lceil s/L \rceil\, L]\,, & \text{if } \lceil s/L \rceil\, L \le n; \\ [(\lceil s/L \rceil - 1)\, L + 1,\, n]\,, & \text{if } \lceil s/L \rceil\, L > n. \end{cases}$$

The greedy formulation of the Shortest Path approach is outlined in Algorithm 3.2.

For the running time of Algorithm 3.2, define $L' := \max\{L_1, \dots, L_k\}$ and note that computing $C^{\mathrm{M}}(W)$ for all time windows W of length L that contain a given slot s can take up to $\mathcal{O}(kL'^2)$ operations using a naive implementation. Of course, this runtime bound can be improved by storing the values of $C^{\mathrm{M}}(W)$ for all relevant time windows and simply updating those values in each iteration of the "for" loop in Algorithm 3.2. As we move forward one slot at a time (changing the slot configuration only at that one slot), the update is fairly straightforward and can be performed in $\mathcal{O}(kL')$ steps, yielding an overall runtime bound of $\mathcal{O}(nkL')$ operations for the greedy implementation. As L' and k are usually small constants compared to n, this algorithm is of linear order in most applications, whereas the pure Dijkstra implementation has complexity $\mathcal{O}(n \log n)$ for constant k.

Algorithm 3.2: Greedy algorithm for Maximum Movements Slot Packing

```
Input: A slot count $n \in \mathbb{N}$ and reference value system $\mathcal{R} = \left\{(L_1, b_1)^{(\sigma)}, \dots, (L_k, b_k)^{\sigma}\right\}$ on
       the slot set $\mathcal{S} = \{1, \dots, n\}$.
Output: A movements slot configuration $C^{\mathrm{M}} : \mathcal{S} \to \mathbb{N}_0$, which is feasible with respect to $\mathcal{R}$
        and has maximum size.
1 Set $C_i^{\mathrm{M}} \leftarrow 0$ for all $i \in \mathcal{S}$.
2 for s=$1, \dots, n$ do
3     Set $C_s^{\mathrm{M}} \leftarrow \min\left\{\max\left\{0, b^{\mathrm{M}} - C^{\mathrm{M}}(W)\right\} : W \in \mathcal{W}((L,b)^{(\sigma)}, s) \ \wedge \ (L,b)^{(\sigma)} \in \mathcal{R}\right\}$.
4 end
```

3.4.2 Algorithms for Circular Maximum Movements Slot Packing

A Shortest Path Formulation

For the algorithms of Section 3.4.1 the consecutive ones property of the reference value system's incidence matrix is crucial, as it guarantees that the reformulation (3.12) yields a linear programming problem with exactly two variables per inequality, one with a positive and one with a negative sign, which corresponds to a digraph's arc-node incidence matrix. Alas, this property is lost when we consider circular bounds, which correspond to a circular incidence matrix representing the reference value system. In this subsection, we will therefore investigate a method to carry over the idea of the Shortest Path approach to circular shifting bounds, and afterwards present a greedy algorithm that can be applied for both circular and non-circular bounds. Recall that for circular constraints we only consider shifting bounds.

It has already been said that the Shortest Path algorithm of Section 3.4.1 can be generalized to the circular setting, and the way to do this is by resorting to Parametric Shortest Path problems. A Parametric Shortest Path problem is defined by a digraph $G = (V, A)$ with a distinguished node $0 \in V$ and an arc length function $l^{\lambda} : A \to \mathbb{R}$ dependent on a parameter $\lambda \in [0, \infty[$:

$$l_{uv}^{\lambda} := l((u,v)) := l_{uv}^{*} + \lambda \tilde{l}_{uv},$$

where $l^*, \tilde{l} : A \to \mathbb{R}$ are two unparameterized arc length functions. The task of Parametric Shortest Path is now to compute the shortest path distances w_u^{λ} from the node 0 to all nodes $u \in V$ with respect to the arc length function l^{λ} for all values of $\lambda \geq 0$.

We give an outline of how to solve this problem (details can be found in [AMO93] and [YTO91]):

- First, consider a shortest path tree T^{λ} for any given value of λ. A shortest path tree $T^{\lambda} \subset A$ is a tree in G such that the shortest 0-u paths with respect to l^{λ} are contained in T^{λ} for all nodes $u \in V$. One can then show that $w_u^{\lambda} = (w^*)_u^{\lambda} + \lambda \cdot (\tilde{w})_u^{\lambda}$, where $(w^*)_u^{\lambda}$ and $(\tilde{w})_u^{\lambda}$ are the shortest path distances in the subgraph $\left(V, T^{\lambda}\right)$ with respect to the arc length functions l^* and $\tilde{l}$, respectively.

- Starting with $\lambda = 0$, determine a "breakpoint parameter value" λ' greater than the current value of λ such that the shortest path tree stays unaltered for all parameter values between the current λ and λ'. To this end, consider all non-tree arcs in the graph and determine the minimum parameter value increase where at least one of these arcs must enter the shortest path tree (leading to a new shortest path tree via an arc exchange).

- Successively determine all breakpoint parameter values, i. e., those values for the parameter λ where the shortest path tree changes, and compute the corresponding shortest path trees.

One can show that is algorithm produces only a finite number of different shortest path trees, thus the Parametric Shortest Path problem reduces to successively computing all (finitely many) breakpoint parameter values and applying a shortest path algorithm with respect to l^* and $\tilde{l}$ for all the resulting shortest path trees. A refined version of this idea is presented in [YTO91], yielding an algorithm with running time $\mathcal{O}\left(|V|\,|A| + |V|^2 \log |V|\right) = \mathcal{O}\left(|V|^3\right)$ that produces at most $|V|$ different shortest path trees for a digraph $G = (V, A)$

We will now apply the Parametric Shortest Path algorithm to Maximum Movements Slot Packing on the slot set $\mathcal{S} = \{1, \dots, n\}$ with reference value system $\mathcal{R}$ and circular time window bounds. Recall that we only consider shifting bounds for circular problems, so using again the transformation (3.12), the problem is equal to (the integer programming version of) the linear program

$$\begin{array}{lll} \max & w_n & \\ \text{s.t.} & w_{s+L_i} - w_s \le b_i^{\mathrm{M}} & \text{for all } (L_i, b_i) \in \mathcal{R} \text{ and all } s \in \{0, \dots, n - L_i\} \\ & w_{s-n+L_i} - w_s \le b_i^{\mathrm{M}} - w_n & \text{for all } (L_i, b_i) \in \mathcal{R} \text{ and all } s \in \{n - L_i + 1, \dots, n-1\} \\ & w_{s-1} - w_s \le 0 & \text{for all } s \in \{1, \dots, n\} \\ & w_0 = 0 & \end{array}$$

We turn this into a parameterized linear program by replacing the variable w_n on the right hand side by the parameter λ. Then, analogous to the above treatment, a corresponding graph $\mathring{G} = (V, \mathring{A})$ can be defined (see Figure 3.6) on the node set $V = \{0, \dots, n\}$ and the arc set

$$\mathring{A} := \left(\bigcup_{i=1,\dots,k} (\mathring{A}_i \cup (\mathring{A})'_i \right) \cup B,$$

$$\begin{array}{rll} \text{where} \quad \mathring{A}_i := & \{(s, s+L_i) : s = 0, \dots, n - L_i\} & \text{for all } (L_i, b_i) \in \mathcal{R} \\ (\mathring{A})'_i := & \{(s, s-n+L_i) : s = n - L_i + 1, \dots, n-1\} & \text{for all } (L_i, b_i) \in \mathcal{R} \\ B := & \{(n, n-1), (n-1, n-2), \dots, (1, 0)\}\,. & \end{array}$$

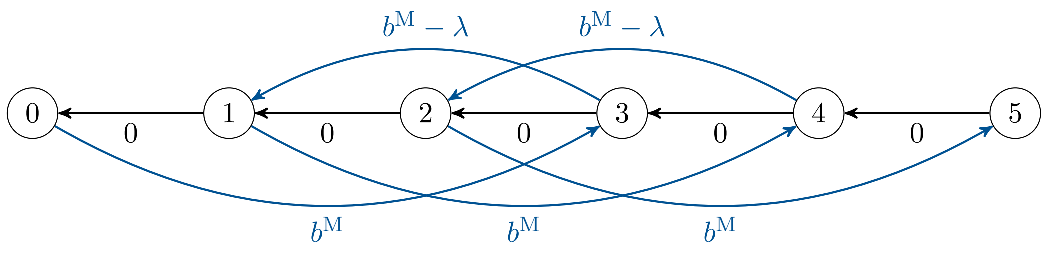


Figure 3.6: Circular Slot Packing as Shortest Path problem. Shown is an example for $n = 5$ slots and the reference value system $\mathcal{R} = \left\{\left([3]_\circ, b\right)\right\}$, containing one shifting bound that is applied circularly.

The arc length function $l^\lambda : A \to \mathbb{R}_0$ is now defined as

$$l^\lambda((u,v)) := l^\lambda_{uv} := \begin{cases} b^{\mathrm{M}}_i, & \text{if } (u,v) \in \mathring{A}_i; \\ b^{\mathrm{M}}_i - \lambda, & \text{if } (u,v) \in (\mathring{A})'_i; \\ 0, & \text{if } (u,v) \in B. \end{cases}$$

The task at hand is then to solve the Parametric Shortest Path problem on $\mathring{G}$ starting at $\lambda = 0$, thus obtaining the breakpoint parameter values for λ and the corresponding values for the decision variables w^λ_u (which are equal to the shortest path distances from node 0 to the nodes u) for $u \in V$. For a solution to the original problem, we impose the additional requirement that

$$w^\lambda_n = \lambda. \tag{3.24}$$

Having obtained a solution to Parametric Shortest Path, it is then an easy task to determine a value of λ with that property and the corresponding shortest path distance w^λ_n.

Note, however, that the graph $\mathring{G}$ may contain arcs of negative length depending on the parameter λ. Thus we may no longer use Dijkstra's algorithm for the shortest 0-n path computations needed for the solution of vParametric Shortest Path, but rather an algorithm that can handle negative arc lengths, e. g., the algorithms of Bellman-Ford or Floyd-Warshall (see [PS98] for details on these algorithms). This fact also accounts (at least partly) for the runtime $\mathcal{O}\left(|V|^3\right)$ of the algorithm from [YTO91] mentioned above. For that algorithm, a certain procedure for producing the shortest path trees is employed that guarantees that a maximum of $|V|$ such trees are needed for the solution. Application of the proposed Parametric Shortest Path algorithm on the graph $\mathring{G}$ with arc length function l^λ as defined above and afterwards computing the correct parameter value now yields the desired algorithm for circular Maximum Movements Slot Packing.

Theorem 3.31
An instance of circular Maximum Movements Slot Packing *on n slots and with a reference value system consisting of k shifting bounds can be solved in running time $\mathcal{O}\left(kn^2 + n^2 \log n\right)$.*

Proof. Transforming the instance into a Parametric Shortest Path problem as outlined above and applying the algorithm of [YTO91] we get a complete set of shortest path trees in running time at most $\mathcal{O}\left(|V|\,|\mathring{A}| + |V|^2 \log |V|\right) = \mathcal{O}\left(kn^2 + n^2 \log n\right)$. By [YTO91], the algorithm yields at most $(n+1)^2$ shortest path trees, hence using (3.24), the correct value for λ can be computed in running time $\mathcal{O}\left(n^2\right)$. The optimal slot packing can then be devised by computing the dual value to the shortest path distances obtained by the algorithm (which can certainly be done in course of the algorithm as well), analogous to the case of non-circular time window bounds treated above. Therefore, the overall running time is dominated by the parametric shortest path algorithm. As $|\mathring{A}| \leq (n+1) + k(n+1)$, this yields the desired runtime bound. □

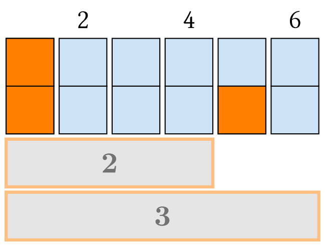
A Greedy Algorithm for (Circular) Maximum Movements Slot Packing

While the Shortest Path approach of Section 3.4.1 can somehow be carried over to the situation of circular shifting bounds, the greedy interpretation of Section 3.4.1 fails, because the arc set is more complex for the graph $\mathring{G}$ defined in the foregoing subsection. In this subsection, we will investigate a different greedy-like approach to (circular) Maximum Movements Slot Packing, which will also reveal some structural results about relevant classes of solutions for the problem. More specifically, we will present an algorithm that produces a uniform optimal solution, thereby also providing an alternative proof of polynomial solvability of the circular and non-circular variant of Maximum Slot Packing (more precisely, their respective verification versions) as well as proving their equivalence in terms of objective value under minor restrictions. As this section is only concerned with movements slot configurations, we will only denote the movements bound of a time window bound, thus by writing $(L, b)^{(\sigma)}$ we imply that $b \in \mathbb{R}_{\geq 0}$ is the movements bound value if not explicitly stated otherwise.

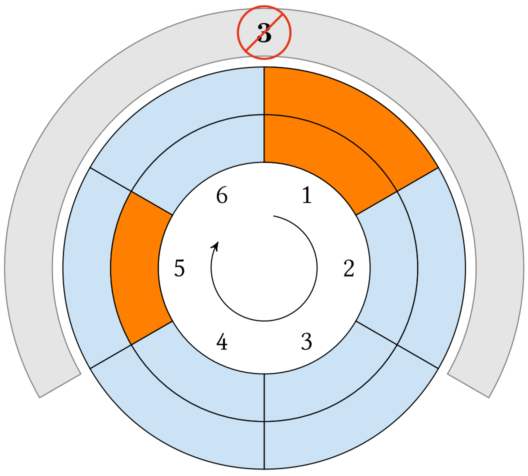
Algorithm 3.2 can be improved in one aspect: It does not necessarily yield a slot configuration that is also feasible when the shifting bounds are applied as circular shifting bounds, as Example 3.32 shows. This immediately raises the question whether there is an easy algorithm for Maximum Movements Slot Packing that yields a circular feasible optimal solution.

Example 3.32
Consider the reference value system $\mathcal{R} = \{([6]_\circ, 3); ([4]_\circ, 2)\}$ (only movements bound values are given) on the slot set $\mathcal{S} = \{1, \ldots, 6\}$. Application of Algorithm 3.2 yields the movements slot configuration $C^{\mathrm{M}} = (2, 0, 0, 0, 1, 0)$ depicted in Figure 3.7a. This solution is certainly feasible with respect to $\mathcal{R}$, but $C^{\mathrm{M}}(\llbracket 5 + [4]_\circ \rrbracket_n) = 3 > 2$, thus C^{M} is infeasible for the circular variant of Maximum Movements Slot Packing, see Figure 3.7b.

For circular shifting bounds, the flight movements have to be distributed in a more uniform way. Algorithm 3.3 presents an approach to do this. The algorithm works in two steps: First, a number of $\lfloor q \rfloor$ movements, where $q = {}^{b_k}/_{L_k}$ for the longest shifting bound L_k, is allocated to each slot. This is exactly the number of movements that can be allocated uniformly to each slot without violating any of the shifting bounds. Then at most one movement is added to each slot, starting at slot 1 and proceeding by increasing slot number. The movements are added in a way that keeps



(a) Non-circular bounds



(b) Circular bounds

Figure 3.7: Greedy movements slot configuration of Example 3.32

the ratio of movements per slot close to the quotient q, and hence a very uniform slot configuration is obtained. See Example 3.33 for an exemplary application of Algorithm 3.3.

Example 3.33
Let us illustrate Algorithm 3.3 by a small example: We consider the slot set $\mathcal{S} = \{1, \dots, 15\}$ and the reference value system $\mathcal{R} = \{([6]_\circ, 3); ([4]_\circ, 2)\}$ (only movements bound values are given) as in Example 3.32. As $3/6 = 2/4 = 1/2$, the system is clearly monotone and $q = 1/2$. The algorithm then computes a slot packing as follows:

1. $\lfloor q \rfloor = 0$, so start with $z = 0 \in \mathbb{N}_0^6$ and $v = 0$.
2. $\mathbf{j = 1}$: $v/j = 0 < q \Rightarrow z_1 \leftarrow 1$ and $v \leftarrow 1$.
3. $\mathbf{j = 2}$: $v/j = 1/2 \nless q \Rightarrow$ no change.
4. $\mathbf{j = 3}$: $v/j = 1/3 < q \Rightarrow z_3 \leftarrow 1$ and $v \leftarrow 2$.
5. $\mathbf{j = 4}$: $v/j = 2/4 \nless q \Rightarrow$ no change.
6. $\mathbf{j = 5}$: $v/j = 2/5 < q \Rightarrow z_5 \leftarrow 1$ and $v \leftarrow 3$.
7. $\mathbf{j = 6}$: $v/j = 3/6 \nless q \Rightarrow$ no change.
8. The output of the algorithm is the movements slot configuration $z = (1, 0, 1, 0, 1, 0)$, as shown in Figure 3.8. ♢

Algorithm 3.3: Greedy Algorithm for (Circular) Maximum Movements Slot Packing

Input: A monotone reference value system $\mathcal{R} = \{(L_1, b_1), \dots, (L_k, b_k)\}$ with $L_1 < L_2 < \cdots < L_k$.

Output: A circular maximum movements slot configuration $z \in \mathbb{N}_0^n$ for $n = L_k$.

```
1  Set $q \leftarrow \frac{b_k}{L_k}$.
2  Set $z \leftarrow \lfloor q \rfloor \cdot \mathbb{1} \in \mathbb{N}_0^n$.
3  Set $v \leftarrow 0$.
4  for $j = 1$ to $n$ do
5      Set $v \leftarrow v + z_j$.
6      if $\frac{v}{j} < q$ then
7          Set $z_j \leftarrow z_j + 1$.
8          Set $v \leftarrow v + 1$.
9      end
10 end
```

Algorithm 3.3 produces a maximum movements slot configuration as long as the reference value system is monotone. This requirement is not very surprising, as it is generally impossible to get a uniform slot configuration for a non-monotone reference value system. What is more surprising is the fact that the algorithm only relies on the values of L_k and b_k, completely ignoring the other shifting bounds. Of course, this is a direct consequence of monotonicity, which fundamentally states that the longest bound (L_k, b_k) is also the relatively strictest, allowing for the least number of movements per slot. As the algorithm takes this very ratio as a “target measure”, the solution will automatically be feasible for *each* shifting bound in the reference value system.

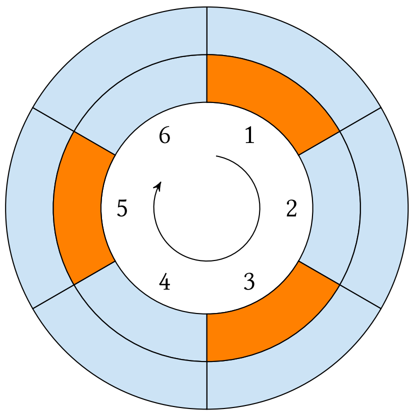


Figure 3.8: Result of Algorithm 3.3 for Example 3.33.

Lemma 3.34
Let $\mathcal{R} = \{(L_1, b_1), \ldots, (L_k, b_k)\}$ be a monotone reference value system that consists of only shifting bounds and assume $L_k \geq L_{k-1} \geq \cdots \geq L_1$. Then Algorithm 3.3 computes a circular maximum movements slot configuration on the slot set $\{1, \ldots, L_k\}$.

Proof. Set $n := L_k$ and let $v^{(j)}$ and $z^{(j)}$ be the values of v and z at the end of the j-th pass of the for loop in Algorithm 3.3. Furthermore, define $q = \frac{b_k}{L_k}$ and $q' := \lfloor q \rfloor$. Of course, if $q = q'$, then the algorithm never enters the "then"-block in lines 7 and 8, thus $v^{(j)} = j \cdot q'$ for all $j \in \{1, \ldots, n\}$ and $\sum_{j=1}^{n} z_j = n \cdot q' = b_k$. The vector z is then clearly a slot configuration of maximum size and it is feasible, thus a maximum slot packing as asserted.

In the following, let us assume that $q' < q$, which in particular implies $\frac{v^{(0)}+q'}{1} < q$. The idea of Algorithm 3.3 is to compute a slot packing z whose "movements to slots ratio" stays "close to q". More precisely, in each iteration either q' or $q'+1$ movements are added to the slot configuration z, and we will first show, that the algorithm guarantees that for every $j \in \{1, \ldots, n\}$ the inequalities

$$\frac{v^{(j)} - 1}{j} < q \leq \frac{v^{(j)}}{j} \tag{3.25}$$

hold. We prove these by induction on j. For $j = 1$, notice that $\frac{q'}{1} < q$, hence $z_1 = q' + 1$ and also $v^{(1)} = q' + 1$. We then get

$$\frac{v^{(1)} - 1}{1} < q \leq \frac{v^{(1)}}{1} \quad \Leftrightarrow \quad q' < q \leq q' + 1,$$

which is obviously true by definition of q'.

For the inductive step, assume (3.25) is true up to $j - 1$, so our induction hypothesis reads

$$\frac{v^{(j-1)} - 1}{j - 1} < q \leq \frac{v^{(j-1)}}{j - 1}.$$

We distinguish two cases:

Case $\mathbf{v^{(j)} = v^{(j-1)} + q'}$: From the first inequality of the induction hypothesis, we get

$$\begin{aligned} & v^{(j-1)} - 1 < (j-1)q \\ \Rightarrow\ & v^{(j-1)} + q' - 1 < (j-1)q + q' \leq (j-1)q + q = jq \\ \Rightarrow\ & \frac{v^{(j)} - 1}{j} < q. \end{aligned}$$

For the second inequality, notice that $v^{(j)} = v^{(j-1)} + q'$ implies $z_j = q'$ and thereby

$$\frac{v^{(j)}}{j} = \frac{v^{(j-1)} + q'}{j} \geq q,$$

so the statement is obvious.

Case $\mathbf{v^{(j)} = v^{(j-1)} + q' + 1}$: This case implies $z_j = q' + 1$ and thereby

$$\frac{v^{(j-1)} + q'}{j} < q.$$

Hence we get

$$\frac{v^{(j)} - 1}{j} = \frac{v^{(j-1)} + q'}{j} < q,$$

the first part of the claimed inequality. For the second part, suppose $\frac{v^{(j-1)}+q'+1}{j} = \frac{v^{(j)}}{j} < q$. By induction, $\frac{v^{(j-1)}}{j-1} \geq q$, thus

$$\begin{aligned} \frac{v^{(j-1)} + q' + 1}{j} &< \frac{v^{(j-1)}}{j-1} \\ \Leftrightarrow (j-1)(v^{(j-1)} + q' + 1) &< jv^{(j-1)} \\ \Leftrightarrow j(q'+1) &< v^{(j-1)} + q' + 1 = v^{(j)} \\ \Leftrightarrow q' + 1 &< \frac{v^{(j)}}{j}. \end{aligned}$$

But this means $q' + 1 < q$, clearly a contradiction (recall $q' = \lfloor q \rfloor$).

The inequalities (3.25) provide a bound on the final solution $v^* = v^{(n)}$:

$$\frac{v^* - 1}{n} < q \leq \frac{v^*}{n},$$

yields (recall $n = L_k$ and $q = {}^{b_k}/_{L_k}$)

$$v^* - 1 <_k^b \leq v^* \Rightarrow v^* = b_k.$$

Due to the shifting bound (L_k, b_k), there can be no better solution, thus the algorithm produces a slot packing of maximum size. Hence to prove optimality of the solution, we have to show that the solution is indeed feasible with respect to *all* shifting bounds in $\mathcal{R}$.

In order to prove feasibility of the solution z^* produced by the algorithm, let (L_r, b_r) be some shifting bound contained in $\mathcal{R}$ and let $t \in \{1, \ldots, n\}$ such that $t + (L_r - 1) \leq n$. Then

$$\begin{aligned} \sum_{j=t}^{t+L_r-1} z_j &= \left(v^{(t+L_r-1)} - v^{(t-1)}\right) \\ &< (q(t + L_r - 1) + 1 - q(t-1)) \quad \text{due to (3.25)} \\ &= qL_r + 1 = \frac{b_k}{L_k} L_r + 1. \end{aligned}$$

Hence
$$\sum_{j=t}^{t+L_r-1} z_j < qL_r + 1 = \frac{b_k}{L_k} L_r + 1 \leq b_r + 1,$$
as $\frac{b_k}{L_k} \leq \frac{b_r}{L_r}$ by monotonicity of $\mathcal{R}$. As $z \in \mathbb{N}_0^n$, the sum above has to be integral, meaning
$$\sum_{j=t}^{t+L_r-1} z_j \leq b_r,$$
thus z is feasible with respect to (L_r, b_r) for all shifting bounds $(L_r, b_r) \in \mathcal{R}$. This proves z is a maximum slot packing. □

The following important result is an immediate consequence of Algorithm 3.3 and Lemma 3.34.

Theorem 3.35
Let $\mathcal{R} = \{(L_1, b_1), \ldots, (L_k, b_k)\}$ be a monotone reference value system with $L_k \geq L_{k-1} \geq \cdots \geq L_1$ and let $n = rL_k$ for some $r \in \mathbb{N}$. Then a maximum movements slot configuration on $\mathcal{S} = \{1, \ldots, n\}$ with respect to $\mathcal{R}$ has size rb_k.

This result also provides us with a polynomial algorithm for a restricted class of instances of the decision version of MAXIMUM MOVEMENTS SLOT PACKING. Recall that the decision problem only asked for the value of an optimal solution (more precisely, whether there is a solution exceeding some given objective value N), not for the solution itself (and producing an actual solution would necessarily result in an exponential algorithm).

Corollary 3.36
Let $(n, \mathcal{R}, N)$ be an instance of the decision version of MAXIMUM MOVEMENTS SLOT PACKING *with the following additional properties:*

1. *$\mathcal{R} = \{(L_1, b_1), \ldots, (L_k, b_k)\}$ is a monotone reference value system consisting exclusively of shifting bounds,*
2. *n is a multiple of $\max\{L_j : (L_j, b_j) \in \mathcal{R}\}$.*

Then the instance can be solved in time polynomial in k (and thus in time polynomial in the size of the input).

Proof. Let us first assume that $L_k = \max\{L_j : (L_j, b_j) \in \mathcal{R}\}$. As n is a multiple of L_k, there is some $r \in \mathbb{N}$ such that $n = rL_k$. Then application of Theorem 3.35 yields the result that a maximum slot packing on $\mathcal{S} = \{1, \ldots, n\}$ has size rb_k. Thus we only need to find the maximum among the $L_1, \ldots, L_k$, which can be done in time $\mathcal{O}(k \log k)$ by simply sorting these numbers. Then computing r and rb_k are two elementary operations, and so is the comparison of rb_k and N, which promptly yields the desired answer. Thus the algorithm has running time $\mathcal{O}(k \log k)$, which is polynomial in the size of the input. □

3.5 Minimum Slot Cover

3.5.1 Covering Slot Configurations

In the context of Flight Scheduling and Maximum Slot Packing, a natural question to ask is:

> "How does a bad flight schedule look like? Is it even possible to produce a non-optimal flight schedule? How large can the gap between a bad and a good flight schedule possibly be?"

This section will be devoted to investigations into "bad" slot configurations, so called *slot covers*, while Section 3.6 will deal with the question of how to avoid such covers in the flight scheduling process. All of the above questions naturally call for an answer in the context of slot configurations, i. e., with no actual flight requests considered. If we know how a bad slot configuration looks like, producing an example with suitable flight requests that would actually result in such a schedule is then an easy task.

First, let us make the notion of a "bad" flight schedule more precise. A flight schedule that is not maximum, but that would allow for additional flights (hence non-optimality is simply due to lack of suitable demand) would not necessarily be a bad thing, so the precise question is "Are there non-maximum slot configurations such that no more flights can be added to them?"

Definition 3.37 (Slot Cover)
Let $\mathcal{S} = \{1, \ldots, n\}$ be a slot set and $\mathcal{R}$ a reference value system on $\mathcal{S}$. A movements slot configuration $C^{\mathrm{M}} : \mathcal{S} \to \mathbb{N}_0$ is said to *cover* a slot $t \in \mathcal{S}$ with respect to $\mathcal{R}$ if there is a time window bound $(L, b)^{(\sigma)} \in \mathcal{R}$ and $W \in \mathcal{W}((L, b)^{(\sigma)}, t)$ (cf. Definition 3.30) such that

$$C^{\mathrm{M}}(W) = b^{\mathrm{M}}.$$

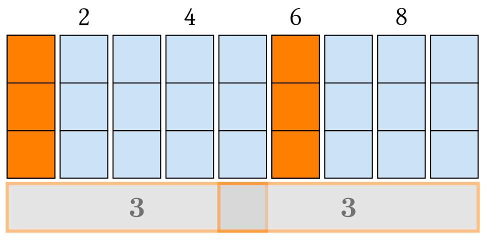
A slot configuration is called a *slot cover* or *covering slot configuration* if it is feasible with respect to $\mathcal{R}$ and covers each slot in $\mathcal{S}$.

Notice that we require a slot to be covered by a movements time window bound, thus a slot that does not allow for an additional departure, but could accommodate one more arrival or vice versa, is not covered. In this section, our primary interest is in identifying structures that lead to bad flight schedules, and for a symmetric reference value system, especially in a real-world context, one would expect the movements bounds to be decisive for slot covering configurations. Hence we concentrate solely on movements in this section, thereby avoiding some "exotic cases" that can be introduced by considering also arrival and departure bounds. Also, we will generally assume that all reference value systems are symmetric and denote time window bounds $\big(L, (b^{\mathrm{A}}, b^{\mathrm{D}}, b^{\mathrm{M}})\big)^{(\sigma)}$ by just $\big(L, b^{\mathrm{M}}\big)^{(\sigma)}$ for the rest of this section, as only the movements value is relevant for a slot cover. Thus if not explicitly stated otherwise, a time window bound $(L, b)^{(\sigma)}$ is to be read as $(L, (b, b, b)^{T})^{(\sigma)}$ in this section and the next.

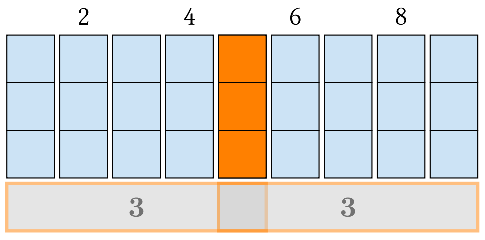
The question above can now be rephrased as the query for a slot cover that minimizes the number of flight movements.

Problem 3.38: Minimum Slot Cover
Instance: A slot count $n \in \mathbb{N}$ and a reference value system $\mathcal{R}$ on the slot set $\mathcal{S} = \{1, \dots, n\}$.
Question: Find a slot cover $C^{\mathrm{M}} : \mathcal{S} \to \mathbb{N}_0$ with respect to $\mathcal{R}$ that minimizes the number of flight movements $C^{\mathrm{M}}(\mathcal{S})$.

See Figure 3.9 for an illustration of a slot cover for the reference value system $\mathcal{R} = \{([5]_\circ, 3)\}$. The figure shows a maximum slot packing with six flight movements in two different slots in Figure 3.9a, and a minimum slot cover with three flight movements in one slot in Figure 3.9b. In both cases, no more flights can be added without violating the shifting bound. Similar situations are possible for more than one shifting bound.



(a) a maximum slot packing



(b) a minimum slot cover

Figure 3.9: Example of a minimum slot cover with three movements versus a maximum slot packing with six movements. The single shifting bound $([5]_\circ, 3)$ is applied in this example on the slot set $\mathcal{S} = \{1, \dots, 9\}$.

Minimum Slot Cover is interesting under two different aspects: On the one hand, a solution to Minimum Slot Cover exhibits a lower bound on the number of flights that can be handled at an airport for any given planning horizon and reference value system; so together with Maximum Slot Packing we get an estimate of how much can be lost by implementing a non-optimal flight schedule. This is also highly useful when one is interested in finding a reference value system that does not permit grave errors in the sense of exhibiting not too large a gap between an optimal flight schedule and any non-optimal solution. We will evaluate reference value systems that avoid such a gap in a more realistic setting in Chapter 4.

On the other hand, Minimum Slot Cover is a problem regularly faced by large airlines at their home airport. Every airline considers one airport (or several airports, for very large airlines) their home airport, a notion that commonly correlates with a high number of in- and outgoing flights for that particular airline (e. g., Lufthansa has their home airports at Frankfurt/Main and Munich airport). Thus a large airline operates a major fraction of all flights taking place at their home airport, which gives them the opportunity to control a major fraction of the available slots. Naturally, an airline is not only interested in servicing as much profitable flights as possible, but also in blocking its competitors from doing the same, if they get the opportunity. So the question of Minimum Slot Cover arises in this “unfriendly” context: How many and which slots would an airline need to acquire in order to block competitors from entering the market? Of course, for competition to function properly, government and airport authorities are interested in detecting and preventing such "blocking behavior". Knowledge about the structure of minimum slot covers can help in this task and in creating reference value systems (or other additional constraints) that avoid such situations.

3.5.2 Integer Programming Formulation

In a sense, Minimum Slot Cover is dual to Maximum Slot Packing — while the latter problem aims at "blocking" a slot set with as many flights as possible, the former asks for a minimal set of flights that do the same. However, this duality does not correspond to classical LP-duality (not even when we ignore the integer variable requirements), although there is some relationship. To clarify the relations between Maximum Slot Packing and Minimum Slot Cover, we will briefly look at an appropriate integer programming model here. For a more concise presentation, we restrict ourselves to shifting bounds here; integration of non-shifting bounds into the model is straightforward.

Recall the LP relaxation of the integer programming formulation for Maximum Movements Slot Packing with respect to the reference value system $\mathcal{R} = \{(L_1, b_1), \dots, (L_k, b_k)\}$ (with only shifting bounds) on the slot set $\mathcal{S} = \{1, \dots, n\}$ (cf. Section 3.3.1):

$$\begin{array}{lll} \max & \mathbb{1}^T x & \\ \text{s. t.} & x(s + [L_i]_\circ) \le b_i^{\mathrm{M}} & \text{for all } s \in \mathcal{S} \text{ and all } (L_i, b_i) \in \mathcal{R} \\ & x \ge 0 & \end{array}$$

The LP-dual of this problem is

$$\begin{array}{lll} \min & \displaystyle\sum_{i=1}^{k} \left(b_i^{\mathrm{M}} \cdot \sum_{s=1}^{n} y_{is} \right) & \\ \text{s. t.} & \displaystyle\sum_{i=1}^{k} \left(\sum_{s=t-L_i}^{t} y_{is} \right) \ge 1 & \text{for all } t = 1, \dots, n \\ & y \ge 0, & \end{array}$$

where we used the fact that for a shifting bound (L_i, b_i)

$$t \in s + [L_i]_\circ \quad \Leftrightarrow \quad s \in t - [L_i]_\circ \,.$$

If one interprets y_{is} as indicator variable for the i-th shifting bound with time window starting at slot s (which is not quite valid, because y_{is} is not necessarily a binary variable), the constraints bear some similarity to Minimum Slot Cover. Alas, the objective function does not count flight movements (which is impossible, because there are no variables corresponding to flight movements in the dual), but rather "active" shifting bounds time windows weighted by their respective bound value.

In fact, a proper formulation of Minimum Slot Cover as integer programming problem needs to incorporate aspects from both primal and dual Maximum Slot Packing. We propose the following formulation:

$$\begin{aligned}
\min \quad & \mathbb{1}^T x && && (3.26)\\
\text{s.t.} \quad & x(s + [L_i]_\circ) \le b_i^{\mathrm{M}} && \text{for all } s = 1, \dots, n \text{ and all } i = 1, \dots, k && (3.27)\\
& x(s + [L_i]_\circ) \ge y_{si} \cdot b_i^{\mathrm{M}} && \text{for all } s = 1, \dots, n \text{ and all } i = 1, \dots, k && (3.28)\\
& \sum_{i=1}^{k} \left(\sum_{s=t-L_i}^{t} y_{is} \right) \ge 1 && \text{for all } t = 1, \dots, n && (3.29)\\
& x \in \mathbb{N}_0^n && && (3.30)\\
& y \in \{0,1\}^{nk}
\end{aligned}$$

This program combines elements from primal (the packing constraint (3.27) and the objective (3.26), changed to minimization) and dual (the covering constraint (3.29)) of Maximum Slot Packing and links them via (3.28).

3.5.3 Slot Cover with a Single Shifting Bound

Let us first investigate slot cover with only one shifting bound (L, b). Besides giving a general idea about how minimum slot covers can look like, this situation fairly accurately represents the case of multiple shifting bounds where only one bound has an actual influence on the number of flight movements in a maximum slot packing or a minimum slot cover (a situation not uncommon in practice, also see Section 3.4.2). An illustration of the situation is displayed in Figure 3.9b.

Theorem 3.39
Let (L, b) be a shifting bound, let $n = 2L - 1$ and denote the slot set by $\mathcal{S} = \{1, \dots, n\}$. Then the size of a maximum slot packing on $\mathcal{S}$ with respect to $\mathcal{R} = \{(L, b)\}$ is $2b$, and the size of a minimum slot cover on $\mathcal{S}$ is b. Furthermore, for $n' > n$, any slot cover of $\{1, \dots, n'\}$ has size strictly greater than b.

Proof. Consider the slot configurations $P, C : \mathcal{S} \to \mathbb{N}_0$ defined by

$$P(t) := \begin{cases} b, & \text{for } t \in \{1, L+1\}, \\ 0, & \text{for } t \notin \{1, L+1\}, \end{cases} \quad \text{and} \quad C(t) := \begin{cases} b, & \text{for } t = L, \\ 0, & \text{for } t \neq L. \end{cases}$$

Clearly, $P(s + [L]_\circ) = b$ for every $s \in \{1, \dots, n - L + 1\}$, so P is a feasible slot configuration. On the other hand, P is maximum, because the time windows $1 + [L]_\circ$ and $(n - L + 1) + [L]_\circ$ can accommodate for a maximum of b movements each, and these two time windows contain all slots in $\mathcal{S}$, thus no slot configuration can have a size of more than $2b$ movements under the shifting bound (L, b).

For the slot configuration C, we get $C(t + [L]_\circ) = b$ for all $t \in \{1, \ldots, n - L + 1\}$, thus C is a slot cover. On the other hand, a slot configuration $C^<$ with less than b movements cannot be a slot cover, because that would imply $C^<(t + [L]_\circ) < b$ for each $t \in \{1, \ldots, n - L + 1\}$, and thus no slot in $\mathcal{S}$ could be covered by $C^<$, proving minimality of C.

Finally, consider $n' > n$ and suppose there was some slot cover $C' : \{1, \ldots, n'\} \to \mathbb{N}_0$ of size b or less. Let $t_0 \in \{1, \ldots, n'\}$ be a slot with $C'(t_0) > 0$, then

$$S' := \{1, \ldots, n'\} \setminus \big([t_0 - L + 1,\, t_0] \cup [t_0,\, t_0 + L - 1]\big)$$

is not empty. Let $s_0 \in S'$ be a slot in S', then s_0 is only contained in time windows that do not contain the slot t_0, thus $C'(t + [L]_\circ) \le b - 1$ for each $t \in \{1, \ldots, n'\}$ such that $s_0 \in t + [L]_\circ$. Hence s_0 is not covered by C', a contradiction. □

3.5.4 Slot Cover with two Shifting Bounds

The Minimum Slot Cover problem with two shifting bounds is similar to the problem with one bound. The general idea for defining a slot cover is the same: Produce a slot configuration with a local peak of flight movements and make that peak as narrow as possible. The problem with this approach is that the smaller shifting bound will generally not allow for an arbitrarily narrow peak, so we have to take that aspect into account.

Theorem 3.40
Let $\mathcal{R} := \{(L', b'); (L, b)\}$ be a monotone, symmetric reference value system with $L' < L$ and let $q \in \mathbb{N}_0$, $q \le L$ such that there exists a slot configuration $P' : \{1, \ldots, q\} \to \mathbb{N}_0$ of size b^{M} that is feasible with respect to (L', b'). Then there is a slot cover of size b^{M} with respect to $\mathcal{R}$ on the slot set $\mathcal{S} = \{1, \ldots, 2L - q\}$.

Furthermore, if P' is circular feasible with respect to $\{(L', b')\}$ or if $q + L' \le L + 1$ there is also a feasible slot configuration of size $b^{\mathrm{M}} + P'([1,\, \min\{q, L - q\}])$. In particular, if $L \ge 2q$ then there is a maximum slot packing of size $2b^{\mathrm{M}}$.

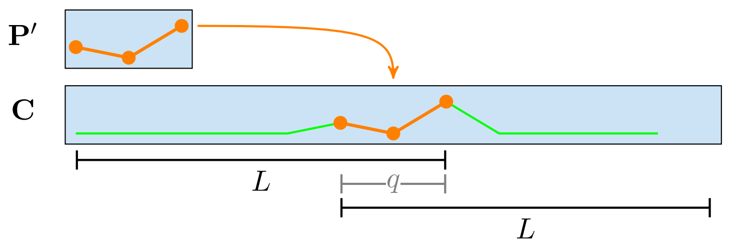
Proof. For q and $P' : \{1, \ldots, q\} \to \mathbb{N}_0$ as required by the theorem define the slot configuration $C : \mathcal{S} \to \mathbb{N}_0$ (see Figure 3.10 for an illustration) by

$$C(t) := \begin{cases} P'(t - (L - q)), & \text{for } t \in \{L - q + 1, \ldots, L\}, \\ 0, & \text{for } t \notin \{L - q + 1, \ldots, L\}. \end{cases}$$

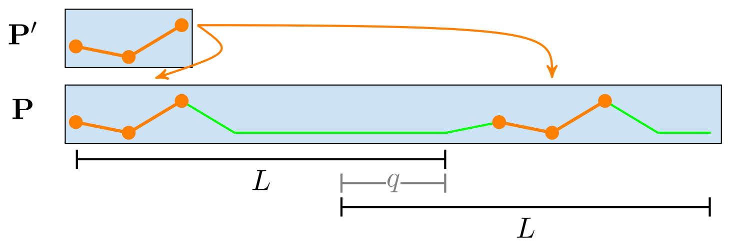
The union of the two time windows $1 + [L]_\circ$ and $(L - q + 1) + [L]_\circ$ is the whole slot set $\mathcal{S}$, and

$$C(1 + [L]_\circ) = C\big((L - q + 1) + [L]_\circ\big) = b^{\mathrm{M}},$$

thus C covers every slot in $\mathcal{S}$. Furthermore, C is feasible with respect to (L', b') by feasibility of P', and it is also feasible with respect to (L, b), as it has size b^{M}. Hence C is a slot cover for the slot set $\mathcal{S}$ as desired.



(a) The slot cover C



(b) The slot packing P

Figure 3.10: Illustrations for the proof of Theorem 3.40.

Now consider the slot configuration $P : \mathcal{S} \to \mathbb{N}_0$ defined by (see Figure 3.10)

$$P(t) := \begin{cases} P'(t), & \text{for } t \in \{1, \dots, q\}, \\ P'(t-L), & \text{for } t \in \{L+1, \dots, \min\{L+q, n\}\}, \\ 0, & \text{for } t \notin \{1, \dots, q\} \cup \{L+1, \dots, \min\{L+q, n\}\}. \end{cases}$$

Clearly, P has size $b^{\mathrm{M}} + P'([1,\ \min\{q, L-q\}])$, as $n - L = L - q$. P is a feasible slot configuration with respect to the shifting bound (L, b): Consider a time window $s + [L]_\circ$ for $s \in \mathcal{S}$. If $s = 1$ or $s \geq q+1$ then the time window intersects only one of the intervals $[1,\ q]$ and $[L+1,\ \min\{L+q, n\}]$, hence $P(s + [L]_\circ) \leq b^{\mathrm{M}}$. So suppose $2 \leq s \leq q$, then by definition of P' we have

$$(s + [L]_\circ) \cap ([1,\ q] \cup [L+1,\ \min\{L+q, n\}]) = [s,\ q] \cup [L+1,\ \min\{L+s, n\}]\,,$$

hence $P(s + [L]_\circ) \leq P'([1,\ q]) = b^{\mathrm{M}}$. Also, if P' is circular feasible then P is feasible with respect to the shifting bound (L', b'); and if $q + L' \leq L + 1$ then there is no time window of length L' that contains slots in both $[1,\ q]$ and $[L+1,\ \min\{L+q, n\}]$, also yielding feasibility of P with respect to (L', b'). Thus P is a slot packing of the desired size.

If, in particular, $L \geq 2q$ then $\min\{q, L-q\} = q$ and $q + L' < 2q \leq L$, hence $q + L' \leq L + 1$. That means P is feasible with respect to $\mathcal{R}$ and has size $2b^{\mathrm{M}}$. Furthermore, the time windows $1 + [L]_\circ$ and $L - q + 1 + [L]_\circ$ each bound the number of flights movements within their respective domain by b^{M}, and their union is $\mathcal{S}$. Therefore there can be no slot packing of size larger than $2b^{\mathrm{M}}$ in $\mathcal{S}$, meaning that P is a maximum slot packing. □

Notice that in Theorem 3.40 we did not claim that the slot cover $C : \mathcal{S} \to \mathbb{N}_0$ of size b^{M} is a minimum cover. While the results for one shifting bound would suggest such a conjecture, the situation is more difficult for two shifting bounds.

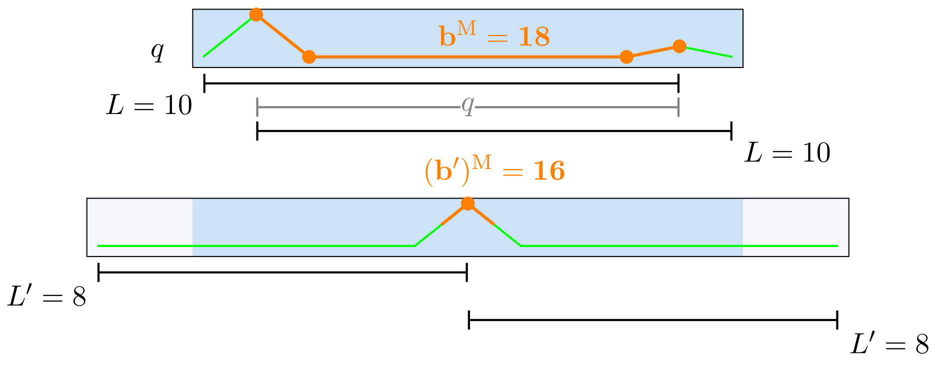

q
$\mathrm{b}^{\mathrm{M}} = 18$
$L = 10$
q
$L = 10$
$(\mathrm{b}')^{\mathrm{M}} = 16$
$L' = 8$
$L' = 8$

Figure 3.11: The slot cover of Theorem 3.40 need not be a minimum slot cover.

Example 3.41
Consider the example of $\mathcal{R} = \{([10]_\circ, 18); ([8]_\circ, 16)\}$ (only movement bounds values are given), which is clearly monotone. A slot packing $P' : \{1, \ldots, q\} \to \mathbb{N}_0$ is, e. g., given by $P' = (16, 0, 0, 0, 0, 0, 0, 0, 2)$ for $q = 9$. As eight slots can only accommodate up to 16 flight movements, $q = 9$ is even the least possible value for q. This yields a slot cover of $2 \cdot 10 - q = 11$ slots by 18 flight movements. However, $C' = (0, 0, 0, 0, 0, 16, 0, 0, 0, 0, 0)$ is also a slot cover (this time, all slots are covered by time windows of length 8) of the slot set $\{1, \ldots, 11\}$ using only 16 movements (and the same idea can be extended to a slot cover of $\{1, \ldots, 15\}$ using 16 movements), thus the slot cover constructed in the proof of Theorem 3.40 is obviously not a minimum cover in this situation. $\diamondsuit$

However, in some cases the slot cover of Theorem 3.40 is actually a minimum slot cover. A class of instances where this happens is somehow "in between" the cases of a single shifting bound and two shifting bounds, namely the case where one of the two bounds has a length of just one slot.

Theorem 3.42
Let $B \in \mathbb{N}$, (L, b) be a shifting bound with $^{b^{\mathrm{M}}}/_{L} \leq B$ and consider the (monotone) reference value system $\mathcal{R} := \{([1]_\circ, B); ([L]_\circ, b)\}$ on the slot set $\mathcal{S} = \{1, \ldots, 2L - q\}$, where $q = \lceil ^{b^{\mathrm{M}}}/_{B} \rceil$. Then the size of a minimum slot cover on $\mathcal{S}$ with respect to $\mathcal{R}$ is b^{M}, and for $n' > 2L - q$, any slot cover of $\{1, \ldots, n'\}$ has size strictly greater than b^{M}. Furthermore, if $L \geq 2q$, the size of a maximum slot packing on $\mathcal{S}$ with respect to $\mathcal{R}$ is $2b^{\mathrm{M}}$.

Proof. First notice that $\mathcal{R}$ is indeed monotone due to the requirement $^{b^{\mathrm{M}}}/_{L} \leq B$, which also implies $L \geq q = \lceil ^{b^{\mathrm{M}}}/_{B} \rceil$, as $L \in \mathbb{N}$. Define a slot configuration $C : \mathcal{S} \to \mathbb{N}_0$ (similar to the illustrations in Figure 3.10) by

$$C(t) := \begin{cases} B, & \text{for } t \in [L-q+1,\, L-1], \\ b^{\mathrm{M}} - (q-1)B, & \text{for } t = L, \\ 0, & \text{otherwise.} \end{cases}$$

Then C has size b^{M} and is clearly feasible with respect to $\mathcal{R}$. Furthermore, as

$$C(1 + [L]_\circ) = C((L-q+1) + [L]_\circ) = b^{\mathrm{M}}$$
$$\text{and } (1 + [L]_\circ) \cup ((L-q+1) + [L]_\circ) = \mathcal{S},$$

the slot configuration C is a slot cover. Suppose C is not a minimum slot cover, then there exists a slot cover $G : \mathcal{S} \to \mathbb{N}_0$ of size $b^{\mathrm{M}} - 1$ or less. This means that no slot is covered by a time window bound of length L, thus $G(t) = B$ for each slot $t \in \mathcal{S}$. But then $G(1 + [L]_\circ) = LB \geq b^{\mathrm{M}}$, contradicting our assumption about the size of G. Thus C is a minimum slot cover.

Now consider $n' > 2L - q$ and suppose there was some slot cover $C' : \{1, \ldots, n'\} \to \mathbb{N}_0$ of size b^{M}. Choose $t_0, t_1 \in \mathcal{S}$ with $t_0 \leq t_1$ such that $C'([t_0,\, t_1]) = b^{\mathrm{M}}$, and such that $C'([t_0+1,\, t_1]) < b^{\mathrm{M}}$ and $C'([t_0,\, t_1 - 1]) < b^{\mathrm{M}}$; then $|[t_0,\, t_1]| \geq q$. For a slot cover, every slot in $\{1, \ldots, n'\}$ must be contained in some time window of length L that completely contains $[t_0,\, t_1]$. Consider a slot

$$s_0 \in \{1, \ldots, n'\} \setminus ([t_1 - L + 1,\, t_1] \cup [t_0,\, t_0 + L - 1]),$$

which exists due to $n' - (2L - q) > 0$. Then s_0 is only contained in time windows that do not contain $[t_0,\, t_1]$ as a subset, and hence $C'(s_0 + [L]_\circ) \leq b^{\mathrm{M}} - 1$, contradicting the assumption of C' being a slot cover. Hence every slot cover on $\{1, \ldots, n'\}$ must contain at least $b^{\mathrm{M}} + 1$ movements.

For the slot packing, let $L \geq 2q$ and define a slot configuration $P : \mathcal{S} \to \mathbb{N}_0$ by (for illustrations, see the similar situation in Figure 3.10)

$$P(t) := \begin{cases} B, & \text{for } t \in [1,\, q-1] \cup [L+1,\, L+q-1], \\ b^{\mathrm{M}} - (q-1)B, & \text{for } t \in \{q, L+q\}, \\ 0, & \text{otherwise.} \end{cases}$$

As $L \geq 2q \Rightarrow 2L - q \geq L + q$, all flight movements defined by P are indeed assigned to slots in $\mathcal{S}$, so P has size $2b^{\mathrm{M}}$. Furthermore, for each $s \in \{1\} \cup \{q+1, \ldots, 2L-q\}$ we have $P(s + [L]_\circ) \leq b^{\mathrm{M}}$, and for each $s \in \{2, \ldots, q\}$ we have

$$P(s + [L]_\circ) = b^{\mathrm{M}} - (q-1)B + (q-s)B + (s-1)B = b^{\mathrm{M}},$$

hence P is feasible with respect to $\mathcal{R}$ (notice $P(t) \leq B$ for all $t \in \mathcal{S}$). Furthermore, the union of the time windows $1 + [L]_\circ$ and $(L-q+1) + [L]_\circ$ is the whole slot set $\mathcal{S}$, and the shifting bound (L, b) allows for a maximum of b^{M} movements within each of these time windows, thus P is a maximum slot packing. □

Remark 3.43
Theorem 3.42 can be slightly generalized to monotone reference value systems $\mathcal{R} = \{(L', b'); (L, b)\}$, where L is an integer multiple of L'. In such a case, by simply regarding an interval of length L' as *one* slot and scaling down L appropriately we get the reference value system $\mathcal{R}' = \{(1, b'); (L/L', b)\}$, where Theorem 3.42 can be applied to the slot set $\mathcal{S}' := \{1, \dots, 2L/L' - q\}$ with $q = \lceil b^{\mathrm{M}}/(b')^{\mathrm{M}} \rceil$. To "scale up" the result to the original system afterwards, note that each slot containing $B = b'$ movements can be replaced by a circular slot packing of size $(b')^{\mathrm{M}}$ on L' slots, see Lemma 3.34 and Theorem 3.35, which retains feasibility of the scaled up versions of the configurations C and P in the above proof with respect to $\mathcal{R}$. Thus the results of Theorem 3.42 are, appropriately scaled, also true for a setting with two shifting bounds where the larger bound's length is an integer multiple of the shorter one's.

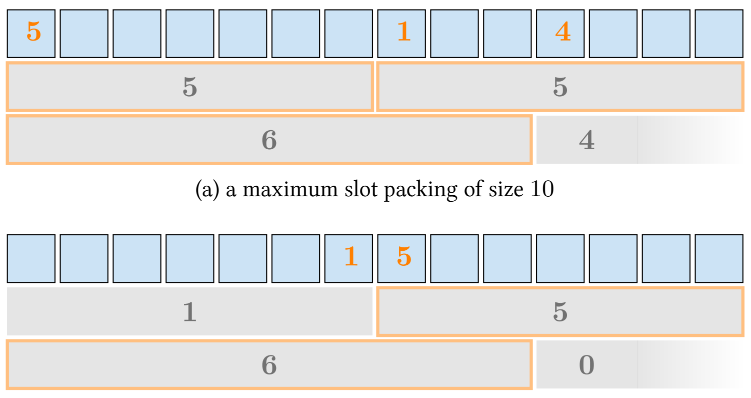
3.6 Avoiding the Gap between Slot Packing and Slot Cover

The Minimum Slot Cover problem is in some sense dual to Maximum Slot Packing. The difference between a slot packing and a slot covering configuration (if there is any difference) provides for a measure of how much one can possibly gain by optimization under the given circumstances, we will refer to that difference as the *packing-covering gap* in the following. From a different point of view, one could also express this question as "How much can be lost by employing some flight planning procedure that does not necessarily yield an optimum?" Of course, this question will at once be followed by "How can one avoid ending up with non-optimal solutions (or at least enforce solutions that are close to optimal)?" An answer to this question can be particularly valuable in an online scheduling context, where not all requests are known in advance. In such a situation, one can produce an optimal schedule for the known requests that might turn out to be a rather bad choice when the complete request set is revealed afterwards. But if the constraint set would not allow for non-optimal flight plans, such a problem could never emerge. This setting does have some practical relevance, although the impact is limited, as sometimes an airline decides to offer a new connection amidst a planning season, and then requests a suitable slot pair.

So the focus of this section, in contrast to the rest of this chapter, is not finding an optimal flight schedule subject to the given constraints, but to modify the constraints in such a way that any feasible schedule will be optimal, or can at least be extended to an optimal schedule. This might seem a little strange at first sight — if one knows how to get to an optimum, why not just apply that knowledge and thus solve the problem? However, this is sometimes not the easiest solution from a practical point of view. After all, implementing an optimal solution to the flight planning problem would require major changes for all parties involved, new procedures would have to be developed, new software would have to be implemented on the existing systems. In contrast to that, minor adjustments of the constraints (in this case, the time window bounds) do occur on a very regular basis (e. g., the flight movements bound values are adjusted at least once a year). Hence all systems and all parties involved in the planning process are prepared for such small adjustments, meaning a small change of the reference value system could be implemented without too many (potential) problems. Another motivation for changing the constraint system could be the suppression of uncooperative behavior on the part of some airlines. As was outlined in Section 3.5, large airlines may have the potential of “blocking” a fair amount of slots by placing their requests such as to form a configuration close to being a slot cover. This way, some of the airline’s competitors can be barred from offering services at an airport, much to the airport’s (and the customers’) disadvantage. Thus an airport will not only be interested in detecting such “blocking behavior”, but also in constraints that inhibit it.

If one tries to sum up the results of the previous sections in one “rule of thumb”, this could be “For a maximum slot packing, aim at a uniform distribution of the flight movements and try to avoid narrow local peaks in the slot configuration.” Of course, this is not the whole truth, but as a guideline, this rule will turn out to be quite effective, even in practical settings with many more constraints for the scheduling process. So to avoid (or at least reduce) the gap between a maximum slot packing and a minimum slot cover, a modification of the rules is in order that avoids narrow local peaks or the negative effects (“spreading out”) of these peaks. In addition, such a modification should not be too complicated, so that it may be integrated in more complex scenarios without too much modifications to existing models and procedures.

With Section 3.5 in mind, one might first think of resorting to non-shifting bounds only, i. e., replacing all shifting bounds with non-shifting bounds with the same bound values. However, this approach does not work in general. One problem here is due to boundary effects. Consider Figure 3.12 for an example, where a maximum slot packing and a minimum slot cover for two non-shifting bounds are depicted. In this example, the fact that n is not the least common multiple (or a multiple thereof) of the two bound lengths involved, is exploited in order to create an “open boundary” that is in turn responsible for a positive packing-covering gap.

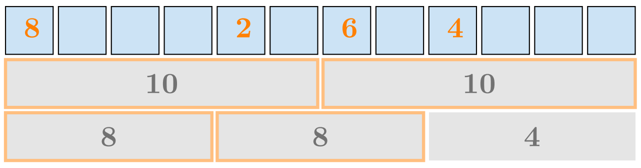


(a) a maximum slot packing of size 10

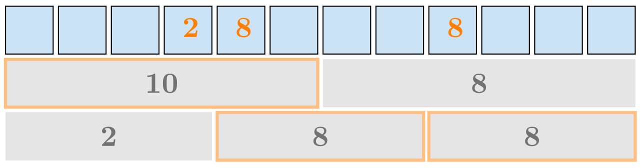
(b) a minimum slot cover of size 6

Figure 3.12: Example of boundary effects causing a positive packing-covering gap even for non-shifting bounds with the reference value system $\mathcal{R} = \left\{([10]_\circ, 6)^{(10)}; ([7]_\circ, 5)^{(7)}\right\}$ and $n = 14$.

However, this is not the only issue. Even if we restrict ourselves to monotone reference value systems $\mathcal{R}$ and define n to be some common multiple of all bound lengths in $\mathcal{R}$ (thus avoiding boundary effects as shown in Figure 3.12), a positive packing-covering gap can occur. Figure 3.13 shows an example of such a situation. Here, an effect similar to situations of shifting bounds occurs (cf. Section 3.5): Instead of two shifting bounds of the same length overlapping on a common "local traffic peak", the overlap is now provided by two non-shifting bounds of different length. Two time windows, namely $1 + [6]_\circ$ and $5 + [4]_\circ$, both contain slot number 5, and thus 8 flight movements. Thus very few flights suffice to make both these windows active and cover a large amount of the slot set. (The situation is similar to the "local peaks" used to construct slot covers in Section 3.5.)



(a) a maximum slot packing of size 20



(b) a slot cover of size 18

Figure 3.13: Example of a packing-covering gap when larger bound lengths are not a multiple of the next one's length down. The reference value system used here is $\mathcal{R} = \left\{([6]_\circ, 10)^{(6)}; ([4]_\circ, 8)^{(4)}\right\}$, and the slot count $n = 12$ is the least common multiple of 6 and 4.

These observations motivate the following definition.

Definition 3.44
A reference value system $\mathcal{R} = \left\{(L_1, b_1)^{(\sigma_1)}, \dots, (L_k, b_k)^{(\sigma_k)}\right\}$ with $L_1 \leq L_2 \leq \dots \leq L_k$ is said to have the *inclusion property*, if for each $j \in \{1, \dots, k-1\}$ the number L_{j+1} is an integer multiple of L_j.

Reference value systems having the inclusion property are not uncommon in practical applications. Often, a system with lengths 60 minutes, 30 minutes and 10 minutes or 60 minutes and 20 minutes is used,[3] these all have the inclusion property.

Theorem 3.45
Let $\mathcal{R} = \left\{(L_1, b_1)^{(L_1)}, \dots, (L_k, b_k)^{(L_k)}\right\}$ be a monotone reference value system that consists of only non-shifting bounds and has the inclusion property, and let $n \in \mathbb{N}$ be an integer multiple of $\max\left\{L_i : (L_i, b_i)^{(L_i)} \in \mathcal{R}\right\}$. Then the size of a maximum slot packing with respect to $\mathcal{R}$ on $\mathcal{S} = \{1, \dots, n\}$ is equal to the size of a minimum slot cover.

[3]For parameters actually used in practice see Chapter 4 and `http://sws.fhkd.org/`, the website of the German airport coordinator.

Proof. For easier notation, define for $j \in \{1, \ldots, k\}$ the "start slots set" of the time window bound $(L_j, b_j)^{(L_j)}$ by $S_j := \{1, L_j + 1, \ldots, n - L_j + 1\}$.

We may assume $L_1 \leq \cdots \leq L_k$, and define $r = {}^{n}/_{L_k} \in \mathbb{N}$. According to Section 3.4, in particular Theorem 3.35, a feasible slot configuration of size $r \cdot b_k^{\mathrm{M}}$ exists on the slot set $\mathcal{S} = \{1, \ldots, n\}$. Due to the non-shifting constraints, a maximum of b_k^{M} flight movements can be contained in each of the time windows $t + [L_k]_\circ$ for $t \in \{1, L_k + 1, \ldots, (r-1)L_k + 1\}$, hence a slot configuration that contains rb_k^{M} movements is a maximum slot packing.[4]

Now let $C : \mathcal{S} \to \mathbb{N}_0$ be a minimum slot cover. For $s \in S_k$ choose $j \in \{1, \ldots, k-1\}$ and $s' \in S_j$ such that $(s' + [L_j]_\circ) \cap (s + [L_k]_\circ) \neq \emptyset$, then $(s' + [L_j]_\circ) \subset (s + [L_k]_\circ)$ due to the non-shifting nature of the time window bounds in $\mathcal{R}$ and due to the inclusion property. Thus the restriction of C to each of the time windows $s + [L_k]_\circ$, $s \in S_k$, also has to be a slot cover with respect to $\mathcal{R}$ on the slot set $s + [L_k]_\circ$. Hence it suffices to proof the claim of the theorem for $n = L_k$.

Let $\mathcal{S}' := \{1, \ldots, L_k\}$ and let $C' : \mathcal{S}' \to \mathbb{N}_0$ be a minimum slot cover with respect to $\mathcal{R}$ on the slot set $\mathcal{S}'$. Naturally, the size of C' is at most b_k^{M}, and we have argued above that a slot configuration of that size exists and is, in fact, a maximum slot packing. It remains to show that C' cannot have a size less than b_k^{M}. To see this, we use induction on $j = 1, \ldots, k$ to show the following statement: If a slot configuration $C' : \mathcal{S}' \to \mathbb{N}_0$ covers all slots in $\{s+1, \ldots, s+L_j\}$ with respect to the reference value system $\mathcal{R}_j := \left\{(L_1, b_1)^{(L_1)}, \ldots, (L_j, b_j)^{(L_j)}\right\}$ for some $s \in S_j \cap \mathcal{S}'$, then $C'(s + [L_j]_\circ) = b_j^{\mathrm{M}}$.

For $j = 1$, there is only one non-shifting bound in $\mathcal{R}_1$, hence $C'(s + [L_1]_\circ) = b_1^{\mathrm{M}}$ for all $s \in S_1 \cap \mathcal{S}'$. Now consider some $j \in \{2, \ldots, k\}$ and assume the assertion is true up to $j-1$. Suppose there is some $s \in S_j \cap \mathcal{S}'$ with $C'(s + [L_j]_\circ) < b_j$, then the slots in $s + [L_j]_\circ$ must be covered with respect to the reference value system $\mathcal{R}_{j-1}$ already. Application of the induction hypothesis yields $C'(s' + [L_{j-1}]_\circ) = b_{j-1}^{\mathrm{M}}$ for all $s' \in \{s, s + L_{j-1}, \ldots, s + ({}^{L_j}/_{L_{j-1}} - 1)\, L_{j-1}\}$ by the inclusion property, and thus

$$C'(s + [L_j]_\circ) = {}^{L_j}/_{L_{j-1}} \cdot b_{j-1}^{\mathrm{M}} \geq b_j^{\mathrm{M}}$$

by monotonicity of $\mathcal{R}$ (and thus $\mathcal{R}_j$), a contradiction.

This asserts in particular that $C'(1 + [L_k]_\circ) = b_k$, thereby proving the statement of the theorem. □

Theorem 3.46
Consider a reference value system $\mathcal{R} = \{(L_1, b_1), \ldots, (L_k, b_k)\}$ consisting of only shifting bounds, and define

$$b^* := \min \left\{ \left\lfloor \frac{b_j}{L_j} \right\rfloor : (L_j, b_j) \in \mathcal{R} \right\}.$$

[4]This also shows that the size of a maximum slot packing does not increase by going from shifting to non-shifting constraints (which corresponds to removing many of the constraints). This is important in practical considerations, because by changing the constraints one will usually not want to allow for more flights in total.

Then with respect to the new reference value system $\mathcal{R}^* := \mathcal{R} \cup \{([1]_\circ, b^*)\}$, the size of a Maximum Slot Packing *is equal to the size of a* Minimum Slot Cover *for every slot set $\mathcal{S} = \{1, \dots, n\}$ where $n \in \mathbb{N}$.*

Proof. For each slot configuration $P : \mathcal{S} \to \mathbb{N}_0$ that is feasible with respect to $\{(1, b^*)\}$, each shifting bound $(L_j, b_j) \in \mathcal{R}$, and each $s \in \mathcal{S}$, the inequality

$$P(s + [L_j]_\circ) \leq b^* \cdot L_j \leq \frac{b_j}{L_j} L_j = b_j$$

holds, thus a slot configuration is feasible with respect to $\mathcal{R}^*$ if and only if it is feasible with respect to $\{(1, b^*)\}$. Hence no slot configuration on $\mathcal{S} = \{1, \dots, n\}$ that is feasible with respect to $\mathcal{R}^*$ can contain more than $n(b^*)^{\mathrm{M}}$ movements.

Now consider an arbitrary slot cover $C : \mathcal{S} \to \mathbb{N}_0$ on $\mathcal{S}$ with respect to $\mathcal{R}^*$, then $C(s) = (b^*)^{\mathrm{M}}$ for every $s \in \mathcal{S}$: Suppose there was some slot $s' \in \mathcal{S}$ with $C(s') < (b^*)^{\mathrm{M}}$ and consider any $(L_j, b_j) \in \mathcal{R}$ and a slot $t \in \mathcal{S}$ such that $s' \in t + [L_j]_\circ$. Then

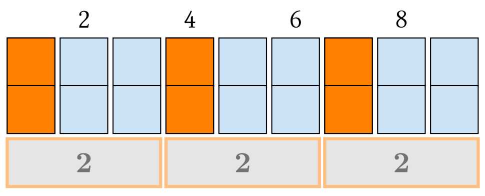
$$C(t + [L_j]_\circ) \leq (L_j - 1)(b^*)^{\mathrm{M}} + C(s') < L_j (b^*)^{\mathrm{M}} \leq b_j^{\mathrm{M}},$$

thus s' is not covered, contradicting our assumption of C being a slot cover. Therefore, every slot cover has size exactly $n(b^*)^{\mathrm{M}}$, which is equal to the size of a maximum slot packing. □

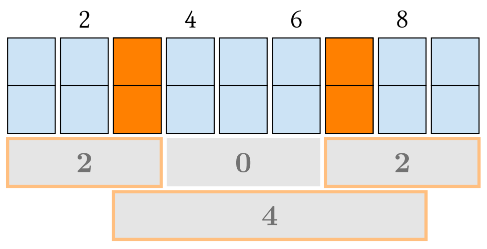
The modification suggested in Theorem 3.46 is basically to replace the complex interaction of several shifting bounds by one very simple non-shifting bound that ensures feasibility of the result with respect to the original reference value system while at the same time being as “loose” as possible. Usually, the new reference value system $\mathcal{R}^*$ will allow for less flight movements than the original system $\mathcal{R}$, but it will have the benefits of simplifying the optimization process and of not allowing for any blocking behavior by airlines. In Chapter 4, more precisely in Section 4.4.2, we will investigate the effects of this idea in a real-world setting and suggest a more realistic approach based on this result.

Remark 3.47
As above, a slight generalization seems natural: Consider a reference value system $\mathcal{R} = \{(L_1, b_1), \dots, (L_k, b_k)\}$ consisting of only shifting bounds as above, let $L^* := \gcd\{L_1, \dots, L_k\}$ and $b^* := \min\{\lfloor b_j/L_j \cdot L^* \rfloor\}$. As outlined above, one can then “scale down” the problem by contracting each of the sets $\{1, \dots, L_k\}, \{L_k + 1, \dots, 2L_k\}, \dots$ to *one* slot and then modify the time window bounds accordingly. By imposing an additional bound of $([1]_\circ, b^*)$ for each of these “new slots”, we are exactly in the situation required by the theorem, thus planning on the new system would result in no packing-covering gap. Of course, after the planning has been done, one would “scale back up” by replacing every “new slot” by L^* “old slots”, using a circular feasible slot configuration of suitable size for each of the intervals $\{1, \dots, L_k\}, \{L_k + 1, \dots, 2L_k\}, \dots$ (see outline above).

It is tempting to simplify that approach by defining L^* and b^* as above and then replace the original reference value system $\mathcal{R}$ by $\mathcal{R} \cup \left\{([L^*]_\circ, b^*)^{(L^*)}\right\}$ instead of scaling down. Unfortunately, this approach will generally not work, as is illustrated by the example in Figure 3.14. The figures show a maximum slot packing and a minimum slot cover with respect to the reference value system $\mathcal{R} = \left\{([3]_\circ, 2)^{(3)}, ([6]_\circ, 4), ([9]_\circ, 6)\right\}$, which would arise by applying the above procedure to $\{([6]_\circ, 4), ([9]_\circ, 6)\}$ (meaning contract the slots in groups of three, according to the bound of smallest length $([3]_\circ, 2)$). In the minimum slot cover shown in Figure 3.14b, no flight movements are placed in the middle group of three slots (fourth to sixth slot), while the first and last group of three slots each contain two flight movements. This certainly covers slots one to three and seven to eight using a time window of length three. But the slots four to six are also covered, namely by the time window $3 + [6]_\circ$, which contains all four flight movements in the slot configuration displayed. On the contrary, the maximum slot packing shown in Figure 3.14a contains six flight movements, thus a positive packing-covering gap exists in this example.



(a) a maximum slot packing of size 6



(b) a minimum slot cover of size 4

Figure 3.14: Maximum slot packing and minimum slot cover for the reference value system $\mathcal{R} = \left\{([3]_\circ, 2)^{(3)}, ([6]_\circ, 4), ([9]_\circ, 6)\right\}$, see Remark 3.47.

3.7 Concluding Remarks

In this chapter, we introduced the Flight Scheduling problem and the related optimization problems Maximum Slot Packing and Minimum Slot Cover. The first two problems aim at a flight schedule that integrates a maximum number of flight movements, while our studies of Minimum Slot Cover gave some insight into the structure of bad flight schedules. In combination, these considerations provide valuable information about the inherent mechanism governing Flight Scheduling problems.

Of course, the model we looked into in this chapter is still substantially simplified compared to the constraints that govern flight scheduling in real-world applications. Hence, to test our result for viability in practice, we will have to consider realistic instances and a much more complex mathematical model. We will do so in the following chapter, building upon the foundation laid out here.

Although some insight into complexity issues was provided in this chapter, there are so many variations of scheduling, packing and covering problems that we could not possibly investigate all of them. To conclude this chapter, let us briefly comment on one such variant that is somehow "in between" Flight Scheduling and Slot Packing and demonstrates the limitations of the matroid approach that can be used for certain subclasses of Flight Scheduling problems. More precisely, consider a situation where the "ground time", i. e., the time between an arrival and the subsequent departure for a flight, is a fixed amount of time $g \in \mathcal{S}$ for all flights, where $\mathcal{S} = \{1, \dots, n\}$ is a slot set.

Any choice of the ground time g induces a one to one mapping between a collection of slot requests and a corresponding collection of flight requests by considering G_i as an arrival slot request and associating with each slot request G_i the flight request $F_i := \Big\{(a, [\![a+g]\!]_{[n]}) : a \in G_i\Big\}$. For slot requests, we were able to obtain some complexity results via matroid theory, while for general flight requests the Flight Scheduling problem is known to be $\mathcal{NP}$-hard. Here, we formally deal with flight requests (i. e., slot pairs), but find them to be very similar to slot requests via a fixed ground time coupling. The problem can then be stated as:

Problem 3.48: Maximum Slot Packing with A/D Coupling
Instance: The number of slots $n \in \mathbb{N}$, a shifting bound (L, b) and a fixed "ground time" $g \in \mathbb{N}$.
Question: Find a slot configuration C on $\mathcal{S} = \{1, \dots, n\}$ that is feasible with respect to $\mathcal{R}$ and maximizes the number of flight movements $C^{\mathrm{M}}(\mathcal{S})$ subject to the constraints

$$\begin{aligned} C^{\mathrm{D}}(s) &= C^{\mathrm{A}}(s-g) && \text{for all } s \in \mathcal{S} \text{ where } s-g \in \mathcal{S}, \\ C^{\mathrm{D}}(s) &= C^{\mathrm{A}}(n+1-g) && \text{for all } s \in \mathcal{S} \text{ where } s-g \notin \mathcal{S}. \end{aligned}$$

In this setting it is natural to assume a circular slot set, i. e., a slot set $\mathcal{S} = \{1, \dots, n\}$ where n is a multiple of L (so we do not run into problems with circular non-shifting bounds, cf. the discussion in Section 3.1.2) and interpret a slot number $t > n$ as $[\![t]\!]_{[n]}$. In reality, this could either mean that the same flight pattern repeated itself over and over again, or we could think of a flight request $\{(a, d)\}$ with $d > a$, but $[\![d]\!]_{[n]} \leq a$, as a flight that stays at the airport for the night and leaves in the early morning. Now, only the bound b^{M} should be considered, because separate arrival/departure bounds are of no value in this situation. The notation used here is similar to that in Section 3.2.3. We will now show that — in spite of the similarities to the slot request considerations — the matroid approach used there (cf. Section 3.2.3) does not carry through.

For the flight request set $\mathcal{F} = \{F_1, \dots, F_m\}$ associated to the slot requests $\mathcal{G} = \{G_1, \dots, G_m\}$, define $F_i' := \{(i, a, d) : (a, d) \in F_i\}$ and for $t \in \mathcal{S}$ define the set of flight requests with arrival (departure) at slot t as

$$\begin{aligned} H_t^{\mathrm{A}} &:= \bigcup_{i=1,\dots,m} \{(i, t, d) : (t, d) \in F_i\} \\ H_t^{\mathrm{D}} &:= \bigcup_{i=1,\dots,m} \{(i, a, t) : (a, t) \in F_i\}, \end{aligned}$$

respectively. For a subset $S \subset \mathcal{S}$ of the slot set, we can define the set of flights arriving (departing) within the time window $s + [L]_\circ$ (cf. Section 3.2.3) by

$$(H^{\mathrm{A}})'_s := \bigcup_{[\![t \in s+[L]_\circ]\!]_{[n]}} H^{\mathrm{A}}_t \quad \text{for } s \in S$$
$$(H^{\mathrm{D}})'_s := \bigcup_{[\![t \in s+[L]_\circ]\!]_{[n]}} H^{\mathrm{D}}_t \quad \text{for } s \in S.$$

A natural ground set is then

$$\mathcal{B}_{\mathcal{F}} := \bigcup_{i=1,\dots,m} F'_i,$$

and the set

$$\mathcal{M}^{\infty}_{(L,b);g} := \left\{ B \subset \mathcal{B}_{\mathcal{G}} : \left|B \cap (H^{\mathrm{A}})'_s\right| + \left|B \cap (H^{\mathrm{D}})'_s\right| \leq b^{\mathrm{M}} \text{ for all } s \in S \right\}$$

is an independence system over $\mathcal{B}_{\mathcal{F}}$.

Unfortunately, $\mathcal{M}^{\infty}_{(L,b);g}$ is not a matroid. To see this, we exhibit an example demonstrating that the matroid exchange property (cf. Theorem 1.5) does not hold for $\mathcal{M}^{\infty}_{(L,b);g}$. Consider a non-shifting bound $(L,b)^{(L)}$ such that $n = 2L$, $g < L$ and $b^{\mathrm{M}} = 2k$ is even and define a flight request set $\mathcal{F} = \{F_1, \dots, F_{4k-1}\}$ by

$$\begin{aligned} F_1 := \cdots := F_k &:= \{(1, 1+g)\}, \\ F_{k+1} := \cdots := F_{2k} &:= \{(L+1, L+1+g)\}, \\ F_{2k+1} := \cdots := F_{3k} &:= \{(L, L+g)\}, \\ F_{3k+1} := \cdots := F_{4k-1} &:= \{(2L, g)\}. \end{aligned}$$

Then the sets

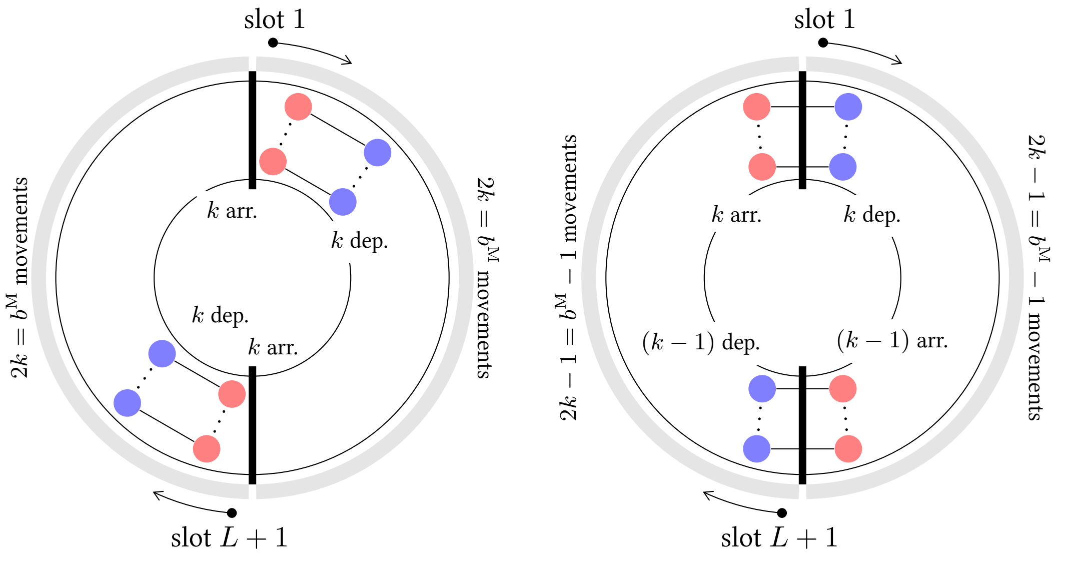
$$U := \bigcup_{i=1,\dots,2k} \{(i,a,d) : (a,d) \in F_i\}, \quad V := \bigcup_{i=2k+1,\dots,4k-1} \{(i,a,d) : (a,d) \in F_i\}$$

correspond to the flight requests $F_1, \dots, F_{2k}$ and $F_{2k+1}, \dots, F_{4k-1}$, respectively, and their respective cardinalities are $|U| = 2k = |V| + 1$, see Figure 3.15 for an illustration of U and V. Furthermore, the requests in U and V account for the following numbers of arrivals and departures within the time windows $1 + [L]_\circ$ and $(L+1) + [L]_\circ$:

	$\mathbf{1} + [\mathbf{L}]_\circ$		$(\mathbf{L}+\mathbf{1}) + [\mathbf{L}]_\circ$	
	Arrivals	Departures	Arrivals	Departures
U	k	k	k	k
V	k	$k-1$	$k-1$	k

Hence, both U and V are independent sets in $\mathcal{M}^{\infty}_{(L,b);g}$, as they account for at most $b^{\mathrm{M}} = 2k$ flight movements for each of the relevant time intervals. However, for every element $(i, a, d) \in U$, the set $V' := V \cup \{(i, a, d)\}$ would represent at least $b^{\mathrm{M}} + 1$ flight movements for at least one of the two time windows $1 + [L]_\circ$ and $(L + 1) + [L]_\circ$ (as for each element of U inserted in V, *two* flight movements are added to one of those time window), thus $V' \notin \mathcal{M}^{\infty}_{(L,b);g}$. This asserts that the matroid exchange property does not hold for $\mathcal{M}^{\infty}_{(L,b);g}$, thus the independence system is not a matroid.

Coupling arrival and departure of a flight request destroys the matroid structures of Flight Scheduling problems with only slot requests, even if the coupling is not flexible (if it was, we could interpret the problem as an instance of 3D-Matching for a suitable shifting bound, cf. Section 3.2.1). Of course, similar variations on Flight Scheduling and Maximum Slot Packing can be thought of, e. g. allowing for different ground times with an upper and a lower bound, or using more than one shifting bound. Identifying a property (preferably one that bears some relevance to applications) to classify the complexity of these problems is an interesting question following up on the results of this thesis.


slot 1
k arr.
k dep.
k dep.
k arr.
$2k = b^{\mathrm{M}}$ movements
$2k = b^{\mathrm{M}}$ movements
slot $L + 1$
slot 1
k arr.
k dep.
$(k-1)$ dep.
$(k-1)$ arr.
$2k - 1 = b^{\mathrm{M}} - 1$ movements
$2k - 1 = b^{\mathrm{M}} - 1$ movements
slot $L + 1$

(a) Arrivals and departures in the set U (b) Arrivals and departures in the set V

Figure 3.15: Illustration of the flight request sets U and V. A set of $n = 2L$ slots is shown in a circle, together with the two time windows $1 + [L]_\circ$ and $(L + 1) + [L]_\circ$ (gray). It is impossible to move an arrival/departure pair from the larger set U to the smaller set V without violating one of the time window bounds for V.

Chapter 4

Applications of Flight Scheduling at Airports

In this chapter, we will expand the theoretical considerations of Chapter 3 into a model suitable for practical application. To this end, we will first describe the practice of flight scheduling, clarifying and motivating various classes of constraints in detail in Section 4.1. In Section 4.2, the constraints will be translated into a concise mathematical model of the flight scheduling problem, different aspects of the model will be discussed and refined along the way. Section 4.3 evaluates our model on both real-world and realistic random data. The results obtained will be discussed under aspects of both solution quality and solution structure, relating the practical results to our findings of Chapter 3. In Section 4.4, we present some practical considerations on alternative reference value structures that aim at avoiding bad flight schedules, again expanding on the work of Chapter 3, specifically the ideas and results on Minimum Slot Cover and the gap between Minimum Slot Cover and Maximum Slot Packing, see Sections 3.5 and 3.6. Finally, Section 4.5 contains some concluding remarks.

4.1 Flight Scheduling in Practice

4.1.1 The Slot System and the Constraints for the Flight Scheduling Process

As was already explained in Section 1.3, the capacity at most airports is limited and often does not bear up against the vastly growing demand for national and international air traffic. To resolve this situation, the so called *slot system* has been developed as a means of allocating the scarce resource of airport capacity to the airlines wishing to offer a connection to and from an airport. In the rest of this section we will describe the slot system and its constraints including some special regulations for German airports in greater detail. Although some of that information was already given in Chapter 3, the descriptions here will be much mored detailed, aiming at a concise mathematical model for flight scheduling that can be used for practical purposes.

A *slot* is defined as "the scheduled time of arrival or departure [...] for an aircraft movement on a specific date at a coordinated airport." ([IATA07]) The procedure of slot allocation is applied only at airports where demand exceeds capacity by a noticeable factor during certain periods, these airports are referred to as *fully coordinated airports*. In Germany, by summer 2008 the following airports are fully coordinated: Frankfurt, Berlin (Tegel, Schönefeld, Tempelhof), Düsseldorf, München and Stuttgart. Thus, in order to provide air service to and from a coordinated airport, an airline first needs to acquire a pair of corresponding arrival and departure slot at this airport for the desired landing and take-off times.

The slot system is implemented according to the IATA *Worldwide Scheduling Guidelines* (cf. [IATA07]) by the *airport coordinator*, who is appointed by the government of every EU member state for the coordination of its national airports. The coordinator has to be independent from both the airports and the airlines and acts "in a neutral, non-discriminatory and transparent way" as "the sole person responsible for the allocation of slots" ([EU93]). At each coordinated airport, there is also a *coordination committee* (composed of members of the managing body of the airport and the air carriers operating at that airport) to advise the coordinator. The coordinator assigns available slots according to the airlines' requests "while taking account of all relevant technical, operational and environmental constraints" ([EU93]). In Germany, the airport coordinator is Claus Ulrich (as of 2008), head of the *FHKD (Flughafenkoordination Deutschland)*[1], an authority that reports to the German Federal Ministry of Transport and is financed by all major airlines in Germany. Similar authorities exist in all EU member states (e. g. *Airport Coordination Limited*[2] in the UK).

Flight scheduling as considered here is a long-term planning problem, thus a meticulous schedule is not necessary — exact airport operations can only be planned on a daily or hourly basis, as conditions such as winds and weather inadvertently introduce minor deviations from the schedule. For this reason, slots are usually allocated on a time scale discretized in ten minute intervals, with each interval possibly accommodating more than one flight. Each year is divided into two scheduling periods, a summer and a winter season. The summer season usually lasts from the end of March until the end of October, the winter season from October until March. (The summer season normally coincides with the period of daylight savings time).

Historic Slots

About six months before the start of a scheduling period the planning process starts with the coordinator passing to the airlines information about which slots are regarded *historic*. A series of slots allocated to an airline will be declared historic when they have been operated for at least 80% of the time during the period for which they have been allocated. Under the so-called *use it or loose it* provision, airlines fulfilling that usage criterion thereby acquire a *grandfather right* for a series of slots, meaning they have the right to claim the same series of slots for the coming planning period. That right is generally based on data from the preceding period of the same type, i. e., the last winter period for the upcoming winter and the last summer period for the upcoming summer. Having acknowledged the information about historic slots, the airlines then submit their *slot clearance requests* to the coordinator. A slot clearance request usually contains the desired

[1]Up-to-date information can be found on the official website `http://www.fhkd.org`.
[2]For further information, see the official website `http://www.acl-uk.org`.

slots for arrival and departure, the type of aircraft that will be used and the intended days of service.

In allocating the available slots to the airlines the coordinator will try to produce a “uniform” schedule where possible, respecting series requests (i. e., a request for several slots on different days that must be scheduled for the same time of day on every requested day). However, the coordinator is free to break up a series request after consultation with the airline, e. g., a series request for a slot pair on every day of the week might be broken up into one request for Monday–Friday and one for Saturday–Sunday with slightly different slots assigned to each of the requests to give the coordinator more flexibility.

The coordinator starts by allocating the slots that are considered historic. These historic slots are grouped into three categories:

Historic Slots (H): A request is made for a series of historic slots. Slots must necessarily be allocated exactly as demanded for these requests.

Changed Historic Slots (CL/CR): A request is made to move a flight from a historic slot (or series of slots) to a different slot (or series of slots). Such a request must either be fulfilled, or the original historic slots must be allocated to the request. For a *CL request*, there is only the alternative of allocating the desired slot(s) or the historic slot(s), whereas for a *CR request* any slot(s) between the desired and the historic slot(s) may be allocated (as long as some operational constraints are met), with precedence towards the desired slot(s).

Year Round Service (CI): A request is made for a series of slots that has been operated in the previous planning period. These requests are allocated immediately after the historic slots, i. e., they must be allocated when capacity is left after the historic slots are allocated. Notice that CI requests are not exactly requests for historic slots, as these claims are not based on the preceding season of the same type. Nevertheless, they are regarded as “almost historic” in practice.

New Entrants and Incumbents

After the slots with grandfather rights have been assigned, the remaining slots are entered into the *slot pool*, which is simply standard terminology meaning all slots that are available at a certain stage of the allocation process (keep in mind that slot allocation is implemented as a sequential process in practical operations). The *new entrants* rule is described in the IATA Worldwide Scheduling Guidelines (cf. [IATA07]) as follows:

> “Of the slots contained within the slot pool at the initial allocation, 50% must be allocated to new entrants, unless requests by new entrants are less than 50%.”

The “slot pool” here refers to all slots that are available after the allocation of slots to all requests with historic rights. A *new entrant (NE request)* is an airline that does not yet operate at the airport considered or operates no more than four slots on the requested date. This rule is designed to lower market entrance barriers and facilitate competition. For an exact definition, see [EU93; EU02; EU03; EU04] and [IATA07].

However, a burden is attached to new entrant status: If an airline is allocated slots under new entrant status, these come with the obligation to operate the new service for at least two scheduling periods of the same type (i. e., two winter periods or two summer periods). Furthermore, slots allocated under the new entrant rule may not be exchanged with other airlines. As a consequence, most airlines are reluctant to claim new entrant status in practice even if they could legally do so, because it severely reduces their ability to reschedule a flight or exchange slots and thus gives them less flexibility. For new entrant requests a certain flexibility is granted to the airport coordinator; the exact time allocated to a new entrant may vary from -60 minutes to $+60$ minutes around the requested time.

There is, however, a problem with the new entrants rule we have to deal with in our optimization model. As we know from Chapter 3, the total number of available slots depends heavily on the allocations already set up and is therefore subject to change under different allocations, see Sections 3.5 and 3.6 for more details. Furthermore, it is unclear how many slots are to be considered "available" (and thus part of the "slot pool") after a partial allocation (e. g., of requests bearing historic rights) has been performed. It is certainly *not* obvious how many slots are still "remaining" after some of the flights have been integrated, because this number may vary depending upon how the remaining flights are integrated, and also upon the exact slots that are allocated to the historic flights (where there is a choice). Of course, if the number of available slots is not clear at any stage of the scheduling process, the number of slots reserved for new entrants can not at all be calculated.

There are two apparent approaches to deal with this situation. The "foolproof" way is to use the 50% as an ex-post measure, meaning that in the final allocation the new entrants should make up for at least 50% of all flights that have been integrated into the schedule and that are not historic flights, provided there are new entrants' requests that have not been integrated. A different (but computationally much simpler) approach is to somehow estimate the number of free slots and set up a constraint based on that estimation. This can be done, e. g., by simply taking the maximum number of flights that could be integrated into a partial flight schedule, or (a little more sophisticated) by first computing a preliminary flight schedule without the new entrants rule, deduce from that the number of slots allocated to non-historic flights and demand that at least half of that number is allocated to new entrants, if there are enough requests. The final flight schedule is then computed in a second pass with the new entrants rule in effect. As a drawback, this method not only requires two flight schedule computations, but also does not guarantee ex-post validity of the final schedule with respect to the new entrants rule, because the numbers taken from the preliminary schedule can naturally only be an approximation to the final results.

As a more pragmatic way of dealing with the new entrants rule, one could simply get rid of the 50% constraint and demand that all new entrants' requests be integrated into the flight schedule. As mentioned before, in practice most airlines are very reluctant to claim new entrant status, because of the liabilities involved. Hence, in practice, one would expect that there are far less new entrants' requests than to make up for more than 50% of the available slots (however these would be determined), thus all of these request would have to be integrated anyway.

After the allocation of slots to new entrants or after the reserved 50% of the slot pool are used up, all other requests (called *incumbents' requests*) are processed. As with new entrants, the slot assigned to an incumbent request (if any) may vary up to 60 minutes around the requested time.

When the slot assignment has been communicated to the airlines, they may re-route their flights in order to make the best possible use of their slots and they may also exchange slots one for one, but without monetary compensation, as long as they obtain the consent of the coordinator for these changes (however, neither is possible for new entrants' slots). To facilitate these arrangements, the IATA hosts a *Schedules Conference* some time after the coordinators have communicated their preliminary slot allocation to the airlines, where the coordinators, airport representatives and the airlines' planners meet to finalize the schedules.

Reference Value Systems

At German airports the air and ground control as well as environmental and noise protection restrictions are usually cast into so-called *reference value systems*, cf. Section 3.1.2. Recall that a reference value system is a collection of *time window bounds* that constrain the number of arrivals, departures and flight movements that can be allocated within a certain period of time. While we considered both shifting and non-shifting bounds in Chapter 3, in practical applications only shifting bounds are used. For this reason, we will restrict our considerations in this chapter to shifting bounds. However, this is mainly a simplification of notation, integration of non-shifting bounds into our model is certainly possible and quite straightforward. Furthermore, we will only consider symmetric and monotone reference value systems (cf. Definition 3.6) in this chapter, as all practical data that we collected only used such systems.

A *shifting bound* consists of a period length (e. g., 60 minutes) and three bound values that define upper bounds on the number of arrivals and departures as well as on the total flight movements for every time window of the specified length within the planning horizon. These values are referred to as *arrival (A)*, *departure (D)* and *movements (M)* values. Normally, several shifting bounds for different time periods interact to define implicit bounds on the number of slots that can be allocated, see Sections 3.3 and 3.4 for a closer look into the nature of these interactions. For a flight schedule to be feasible, it must conform to the complete set of shifting bounds contained in a reference value system.

In Germany, there are usually shifting bounds for time periods of 10, 30 and 60 minutes, where the bounds for the longer periods are relatively more restrictive than those for the shorter periods (i. e., the reference value systems are monotone). There is, however, no "hard legislation" that would allow only for 10, 30 and 60 minute time windows, so a different reference value system might be put into practice if there is consensus among the participating parties (including the airport coordinator, the airports, the airlines and the local authorities). Notice that a reference value system with time window lengths 10, 30 and 60 minutes has the inclusion property, cf. Definition 3.44.

It is important to note that, when talking about a flight schedule, one usually refers to *runway* *times*, i. e., the time where the aircraft touches or takes off the runway. In contrast to this, the flight schedule published to the passengers generally gives *gate times*, referring to the time when the aircraft arrives at or departs from the terminal building. If not explicitly stated otherwise, the shifting bounds and all other time-dependent constraints always refer to runway times. This difference is important, because arrival and departure flights undergo a different time shift in the conversion between runway and gate times. An arriving flight needs to taxi from the runway to a parking position and possibly connect to the gate via a jetway. A departing flight, in addition to disconnection from the jetway and taxi time, also needs to be pushed back and turned around by a tow truck, because airplanes cannot taxi backwards on their own. As a standard procedure, for an arriving flight the gate time is five minutes after its arrival at the runway, while a departing flight has to leave the gate ten minutes before it is scheduled for take-off on the runway. Recall that time is usually discretized in units of ten minutes, which leads to a five minute "pattern" in published flight schedules — this is due to the different conversion factors between gate and runway times for arrivals and departures, respectively, and the fact that flight scheduling is performed in runway time notion, while the published flight schedule lists gate times.

In a reference value system, the shifting bounds are usually referred to by a suggestive abbreviation containing most of the necessary information. A shifting bound is denoted by the leading letter `R` (for the German word "Regel", "rule"), followed by the length of the respective time window in minutes and one of the letters `A`, `D` or `M` to denote the arrival, departure or movements bound value. For example, `R30A` $= 25$ describes a shifting bound of length 30 minutes allowing for a maximum of 25 arrivals. We will frequently give all three values as a triple according to the scheme `R10 A/D/M` $= 9/9/16$ or simply `R10` $= 9/9/16$. The shifting bounds may also differ by time of day, in which case the respective times of validity are given with the shifting bounds. This can, for instance, be used to implement restrictions on night traffic, which is a very common scenario at German airports. For an example of a reference value system, we refer the reader to Figure 3.1 in Section 3.1.3.

Linking Arrival and Departure

A flight request as described above generally consists of a pair of arrival and subsequent departure. When it is not possible to integrate a request at the exact time that was demanded, this arrival-departure link has to be respected. More specifically, there has to be enough time scheduled between arrival and departure so that the aircraft can be safely unloaded, necessary checks can be performed, fuel can be replenished, the cabin can be cleaned and boarding can be completed comfortably. The time that is needed for these processes naturally varies with the size of the aircraft; while a small regional jet can be handled within maybe half an hour, a larger aircraft servicing an intercontinental flight with several hundred passengers will naturally need a longer ground time. The minimal ground time requirement can be deduced from the aircraft type reported by the airline as part of the flight request. On the other hand, every airline wants their equipment to be airborne for as long as possible, because long ground times reduce the airline's profit. Consequently, there is also an upper bound to the ground time that an airline will accept. Thus, when the airport coordinator schedules a flight, both minimal and maximal ground time constraints have to be observed.

There is, however, one notable exception to that rule: At some airports, one airline serves so many flights that it may prefer to request only single arrival and departure slots and link these suitably on its own account later. This provides for greater flexibility for the airline, which is of special importance if major maintenance works are carried out at the airport for that airline. Such maintenance operations, which must be performed on a regular basis, take an aircraft out of the regular circulation for hours or maybe days, so it must be replaced by a different model. The ability to change the linking of arrivals and departures with some flexibility allows the airline to react to maintenance needs as well as unscheduled delays quickly. This exception generally applies to the home base airport of any major carrier; as an example, Lufthansa flights are considered as single slot requests instead of as pairs by the airport coordinator for Frankfurt/Main. However, for a flight schedule to be applicable, the difference between the number of arrival and departure slots may certainly not become too large (some deviation is allowed, because aircraft may stay at the airport for the night or stay at the airport for scheduled maintenance). While an experienced coordinator will intuitively adhere to that rule, any automated solution must integrate an upper bound to the deviation allowed between arrivals and departures on a daily basis and/or for the whole planning season.

North America Rule

As the airways between Europe and Northern America are heavily used, there are restrictions on the number of take-offs in the direction of North America at some German airports. As an example, a valid flight schedule may only contain up to four flights headed for a destination in Northern America within each fifteen minutes. These bounds may also be applied in a shifting fashion, as described for the reference value systems above. Notice that a bound with a 15 minute time window would require a five minute discretization instead of the usual ten minute steps. As all other requirements can be expressed in a ten minute discretization, the North America rule rule is frequently modified to allow for a maximum of eight North America bound departures within each thirty minutes (usually non-shifting) for compatibility.

Coupling of Flight Requests

Under certain circumstances, it may be desirable to couple selected flight requests to each other in various ways. One such case is the coupling of *hub and spoke* flights. In such a setting, an airline combines several regional flights with one intercontinental flight in the following way: A number of regional flights with roughly the same arrival time (the *spokes* or *feeder flights*) transport passengers from smaller airports to a large airport (the “hub”). The passengers then change into one larger aircraft for their international destination. Using this system helps the airline to increase capacity utilization of its large (and expensive) aircrafts while offering passengers a convenient means of traveling from an airport located in their home region. Of course, the same system is also used “backwards”, i. e., passengers arriving at a hub from some destination overseas are forwarded to their home airport via a series of regional spoke flights. To implement such a setting, it is necessary to have at least a certain number of feeder flights integrated into the flight schedule if the intercontinental flight is to take place.

Additional Local Rules

In addition to the “global process” outlined above the coordinator has to conform to a number of local rules that may differ from airport to airport to reflect particular technical, operational and environmental constraints. Among these are air and ground traffic capacity restrictions due to safety considerations or airport handling capacity, restrictions on the types of aircraft being used, restrictions on night flight operations, noise protection legislation, restrictions due to special handling requirements (e. g., for flights to Israel) or air corridor limitations (e. g., the North America rule mentioned above). Of course, the interests of the airport also have to be taken into account for the scheduling process. For example, an airportairport usually prefers periodic and “uniform” flight schedules to facilitate organization and short-time rescheduling. In this context, the results of Sections 3.3–3.5 are important, because we can be sure that an optimal outcome is still possible with a “uniformity constraint” in place. (Of course, the results mentioned were obtained for a much more restricted setting, so in practice we could indeed forfeit a possible optimum when enforcing uniformity. However, as test runs will show, this is not very likely for realistic data.) Other aspects include reliability of carriers (carriers that have proven to be reliable in the past may get priority over others) and special events like the Olympic Games or the Soccer World Cup.

Objective Functions

Although the objective of flight scheduling might seem obvious (namely to schedule as many flights as possible, given the above constraints), there are a number of alternatives that one frequently considers in practice. First, notice that there is generally a difference between the number of series requests that are scheduled and the number of flight movements, as the latter also includes information about the days of service and the length of the planning horizon. Revenue for an airport is mainly generated by starting, landing and service fees, so the number of movements is naturally the most important objective from an airport’s point of view.

On the other hand, other sources of revenue are becoming increasingly important, for instance income from the lease of shopping areas and passenger service facilities such as restaurants. These revenues correlate to the number and type of passengers, so an airport might prefer to take the expected total number of passengers (normally estimated by the total number of seats) as an objective. Of similar nature is the sum of the *Maximum Take-Off Weights (MTOW)*.

As outlined before, other important objectives include average punctuality and customer satisfaction (where “customer” can refer to both the airlines and the passengers from an airport’s point of view). As a general approach to these objectives and variants thereof we will use an alternative cost minimization objective. This will take into account the satisfaction level of the airlines with the flight schedule, which is likely to be higher for a schedule that grants the airlines their favorite slots whenever possible, while at the same time integrating a maximum number of flights. Also, getting a slot closer to the requested one should increase overall punctuality, because the airline naturally needs a corresponding slot at the destination preceding or succeeding the airport we are considering for optimization, and the airline's request for a specific slot should be backed by a slot request at other airports taking into account the average flight time. However, if the integration of some flight into the schedule came at a disproportionate cost in terms of large deviations from the requested slots for lots of other flights, it might ultimately be a better choice not to integrate that flight and increase overall satisfaction for all the other airlines. By using a cost function that assigns every arrival and departure request a separate costcost for deviating from the desired slot as well as a cost for non-integration of a flight request, a very fine-grained control is possible that can take into account punctuality levels, steadiness, the number of seats or the MTOW as well as the potential income generated by a flight for the airport.

4.1.2 Flight Scheduling in Practice

We conclude this section with a short note on the implementation of the flight scheduling process in practice. As noted above, the coordinator needs to start by scheduling all historic flights, these make up for about 60% of all flight requests. Generally, H-requests are scheduled first, then those with CL- and CR-classification (the latter at the requested slots or the nearest possible slot). After that, year-round requests are integrated into the flight plan. When all historic flights have been processed, the number of available slots is calculated (which can only be done approximately, because that number depends upon the flight schedule) and a number of 50% of the remaining slots are reserved for new entrants' flight requests (classification NE).

The new entrant requests are drawn at random, using a computer software (this is to ensure neutrality of the coordination process) and integrated into the schedule if possible, at a free slot closest to the requested time (some manual adjustments to the CR and CL flights might be necessary at this stage). When 50% of the non-historic slots have been used or when all NE flights have been integrated into the schedule (in practice, the latter is much more common, because airlines are quite reluctant to have their requests classified as new entrants'), the incumbents' flight requests are also considered. One request after the other is drawn at random and integrated as close to the requested slot as possible (or discarded if it cannot be integrated within the admissible time intervals).

All these steps are performed by a computer software with minimal manual intervention. When all flights have been considered, a draft schedule is presented together with the values of all shifting bounds' time windows. The coordinator and his/her coworkers then start to review the schedule, possibly integrating more flights by manual adjustments. After the coordinator completes this step, the (still preliminary) schedule is sent to the airlines and to the airport, who both get a chance to comment. The scheduling process up to here takes about six to eight weeks in total. The schedule is then finalized at an international Schedules Conference.

4.2 An Integer Programming Model for Flight Scheduling

4.2.1 Notation

In this section we will develop an integer programming formulation for flight scheduling and translate the various constraints described in Section 4.1 into integer linear equations or inequalities. In addition, we propose different objective functions for our model.

To formally model flight scheduling as an integer program we need to adapt some of the notation introduced in Chapter 3 to accommodate for the additional level of detail we need in this chapter.[3] We will generally use one day as the minimal time horizon we consider for scheduling, and every other time horizon will consist of whole days only. Throughout this chapter, we denote by $n \in \mathbb{N}$ the number of slots in one day (usually, $n = 24 \cdot 6 = 144$), $d \in \mathbb{N}$ will be the number of days we consider, and $N := n \cdot d$ is the total number of slots.

Definition 4.1 (Series Request)
Let $n, d \in \mathbb{N}$ and $N := nd$. A *series request* or *flight series request* is a tuple $(T, S, g, \mathcal{D}, c)$ where

- $T = (T^{\mathrm{A}}, T^{\mathrm{D}})^T \in ([n])^\star \times ([n])^\star$, where at least one of $T^{\mathrm{A}}, T^{\mathrm{D}}$ has a value different from ∞, is the requested arrival/departure slot pair for every day of service;
- $S = (S^{\mathrm{A}}, S^{\mathrm{D}})^T \in [n]^2$ is the maximum allowable shift in slots, allowing for a shift of S^{A} around T^{A} for the arrival slot and of S^{D} around T^{D} for the departure slot, respectively;
- $g = (g^{\min}, g^{\max})^T \in [n]^2$ are the minimum and maximum allowable ground times in slots, respectively;
- $\mathcal{D} \subset \{1, \dots, d\}$ is the set of requested days of service (if T^{A} and T^{D} are both unequal to ∞, then $\mathcal{D}$ lists the days for the arrival slot);[4]
- $c = (c^{\mathrm{A}}, c^{\mathrm{D}}, c^{\mathrm{M}})^T \in \mathbb{R}^3_{\geq 0}$ is a cost vector denoting the costs c^{A} and c^{D} for a deviation of one slot from T^{A} and T^{D}, respectively, and the cost c^{M} for not allocating a slot pair to the series request.

A series request $(T, S, g, \mathcal{D}, c)$ is called *arrival request* if $T^{\mathrm{D}} = \infty$, and *departure request* if $T^{\mathrm{A}} = \infty$. A series request with $T^{\mathrm{A}}, T^{\mathrm{D}} \neq \infty$ will sometimes be termed a *mixed series request*. For a series request $F = (T, S, g, \mathcal{D}, c)$ we will sometimes write T_F, S_F, g_F, $\mathcal{D}_F$ and c_F to denote its components.

[3]For that reason, we will use the additional qualifier "abstract" to refer to the definitions of Chapter 3, when the definition we refer to is not clear from the context.

[4]As noted earlier, we will also consider requests for single arrival or departure slots. In these cases, $\mathcal{D}$ simply contains the requested days of service. In case of a slot pair, however, the departure might take place on the day following arrival (a so called "overnight request", because the aircraft stays at the airport overnight), thus we need a convention of how to interpret $\mathcal{D}$ in these cases.

A series request is a “realization” of an abstract flight request as it is stated in practice, with a desired arrival/departure slot, an acceptable maximum deviation from those slots, minimum and maximum ground times and the intended days of service. A series request corresponds to an abstract series request $(F, \mathcal{I})$ (cf. Definition 3.1) defined by

$$F = \Big\{(a,d) \in \{1,\dots,n\}^2 : |T^{\mathrm{A}} - a| \leq S^{\mathrm{A}} \ \wedge \ |T^{\mathrm{D}} - d| \leq S^{\mathrm{D}} \\ \wedge \ g^{\min} \leq \mathrm{dist}_n(a,d) \leq g^{\max}\Big\}$$

and

$$\mathcal{I} := \{(t-1)n : t \in \mathcal{D}\} \,.$$

For a series request we employ the convention that arrival and/or departure must (if at all) always take place on the day that was requested. As a consequence, if an airline desires a very late arrival slot for a request with positive shift S^{A}, the arrival cannot be shifted into the very early hours of the following day. (and, of course, similar situations arise for any early or late slot requested). This restriction is made primarily for the benefit of a more concise notation. It is indeed possible to allow for such situations in our model, but it would often require technical and tedious notation for the different cases involved, and we thus opted in favour of a clearer notation. Let us mention that for practical purposes no loss is incurred by this decision, as night flight traffic is very low at German airports, often due to severe night flight restrictions, so the number of relevant very late and very early flight movements is virtually zero. There are, however, flights that are scheduled to stay at the airport for the night, often for maintenance work, and we will certainly respect those in our model. These overnight series requests are recognizable by the fact that $T^{\mathrm{D}} < T^{\mathrm{A}}$, hence the departure has to take place on the day following arrival. Let us remark once again that for such requests $\mathcal{D}$ contains the days of the desired arrival slot, the departure is then one day later.

The cost vector contained in a series request can be used to represent a flight’s weight relative to others, e. g., because an airline that has proven to be unreliable in the past may get lower preference (and thus lower cost coefficients) than a reliable carrier for the creation of a flight schedule. These preferences can be expressed using an alternative cost objective that works with the parameters c^{A}, c^{D}, and c^{M} of a series request, see below.

One important aspect not expressed in a series request is the classification of the request. This will instead be encoded using a partition of all series requests.

Convention 4.2
For the rest of this chapter, we denote the collection of all series requests that are to be considered for the optimization by $\mathcal{F}$ and refer to that collection as the *flight request set.* Furthermore, we denote by

$$\mathcal{F} = \mathcal{F}^{\mathrm{H}} \cup \mathcal{F}^{\mathrm{CR}} \cup \mathcal{F}^{\mathrm{CI}} \cup \mathcal{F}^{\mathrm{CL}} \cup \mathcal{F}^{\mathrm{NE}} \cup \mathcal{F}^{\mathrm{I}}$$

a partition of $\mathcal{F}$ into disjoint sets meaning by

- $\mathcal{F}^{\mathrm{H}}$, $\mathcal{F}^{\mathrm{CR}}$, $\mathcal{F}^{\mathrm{CI}}$, $\mathcal{F}^{\mathrm{CL}}$: historic flights of H-classification and rescheduled historic flights of CR-, CI- or CL-classification, respectively;
- $\mathcal{F}^{\mathrm{NE}}$: flights with new entrant status;
- $\mathcal{F}^{\mathrm{I}}$: all other flights, i. e., those flight bearing incumbent status.

For convenient notation, we will also use the abbreviation

$$\mathcal{F}^{\mathrm{H/C}} := \mathcal{F}^{\mathrm{H}} \cup \mathcal{F}^{\mathrm{CR}} \cup \mathcal{F}^{\mathrm{CI}} \cup \mathcal{F}^{\mathrm{CL}}$$

for the set of all historic and re-timed historic flight requests.

It will also be convenient to be able to easily distinguish between arrivals and departures later. Notice that a series request $(T, S, g, \mathcal{D}, c)$ corresponds to a tuple of an arrival and a departure series request $\big((T', S', g', \mathcal{D}', c'); (T'', S'', g'', \mathcal{D}'', c'')\big)$ where

$$T' = (T^{\mathrm{A}}, \infty), \quad S' = (S^{\mathrm{A}}, 0), \quad T'' = (\infty, T^{\mathrm{D}}), \quad S'' = (0, S^{\mathrm{D}})$$

and

$$g' = g'' = g, \quad \mathcal{D}' = \mathcal{D}'' = \mathcal{D}, \quad c' = c'' = c,$$

thus a series request can also be viewed as a pair of arrival and departure series request.

Convention 4.3
Let $\mathcal{F}$ denote the flight request set. By

$$\mathcal{F} = \mathcal{F}^{\mathrm{A}} \cup \mathcal{F}^{\mathrm{D}} \cup \mathcal{F}^{\mathrm{M}}$$

we will denote a partition of $\mathcal{F}$ into disjoint sets, where $\mathcal{F}^{\mathrm{A}}$ contains all arrival series requests, $\mathcal{F}^{\mathrm{D}}$ contains all departure series requests and $\mathcal{F}^{\mathrm{M}}$ contains all mixed series requests. We shall also use the abbreviations

$$\mathcal{F}^{\mathrm{A/M}} := \mathcal{F}^{\mathrm{A}} \cup \mathcal{F}^{\mathrm{M}} \quad \text{and} \quad \mathcal{F}^{\mathrm{D/M}} := \mathcal{F}^{\mathrm{D}} \cup \mathcal{F}^{\mathrm{M}}.$$

Furthermore, by $\mathcal{F}^{☾} \subset \mathcal{F}^{\mathrm{M}}$ we will denote the set of *overnight series requests*, i. e.,

$$\mathcal{F}^{☾} := \left\{ F = (T, S, g, \mathcal{D}, c) \in \mathcal{F}^{\mathrm{M}} : T^{\mathrm{A}} > T^{\mathrm{D}} \right\}.$$

As in Definition 3.3, we define a flight schedule to be a mapping from the flight request set into the set of (or "∞" to denote that a series request could not be allocated a slot pair) within the constraints defined by the series request.

Definition 4.4 (Flight Schedule)
Let $\mathcal{F}$ be a flight request set. A *flight schedule* for $\mathcal{F}$ is a function

$$\begin{aligned} f : \mathcal{F} &\to (([n])^\star \times ([n])^\star)^\star , \\ F &\mapsto (f^{\mathrm{A}}(F), f^{\mathrm{D}}(F)) \end{aligned}$$

such that the following properties hold for all $F = (T, S, g, \mathcal{D}, c) \in \mathcal{F}$ (notice that we identify $f(F) = \infty$ with $f^{\mathrm{A}}(F) = f^{\mathrm{D}}(F) = \infty$ here and in the following):

- **Historic flights**: $f^{\mathrm{A}}(F) \in \{1, \dots, n\}$ for all $F \in \mathcal{F}^{\mathrm{H/C}} \cap \mathcal{F}^{\mathrm{A/M}}$ and $f^{\mathrm{D}}(F) \in \{1, \dots, n\}$ for all $F \in \mathcal{F}^{\mathrm{H/C}} \cap \mathcal{F}^{\mathrm{D/M}}$.
- **Shift limits**: $|f^{\mathrm{A}}(F) - T^{\mathrm{A}}| \le S^{\mathrm{A}}$ for all $F \in \mathcal{F}^{\mathrm{A/M}}$ and $|f^{\mathrm{D}}(F) - T^{\mathrm{D}}| \le S^{\mathrm{D}}$ for all $F \in \mathcal{F}^{\mathrm{D/M}}$.
- **Single slots**: If $T^{\mathrm{A}}_F = \infty$, then $f^{\mathrm{A}}(F) = \infty$, and if $T^{\mathrm{D}}_F = \infty$, then $f^{\mathrm{D}}(F) = \infty$.
- **Slot pairs**: $f^{\mathrm{A}}(F) \neq \infty \Leftrightarrow f^{\mathrm{D}}(F) \neq \infty$ for all $F \in \mathcal{F}^{\mathrm{M}}$ with $f(F) \neq \infty$.
- **Ground times**: $g^{\min} \le \operatorname{dist}_n(f^{\mathrm{A}}(F), f^{\mathrm{D}}(F)) \le g^{\max}$ for all $F \in \mathcal{F}^{\mathrm{M}}$ with $f(F) \neq \infty$.

A series request F is said to *integrated* or *scheduled* in the flight schedule f, if $f(F) \neq \infty$.

For the objective functions we shall consider for our integer programming model, some notions measuring the “quality” of a flight schedule in various ways will be helpful.

Definition 4.5 (Size and Cost of a Flight Schedule)
Let $\mathcal{F}$ be a flight request set and $f : \mathcal{F} \to (([n])^\star \times ([n])^\star)^\star$ a flight schedule. Then the *size* of the flight schedule f is the number $|f|$ of scheduled flight movements, i. e.,

$$|f| := \sum_{\substack{F \in \mathcal{F}^{\mathrm{A/M}} \\ f^{\mathrm{A}}(F) \neq \infty}} |\mathcal{D}_F| + \sum_{\substack{F \in \mathcal{F}^{\mathrm{D/M}} \\ f^{\mathrm{D}}(F) \neq \infty}} |\mathcal{D}_F| .$$

The *cost* of the flight schedule f is

$$c(f) := \sum_{\substack{F \in \mathcal{F}^{\mathrm{A/M}} \\ f^{\mathrm{A}}(F) \neq \infty}} |f^{\mathrm{A}}(F) - T^{\mathrm{A}}_F| \cdot c^{\mathrm{A}}_F + \sum_{\substack{F \in \mathcal{F}^{\mathrm{D/M}} \\ f^{\mathrm{D}}(F) \neq \infty}} |f^{\mathrm{D}}(F) - T^{\mathrm{D}}_F| \cdot c^{\mathrm{D}}_F + \sum_{\substack{F \in \mathcal{F} \\ f(F) = \infty}} c^{\mathrm{M}}_F .$$

The two measures size and cost stress very different aspects of a flight schedule. Notice that the size does not simply count the number of flight series integrated into the flight plan, but weighs each integrated series with the number of flight movements it accounts for. In practice, this is the commonly used measure. In contrast to this, the cost of a flight schedule measures the degree of deviation from the originally requested slots that is incurred by a flight scheduleflight schedule. The cost function may also incorporate a flight-dependent weight on shifted arrivals and departures and takes into account a (virtual) cost for not integrating a flight into the schedule.

As discussed in Chapter 3, the most prominent constraints for flight scheduling are the shifting bounds contained in some reference value system. To see if a flight schedule is feasible with respect to a reference value system, we take the very same approach as in Chapter 3 and associate to that flight schedule f a slot configuration $C_f : \{1, \ldots, N\} \to \mathbb{N}_0^3$ counting the arrivals, departures and total movements induced by f for the whole planning horizon. Feasibility of a flight schedule is also defined in terms of C_f, exactly as in Section 3.1.3. As these notions should be obvious, we do not reproduce them here.

4.2.2 Problem Statement

With the above notation, we can now formally state the flight scheduling problem on real-world data.

Problem 4.6: Maximum Flight Schedule
Instance: A number $n \in \mathbb{N}$ of slots per day, a number $d \in \mathbb{N}$ of days, a flight request set $\mathcal{F}$ and a reference value system $\mathcal{R}$.
Question: Find a flight schedule for $\mathcal{F}$ that is feasible with respect to $\mathcal{R}$ and has maximum size.

Problem 4.7: Minimum Cost Flight Schedule
Instance: A number $n \in \mathbb{N}$ of slots per day, a number $d \in \mathbb{N}$ of days, a flight request set $\mathcal{F}$ and a reference value system $\mathcal{R}$.
Question: Find a flight schedule for $\mathcal{F}$ that is feasible with respect to $\mathcal{R}$ and has minimum cost.

4.2.3 Modeling Flight Scheduling as Integer Linear Program

We are now ready to transform the formal description of Sections 4.2.1 and 4.2.2 into an integer linear program. We will first introduce the variables of the integer programming formulation and then discuss the various constraints and possible objective functions. As in the foregoing subsection, let $n \in \mathbb{N}$ be the number of slots per day, $d \in \mathbb{N}$ the number of days in the planning horizon and $N := nd$ the total number of slots. Furthermore, let $\mathcal{F} = \{F_1, \ldots, F_m\}$ denote the flight request set with partitions as defined in Section 4.2.1, and let $\mathcal{R} = \{(L_1, b_1), \ldots, (L_k, b_k)\}$ be a reference value system consisting of only shifting bounds. For a series request F_i we will without further notion denote its components by $(T_i, S_i, g_i, \mathcal{D}_i, c_i)$.

Decision Variables

For every series request in $F \in \mathcal{F}$, we introduce a number of binary decision variables, one for each slot that can be allocated to a particular request. In order to reduce the number of variables, we follow a "template" approach here: We only use variables for slots $t \in [n]$ (i. e., one day of the planning horizon) and "extrapolate" the values of these variables to the whole planning horizon using the set $\mathcal{D}_F$. Thus, the number of variables can be reduced by a factor of d (recall $d \approx 180$ for the complete problem!) compared to the naive approach of using a full set of variables for *every* slot in $\{1, \ldots, N\}$.

As remarked before, we will assume that arrivals and departures must be scheduled for the day of the desired slot, i. e., a flight series is scheduled as overnight series if and only if it is requested as such. For the data that we will use for our practical explorations later, no problems arise from that restriction for two reasons: First, there is a (more or less strict) night flying restriction on German airports, meaning that very late and very early flight movements may not be scheduled anyway. Second, many airlines arrange for necessary maintenance work to be performed during the night at the airport. Thus a flight can either be processed within a single day, or it can stay at the airport over night. In either case, what happens for a particular flight is known in advance and cannot change in the course of optimization, so we need not take into account this possibility. Yet, as mentioned before, a modified model would pose no added difficulties to our approach (but would require some technical case differentiations).

We denote by x_{it} decision variables for arrival requests and by y_{it} decision variables for departure requests, with $i \in [m]$, $t \in [n]$, and with the understanding that

$$x_{it} = \begin{cases} 1, & \text{if slot } t \in [n] \text{ is allocated for arrival to series request } F_i \in \mathcal{F} \\ & \text{for every day of service in } \mathcal{D}_i, \\ 0, & \text{otherwise;} \end{cases}$$
$$y_{it} = \begin{cases} 1, & \text{if slot } t \in [n] \text{ is allocated for departure to series request } F_i \in \mathcal{F} \\ & \text{for every day of service in } \mathcal{D}_i, \\ 0, & \text{otherwise.} \end{cases}$$

Of course, most of these variables will usually be 0, more precisely:

$$\begin{aligned} &\text{for all } F_i \in \mathcal{F}^{\mathrm{A/M}}: \quad x_{it} = 0 \quad \text{for all } t \in [n] \text{ with } |t - T^{\mathrm{A}}| > S^{\mathrm{A}} \\ &\text{for all } F_i \in \mathcal{F}^{\mathrm{D/M}}: \quad y_{it} = 0 \quad \text{for all } t \in [n] \text{ with } |t - T^{\mathrm{D}}| > S^{\mathrm{D}} \end{aligned}$$

For performance and storage space reasons, our implementation of the integer programming model will only allocate space for those decision variables that are not fixed to zero anyway. However, for the formulation of the model it is more convenient to work with the full set of variables and add the “non-zero constraints” above.

Allocation Constraints

In order to guarantee a valid allocation of slots to series requests, we need to integrate the usual allocation constraints into our model.

$$
\begin{aligned}
&\sum_{t\in[n]} x_{it} \le 1 \quad \text{for all } F_i \in \mathcal{F}^{\mathrm{A/M}} \backslash \mathcal{F}^{\mathrm{H/C}} && \sum_{t\in[n]} y_{it} \le 1 \quad \text{for all } F_i \in \mathcal{F}^{\mathrm{D/M}} \backslash \mathcal{F}^{\mathrm{H/C}}\\
&\sum_{t\in[n]} x_{it} = 1 \quad \text{for all } F_i \in \mathcal{F}^{\mathrm{A/M}} \cap \mathcal{F}^{\mathrm{H/C}} && \sum_{t\in[n]} y_{it} = 1 \quad \text{for all } F_i \in \mathcal{F}^{\mathrm{D/M}} \cap \mathcal{F}^{\mathrm{H/C}}\\
&x_{it} = 0 \quad \text{for all } F_i \in \mathcal{F}^{\mathrm{D}} && y_{it} = 0 \quad \text{for all } F_i \in \mathcal{F}^{\mathrm{A}}
\end{aligned}
$$

In addition, for mixed series requests that involve both arrival and departure either both or none of these must be allocated a slot:

$$\sum_{t\in[n]} (x_{it} - y_{it}) = 0 \quad \text{for all } F_i \in \mathcal{F}^{\mathrm{M}}$$

Ground Time Constraints

Ground time constraints are only relevant for flight that do not stay over night (we assume that overnight flights stay on the ground long enough anyway, due to the night flying restrictions), but as remarked before, ground time constraints for overnight flights could be integrated into the model with some notational effort. For modeling time dependent restrictions like the minimum/maximum ground time constraints notice that

$$\sum_{t\in[n]} t \cdot x_{it} \quad \text{and} \sum_{t\in[n]} t \cdot y_{it}$$

can be used to express the arrival and departure slot, respectively, allocated to a flight F_i. With this observation we can model the ground time constraints for a series request $F_i \in \mathcal{F}^{\mathrm{M}} \backslash \mathcal{F}^{☾}$ as

$$
\begin{aligned}
\sum_{t\in[n]} \left((t + g_i^{\min}) \cdot x_{it} - t \cdot y_{it}\right) &\le 0\\
-\sum_{t\in[n]} \left((t + g_i^{\max}) \cdot x_{it} + t \cdot y_{it}\right) &\le 0
\end{aligned}
\tag{4.1}
$$

Notice that a formulation like $\sum_{i\in[n]}(t \cdot x_{it} + g_i^{\min} - t \cdot y_{it}) \le 0$ does not yield the desired result, because it would constitute an infeasibility for every series request that is not allocated a slot pair (and at a congested airport there will naturally be such flight requests). On the other hand, for a series request that is guaranteed to receive a slot (because it has historic rights), this formulation is possible and it is stronger in an LP sense than (4.1), i. e., the inequality is stronger for the LP relaxation. So it is indeed advisable to use (4.1) for all $F_i \in \mathcal{F}^{\mathrm{M}} \setminus \mathcal{F}^{\mathrm{H/C}}$ and

$$\begin{aligned} \sum_{t\in[n]} \left(t \cdot x_{it} + g_i^{\min} - t \cdot y_{it}\right) &\le 0 \\ -\sum_{t\in[n]} \left(t \cdot x_{it} + g_i^{\max} + t \cdot y_{it}\right) &\le 0 \end{aligned} \tag{4.2}$$

instead for all historic flights $F_i \in \mathcal{F}^{\mathrm{M}} \cap \mathcal{F}^{\mathrm{H/C}}$.

Reference Value System

Modeling shifting bounds for a flight schedule is quite straightforward, because one can almost literally translate the abstract definition of a feasible slot allocation into integer programming terminology. The only caveat here is that we have to transform the “template representation” into a real flight schedule on the complete slot set $[N]$ using the “days of service”-set $\mathcal{D}_i$ for every request $F_i \in \mathcal{F}$. Hence for all shifting bounds $(L, b) = \left(L, (b^{\mathrm{A}}, b^{\mathrm{D}}, b^{\mathrm{M}})^T\right) \in \mathcal{R}$ we have the following inequalities:

For all $(L, b) \in \mathcal{R}$, all $s \in \{1, \dots, n - L\}$ and all $v \in \{1, \dots, d\}$:

$$\sum_{t\in s+[L]_\circ} \left(\sum_{\substack{F_i\in\mathcal{F}^{\mathrm{A}} \\ v\in\mathcal{D}_i}} x_{it} + \sum_{\substack{F_i\in\mathcal{F}^{\mathrm{M}} \\ v\in\mathcal{D}_i}} x_{it} \right) \le b^{\mathrm{A}}$$

$$\sum_{t\in s+[L]_\circ} \left(\sum_{\substack{F_i\in\mathcal{F}^{\mathrm{D}} \\ v\in\mathcal{D}_i}} y_{it} + \sum_{\substack{F_i\in\mathcal{F}^{\mathrm{M}}\setminus\mathcal{F}^{\rightmoon} \\ v\in\mathcal{D}_i}} y_{it} + \sum_{\substack{F_i\in\mathcal{F}^{\mathrm{M}}\cap\mathcal{F}^{\rightmoon} \\ (v-1)\in\mathcal{D}_i}} y_{it} \right) \le b^{\mathrm{D}}$$

$$\sum_{t\in s+[L]_\circ} \left(\sum_{\substack{F_i\in\mathcal{F}^{\mathrm{A}} \\ v\in\mathcal{D}_i}} x_{it} + \sum_{\substack{F_i\in\mathcal{F}^{\mathrm{D}} \\ v\in\mathcal{D}_i}} y_{it} + \sum_{\substack{F_i\in\mathcal{F}^{\mathrm{M}}\setminus\mathcal{F}^{\rightmoon} \\ v\in\mathcal{D}_i}} (x_{it} + y_{it}) + \sum_{\substack{F_i\in\mathcal{F}^{\mathrm{M}}\cap\mathcal{F}^{\rightmoon} \\ v\in\mathcal{D}_i}} x_{it} + \sum_{\substack{F_i\in\mathcal{F}^{\mathrm{M}}\cap\mathcal{F}^{\rightmoon} \\ (v-1)\in\mathcal{D}_i}} y_{it} \right) \le b^{\mathrm{M}}$$

We use the convention that $x_{it} = y_{it} = 0$ for all $t \notin [n]$ here. Notwithstanding the definition of reference value systems used in practice (cf. Section 4.1.1), one might also want to consider *circular shifting bounds*. Here, the same inequalities apply, but we now consider the index t to be taken modulo n and we apply the bounds for all $s \in [n]$, thus “wrapping” the time windows around the left and right border of the slot set. In practice, the set of slots usually starts at 0:00 and ends at 23:59, and during the early hours of the night there is only little air traffic at German (and generally European) airports, due to night flight restrictions and low passenger volume during this time. So there being no congestion at the “wraparound times”, circular shifting bounds are usually not an issue in applications.

North America Rule

The North America rule introduced in Section 4.1.1 is usually applied as a non-shifting bound. Let (L, b^{D}) be a tuple with $L \in \mathbb{N}$ denoting the length of the North America rule in slots and a bound $b^{\mathrm{D}} \in \mathbb{N}$ on the number of departures with destination in North America. Furthermore, let $\mathcal{F}^{\mathrm{USA}} \subset \mathcal{F}^{\mathrm{D/M}}$ be the set of series requests with destination in North America. Then the North America rule rule can be modeled as

$$\sum_{t \in s+[L]_\circ} \left(\sum_{\substack{F_i \in \mathcal{F}^{\mathrm{USA}} \setminus \mathcal{F}^{\leftmoon} \\ v \in \mathcal{D}_i}} y_{it} \quad + \sum_{\substack{F_i \in \mathcal{F}^{\mathrm{USA}} \cap \mathcal{F}^{\leftmoon} \\ (v-1) \in \mathcal{D}_i}} y_{it} \right) \leq b^{\mathrm{D}}$$

for all $s \in \{1, L+1, 2L+1, \ldots\} \cap [N]$ (or $s \in [N]$, if applied as shifting bound) and all $v \in \{1, \ldots, d\}$.

Notice that for the usual implementation of a "not more than four departures per 15 minutes" North America rule we have to use a discretization of five minutes per slot instead of the usual ten minutes. Alternatively, the rule can be adapted to the ten minute pattern by, e. g., reformulating it as "not more than eight departures per 30 minutes". Our formulation above is flexible enough to support both approaches, and consequently the implementation of the model will support both a five minute and ten minute discretization. However, as the five minute discretization would roughly double the model size without any benefit besides the ability to implement a 15 minutes version of the North America rule, we will normally use the 30 minutes version and stick to a ten minute discretization. We conducted a number of test runs employing a five minute discretization that have shown only marginal differences in the results. Therefore, we cannot deem worthwhile the additional computational efforts required for a five minute discretization for the benefit of a stricter version of the North America rule alone, and will generally resort to the 30 minutes version in our test runs in Section 4.3.

As a side note, notice that a five minute discretization can also have consequences for the shifting bounds in $\mathcal{R}$: Either the bounds can still be applied with time windows starting at *each* slot, giving a somewhat stricter system than with the ten minute discretization. Alternatively, to keep the bounds independent of the discretization, we can define suitable starting slot sets for every time window bound, e. g., one could use only time windows starting at odd slots. To get comparable results, our implementation generally uses that second approach when employing a five minute discretization.

New Entrants Rule

As was argued in Section 4.1.1, neither legal specifications nor the IATA scheduling guidelines are very specific about how to implement the new entrants rule. We therefore proposed two different approaches to the problem, namely taking the "50% of the remaining slots" as an ex-post measure or alternatively estimating the number of slots still available after allocation of those requests that are backed by historic rights. The latter approach is straightforward in the integer programming model, with the constraint

$$\sum_{t\in[n]}\left(\sum_{F_i\in\mathcal{F}^{\mathrm{NE}}}|\mathcal{D}_i|\,(x_{it}+y_{it})\right)\geq K, \tag{4.3}$$

where K is defined by

$$K:=\min\left\{50\%\text{ of the slots remaining after allocation of historic requests,}\right.$$
$$\left.\sum_{F_i\in\mathcal{F}^{\mathrm{NE}}\cap\mathcal{F}^{\mathrm{A/M}}}|\mathcal{D}_i| \quad+\sum_{F_i\in\mathcal{F}^{\mathrm{NE}}\cap\mathcal{F}^{\mathrm{D/M}}}|\mathcal{D}_i|\right\} \tag{4.4}$$

For the determination of K, one can, e. g., compute a preliminary flight schedule or take the maximum number of slots that is potentially available in a flight schedule with the historic flights already integrated, see Section 4.1.1 for a discussion. This method is quite simple from an integer programming viewpoint, as it does not introduce any new variables; as a drawback, it not only requires two flight schedule computations for a thorough estimation of K, but also does not guarantee ex-post validity of the final schedule with respect to the new entrants rule, because the numbers taken from the preliminary schedule or any other estimation method can naturally only be an approximation of the final results.

For the alternative formulation, requiring ex-post validity of the new entrants rule, we can use a formulation like

$$\sum_{t\in[n]}\left(\sum_{F_i\in\mathcal{F}^{\mathrm{NE}}}|\mathcal{D}_i|\,(x_{it}+y_{it})\right)$$
$$\geq\min\left\{\frac{1}{2}\sum_{t\in[n]}\left(\sum_{F_i\in\mathcal{F}\setminus\mathcal{F}^{\mathrm{H/C}}}|\mathcal{D}_i|\,(x_{it}+y_{it})\right),\sum_{F_i\in\mathcal{F}^{\mathrm{NE}}\cap\mathcal{F}^{\mathrm{A/M}}}|\mathcal{D}_i| \quad+\sum_{F_i\in\mathcal{F}^{\mathrm{NE}}\cap\mathcal{F}^{\mathrm{D/M}}}|\mathcal{D}_i|\right\}.$$

Unfortunately, the min-operator induces a nonlinearity into the model, so we will need an additional binary decision variable z_{NE} to model the new entrants rule. A value of $z_{\mathrm{NE}}=1$ indicates that all new entrants' requests are integrated into the flight schedule (thus obliterating the 50% condition), while a value of $z_{\mathrm{NE}}=0$ "activates" the new entrants rule. This can be achieved by the following formulation:

$$\begin{aligned}\sum_{t\in[n]}\left(\sum_{F_i\in\mathcal{F}^{\mathrm{NE}}}|\mathcal{D}_i|\,(x_{it}+y_{it})\right)&\geq z_{\mathrm{NE}}\cdot\left(\sum_{F_i\in\mathcal{F}^{\mathrm{NE}}\cap\mathcal{F}^{\mathrm{A/M}}}|\mathcal{D}_i| \quad+\sum_{F_i\in\mathcal{F}^{\mathrm{NE}}\cap\mathcal{F}^{\mathrm{D/M}}}|\mathcal{D}_i|\right)\\ \sum_{t\in[n]}\left(\sum_{F_i\in\mathcal{F}^{\mathrm{NE}}}|\mathcal{D}_i|\,(x_{it}+y_{it})\right)&\geq\frac{1}{2}(1-z_{\mathrm{NE}})\sum_{t\in[n]}\left(\sum_{F_i\in\mathcal{F}\setminus\mathcal{F}^{\mathrm{H/C}}}|\mathcal{D}_i|\,(x_{it}+y_{it})\right)\end{aligned}\tag{4.5}$$

In practice, "constraint switching" variables like z_{NE} are generally undesirable, because they can have rather negative effects on the strength of the linear programming relaxation. Hence one often uses alternative formulations that, while not capturing the real constraint in the same concise way as the original formulation, may lead to much better LP relaxations without giving away too much in the applications. An example in this context would be to use the "estimation approach" as in (4.3) and (4.4). Also, when the number of new entrants requests is low (as is frequently the case in practice), one would just perform the computations without using any kind of new entrants rule and instead treat new entrants like historic flights (more precisely, CR class flights with a ± 60 minutes scheduling interval around the requested slots), which should be a very accurate and fast approach in practice. Technically speaking, this corresponds to the case of $z_{\mathrm{NE}} = 1$ in (4.5).

Coupling of Flight Requests, Hub and Spoke Flights

For hub and spoke flights comprising of one hub flight (intercontinental flight) and several (regional) feeder flights, two additional constraints must be modeled:

- Only schedule the hub flight, if a minimum number of feeder flights are integrated.
- If the hub flight is not integrated, none of the feeder flights should be integrated.

For a hub flight $F_{i_0} \in \mathcal{F}$, denote the corresponding feeder flights by $F_{i_1}, \dots, F_{i_p}$ and let $q \in \mathbb{N}$, $q \leq p$ be the minimum number of feeder flights required to schedule the hub flight. Then the following constraints are added to the model:

$$\sum_{t\in[n]} y_{i_0 t} \leq \frac{1}{q} \cdot \sum_{i\in\{i_1,\dots,i_p\}} \left(\sum_{t\in[n]} x_{it} \right)$$
$$\sum_{i\in\{i_1,\dots,i_p\}} \sum_{t\in[n]} p_{it} \leq k \cdot \sum_{t\in[n]} y_{i_0 t}$$

Objective Functions

As outlined before in Section 4.1, there are basically two different objective functions that are relevant in practice. As a first goal, an airport is interested in maximizing the number of flight movements that take place, because every movement generates direct income in the form of landing and starting fees as well as indirect income via the use of ground handling services, fuel sales and the use of airport infrastructure such as shops or restaurants by passengers. The corresponding objective function is

$$\max \sum_{t\in[n]} \left(\sum_{F_i\in\mathcal{F}} (x_{it} + y_{it}) \right) \quad \text{or} \quad \max \sum_{t\in[n]} \left(\sum_{F_i\in\mathcal{F}} w(F_i) \cdot (x_{it} + y_{it}) \right),$$

respectively, where $w : \mathcal{F} \to \mathbb{R}_{\geq 0}$ is an arbitrary weight function. The most important example is the case where

$$w(F_i) := |\mathcal{D}_i|\,,$$

measuring the total number of flight movements. As an alternative, one can also multiply $w(F_i)$ with the number of passengers expected for each flight F_i or with the MTOW.

The alternative approach described in Section 4.1 is a minimization of cost terms. To this end every series request is "equipped" with a cost vector $c = (c^{\mathrm{A}}, c^{\mathrm{D}}, c^{\mathrm{M}})$ allowing for fine-grained control of the costs on a per-flight basis. The corresponding objective function is

$$\begin{aligned}
\min \sum_{F_i \in \mathcal{F}^{\mathrm{A/M}}} c_i^{\mathrm{A}} \cdot & \left(\sum_{t \in \left[T_i^{\mathrm{A}}, T_i^{\mathrm{A}} + S_i^{\mathrm{A}}\right]} x_{it}(t - T_i^{\mathrm{A}}) \quad + \sum_{t \in \left[T_i^{\mathrm{A}} - S_i^{\mathrm{A}}, T_i^{\mathrm{A}}\right]} x_{it}(T_i^{\mathrm{A}} - t) \right) \\
+ \sum_{F_i \in \mathcal{F}^{\mathrm{D/M}}} c_i^{\mathrm{D}} \cdot & \left(\sum_{t \in \left[T_i^{\mathrm{D}}, T_i^{\mathrm{D}} + S_i^{\mathrm{D}}\right]} y_{it}(t - T_i^{\mathrm{D}}) \quad + \sum_{t \in \left[T_i^{\mathrm{D}} - S_i^{\mathrm{D}}, T_i^{\mathrm{D}}\right]} y_{it}(T_i^{\mathrm{D}} - t) \right) \\
+ \sum_{F_i \in \mathcal{F}\mathrm{A}} c_i^{\mathrm{M}} \cdot & \left(1 - \sum_{t \in [n]} x_{it} \right) + \sum_{F_i \in \mathcal{F}^{\mathrm{D}}} c_i^{\mathrm{M}} \cdot \left(1 - \sum_{t \in [n]} y_{it} \right) \\
+ \sum_{F_i \in \mathcal{F}^{\mathrm{M}}} c_i^{\mathrm{M}} \cdot & \left(1 - \sum_{t \in [n]} x_{it} \right).
\end{aligned}$$

Notice that the first two lines describe the cost of deviation from the desired arrival and departure slot respectively, the third line represents the cost of not integrating an arrival or departure series request and the fourth line adds the non-integration cost for a mixed series request. Here, it suffices to consider the x_{it} variables for ever series request $F_i \in \mathcal{F}^{\mathrm{M}}$, because these sum up to one if and only if the same holds for the y_{it} variables. For a series request that cannot be integrated, the first two lines have a value of zero, while for an integrated flight the last two lines become zero.

4.2.4 Size of the Integer Programming Formulation

To give an impression of the enormous number of variables and constraints we are dealing with, we will now roughly analyze the problem size and give some numbers from real world data that will later be used for computational evaluations of our model.

The number of variables is of course bounded by $2\,|\mathcal{F}| \cdot |[n]| + 1 = 2nm + 1$ (the "$+1$" is for the z_{NE} variable used in the ex-post formulation of the new entrants rule), but usually there are much less nonzero variables, namely

$$2 \sum_{F_i \in \mathcal{F}} \left(\left|S^{\mathrm{A}}\right| + \left|S^{\mathrm{D}}\right| \right) + 1.$$

For the constraints, there is one allocation constraint per arrival or departure series request and two allocation constraints per mixed series request, totaling to $|\mathcal{F}^{\mathrm{A}}| + |\mathcal{F}^{\mathrm{D}}| + 2|\mathcal{F}^{\mathrm{M}}|$ constraints. For the mixed series requests, we also need a constraint coupling arrival and departure, so we get an additional $|\mathcal{F}^{\mathrm{M}}|$ constraints. Strictly spoken, these constraints are only necessary for non-historic flight requests, hence the correct number is $|\mathcal{F}^{\mathrm{M}} \cap (\mathcal{F}^{\mathrm{NE}} \cup \mathcal{F}^{\mathrm{I}})|$. For each mixed series request, there are also minimum and maximum ground time constraints, making up for $2|\mathcal{F}^{\mathrm{M}}|$ constraints of this type. The new entrants rule can be formulated using just two constraints, the hub and spoke constraints also need two constraints for each hub and spoke combination. A major fraction of the constraints is contributed by the reference value system constraints. For every bound $(L, b) \in \mathcal{R}$ there are 3 constraints for every slot in $[1,\ N - L]$, so each shifting bound contributes a total of $3(N - L)$ constraints, approximately summing up to $3N|\mathcal{R}|$ constraints. Of similar size are the North America rule constraints, contributing $\lfloor N/L^{\mathrm{USA}} \rfloor$ inequalities for the (usual) non-shifting case (where the respective bound is denoted by $(L^{\mathrm{USA}}, b^{USA})$). Table 4.1 summarizes the above numbers.

constraint type	number of constraints
allocation constraints	$\left\|\mathcal{F}^{\mathrm{A}}\right\| + \left\|\mathcal{F}^{\mathrm{D}}\right\| + 2\left\|\mathcal{F}^{\mathrm{M}}\right\|$
arrival/departure coupling	$\left\|\mathcal{F}^{\mathrm{M}} \cap \left(\mathcal{F}^{\mathrm{NE}} \cup \mathcal{F}^{\mathrm{I}}\right)\right\|$
ground time	$2\left\|\mathcal{F}^{\mathrm{M}}\right\|$
new entrants rule	2
h hub and spoke connections	$2h$
shifting boundary system	$3\sum_{(L,b)\in\mathcal{R}}(N-L)$
North America rule $\left(L^{\mathrm{USA}}, b^{USA}\right)$	$\left\lfloor \frac{N}{L^{\mathrm{USA}}} \right\rfloor$

Table 4.1: The number of constraints for the flight scheduling problem for d days, with n slots per day and $N = nd$ total slots.

Altogether, the problem consists of a total of

$$\left(\left|\mathcal{F}^{\mathrm{A}}\right| + \left|\mathcal{F}^{\mathrm{D}}\right| + 2\left|\mathcal{F}^{\mathrm{M}}\right|\right) + \left|\mathcal{F}^{\mathrm{M}} \cap \left(\mathcal{F}^{\mathrm{NE}} \cup \mathcal{F}^{\mathrm{I}}\right)\right| + 2\left|\mathcal{F}^{\mathrm{M}}\right| \\ + 2 + 2h + 3\sum_{(L,b)\in\mathcal{R}}(N-L) + \left\lfloor \frac{N}{L^{\mathrm{USA}}} \right\rfloor = \mathcal{O}\left(|\mathcal{F}| + N\,|\mathcal{R}|\right)$$

constraints, assuming a non-shifting North America rule and $h \leq |\mathcal{F}|$.

To get an idea of the size of a real world instance, we state some approximate numbers for one of the major German airports for the planning season of winter 2004, which consisted of about 180 days. We will describe this data set in more detail later. For that airport, the coordinator received about 5 330 series requests consisting of roughly 190 000 movements). Roughly 60% of those requests are based on historic claims of different classification. The data we received contained neither new entrants status information nor hub and spoke connections, so these cannot be considered. The reference value system applied for that winter season of 2004 consisted of three shifting bounds of lengths 10, 30 and 60 minutes. Using a ten minute discretization we get $n = 6 \cdot 24 = 144$ slots per day, hence $144 \cdot 180 = 25\,920$ slots for the complete season. As a rough estimate, this data set generates in the order of magnitude of 150 000 binary variables and about 300 000 constraints for a typical real-world instance.

4.3 Computational Results for Flight Scheduling

In this section we will apply our model to several real-world instances to evaluate its performance in practice. A discussion of the outcomes of our test cases will show that the results obtained in Chapter 3 are to some extent still applicable even for the much more complex model.

4.3.1 Notes on the Test Environment

Before we present the results of practical tests on various instances, a few words on the implementation and the test environment are in order. The model was implemented using Dash Xpress-MP Release 2007A (64 Bit Version) and the Mosel modeling language (see [Dash07a; Dash06c; Dash06b; Dash07b; Dash06d; Dash06a] for details on both). A user interface and data processing (called SOFIE, see Section 4.5) was implemented in Java Standard Edition 6 (cf. [Sun06]), the data was stored in a PostgreSQL database (cf. [PSQL05]) running on a Sun Blade 1000 workstation on Solaris 10. The user interface and data processing was mainly performed on an IBM ThinkPad T42p equipped with a 2.0 GHz Pentium M processor and 2 GB RAM. The model runs on Xpress-MP Optimizer were performed using either the same ThinkPad computer or a Sun SPARC Fire V440 equipped with 4 UltraSPARC-IIIi processors at 1.3 GHz and 16 GB of RAM running Solaris 10. The reason for using the Sun was mainly the enormous size of some of the instances which required memory beyond 4 GB, hence a 64 Bit capable machine and operating system had to be employed.

We will generally optimize with the size of the resulting flight schedule as the objective. The results for the cost objective are very similar, we will discuss that for a specific example in Section 4.3.3, where we explicitly compare size and cost objective.

4.3.2 Test Instances for the Airport Flight Scheduling Problem

First, let us briefly comment on the instances that we use to test our integer programming model described in the preceding section. We have mainly worked with two large test instances differing in various aspects and we will report on the results for both of them (and also for some subsets to illustrate various effects).

Instance W04

The instance “Winter 2004” or *W*04 for short was prepared in cooperation with Frankfurt/Main airport and Fraport AG, the airport’s operating company. The data is based on real series requests for the winter season of 2004. Using flight data from the preceding winter and summer season, we identified historic rights and classified them as class H or class CR historic requests. We also tried to match arrivals and departures in order to create mixed series requests. This resulted in a data set with the characteristics summarized in Table 4.2. Of course, for the publication of this thesis the data has been anonymized, but without destroying its structural properties, so this data set is a very realistic example of possible input data for the Flight Scheduling problem.

Figure 4.1a shows the average distribution of the flight requests within one day. Obviously, flight activity starts at about 4:00 in the morning and rises almost linearly in the moving average before reaching a steady level at 8:00. It then stays more or less constant up to 21:00, when it starts to decrease, again roughly linearly in the moving average. Unfortunately, the data does not include information about new entrant status nor about hub and spoke connections. We can, however, incorporate the correct reference value system for our test runs, namely the system that was in effect for the winter 2004 season at Frankfurt/Main airport. The reference value system is summarized in Table 4.4.

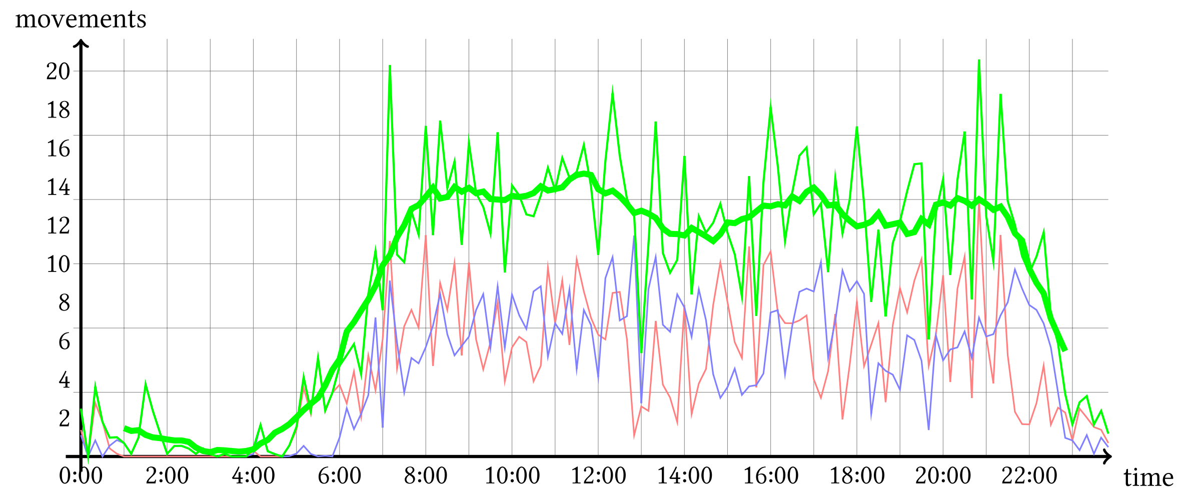
Instance S08

The second data set “Summer 2008” or *S*08 for short is a test set with randomized data designed to evaluate typical structural properties of optimal solutions without reference to a specific airport, its infrastructure and demand structure.[5] The generated data is based on the timetable of Düsseldorf airport, one of Germany’s major airports, yet not among Europe’s leading international hubs, so a fairly “average structure” (in a very informal sense) should be expected from that timetable. We anonymized the data by deleting the destination or departure airports and the specific date, only keeping the day of week and the time of arrival/departure. We then generated arrival and departure series requests for all times and weekdays in the data and mapped all requests to one single week[6] to obtain the instance S08. Thus, S08 provides for an instance reflecting the typical demand curve within a week fairly accurately.

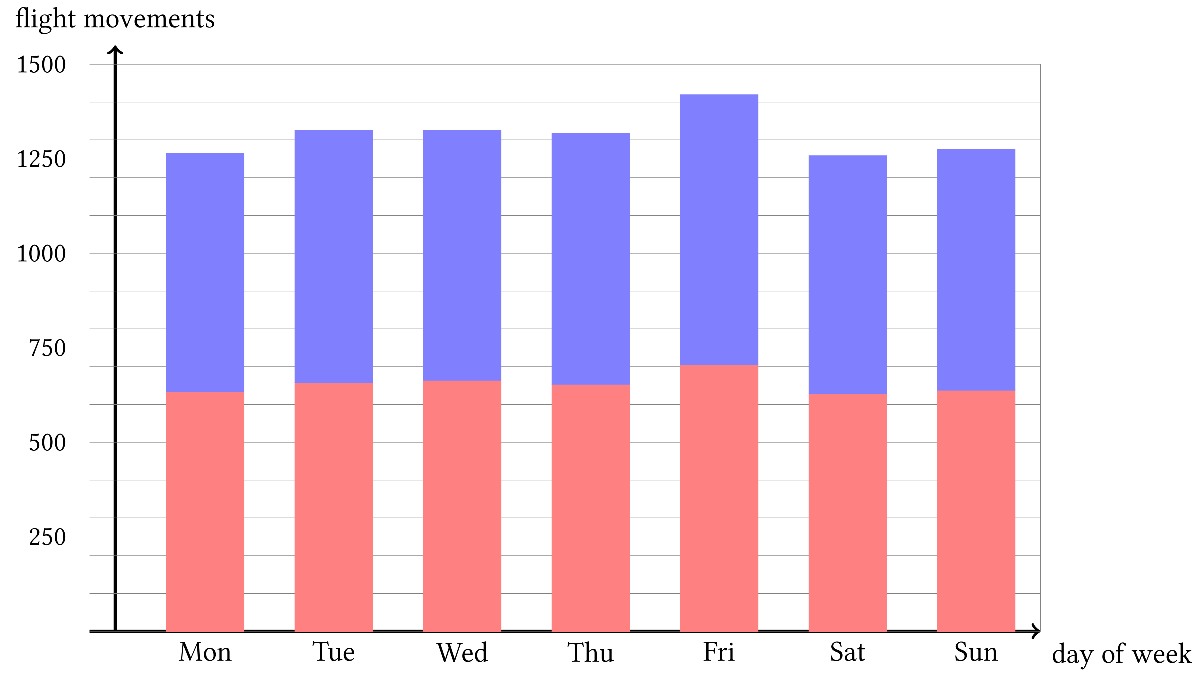
However, as only arrival/departure time and weekdays are available to the public, all requests in S08 are arrival or departure series requests, there are no mixed series requests. For the same reason, no historic rights, no new entrant status information and no hub-and-spoke connection information is available in this instance. The characteristics of S08 are summarized in Table 4.3, while Figure 4.2b shows the distribution of arrivals and departures for the different weekdays. It can be seen that Saturday is a very low traffic day, closely followed by Sunday. In contrast, Friday is clearly the day with most requests, although the difference to other weekdays is not too large.

[5]As an example, both Frankfurt/Main and München airport are large hubs providing for many international connections. One should expect this fact to have an influence on the demand structure at these airports.

[6]To be precise, we chose the week from Monday, Apr 14, 2008, to Sunday, Apr 20, 2008. This choice was made for no particular reason, though.



(a) Average distribution of flight requests over the day with two hours central moving average for the total movements (bold line).



(b) Flight distribution by weekday.

🟥 Arrivals 🟦 Departures 🟩 Total Movements

Figure 4.1: Statistics for the instance W04.

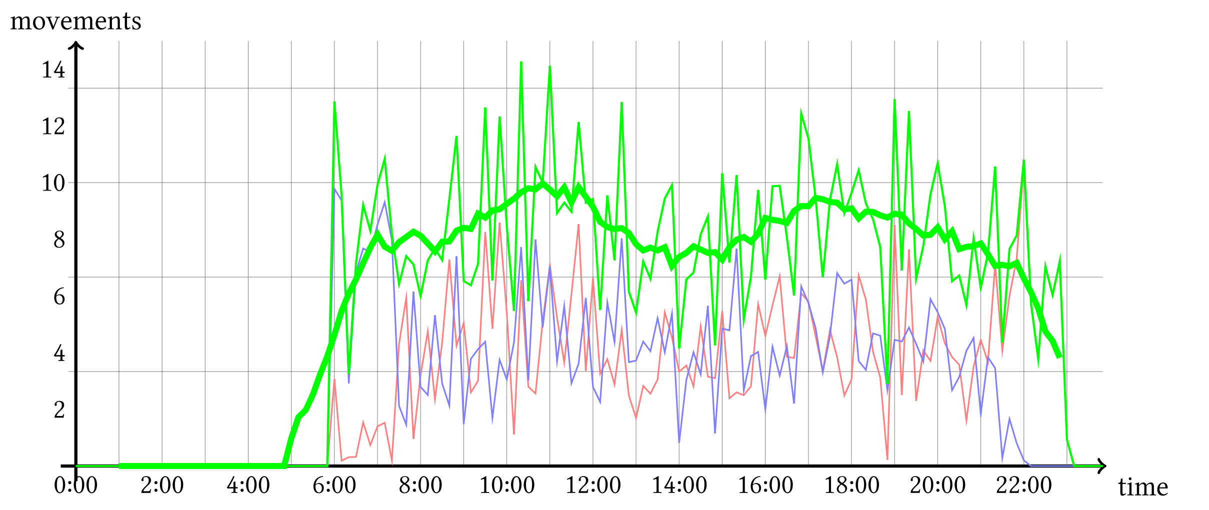
duration of planning period	
absolute dates	from Nov 1, 2004, to Mar 26, 2005
in days	146
in slots	21 024
total number of series requests	5 331
mixed	1 458
arrival	1 967
departure	1 906
North America departure	93
average series length	31.1 movements
average mixed series length	122.2 movements
total number of movements requested	198 003
arrival movements	98 892
departure movements	99 111
average movements per day	1 356.2
average movements per slot	9.4
requests with historic rights	3 468
historic classification H	1 655
historic classification CR	1 813
Maximum difference between “only arrivals” and “only departures”	
for the complete season	10
for each day	10

Table 4.2: Characteristics of the W04 instance.

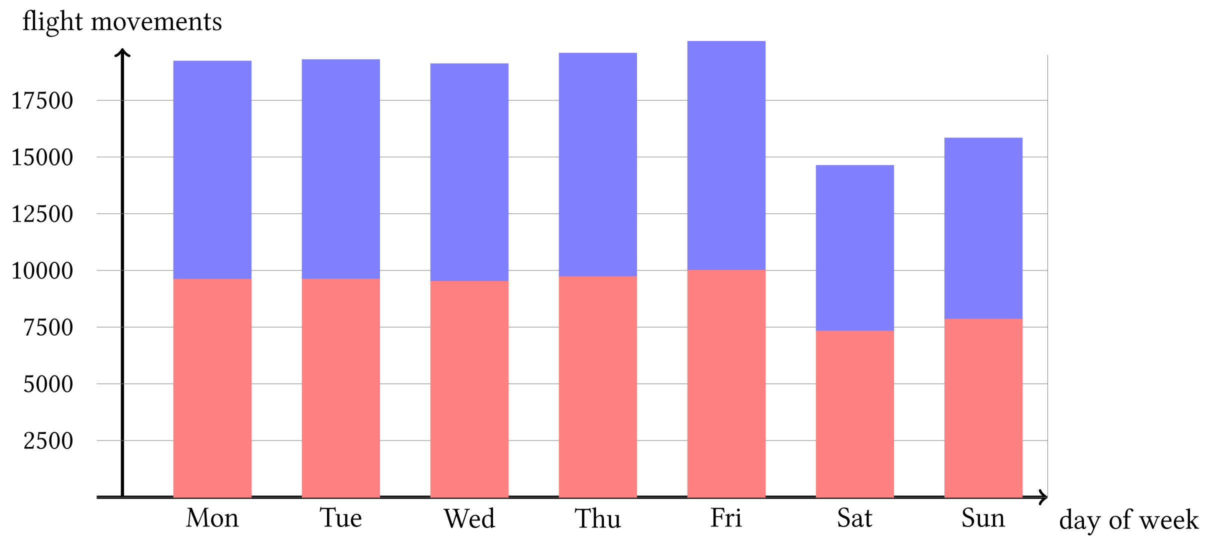
In addition, Figure 4.2a displays the average flight request distribution throughout the day. There is virtually no activity before 4:00 in the morning, then the moving average quickly rises on an approximately linear path until 7:00. Activity stays at about the same level on average between 7:00 and 22:00, after that a decline begins, again roughly linear in the moving average.

Reference Value Systems

Tables 4.4 and 4.5 summarize the reference value systems that will be used for our computational tests. The first system (RW04) is especially suited for use with the instance W04, as it reflects the true shifting bounds as they were in effect at Frankfurt/Main airport during the winter season 2004. Thus the results obtained with W04/RW04 may be compared to the initial schedule proposed by the airport coordinator for Frankfurt/Main airport. Assuming an arbitrary number of single arrival and



(a) Average distribution of flight requests over the day with two hours central moving average for the total movements (bold line).



(b) Flight distribution by weekday.

🟥 Arrivals 🟦 Departures 🟩 Total Movements

Figure 4.2: Statistics for the instance S08.

duration of planning period	
absolute dates	from Apr 14, 2008, to Apr 20, 2008
in days	7
in slots	1 008
total number of series requests	127 723
mixed	*none*
arrival	63 937
departure	63 786
North America departure	*none*
average movements per day	18 246.1
average movements per slot	126.7
requests with historic rights	*none*
Maximum difference between "only arrivals" and "only departures"	
for the complete season	10
for each day	10

Table 4.3: Characteristics of the S08 instance.

departure requests for every slot of the day we could, without any additional constraints besides the reference value system, achieve a maximum of $6 \cdot 78 + 8 \cdot 80 + 7 \cdot 81 + 1 \cdot 82 + 2 \cdot 78 = 1\,913$ flight movements per day. Taking into account the demand structure, the restrictions on night flight and other relevant constraints as outlined in Section 4.1.1, an optimal flight schedule will naturally contain less flights.

time of day	R10A	R10D	R10M	R30A	R30D	R30M	R60A	R60D	R60M
0 : 00 – 5 : 59	9	9	16	23	25	43	43	48	78
6 : 00 – 13 : 59							41	43	80
14 : 00 – 20 : 59							42	44	81
21 : 00 – 21 : 59							43	50	82
22 : 00 – 23 : 59							43	48	78
North America rule: At most four departures per 15 minutes, non-shifting.									

Table 4.4: Reference value system RW04, empty cells mean the value does not change.

The reference value system RS08 (cf. Table 4.5), on the other hand, is not tight enough to be used with the W04 instance (that instance does not contain enough series requests to make the optimization with S08 a challenge), but it is very well suited for computations with the instance

S08 as the test set. Also, we will come back to RS08 in the context of Section 4.4, where we discuss alternative reference value systems. The reference value system RS08 allows for a maximum of $24 \cdot 90 = 2\,160$ flight movements per day. Given a typical demand curve with high demand roughly from 6:00 to 22:00, we should expect at least $16 \cdot 90 = 1\,440$ flight movements per day in an optimal solution.

time of day	R10A	R10D	R10M	R30A	R30D	R30M	R60A	R60D	R60M
0 : 00 – 23 : 59	12	12	15	—	—	—	58	58	90

Table 4.5: Reference value system RS08, "—" means no bound applies.

4.3.3 Optimizing the Instance W04 with Reference Value System RW04

Complete Season

The instance W04 with the corresponding reference value system RW04 is of particular interest, because it is not only a real-world data set, but even one where we have a benchmark value: We can compare our results with the total number of movements in the airport coordinator's initial proposal, which contained 193 430 flight movements for the whole planning horizon. Note, however, that no details on this proposal are known and that it does not represent the final schedule as it was implemented, because every initial proposal is afterwards changed by a series of negotiations between the affected parties. Table 4.6 summarizes the results of the test run for our model, the approach outperforms the coordinator's initial proposal by 2.1%. While this might not seem a large improvement at first, the absolute increase of 4 024 flight movements highlights the value of our approach. Taking into account that an airport charges a considerable fee for every landing and take-off, an increase by more than 4 000 flight movements means a directly proportional increase in fee income (plus added revenue due to other effects such as ground handling services, passenger handling, etc.). The computation of the complete season needed about five hours of CPU time on a single Sun SPARC CPU, as opposed to about six weeks of mainly manual work for the coordinator.

In order to get some insight into the structural properties of the optimal solution, a visualization of some particular aspects will be helpful. Figure 4.3 shows the number of arrivals, departures and total movements per day for the complete season. The number of arrivals and departures varies roughly between 600 and 700 flights, with — up to small variations — equal numbers of arrivals and departures. One can clearly identify a weekly pattern with a small peak on Fridays and significantly less traffic on Saturdays. This pattern can be seen in detail in Figure 4.4b, which shows the number of arrivals, departures and total flight movements for a typical (i. e., outside of the holiday season) week within the winter 2004 season.

A more detailed view is provided by Figure 4.4a, which shows arrivals and departures together with the values of `R10M`, `R30M` and `R60M` shifting bounds for the same week. The values shown for

duration of planning period	Nov 1, 2004, to Mar 26, 2005
duration in days/slots	146 / 21 024
series scheduled	5 271 of 5 331
movements scheduled	197 454 of 198 003
arrival movements	98 725 of 98 892 (65 013 paired)
departure movements	98 729 of 99 111 (65 013 paired)
average movements per day	1 352.4
average movements per slot	9.4
percentage of request series scheduled	98.9%
percentage of requested movements scheduled	99.7%
improvement with respect to total movements	4 024 movements or 2.1%

Table 4.6: Solution summary: Instance W04 with reference value system RW04.

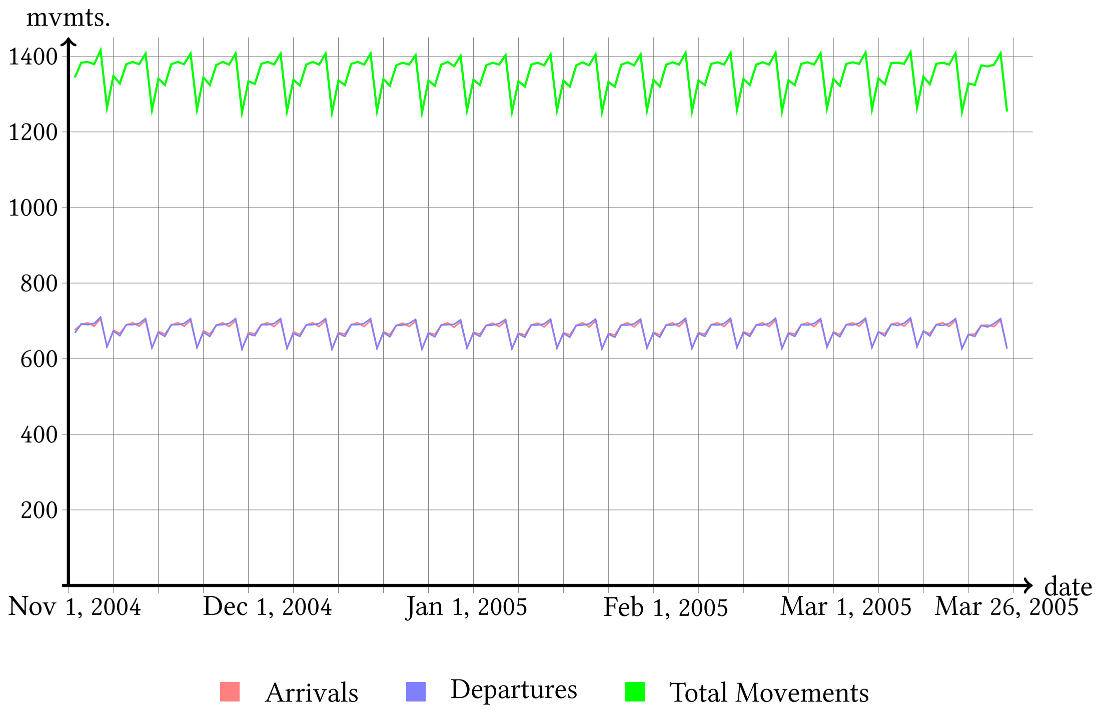


Figure 4.3: Optimization results for the complete season of instance W04 with reference value system RW04.

the shifting bounds represent the number of flight movements within the time window starting at the respective time. The varying value of R60M is clearly visible, and again Friday can be identified as peak traffic day, whereas traffic is considerably lower on Saturday and also a little lower on Monday.

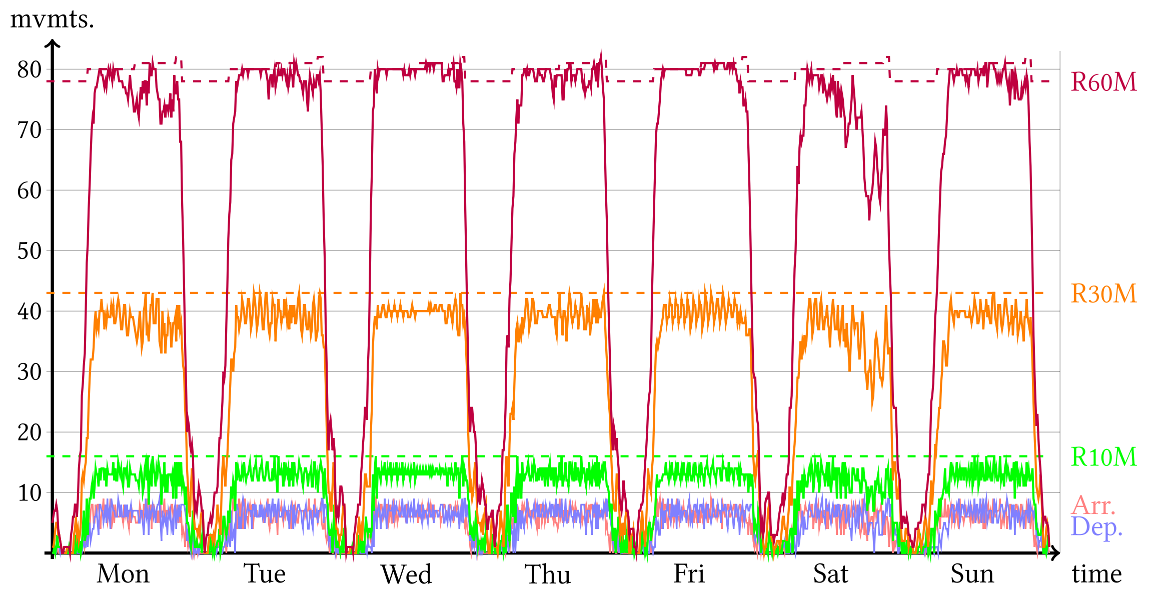
Finally, let us take a closer look at three days of the above week, namely Tuesday (as a prototypical day for the complete season), Friday (the peak day) and Saturday (a low-traffic day).

The chart for Tuesday (Figure 4.5) shows that both arrivals, departures and total movements stay within a corridor of about five movements width for the majority of the day, i. e., roughly from 7:00 to 22:00. While the fact that many flights take place during these hours is not at all surprising,[7] the relatively narrow corridor reflects a uniformity that one might not have anticipated. This uniformity is even more apparent in the R30M values that also move within a corridor of about 7 flights width, a mere 16% with respect to the bound value of 43. Finally, the R60M curve shows the same tendency, with a maximum fluctuation of about 8% with respect to the bound value during daytime. This behavior is very much in accordance with our theoretical considerations in Chapter 3, where we saw that maximum slot packings correlate to uniform slot configurations, whereas minimum slot covers (and thus bad flight schedules in the present context) exhibit a "peaks and valleys" structure. Hence, although the model and data considered here are much more complex than the Maximum Slot Packing problem we considered in Chapter 3, the principal findings still seem to hold within reasonable limits.

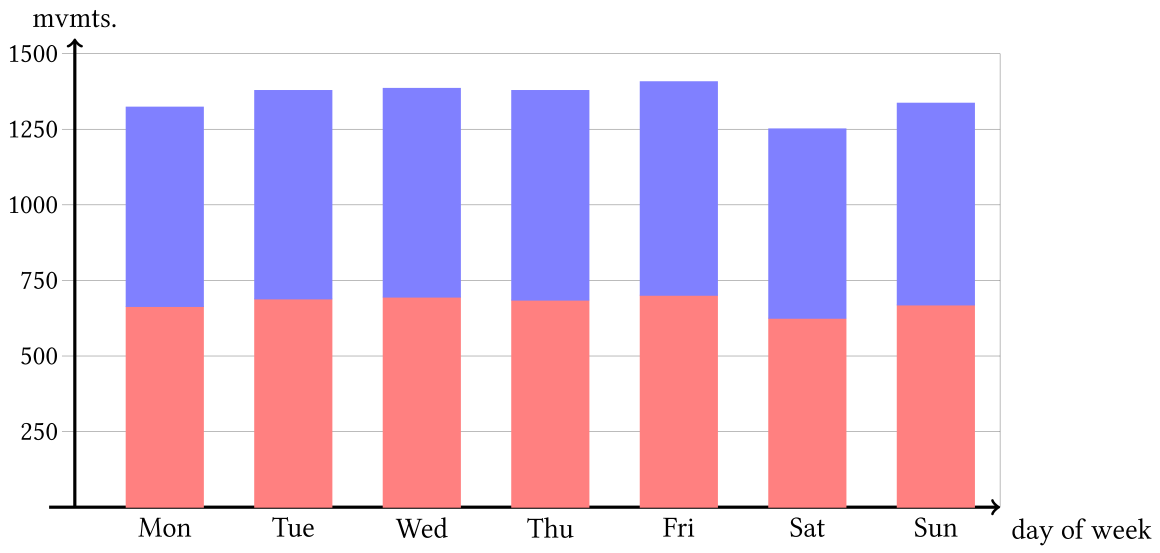
A look at Figures 4.6 and 4.7 for Friday and Saturday, respectively, reveals generally the same structural behavior as discussed for Tuesday. The structure is even more pronounced on Friday, because the very high demand on that day does not leave much room for allocations leading to a suboptimal slot configuration. The fluctuation of the R10M curve is considerably lower, the R60M is almost constantly at maximum level during daytime. This trend is not so apparent for the R30M curve, which exhibits roughly the same amount of fluctuation as for Tuesday. This can be explained by the fact that R30M is a relatively "loose" bound compared to R60M and thus does not have too much influence on the outcome. Basically, the global bound is apparently governed by R60M, while the local structure is mainly due to R10M, leaving little impact for the R30M rule. Notice that these findings are again in accordance with Chapter 3, where we also found that for a monotone reference value system (which RW04 is) the relatively strictest (and for that matter also longest) shifting bound alone governs the optimal objective value.

With all this said, it remains to say that Saturday (cf. Figure 4.7) also displays the uniform behavior of the other days for the first half of the day, while showing a steep descent for the second half, starting at around 13:00. This is simply due to the fact that there are far less flight requests for Saturday afternoon than for any other day — business travel is very low during that time of week (very much in contrast to the Friday afternoon peaks), and most private flight travel also occurs on days other than Saturday, cf. Figure 4.1b for an illustration of the number of requested

[7]In addition to usual business hours, recall that strict night flying restrictions are in place at Frankfurt/Main airport. While we have not integrated these into our model (which would just be a matter of imposing a suitable shifting bound for the respective time of day), the airlines certainly anticipate the restrictions in their requests.



(a) Flight distribution and shifting bound values for `R10M`, `R30M` and `R60M`.



(b) Flight distribution.

🟥 Arrivals 🟦 Departures 🟩 Total Movements

Figure 4.4: Optimization results for the complete season of instance W04 with reference value system RW04. Displayed are the results of the week from Mon, Dec 6, 2004, to Sun, Dec 12, 2004.

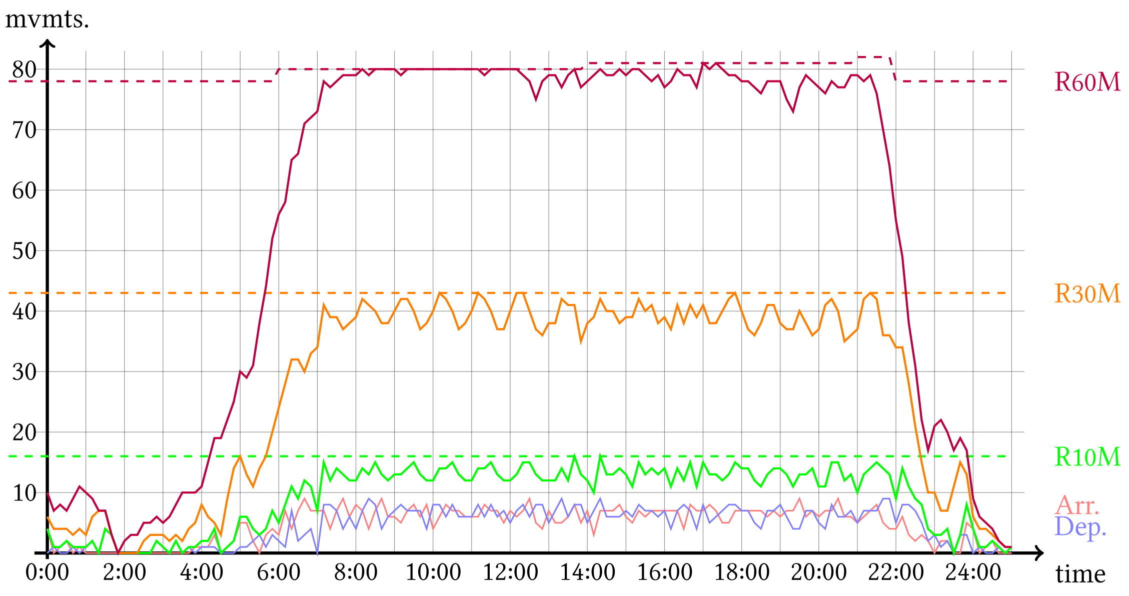


Figure 4.5: Flight distribution and shifting bound values for R10M, R30M and R60M for Tue, Dec 7, 2004, as prototypical "average traffic day" for the instance W04 with reference value system RW04.

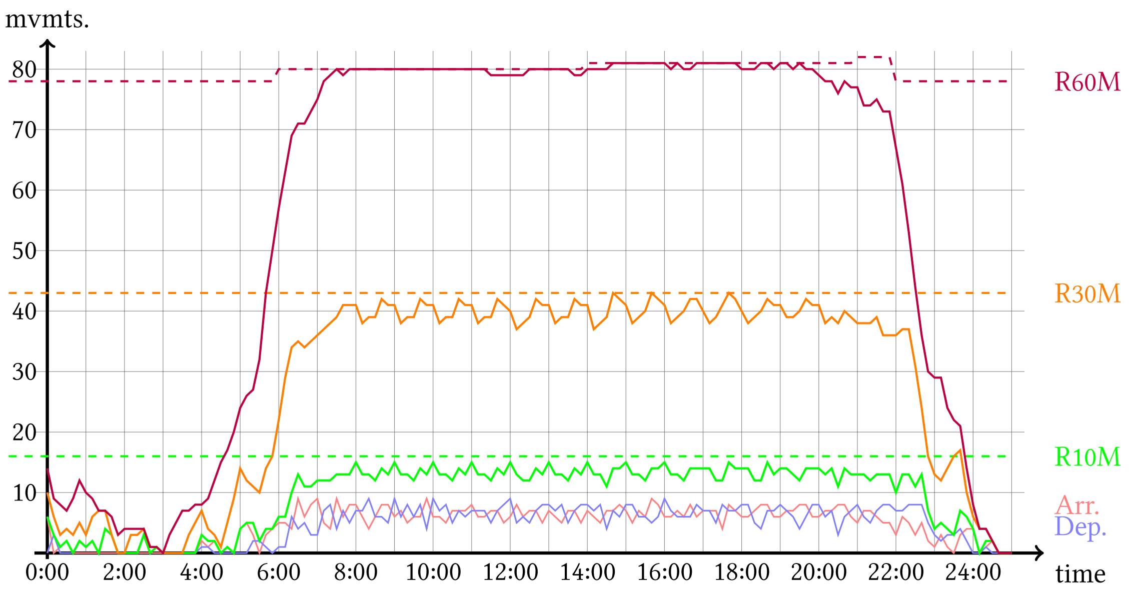


Figure 4.6: Flight distribution and shifting bound values for R10M, R30M and R60M for Fri, Dec 10, 2004, as prototypical "high traffic day" for the instance W04 with reference value system RW04.

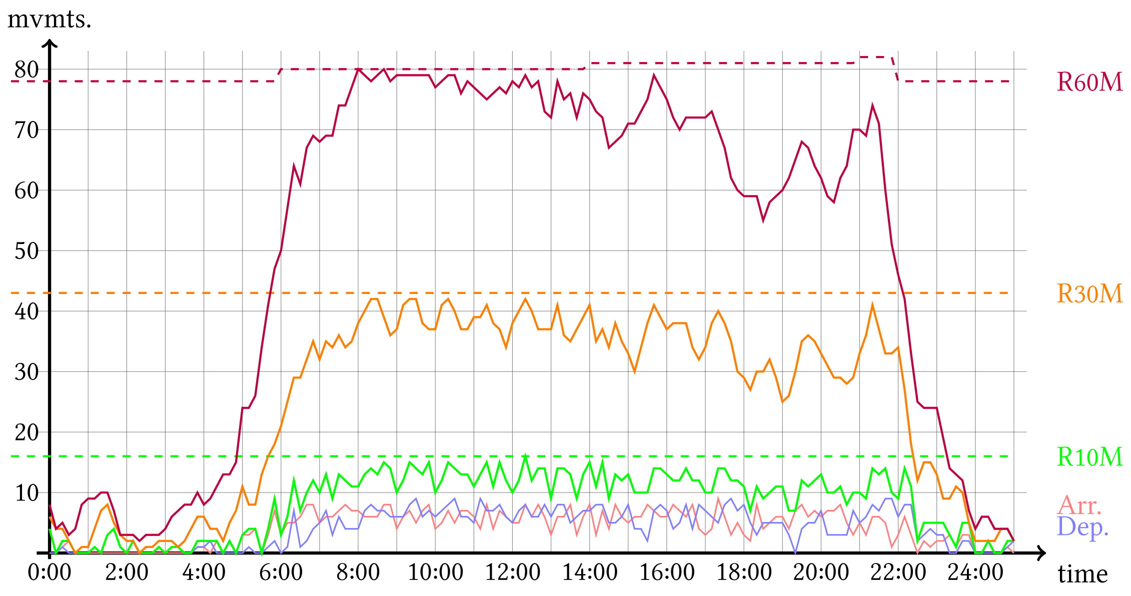


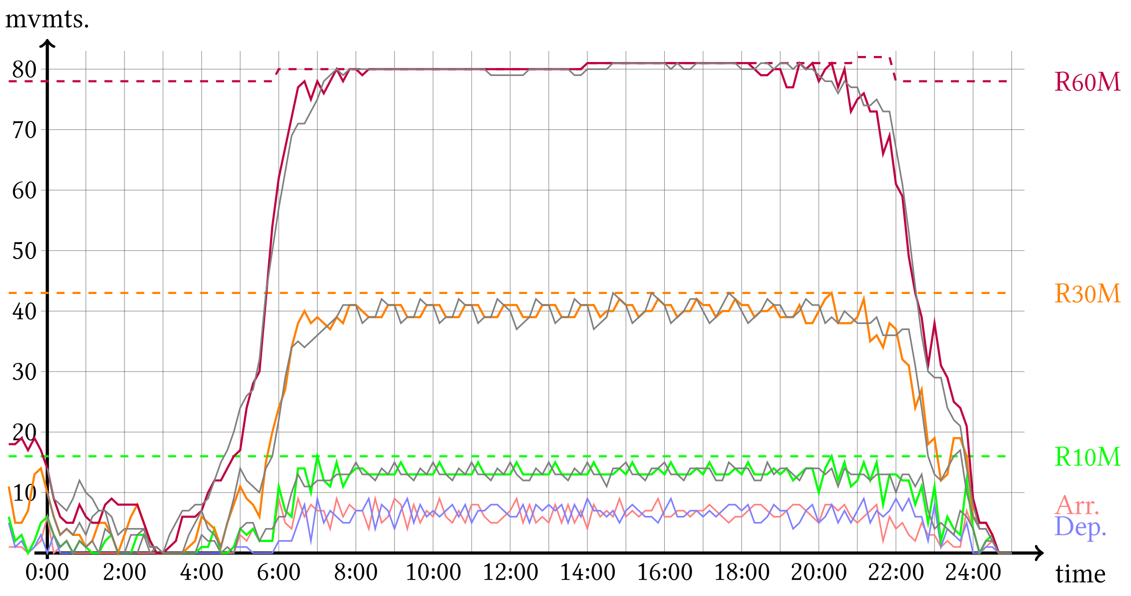
Figure 4.7: Flight distribution and shifting bound values for `R10M`, `R30M` and `R60M` for Sat, Dec 11, 2004, as prototypical "low traffic day" for the instance W04 with reference value system RW04.

flights by weekdays.

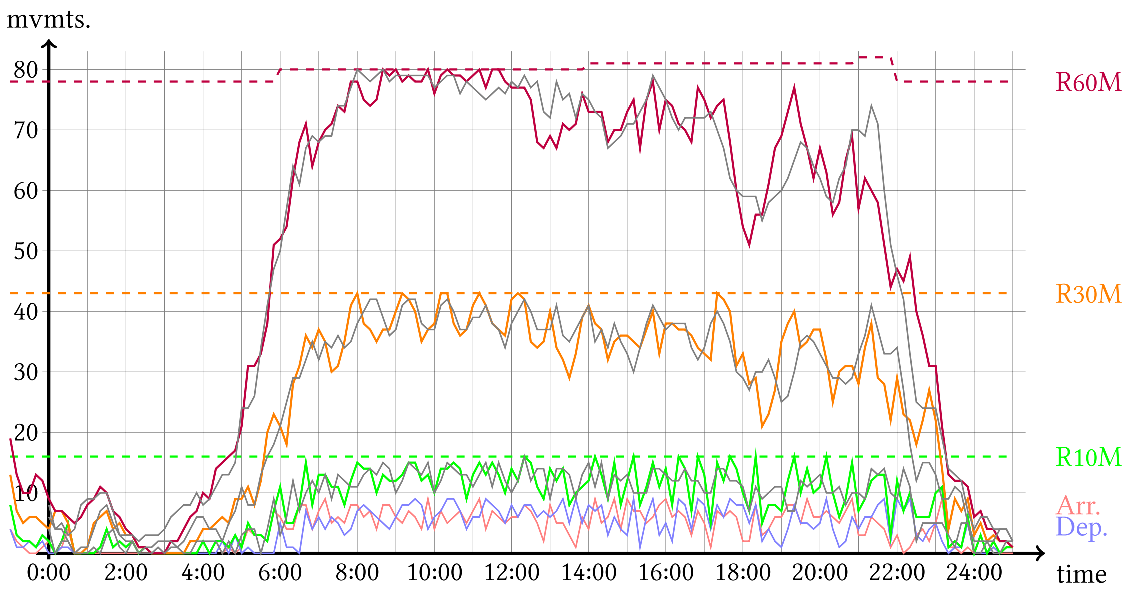
Single Day

In order to systematically investigate the intra-day structure of optimal flight schedules and the effect of long series requests, we will now concentrate optimization on a single day instead of computing a complete season. Thus constraints on days other than the one considered can have no effects on the optimum. With an average series length of 122 days, one might indeed wonder whether long flight series have a negative effect on the objective value. For our test, we will consider two specific dates, namely Friday, Dec 10, 2004, and Saturday, Dec 11, 2004, of the instance W04, using the corresponding reference value system RW04. Thereby we will not only have both an example for a high-traffic and for a low-traffic day, but we are also able to compare the results to the above computations for the complete season and hence estimate to which extent flight series influence the objective value as well as the structure of optimal solutions. All computations in this subsection were performed on the IBM ThinkPad and needed about 30 to 60 seconds of CPU time for each test run.

Tables 4.7 and 4.8 summarize the results. We again allowed for a total difference of ten movements between arrival and departure movements for comparison with our earlier results. The difference to the optimal flight schedule for the complete season is a mere two flights for Friday, Dec 10, 2004. This suggests that the effects of flight series are almost insignificant for the objective value. However, comparing Figure 4.6 and Figure 4.8a (for easier reference, the `R10M`, `R30M` and



(a) Fri, Dec 10, 2004



(b) Sat, Dec 11, 2004

Figure 4.8: Flight distribution and shifting bound values for `R10M`, `R30M` and `R60M` for single day optimization of instance W04 with reference value system RW04.

duration of planning period	Fri, Dec 10, 2004
duration in slots	144
series scheduled	1 409 of 1 419
movements scheduled	1 409 of 1 419
arrival movements	702 of 706
departure movements	707 of 713
average movements per slot	9.8
percentage of requests scheduled	99.3%
improvement with respect to complete season	2 movements or 0.1%

Table 4.7: Solution summary: Fri, Dec 10, 2004, of instance W04 with RW04.

`R60M` values from Figure 4.6 are displayed in gray in Figure 4.8a) reveals a subtle difference especially in the `R10M`, `R30M` and `R60M` lines: The decrease at the end of the day is faster in the single day optimum and the schedule seems to be a little "more tight", i. e., the (almost) same number of flights is concentrated on a smaller time interval. This effect is due to the tendency towards a uniform flight schedule. With no "external effects" (in the form of flight series, which also have to conform to constraints outside of a single day) corrupting the optimum, an even more uniform distribution is possible during the relevant time of day. On the other hand, this relevant time is narrower than before.

The optimization for Saturday, Dec 11, 2004, (cf. Table 4.8 and Figure 4.8b) shows that the low number of flights scheduled for that day in the season optimum is not due to any kind of series side effects, but can simply be attributed to the fact that there is a low number of requests for that particular day (and for Saturday in general). Optimizing that day alone shows little difference to the results for the complete season presented above. Again, a slight tendency towards a uniform schedule can be observed, but there are simply not enough requests to produce a schedule as

duration of planning period	Sat, Dec 11, 2004
duration in slots	144
series scheduled	1 252 of 1 252
movements scheduled	1 252 of 1 252
arrival movements	626 of 626
departure movements	626 of 626
average movements per slot	8.7
percentage of requests scheduled	100%
improvement with respect to complete season	*none*

Table 4.8: Solution summary: Sat, Dec 11, 2004, of instance W04 with RW04.

uniform as, e. g., on Fridays (notice that every request is integrated in the schedule, and this is also true for that particular day in the "complete season" optimization run.)

Effects of the Cost Objective Function

The cost objective function presented in Section 4.2.3 can be used in two different ways. First, one can assign different weights to flights, according, e. g., to their MTOW, the number of passengers or some quality benchmark. Second, as the cost objective penalizes deviation of the assigned time from the requested time, using the cost objective instead of just the size of a flight schedule should result in a schedule that integrates as many flights as possible, as close to their requested times as possible. By setting the "cancellation cost" sufficiently low, one can even model a trade-off between the number of flights integrated and the deviation incurred.

We will briefly demonstrate that second effect on a small example here. To that end, we optimize the days Fri, Dec 10, 2004, and Sat, Dec 11, 2004 from instance W04 with RW04, according to the cost objective. We subsequently compare the results to those obtained for the size objective earlier, in particular with respect to deviation. As our main objective still is the integration of a large number of flights, we set the cancellation cost to 1000 and the deviation cost for arrivals and departures to 1, respectively.

planning period **objective**	Fri, Dec 10, 2004 maximum size	Fri, Dec 10, 2004 minimum cost
movements scheduled	1 409 of 1 419	1 409 of 1 419
total deviation	26 800 minutes	13 685 minutes
average deviation per movement	19.0 minutes	9.7 minutes
change in movements		*no change*
change in deviation		*decrease by* 48.9%

Table 4.9: Solution summary: Cost objective for Fri, Dec 10, 2004 from W04 with RW04.

planning period **objective**	Sat, Dec 11, 2004 maximum size	Sat, Dec 11, 2004 minimum cost
movements scheduled	1 252 of 1 252	1 252 of 1 252
total deviation	25 765 minutes	6 955 minutes
average deviation per movement	20.6 minutes	5.6 minutes
change in movements		*no change*
change in deviation		*decrease by* 72.8%

Table 4.10: Solution summary: Cost objective for Sat, Dec 11, 2004 from W04 with RW04.

The results of the optimization are shown in Tables 4.9 and 4.10. For Friday, we get the same number of flights as with optimizing for maximum size, while the average deviation decreases by almost 50% to 9.7 minutes. For Saturday, the result is even better. Again, each flight is integrated, and the average deviation per movement decreases to 5.6 minutes. This is, of course, due to the low number of flight requests for Saturday leaving a considerable margin for optimization with respect to deviation.

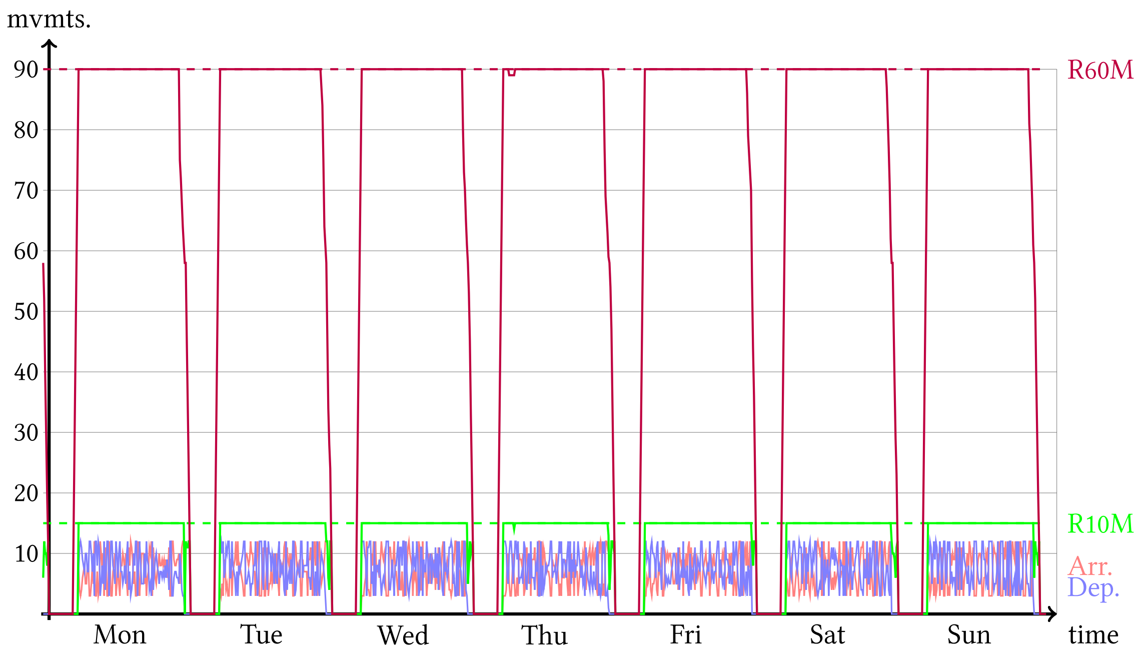
Let us remark that tests of the cost objective for the complete season yielded similar results. However, for lower settings of the cancellation cost, a trade off between low deviation and large schedule size can be observed: Obviously, it sometimes pays off to not integrate a flight and in return gain a much better slot allocation in terms of deviation from the requested slots.

4.3.4 Optimizing the Instance S08 with Reference Value System RS08

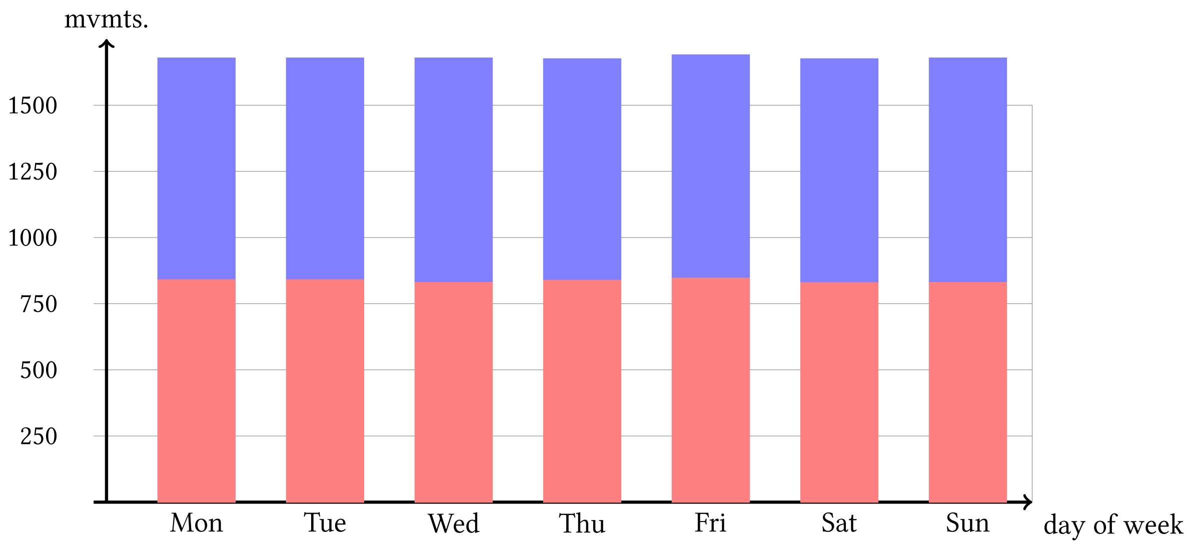
Naturally, a reference value system for the summer 2008 season should be most suitable for a flight request set loosely based on a summer 2008 flight schedule. This is one of the reasons why we examine the results of the instance S08, optimized with the reference value system RS08. A second reason is the fact that RS08 is different from RW04 in a major aspect: It does not include any `R30` shifting bound. The computations were done on the Sun SPARC system and consumed about one hour of CPU time on a single CPU. The results are summarized in Table 4.11; Figure 4.9b shows the number of arrivals and departures for every day of the planning horizon. This clearly shows that Saturday and Sunday are now as strong traffic-wise as Monday to Thursday. This is due to the fact that, while there are still much less requests for the weekend than for workdays, the absolute number of requests in this instance is much higher than what can theoretically be integrated into a flight schedule with respect to the reference value system RS08, even for Saturday and Sunday. On the other hand, Friday still is the one day where air traffic reaches its weekly maximum. The reasons for this phenomenon remain unclear momentarily — just like for Saturday and Sunday, every weekday of the instance S08 has much more flight requests than could possibly be handled, so the “dominance” of Friday is not so natural anymore. However, a look into the local traffic structure below will clarify this point.

duration of planning period	Mon, Apr 14, 2008, to Sun, Apr 20, 2008
duration in days/slots	7 / 1 008
movements scheduled	11 752 of 127 723
arrival movements	5 881 of 63 937
departure movements	5 871 of 63 786
average movements per day	1 678.9
average movements per slot	11.7

Table 4.11: Solution summary: Instance S08 with RS08.



(a) Flight distribution and shifting bound values for `R10M` and `R60M`.



(b) Flight distribution.

🟥 Arrivals 🟦 Departures

Figure 4.9: Optimization results for the complete season of instance S08 with RS08.

For an impression of the local solution structure, Figures 4.10a and 4.10b show the arrivals, departures and `R10M` and `R60M` values for Friday (the peak traffic day) and Monday (a "prototypical" day). One can clearly see that both days are very similar with respect to the flight schedule, but the Friday schedule stretches a little longer into the night. The reason for this is simply the fact that there are more flight requests for later hours on Fridays than on all other days of the week. This explains why the optimal flight schedule still contains more flights for Fridays than for any other day.

4.4 Computational Results for Alternative Reference Value Systems

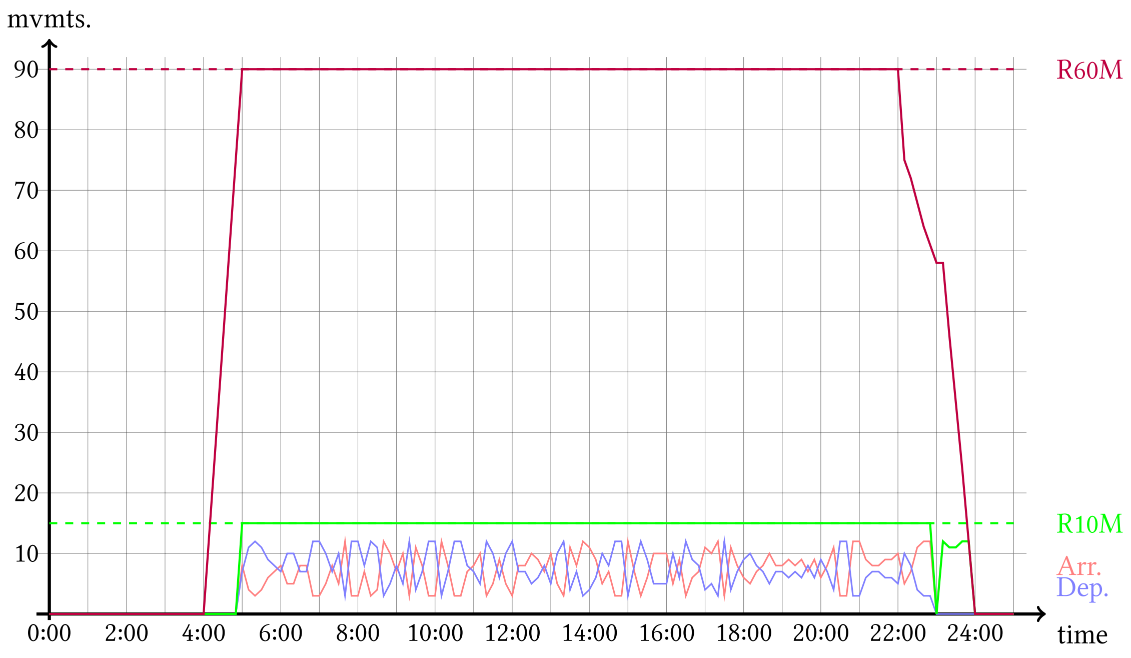
In the last section of this chapter on practical airport slot scheduling, we will present some computational results on alternative reference value systems. The motivation for considering variations of the original constraints were already discussed in Section 3.6, were we also presented some ideas for modifications to the reference value systems that close the gap between a Maximum Slot Packing and a Minimum Slot Cover. Of course, these results are only valid for the simplified setting of Chapter 3, but as we have seen before, many of the results obtained there carry over to the complex real-world situation to some extent. Our results in Chapter 3 and also our findings in Section 4.3 give rise to some "rules of thumb" for practical airport slot scheduling:

- Optimal flight schedules are favored by uniform slot configurations.
- Non-uniform slot configurations generally lead to flight schedules that cannot accommodate additional flights, but on the other hand integrate considerably less flights than an optimal schedule.
- A slot configuration that avoids "local peaks" thereby avoids worst-case scenarios and should thus produce close-to-optimal flight schedules.
- The maximum number of movements of a flight schedule depends primarily on the longest and relatively strictest shifting bound in a monotone and symmetric reference value system.

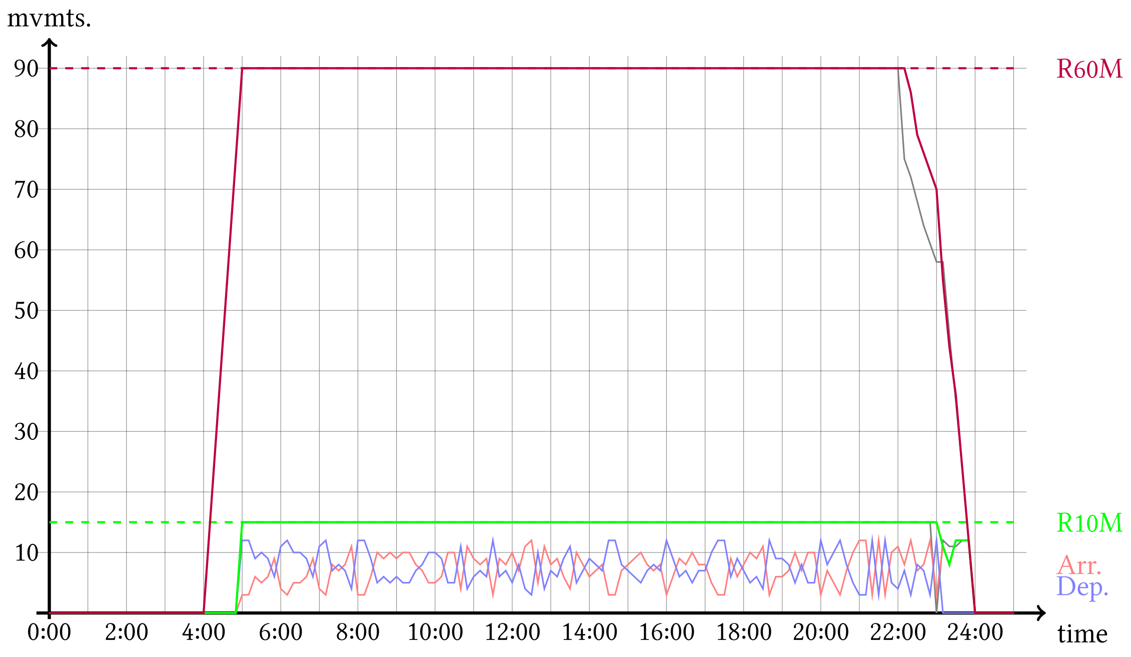
Thus in order to enforce a uniform flight schedule with (almost) the same number of flights that can be obtained with some given reference value system, several approaches seem plausible in the light of Theorems 3.45 and 3.46. In the following, we will implement two of these, apply them to our test cases and discuss the results. All computations in this sections were performed on the IBM ThinkPad and needed roughly 30 to 60 seconds of CPU time.

4.4.1 Reduced Shifting

The effect of local traffic peaks is "spread out" through the shifting time windows which overlap precisely at such a peak. If the time windows were non-shifting, any peak would influence just the



(a) Mon, Apr 14, 2008.



(b) Fri, Apr 18, 2008. For comparison, `R10M` and `R60M` values for Mon, Apr 14, 2008, are drawn in gray.

Figure 4.10: Flight distribution and shifting bound values for `R10M` and `R60M` for instance S08 with reference value system RS08.

slots contained in *one* time window, thereby avoiding the negative effects of local traffic peaks as far as optimality of the flight schedule is concerned. So while non-shifting bounds may be a better choice for the sake of "robustness of optimal solutions" (i. e., optimal or at least good solutions are obtained regardless of the planing procedures), the non-shifting bounds do not inhibit local peaks, but just reduce their "spreading effects". In addition, even non-shifting bounds can cause such effects, although to a limited extent, cf. Figure 3.13 and the discussions in Section 3.6. Fortunately, most reference value systems used in practice have the inclusion property (cf. Definition 3.44) which inhibits such effects.

Traffic peaks are undesirable in practice not only because of their negative effects on optimality of a flight schedule, but also because they negatively affect airport operations on various levels. For instance, handling of air and ground traffic becomes a lot more difficult during such traffic peaks, passenger flow is less continuous and hence congestion at security check lines and baggage check are more likely to occur. Furthermore, baggage needs to be processed more quickly, increasing the risk of misdirection. One of the aims of shifting bounds is to reduce the probability of local traffic peaks (as we have seen in the results obtained so far), and this goal cannot be achieved by non-shifting bounds. Consider, e. g., the situation depicted in Figure 4.11b, where a "double peak" occurs that could have been avoided by using shifting bounds.

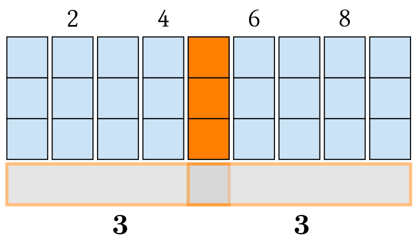
The idea of *reduced shifting* is to mitigate the "spreading-out" problem of local peaks by changing the shifting behavior to combine the effects of shifting and non-shifting bounds: Instead of one time window starting at every slot in the planning horizon, one only considers time windows starting at every k-th slot, for some suitably chosen number $k \in \mathbb{N}$. Let us make this more precise.

Definition 4.8
A *k-shifting bound* or *shifting bound with step k* is a time window bound $(L, b)^{(\sigma)}$ together with an integer $k \in \mathbb{N}$ and is denoted by $(L, b)^{(k)}$ or $([L]_\circ, b)^{(k)}$. A slot configuration $C : \mathcal{S} \to \mathbb{N}_0^3$ on the slot set $\mathcal{S} = \{1, \ldots, n\}$ (and also a flight schedule that induces it) is called feasible with respect to a k-shifting time window bound $(L, b)^{(k)}$, if

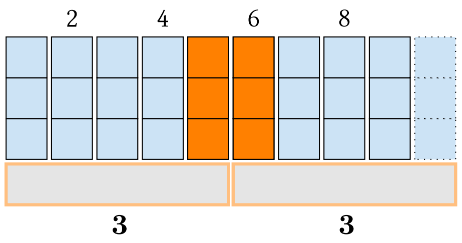
$$C(s + [L]_\circ) \leq b$$

for all $s \in \{1, 1 + k, \ldots, 1 + (\lfloor \frac{n}{k} \rfloor - 1)k, 1 + \lfloor \frac{n}{k} \rfloor k\} \cap \mathcal{S}$. Any k-shifting bound $([L]_\circ, b)^{(k)}$ with $1 < k < L$ is referred to as *reduced shifting bound.*

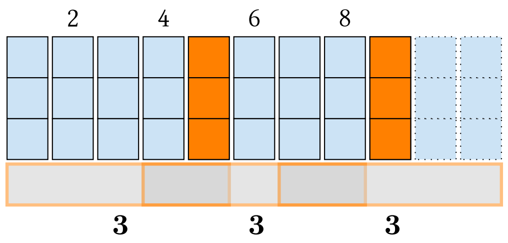
Figure 4.11 shows the effect of k-shifting for a simple example. While ordinary shifting bounds allow for a local peak of three flight movements at slot five to block a total of nine slots, non-shifting bounds result in a flight schedule with six movements within the first ten slots. However, a large peak appears at slots five and six. With reduced shifting (in the figure, the parameter k is set to three), the negative effect of the local peak on the flight schedule cannot be completely avoided; three flight movements can now be placed such that they block only eight slots instead of nine. Yet, the situation is a little better than the one encountered when completely relying on non-shifting bounds.



(a) Shifting bounds



(b) Non-shifting bounds

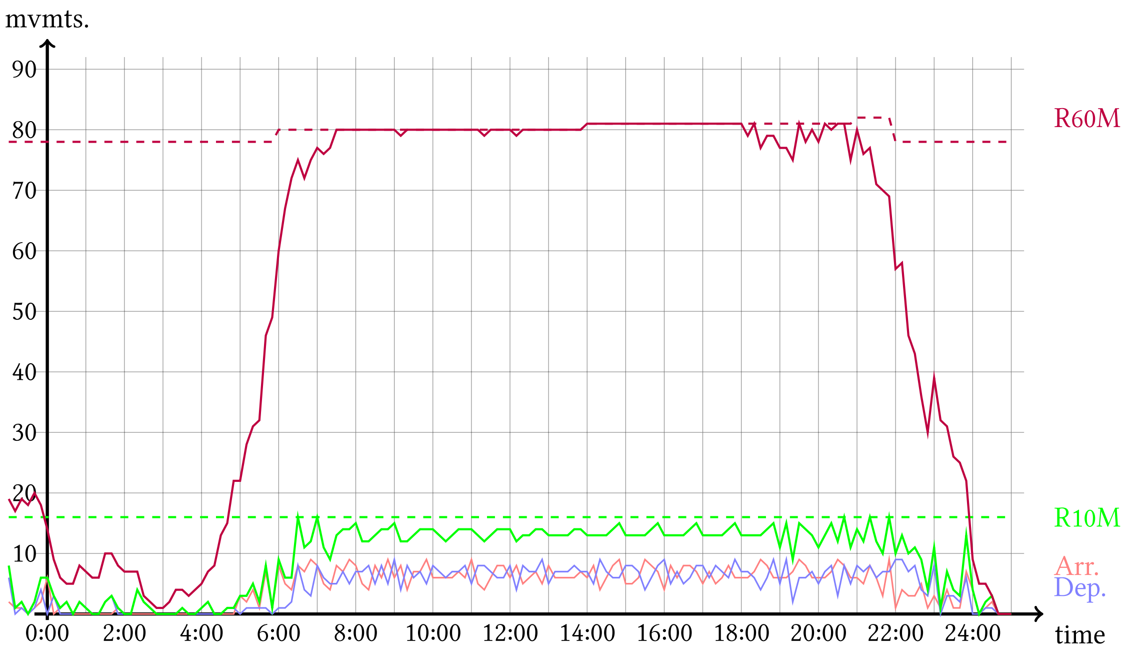


(c) Reduced Shifting (3-shifting bounds)

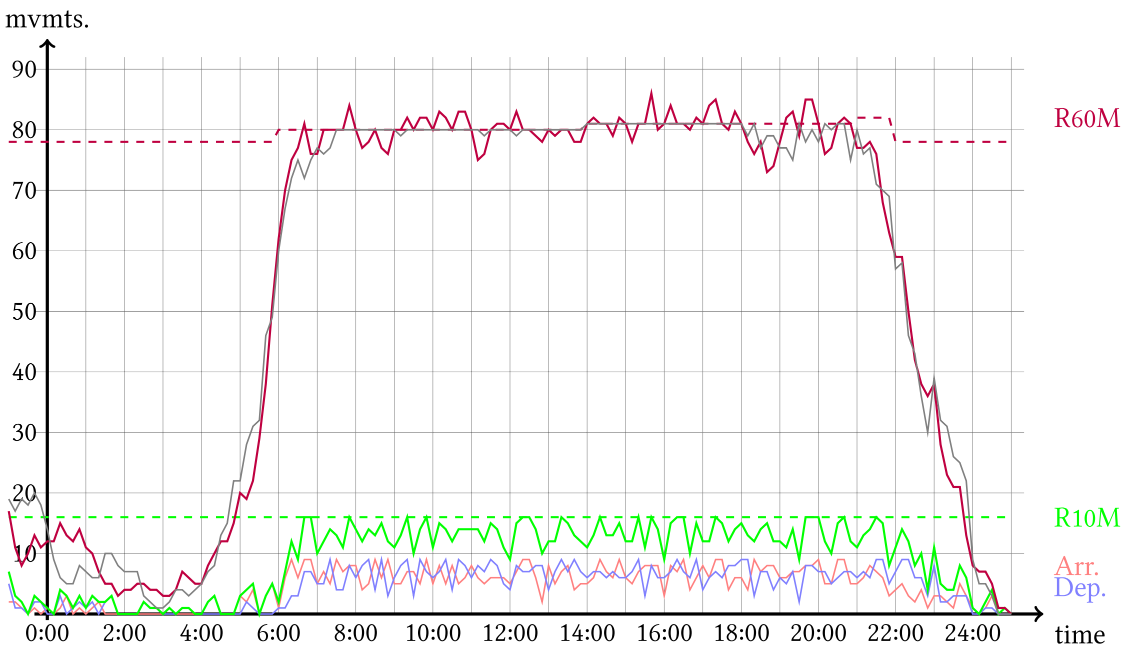
Figure 4.11: Comparison of ordinary shifting, non-shifting and reduced shifting approach with one shifting bound $(5, 3)$ and shifting step $k = 3$.

For the purpose of evaluating the effects of the reduced shifting approach on real-world data, we optimized one day of the instance W04, taking the reference value system RW04 as the basis for our new shifting bounds. The R60 bounds were chosen for the reduced shifting — the other bounds are R10, which entails just one slot and thus is non-shifting anyway, and R30, which is too short (just three slots) to properly evaluate the effects of reduced shifting. As a first step, we removed the R30 shifting bounds for proper assessment of the effects of reduced shifting bounds. We then computed an optimal schedule for one day in the winter season 2004, namely Friday, Dec 10, 2004, for comparison with our earlier experiments. We first used 1-shifting R60 bounds, then a shifting step of three for the R60 bounds. The results are summarized in Table 4.12 and Figures 4.12a and 4.12b.

Obviously, the reduced shifting approach has one disadvantage: If one replaces a shifting bound (L, b) by a reduced shifting bound $(L, b)^{(k)}$ for some suitably chosen $k > 1$, the solution obtained is not necessarily feasible with respect to the original shifting bound any more. Instead, as some of the bounds are now missing, more flights might be integrated into the flight schedule. This effect gets worse with increased shifting step, while on the other hand the packing-covering-gap tends to decrease the larger the shifting step gets. Finding the balance between these two effects usually requires a thorough understanding of the local situation and is dependent on a lot of factors beyond the scope of our modeling approach. If a suitable balance between rigidity and robustness (in terms of packing-covering-gap) of the reference value system can be found for a specific situation, the



(a) With (ordinary) 1-shifting.



(b) With reduced 3-shifting for `R60M`. For comparison, the `R60M` values for 1-shifting are depicted in gray.

Figure 4.12: Flight distribution and shifting bound values for `R10M` and `R60M` for Fri, Dec 10, 2004, of instance W04 with `R10` and `R60` from the reference value system RW04.

duration of planning period	Fri, Dec 10, 2004	
duration in slots	144	
	1-shifting	**3-shifting**
movements scheduled	1 409 of 1 419	1 411 of 1 419
arrival movements	703 of 706	701 of 706
departure movements	706 of 713	710 of 706
average movements per slot	9.8	9.8
percentage of requests scheduled	99.3%	99.4%

Table 4.12: Solution summary for the *reduced shifting* approach: Fri Dec 10, 2004, of instance W04 with R10 and R60 from reference value system RW04. Comparison between 1-shifting and reduced 3-shifting.

reduced shifting approach provides a worthwhile candidate for changing an established reference value system. Furthermore, the approach has a major psychological advantage: The changes are relatively subtle, thus the institutions involved in the scheduling process do not need to change their workflow or computational tools too much, and their experiences will still be valid, at least to some extent. Therefore such a system will have a higher acceptance factor than a more radical change.

4.4.2 Strict Slot Bound

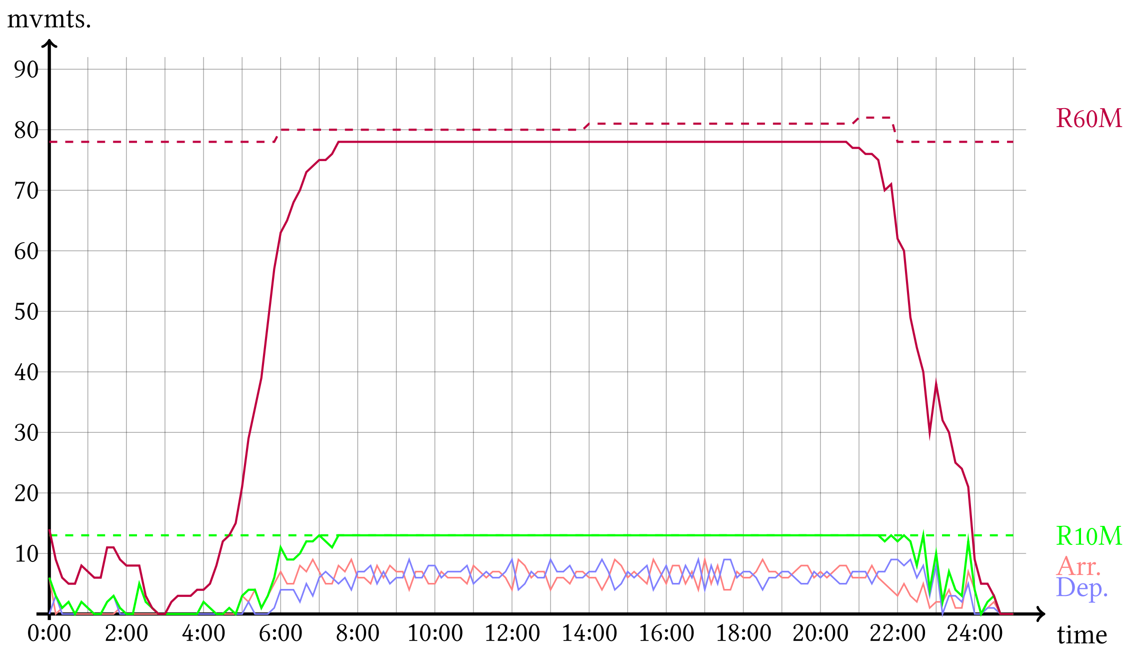
A different approach, called *strict slot bound*, is outlined in Section 3.6, more precisely in Theorem 3.46. The idea is to add a bound on the number of flight movements for each single slot that is strict enough to imply all other bounds. Of course, it suffices to do this for the movements bound, as the movements value is less or equal to the sum of arrivals and departures bound values. Thus for a reference value system $\mathcal{R} = \{(L_1, b_1), \ldots, (L_k, b_k)\}$ define

$$b^* := \min\left\{ \left\lfloor \frac{b_j^M}{L_j} \right\rfloor : (L_j, b_j) \in \mathcal{R} \right\} \cdot \mathbb{1}_3;$$

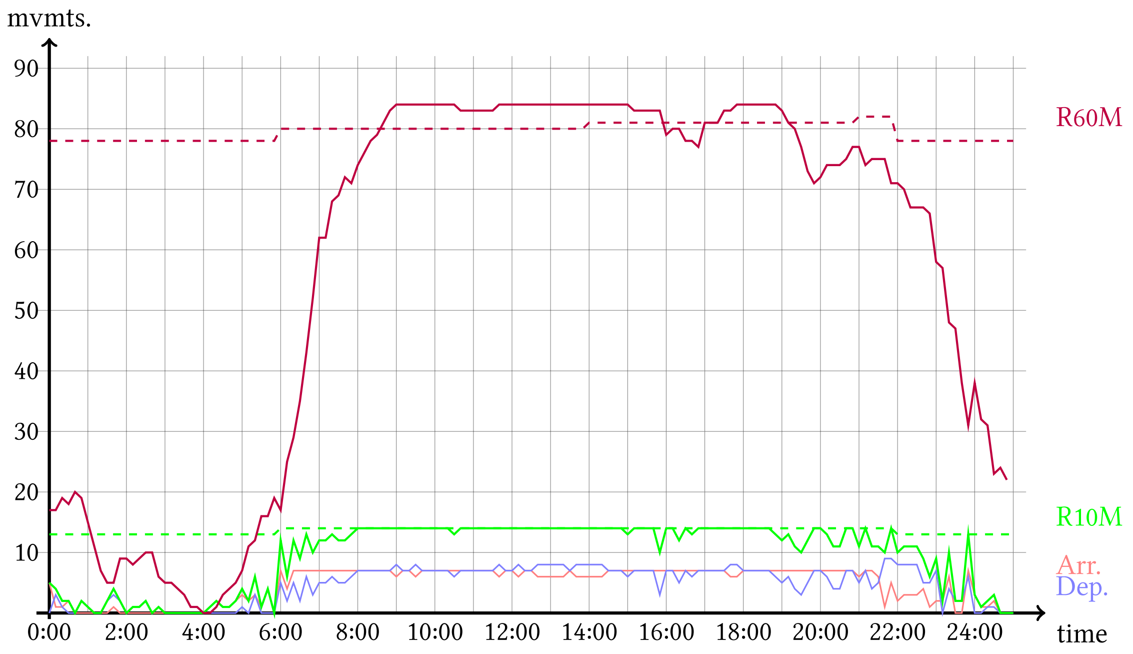
we will refer to this definition of b^* as *rounding down* or *strict slot bound rounded down*. Optimizing with respect to the reference value system $\mathcal{R}^* := \mathcal{R} \cup \{([1]_\circ, b^*)\}$ is then guaranteed to yield a solution that is feasible with respect to $\mathcal{R}$. In addition, the packing-covering gap is zero, i. e., the new reference value system will never allow for a bad flight schedule.

Table 4.13 and Figure 4.13a show the results of a test run with one day (Fri, Dec 10, 2004) of instance W04, based on the reference value system RW04 with strict slot bound rounded down. Due to the rounding, the R60M value is effectively 78 for the whole day, so considerable optimization potential is lost.

If the focus is not so much on strict compliance with the reference value system $\mathcal{R}$ and small



(a) Rounding down R10M



(b) Rounding up R10M

Figure 4.13: Flight distribution and shifting bound values for R10M and R60M for Fri, Dec 10, 2004, of instance W04 with strict slot bound. The values of R60M are added for reference and were not used in the computation.

duration of planning period	Fri, Dec 10, 2004	
duration in slots	144	
	rounding down	**rounding up**
movements scheduled (of 1 419)	1 383	1 419
arrival movements (of 706)	690	706
departure movements (of 713)	693	713
average movements per slot	9.6	9.9
percentage of requests scheduled	97.5%	100.0%

Table 4.13: Solution summary for the *strict slot bound* approach with rounding down and rounding up.

deviations may be acceptable, one could also opt to round up instead of down to avoid giving away too much optimization potential, thus using

$$b' := \min \left\{ \left\lceil \frac{b_j^M}{L_j} \right\rceil : (L_j, b_j) \in \mathcal{R} \right\} \cdot \mathbb{1}_3$$

in place of b^* (*strict slot bound rounded up*). On the downside, while with rounding down we could just use $\{([1]_\circ, b^*)\}$ as the reference value system, using only $\{([1]_\circ, b')\}$ will loosen the reference values to some extent, so this approach might produce a flight schedule that is not feasible with respect to the original reference value system $\mathcal{R}$. Results for this approach are shown in Table 4.13 and Figure 4.13b. While all flight request can now be scheduled, the old `R60M` bounds (which have been disabled for these test runs) are violated by a considerable amount.

To remedy that situation, the approach can be extended by using the reference value system $\mathcal{R}' := \{([1]_\circ, b')\} \cup \mathcal{R}$ instead of just $\{([1]_\circ, b')\}$. This keeps the original shifting bounds in effect, while at the same time adding a requirement for uniformity, as $\mathcal{R}'$ does not allow for considerable local peaks and thus provides for a very small packing-covering gap. In other words, this new reference value system will produce an optimal flight schedule "by design", meaning any flight schedule which is feasible with respect to $\mathcal{R}'$ is nearly optimal (or can at least be extended to a nearly optimal flight schedule by filling the remaining free slots). Still, $\mathcal{R}'$ is stricter than $\mathcal{R}$, so some flight schedules that are optimal with respect to $\mathcal{R}$ might no longer be feasible and the objective value might decrease when switching to $\mathcal{R}'$ due to the loss in flexibility. The test results for this approach can be seen in Table 4.14 and Figure 4.14b.

Finally, to strike a balance between $\{([1]_\circ, b')\}$, i. e., rounding up alone, and $\mathcal{R}'$, one can also employ the time window bounds of $\mathcal{R}'$ as non-shifting bounds. This approach leads to much less constraints than using $\mathcal{R}'$, while still providing for some correction for the rounding-up approach on a larger scale (i. e., rounding up locally, for instance on a ten minute scale, but keeping the original values on a larger scale, e. g., on intervals of 60 minutes).

To evaluate the potential of rounding up combined with the original bounds, we tested both approaches on a single day of instance W04 with bounds based on RW04 again. The results are

duration of planning period	Fri, Dec 10, 2004		
duration in slots	144		
	slot bound excl.	**non-shifting R60**	**shifting R60**
movements scheduled (of 1 419)	1 419	1 405	1 405
arrival movements (of 706)	706	699	699
departure movements (of 713)	713	706	706
average movements per slot	9.9	9.8	9.8
percentage of requests scheduled	100.0%	99.0%	99.0%

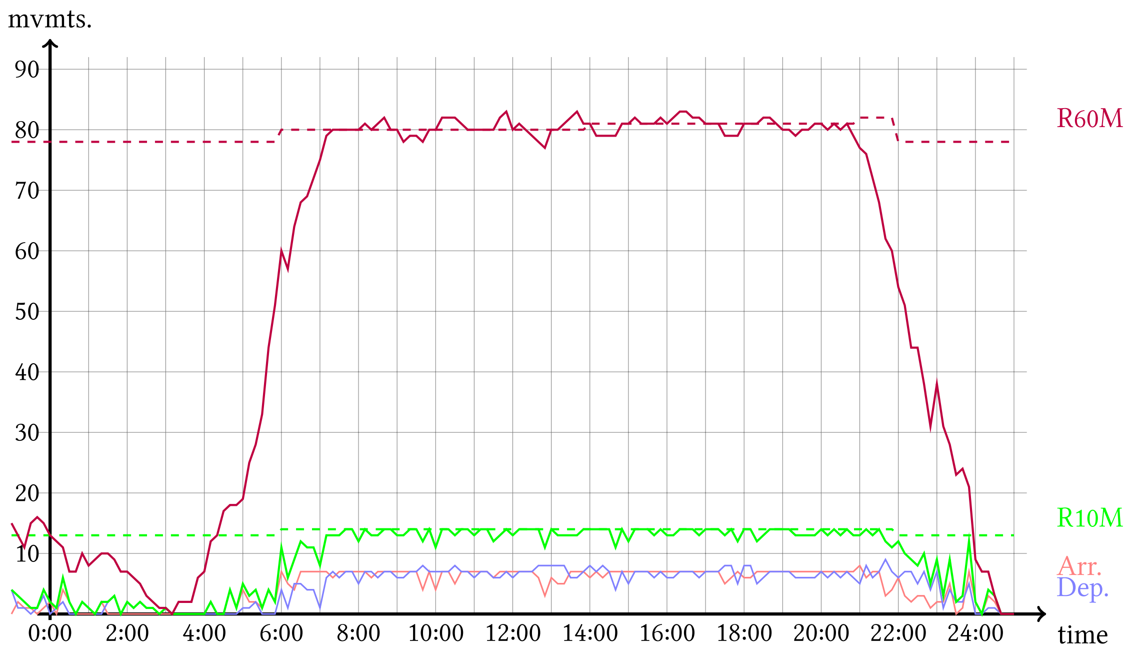
Table 4.14: Solution summary for the *strict slot bound* approach with rounding up: Fri, Dec 10, 2004, of instance W04 with R10 and R60 from reference value system RW04. Comparison between using strict bounds exclusively and combining them with non-shifting and shifting R60 bounds.

summarized in Table 4.14. We compare the schedule for using solely a slot bound obtained from R60M by rounding up as described above, with the results for combining that slot bound with shifting and non-shifting R60 bounds, respectively. Figure 4.13b shows the schedule for just the slot bounds. As stated before, the rounding up leads to considerable violations of the R60 bounds (which have not been used in the computation and are added to the figure for illustration only). In contrast to that, adding non-shifting R60 bounds (cf. Figure 4.14a), we observe a much more uniform schedule. Still, some bounds are violated, but the R60 amplitude is considerably lower. Finally, Figure 4.14b shows the schedule for adding the complete shifting R60 bounds, yielding a flight schedule that is feasible for the original bounds, but still uses a reference value system that does not allow for a significant packing-covering gap.

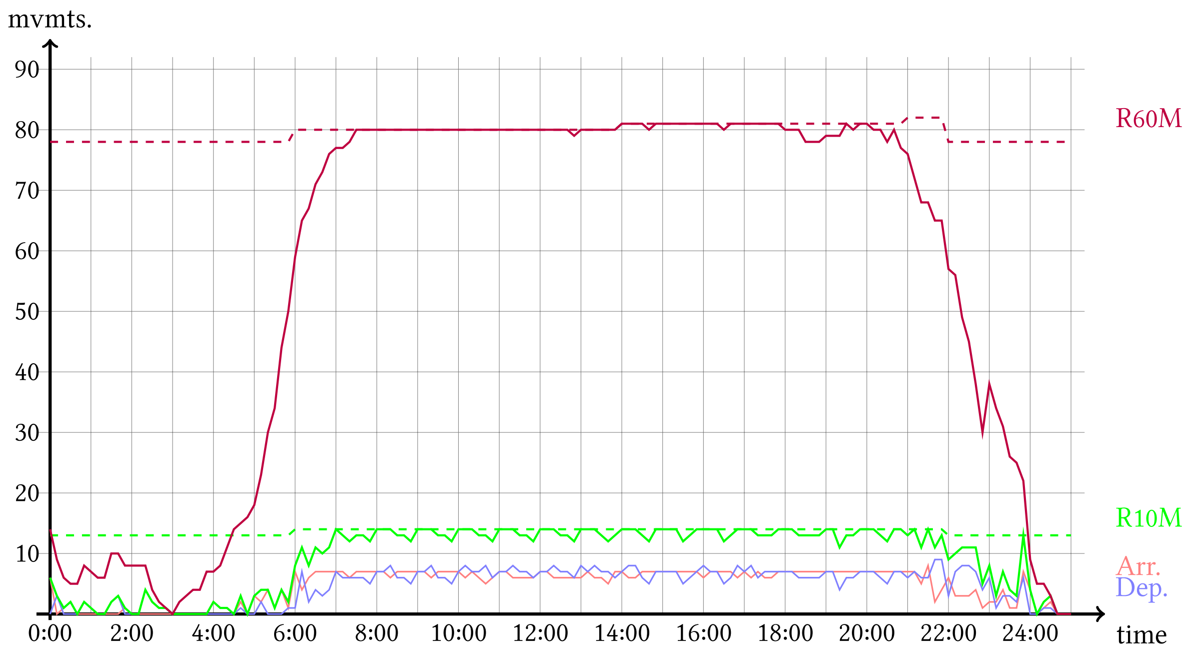
We know from the tests in Section 4.3.3 that the optimum under the original reference value system is 1 409 movements, compared to 1 405 movements with the last strict slot bound approach. However, the new reference value system has a striking advantage: It is "fool-proof", meaning it does not support a very bad flight schedule as the old system would. As was to be expected, there is a price to pay for this benefit, and the decision whether the gain is worth this price can only be taken based on the individual situation at an airport. Finally, observe that there is no difference in the objective value between using shifting and non-shifting bounds. This suggests to generally use the shifting bounds variant, because it produces even more uniform flight schedules while not incurring a considerably higher cost in the form of less flight movements. However, a possible reason for using the non-shifting variant might be the lower number of constraints compared to the shifting variant.

4.5 Concluding Remarks

In this chapter, we gave a detailed description of the real-world flight scheduling process, and all relevant legal, technological and environmental constraints. We then modeled flight scheduling



(a) Non-shifting `R60` bounds.



(b) Shifting `R60` bounds.

Figure 4.14: Flight distribution and shifting bound values for `R10M` and `R60M` for Fri, Dec 10, 2004, of instance W04 with strict slot bound rounded up.

as an integer linear program, implemented it and conducted a series of computational tests. We discussed the results, connecting them with the findings of Chapter 3. Finally, we investigated the possibility of alternative reference value systems, discussed the relevant aspects in the light of Chapter 3 and our foregoing computational experience, and suggested several improvement approaches that were subsequently tested on real-world data.

As the results in this chapter demonstrate, our considerations on Maximum Slot Packing and Minimum Slot Cover in Chapter 3 are not only of theoretical value, but provide the backbone for automating the process of allocating slots to flight requests. For this chapter, we not only demonstrated that such an automated allocation is possible, but we also implemented our model in a software (which is called SOFIE — "Slot-Optimierung für die Flugplanung und integrierte Eckwertstrukturanalyse") to perform this task. This software was successfully employed for a series of test cases. It is thus possible to fully automate the flight scheduling process and to come up with an initial proposed schedule for a complete season within less than 6 hours, whereas the same task means more than 6 weeks of work for the airport coordinator and his or her coworkers now. Of course, through negotiations with airports, airlines and authorities, the initial proposal is changed to some extent before the flight schedule is finalized. In the course of these events, the coordinator may even opt to violate some of the constraints to a minor extent. These procedures will certainly not become superfluous due to our optimization approach, manual intervention is considered desirable by the participants of the process after the initial stage. On the other hand, the computation of a very good initial flight schedule can be greatly simplified by using SOFIE.

Although our computations proved that the results produced by the airport coordinator are already very good, given that no mathematical optimization is involved in those procedures as of today, these results are mainly due to long experience in the field of manual flight scheduling. Naturally, the coordinator will cope with small alterations of the constraints (such as a small change to some of the shifting bound values) without much problems. But problems are to be expected once a more fundamental change of the constraints occurs (e. g., structurally different reference value systems), as it will take some time of "trial and error" to gain experience with a new system. In contrast to this, mathematical optimization does not need to adapt slowly to a new situation — a change in the model parameters is sufficient to provide the coordinator with optimal flight schedules and thereby with structural information about a new constraint system, thus giving valuable information for the evaluation of alternative reference value systems.

Furthermore, the implementation of our model can also be worthwhile for different, but related tasks. For instance, it may be used to quickly evaluate the consequences of manual intervention and to re-optimize the flight schedule afterwards for a choice of different objective functions. This can provide valuable information to support the negotiations leading to the final flight schedule. As we showed in Section 4.4, SOFIE can also be used to benchmark different reference value systems or estimate the effects of other changes to the constraint system quickly and in a very cost-efficient way.

The mathematical model for flight scheduling that we presented, analyzed and solved in Chapters 3 and 4 encompasses all practically relevant constraints for the scheduling process and can easily be extended and modified, should new constraints be incorporated in the future. It solves our test instances in a fraction of the time the airport coordinator needs to produce a schedule, and it generally gives better results than can be achieved by manual “trial and error”, given the complexity of the problem. To put our methods to use in practical slot scheduling, there is of course still work to be done. Apart from exact specifications of some of the constraints (see for instance the discussion on the new entrants rule in Section 4.1.1), accurate data and software interfaces to established visualization and planning tools in airport operations will be necessary. Additionally, a fast heuristic solver based on the algorithms of Chapter 3 might provide useful information to assess the effects of certain changes to the flight schedule on the spot, hence such a tool could be a valuable aid for the negotiations at the IATA schedules conferences. But even now, our structural results and insights can help to produce better flight schedules by incorporating some rather simple rules into the scheduling process and to evaluate the effects of changes in the constraint system without the need for extensive and costly simulations or real-world trials.

Wire Placement Glossary

Crosstalk Signal transitions on one wire may interfere with the signal on a neighboring wire by means of capacitive coupling if the wires are very close and/or the signal level is very high. This interference is called *crosstalk*. Crosstalk may cause a loss of signal integrity and thus lead to malfunction or failure of the circuit.

Design Automation, Routing Routing or chip design refers to the process of constructing a layout of the wires on a chip such that the semiconductor gates on the silicon layer are connected to each other as provided for by the circuit diagram. This process involves a large number of wires and constraints on the possible placement of these wires as well as one or several objectives (e. g. area consumption) and is usually automated by using suitable computer software. The process is then referred to as electronic design automation.

Dummy Wires Wires with switching activity 0, used in the model to avoid border effects. In reality, power or shield wires on a chip have switching activity 0.

Dynamic Power Loss The component of overall power loss that is attributed to dynamic effects, i. e., effects that stem from the change of signal levels on a semiconductor chip. Dynamic power loss can be traced back to switching capacitances between circuit wires and short-timed short circuit currents.

Interconnects The wires connecting the gates on the silicon layer of a semiconductor chip to each other.

International Technology Roadmap for Semiconductors, ITRS A biannual report compiled by the *SIA*, together with experts from the leading nations in the semiconductor industry. The ITRS describes current developments, trends, projections and important future challenges in the semiconductor industry. It is available online at `http://www.itrs.net`.

IP ("Intellectual Property") Module A small semiconductor circuit designed for integration into larger semiconductor chips. An IP module is dedicated to very specific tasks and is connected to the rest of the design and other modules via a well-specified interface.

Layers A semiconductor chip is composed of several layers stacked on top of each other. At the base level there is a silicon layer containing the transistors that are combined into logic gates. To interconnect these logic gates one or more layers containing metal wiring (metal layers) are arranged above the silicon layer.

Minimum Inter Wire Distance Minimum Distance that is admissible between any two wires on a chip.

Permutation Network An extra layer added to a semiconductor circuit to connect locally reordered wires to their original connections in the circuit. The use of permutation networks enables local application of wire ordering within complex semiconductor circuits.

Preferred Routing Direction Within a single metal layer, (almost) all of the wires are routed in parallel direction, called the preferred routing direction. For two metal layers immediately on top of each other, the preferred routing directions are perpendicular.

Semiconductor Industry Association, SIA The Semiconductor Industry Association is a trade association representing the U.S. semiconductor industry. It currently (2008) has 95 member companies (including Advanced Micro Devices, IBM, Intel and Texas Instruments), representing more than 85% of semiconductor production in the United States. Jointly with experts from the leading nations in the semiconductor industry, the SIA regularly publishes the *International Technology Roadmap for Semiconductors (ITRS).* Further information can be found on the official website `http://www.sia-online.org`.

Spacing Range For a given set of wires, the distance available on the chip between the two border wires.

Static Power Loss The component of overall power loss that is attributed to static effects, i. e., effects that occur independently of signal switches. This mainly refers to leakage currents at transistor level.

Switching Frequency The frequency of a signal change on a circuit wire. The switching frequency is usually approximated by simulations of typical applications of a semiconductor circuit or by deducing a probability distribution for specialized wires such as data bus or counting bus wires.

Wire Ordering The process or the mathematical problem of finding an order of the wires on a chip such that a subsequent wire spacing yields the best possible result among all permutations of the wires.

Wire Placement The combination of *wire ordering* and subsequent *wire spacing* to find a power optimal permutation and corresponding distances between the wires on a chip.

Wire Spacing The process or the mathematical problem of finding distances between wires on a chip (for fixed ordering) such that the total power loss is minimized.

Flight Scheduling Glossary

Airport Coordination Limited, ACL The ACL is responsible for airport coordination in the United Kingdom. Its German counterpart is the *FHKD*.

Airport Coordinator Head of a national airport coordination authority. The airport coordinator is responsible for implementing the slot system at national airports where possible overload cannot be avoided by voluntary cooperation (coordinated airports). To this end, the coordinator closely cooperates with airports, airlines and international organizations such as the IATA and exercises the authority of allocating slots to airlines based upon their requests and upon the relevant capacity restrictions at an airport.

Coordinated Airport An airport where the slot system is implemented. In Germany, these are the airports Frankfurt/Main, Berlin (Tegel, Schönefeld, Tempelhof), Düsseldorf, München and Stuttgart.

Flughafenkoordination Deutschland, FHKD The FHKD is the German airport coordination authority. As such, it is responsible for issuing flight schedules for the coordinated German airports, monitoring the proper use of slots and granting historic rights where applicable.

Grandfather Rights, Historic Rights When an airline has been allocated a slot and makes proper use of that slot during the scheduling period (i. e., the slot is used for at least 80% of the time), it acquires the right to receive the same slot in the following scheduling period of the same type (i. e., from winter season to winter season or from summer season to summer season). This rule is also known as *historic slots* or *"use it or loose it" rule.*

Hub and Spoke System For a long distance flight using a large aircraft it is often necessary to first transport passengers from many local airports to a larger airport (called *hub*) in order to offer them a connected service from an airport near their home, but on the other hand make efficient use of large aircrafts and slots for long distance destinations. The regional feeder flights are sometimes referred to as *spoke flights*, because they emanate from the hub like the spokes of a wheel.

Incumbent An airline that has been active at a certain airport in the past and does not classify as new entrant. The term is also used for specific requests by airlines that can neither claim historic status nor classify as new entrants' requests.

International Air Transport Association, IATA The International Air Transport Association is an organization of the commercial airline industry with about 94% of all airlines worldwide being members of the IATA. Among its aims are standardizing and facilitating common procedures in international air traffic, such as ticket handling, security measures, airline cooperation or implementation of the slot system.

Maximum Take-Off Weight, MTOW The maximum take-off weight is the maximum weight of an aircraft (including payload) at which the aircraft is allowed to attempt take-off. This number can be interpreted as a measure of loading capacity for an aircraft.

New Entrant An airline that is not presently active at an airport or to which at most four slots (including its current requests) are allocated at an airport on a specific date. According to EU regulations, 50% of the slots available after historic slots have been allocated are reserved for new entrants' flight requests, provided there is a sufficient number of such requests. A slot allocated under the new entrants rule must be used for the requested service for at least two years.

Reference Value System A collection of *time window bounds* that are all applied simultaneously.

Schedules Conference An international meeting of airport and airline representatives together with national airport coordinators hosted by the *IATA* after preliminary schedules have been fixed. The purpose of a schedules conference is to facilitate slot exchange and re-routing of flights so that airlines can make the best possible use of the slots they are allocated.

Scheduling Period The period during which a flight schedule is in effect. One distinguishes between winter season (from the end of October until the end of March of the following year) and summer season (from the end of March until the end of October).

Slot The scheduled time of arrival or departure for an aircraft movement on a specific date and time, distinguished between arrival slot (for landing) and departure slot (for take-off). One slot can usually be allocated to more than one aircraft movement and represents a time window of five or ten minutes in most real world situations.

Slot System A system for allocating the resources at an airport where there is not enough capacity to facilitate all airlines wishing to offer flight services to and/or from that airport. Under the slot system, every airline that wants to offer a service first needs to obtain a slot, i. e., the right to land and take off at a specified time.

Time Notions In airport operations, two different time notions are distinguished, namely *gate time* and *runway time*. By gate time we mean the time when an aircraft arrives at or departs from the airport's gate or parking position, this time notion is relevant for the passengers.

By runway time we mean the time when the aircraft occupies the runway of the airport, this time notion is relevant for all flight movement bounds, most notably for those imposed by some reference value system. If not stated otherwise, we always refer to runway time in the context of this work.

Time Window Bound, Shifting Bound, Non-Shifting Bound A bound on the number of arrivals, departures and/or total flight movements ("movements" or "mixed" value) that is applied to all flights within a certain time windows of a given length. Time window bounds can be applied in a non-shifting manner, which means that the first time window starts at slot 1, the next time window starts immediately after the end of the first one, and so on. More common is the shifting variant, where a new time window starts at every slot.

List of Symbols

Special Sets and Set Systems

$\mathbb{N}$, $\mathbb{N}_0$	the set of natural numbers (excluding and including 0, respectively)
$\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{R}$	the set of integers, rationals and real numbers, respectively
$\mathcal{P}(S)$	the power set of S
$(M, \mathcal{M})$	an independence system or a matroid over the ground set M
$[a,\, b]$, $]a,\, b]$, $[a,\, b[$, $]a,\, b[$	the sets $\{x \in M : a \leq x \leq b\}$, $\{x \in M : a < x \leq b\}$, $\{x \in M : a \leq x < b\}$ and $\{x \in M : a < x < b\}$, respectively, where M is either $\mathbb{Z}$ (mostly) or $\mathbb{R}$, depending on the context. If the choice of M is not obvious from the context, we denote M by a subscript like $[a,\, b]_{\mathbb{Z}}$.
$[n]$	the set $\{x \in \mathbb{Z} : 1 \leq x \leq n\}$
$[n]_\circ$	the set $\{x \in \mathbb{Z} : 0 \leq x \leq n-1\}$
$A \subset B$	A is a subset of B, where $A = B$ is allowed
$(S)^\star$	the set $S \cup \{\infty\}$
$[\![t + S]\!]_G$	the circular Minkowski sum of $\{t\}$ and $S \subset G$ with respect to G.

Vectors, Matrices, Numbers and Miscellaneous

u_i	the i-th unit vector
$\mathbb{1}$, $\mathbb{1}_n$	the all ones vector, the all ones vector of length n
$A = (a_{ij})$	the matrix A with entries a_{ij}
a_i^T, $a^{(j)}$	the i-th row and the j-th column of a matrix $A = (a_{ij})$, respectively
$[\![m]\!]_n$	the unique integer $k \in \{0, \dots, n-1\}$ such that $k \equiv m \mod n$
$[\![m]\!]_{[n]}$	the unique integer $k \in \{1, \dots, n\}$ such that $k \equiv m \mod n$
$\operatorname{dist}_n(a, d)$	the circular distance from a to d on a circle of length n
$f(S')$	the sum $\sum_{s \in S'} f(s)$ for a function $f : S \to \mathbb{R}$ and $S' \subset S$
$\mathcal{O}(\cdot)$, $\Omega(\cdot)$	Landau symbol and Omega symbol, respectively

Symbols used in Chapter 2

N	number of wires
w, w_i	wires
d	minimum inter-wire distance
r	spacing range
$\widehat{W}, W$	set of wires with or without dummy wires
φ	a wire placement
π	a wire ordering
δ	a wire spacing
α	switching frequency
$\mathcal{P}_N$	the set of all wire orderings on N wires
$\mathcal{D}_N(r, d)$	the set of admissible wire spacings
$\mathcal{A}_N$	the set of switching frequency functions
$\mathcal{S}_N$	the symmetric group on $\{1, \ldots, N\}$ or the symmetric group on $\{0, \ldots, N+1\}$ with the additional property that 0 and $N+1$ are fixed points.
x_i	the distance between the $(i-1)$-th and the i-th wire.
s_i	switching frequency of the i-th wire
q_i	equal to $s_{i-1} + s_i$
P	feasible set of Optimal Wire Spacing
$F(x)$	the objective function for Optimal Wire Spacing
D, $D(x)$	the minimum distance set for the wire spacing x
R, $R(x)$	the complement of $D(x)$
$P_{\mathbb{Q}}$	feasible set of Optimal Rational Wire Spacing
$sqrt(q_i)$	rational approximation of $\sqrt{q_i}$
$SQRT(q_i, \varepsilon')$	number of operations to compute $sqrt(q_i)$ within error bound ε' with respect to $\sqrt{q_i}$ on a binary Turing machine
δ	wellness condition of a δ-well posed instance
x^π	optimal wire spacing for the wire ordering π
$F(\pi, x^\pi)$	the objective function for Optimal Wire Placement
D^π, R^π	equal to $D(x^\pi)$ and $R(x^\pi)$, respectively
ω_i	switching frequencies, sorted in increasing order
τ	a TSP tour or a Hamilton Path
$C(\tau)$	the cost (length) of a tour τ
$\mathscr{R}_{jk}, \overline{\mathscr{R}}_{jk}$	open and closed j-k reversal, respectively
l, $l(x)$, l^π, u, $u(x)$, u^π	lower and upper separation point, respectively

Symbols used in Chapter 3

$\mathcal{S}$	a slot set, i. e., $\mathcal{S} = \{1, \ldots, n\}$ for some $n \in \mathbb{N}$
G, F, $(F, \mathcal{I})$	a slot request, a flight request and a series request, respectively
$\mathcal{I}$	starting point set of a series request
$S^{\mathcal{I}}_{(a,d)}$	feasible slot series
$\mathcal{G}$, $\mathcal{F}$	a collection of slot requests and flight requests, respectively
f	a flight schedule
$\|f\|$	size of a flight schedule
$(L, b)^{(\sigma)}$ or $([L]_\circ, b)^{(\sigma)}$	a time window bound bound of length L with bound value b and shift specification σ
$(L, b) = (L, b)^{(1)}$, $(L, b)^{(L)}$, $(L, b)^{(k)}$	a shifting bound, a non-shifting bound and a k-shifting bound, respectively
b^{A}, b^{D}, b^{M}	arrival, departure and movements bound value
$s + [L]_\circ$	the Minkowski-sum $\{s\} + [L]_\circ = \{s, s+1, \ldots, s+(L-1)\}$
$\mathcal{R}$	a reference value system
R	incidence matrix of a reference value system
$\mathring{R}$	circular incidence matrix of a reference value system
C	a slot configuration
C^{A}, C^{D}, C^{M}	arrivals, departures and movements value of a slot configuration
C_f	the slot configuration associated to a flight schedule f
$\mathcal{B}_{\mathcal{G}}$	a ground set
$\mathcal{M}_{\mathcal{G}}$	flight scheduling indendence system
$\mathcal{M}^{\mathrm{A}}_{(L,b)}, \mathcal{M}^{\mathrm{D}}_{(L,b)}, \mathcal{M}^{\mathrm{M}}_{(L,b)}$	shifting bounds independence systems
H_t	set of slot requests for the slot t
H'_s	set of slot request within a time window starting at s
w_u	node label (length of a shortest 0-u path)
l_{uv}	length of the arc (u, v)
l^{λ}_{uv}	parameterized length of the arc (u, v)
$\mathcal{W}((L, b)^{(\sigma)}, s)$	the set of all time windows for the time window bound $(L, b)^{(\sigma)}$ that contain s

Symbols used in Chapter 4

n	the number of slots per day, usually 144
d	the number of days in the planning horizon
N	the number of slots in the planning horizon, i. e., $N = nd$
$(T, S, g, \mathcal{D}, c)$	a flight series request
$T = (T^{\mathrm{A}}, T^{\mathrm{D}})^T$	the requested arrival/departure slot pair

$S = \left(S^{\mathrm{A}}, S^{\mathrm{D}}\right)^T$	the maximum allowable shift for a fligth request
$g = \left(g^{\min}, g^{\max}\right)^T$	minimum and maximum ground time, respectively
$\mathcal{D}$	requested days of service
$c = \left(c^{\mathrm{A}}, c^{\mathrm{D}}, c^{\mathrm{M}}\right)^T$	cost of deviation from requested slots for arrivals and departures, and cost of not integrating the fligth request at all, respectively
$\mathcal{F}$	a flight request set
$\mathcal{F}^{\mathrm{H}}$	the set of H-classified historic requests
$\mathcal{F}^{\mathrm{CL}}$	the set of CL-classified historic requests
$\mathcal{F}^{\mathrm{CR}}$	the set of CR-classified historic requests
$\mathcal{F}^{\mathrm{CI}}$	the set of CI-classified historic requests
$\mathcal{F}^{\mathrm{NE}}$	the set of new entrants' requests
$\mathcal{F}^{\mathrm{I}}$	the set of incumbents' requests
$\mathcal{F}^{\mathrm{H/C}}$	the set of flight request with any historic classification
$\mathcal{F}^{\mathrm{A}}$	the set of arrival requests
$\mathcal{F}^{\mathrm{D}}$	the set of departure requests
$\mathcal{F}^{\mathrm{M}}$	the set of mixed requests
$\mathcal{F}^{\mathrm{A/M}}$	the set of arrival and mixed requests
$\mathcal{F}^{\mathrm{D/M}}$	the set of departure and mixed requests
$\mathcal{F}^{\leftmoon}$	the set of overnight series requests
$\mathcal{F}^{\mathrm{USA}}$	the set of flight request which depart for a North American airport
$f = \left(f^{\mathrm{A}}, f^{\mathrm{D}}\right)$	a flight schedule
$\lvert f \rvert$	size of a flight schedule f, i. e., number of scheduled movements
$c(f)$	total cost of a flight schedule
`R10A`, `R30D`, `R60M`	shifting bound of length 10, 30 and 60 minutes, for arrivals, departures and movements, respectively
x_{it}, y_{it}	decision variables
W04, RW04	the instance "Winter 2004" and the corresponding reference value system, respectively
S08, RS08	the instance "Summer 2008" and the corresponding reference value system, respectively
$(1, b^*)$, $(1, b')$	a bound for the "strict slot bound" approach, obtained through either rounding down or rounding up
$\mathcal{R}^*$, $\mathcal{R}'$	reference value system with added bound $(1, b^*)$ or $(1, b')$

List of Figures

All figures in this thesis were created by the author. For most of the figures, the wonderful TEX packages TikZ and PGF by Till Tantau were used, see [Tan08].

List of Tables

List of Algorithms

Bibliography

[ABCC03] D. Applegate, R. E. Bixby, V. Chvátal, and W. Cook. *Concorde*. Available online. 2003. URL: `http://www.tsp.gatech.edu/concorde/`.

[ABIVZ92] S. V. Amiouny, J. J. Bartholdi III, J. H. V. Vate, and J. Zhang. "Balanced Loading". In: *Operations Research* 40 (2), 1992, pp. 238–246.

[AMO93] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows. Theory, Algorithms, and Applications*. Prentice Hall, Upper Saddle River, NJ, 1993.

[BDDVW98] R. E. Burkard, V. G. Deineko, R. van Dal, J. A. A. van der Veen, and G. J. Woeginger. "Well-Solvable Special Cases of the Traveling Salesman Problem: A Survey". In: *SIAM Review* 40 (3), 1998, pp. 496–546.

[Ben08] B. Benz. "Pin-Wald versus Pad-Wiese". In: *c't* 7, Mar. 2008, pp. 178–185.

[BIOR80] J. J. Bartholdi III, J. B. Orlin, and H. D. Ratliff. "Cyclic Scheduling via Integer Programs with Circular Ones". In: *Operations Research* 28 (5), 1980, pp. 1074–1085.

[Bix82] R. E. Bixby. "Matroids and Operations Research". In: *Advanced Techniques in the Practice of Operations Research*. Ed. by F. H. Greenberg H. J.Murphy and S. H. Shaw. Vol. 4. Publications in Operations Research. North-Holland, New York, 1982, pp. 333–458.

[BR95] E. E. Bischoff and M. S. W. Ratcliff. "Issues in the development of approaches to container loading". In: *Omega* 23 (4), Aug. 1995, pp. 377–390.

[BT77] S. Baum and L. E. Trotter Jr. "Integer Rounding for Polymatroid and Branching Optimization Problems". In: *Optimization and Operations Research*. Ed. by R. Henn, B. Korte, and W. Oettli. Vol. 157. Lecture Notes in Economics and Mathematical Systems. Springer, Berlin, 1977, pp. 15–23.

[Bur90] R. E. Burkard. "Special cases of travelling salesman problems and heuristics". In: *Acta Mathematicae Applicatae Sinica (English Series)* 6 (3), July 1990, pp. 273–288.

[CGJ97] E. G. Jr. Coffman, M. R. Garey, and D. S. Johnson. "Approximation algorithms for bin packing: a survey". In: *Approximation algorithms for NP-hard problems*. Ed. by Dorit S. Hochbaum. PWS Publishing Co., Boston, MA, USA, 1997, pp. 46–93.

[CI92] Y. L. Le Coz and R. B. Iverson. "A stochastic algorithm for high speed capacitance extraction in integrated circuits". In: *Solid-State Electronics* 35 (7), July 1992, pp. 1005–1012.

[CKP01] J. Cong, C. Koh, and Z. Pan. “Interconnect Sizing and Spacing with Consideration of Coupling Capacitance”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 6 (6), 2001, pp. 1164–1169.

[CY85] D. D. Cochard and K. A. Yost. “Improving Utilization of Air Force Cargo Aircraft”. In: *Interfaces* 15 (1), 1985, pp. 53–68.

[Dash06a] *Xpress-BCL Reference Manual, Release 3.0*. Dash Optimization Ltd. Warwickshire, UK, 2006.

[Dash06b] *Xpress-Mosel Libraries Reference Manual, Release 2.0*. Dash Optimization Ltd. Warwickshire, UK, 2006.

[Dash06c] *Xpress-Mosel Reference Manual, Release 2.0*. Dash Optimization Ltd. Warwickshire, UK, 2006.

[Dash06d] *Xpress-MP Getting Started, Release 2007*. Dash Optimization Ltd. Warwickshire, UK, 2006.

[Dash07a] *Xpress-Mosel User guide, Release 2.0*. Dash Optimization Ltd. Warwickshire, UK, 2007.

[Dash07b] *Xpress-Optimizer Reference Manual, Release 18*. Dash Optimization Ltd. Warwickshire, UK, 2007.

[DB99] A. P. Davies and E. E. Bischoff. “Weight distribution considerations in container loading”. In: *European Journal of Operational Research* 114, May 1999, pp. 509–527.

[DH93] P. Deuflhard and A. Hohmann. *Numerische Mathematik I*. 2nd ed. de Gruyter, Berlin, 1993.

[Dij59] E. W. Dijkstra. “A Note on Two Problems in Connexion with Graphs”. In: *Numerische Mathematik* 1, 1959, pp. 269–271.

[Edm79] J. Edmonds. “Matroid Intersection”. In: *Annals of Discrete Mathematics* 4, 1979, pp. 39–49.

[EU02] *Regulation (EC) No 894/2002 of the European Parliament and of the Council of 27 May 2002*. Official Journal of the European Union L 142. May 2002. URL: `www.eur-lex.europa.eu`.

[EU03] *Regulation (EC) No 1554/2003 European Parliament and of the Council of 22 July 2003*. Official Journal of the European Union L 221. Sept. 2003. URL: `www.eur-lex.europa.eu`.

[EU04] *Regulation (EC) No 793/2004 of the European Parliament and of the Council of 21 April 2004*. Official Journal of the European Union L 138. Apr. 2004. URL: `www.eur-lex.europa.eu`.

[EU93] *Council Regulation (EEC) No 95/1993 of 18 January 1993 on Common Rules for the Allocation of Slots at Community Airports.* Official Journal of the European Union L 14. Jan. 1993. URL: `www.eur-lex.europa.eu`.

[FRA07] Fraport AG. *Frankfurt Airport Luftverkehrsstatistik 2007.* 2007. URL: `http://www.fraport.de/cms/investor_relations/dokbin/281/281211.2007fraportfrankfurt_airport_luftverkehr@de.pdf`.

[FRA08] Frankfurt Airport Services Worldwide. *Geschäftsbericht 2007.* Mar. 2008. URL: `http://www.fraport.de/cms/investor_relations/dokbin/283/283176.geschaeftsbericht_2007_deutsch.pdf`.

[GH62] Alain Ghouila-Houri. "Caractérisation des matrices totalement unimodulaires". In: *Comptes Rendus Hebdomadaires des Séances de l'Academie des Sciences (Paris)* 254, 1962, pp. 1192–1194.

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability. A guide to the theory of NP-completeness.* A Series of Books in the Mathematical Sciences. W. H. Freeman and Co., San Francisco, Calif., 1979.

[GLS85] P. C. Gilmore, E. L. Lawler, and D. B. Shmoys. "Well-solved special cases". In: *The Traveling Salesman Problem. A Guided Tour of Combinatorial Optimization.* Ed. by E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys. John Wiley & Sons, 1985. Chap. 4, pp. 87–143.

[GLS93] Martin Grötschel, László Lovász, and Alexander Schrijver. *Geometric algorithms and combinatorial optimization.* Springer, Berlin, 1993.

[GMM90] H. Gehring, K. Menschner, and M. Meyer. "A computer-based heuristic for packing pooled shipment containers." In: *European Journal of Operational Research* 44 (2), 1990, pp. 277–288.

[Gro89] P. Groeneveld. "Wire ordering for detailed routing". In: *Design & Test of Computers* 6 (6), 1989, pp. 6–17.

[GRZ08] P. Gritzmann, M. Ritter, and P. Zuber. "Optimal wire ordering and spacing in low power semiconductor design". In: *Mathematical Programming*, 2008+ (accepted).

[HH94] G. Hämmerlin and K. H. Hoffmann. *Numerische Mathematik.* Springer, 1994.

[HL06] D. S. Hochbaum and A. Levin. "Optimizing over Consecutive 1's and Circular 1's Constraints". In: *SIAM Journal on Optimization* 17 (2), 2006, pp. 311–330.

[Hof63] A. J. Hoffman. "On simple linear programming problems". In: *Convexity.* Ed. by V. Klee. Vol. VII. Proceedings of Symposia in Pure Mathematics. American Mathematical Society, Providence, 1963, pp. 317–327.

[IATA07] International Air Transport Association (IATA). *Worldwide Scheduling Guidelines, 14th Edition.* July 2007. URL: `www.iata.org`.

[Law75] E. L. Lawler. “Matroid Intersection Algorithms”. In: *Mathematical Programming* 9, 1975, pp. 31–56.

[LM88] G. Laporte and H. Mercure. “Balancing hydraulic turbine runners: A quadratic assignment problem”. In: *European Journal of Operational Research* 35 (3), June 1988, pp. 378–381.

[LMV02] A. Lodi, S. Martello, and D. Vigo. “Heuristic algorithms for the three-dimensional bin packing problem”. In: *European Journal of Operational Research* 141 (2), 2002, pp. 410–420.

[MMK06] S. Michaely, K. Moiseev, and A. Kolodny. “Optimal Bus Sizing in Migration of Processor Design”. In: *IEEE Transactions on Circuits and Systems Part 1: Regular Papers* 53 (5), 2006, pp. 1089–1100.

[MMP01] L. Macchiarulo, E. Macii, and M. Poncino. “Low-Energy Encoding for Deep-Submicron Address Buses”. In: *Proceedings of the 2001 International Symposium on Low Power Electronics and Design*, 2001, pp. 176–181.

[MMP02] L. Macchiarulo, E. Macii, and M. Poncino. “Wire placement for crosstalk energy minimization in address buses”. In: *Proceedings of Design, Automation and Test in Europe Conference and Exhibition, 2002.* 2002, pp. 158–162.

[Mos86] J. Mosevich. “Balancing hydraulic turbine runners–A discrete combinatorial optimization problem”. In: *European Journal of Operational Research* 26 (2), Aug. 1986, pp. 202–204.

[MPS03] E. Macii, M. Poncino, and S. Salerno. “Combining wire swapping and spacing for low-power deep-submicron buses”. In: *Proceedings of the 13th ACM Great Lakes Symposium on VLSI*, 2003, pp. 198–202.

[MUC06] Flughafen München GmbH, Pressestelle. *Press release from 12/27/2006 – Munich breaks the 30-million passenger barrier for first time ever in 2006.* Dec. 2006. URL: `http://www.munich-airport.de/EN/Areas/Company/Medien/textarchiv/textarchiv06/PM79/index.html`.

[MUC07] Flughafen München GmbH. *Statistischer Jahresbericht 2007 – Luftverkehrsstatistik.* 2007. URL: `http://www.munich-airport.de/media/download/bereiche/daten/jahresberichte/deutsch_2007.pdf`.

[MWK06] K. Moiseev, S. Wimer, and A. Kolodny. “Timing Optimization of Interconnect by Simultaneous Net-Ordering, Wire Sizing and Spacing”. In: *Proceedings of the IEEE International Symposium on Circuits and Systems*, 2006, pp. 329–332.

[NW99] G. Nemhauser and L. Wolsey. *Integer and Combinatorial Optimization.* Wiley-Interscience, New York, 1999.

[Pap95] C. H. Papadimitriou. *Computational complexity.* Addison Wesley, New York, 1995.

[Par91] J. K. Park. “A special case of the n-vertex traveling-salesman problem that can be solved in O(n) time”. In: *Information Processing Letters* 40 (5), Dec. 1991, pp. 247–254.

[Per06] T. S. Perry. “Wizard of Watts”. In: *IEEE Spectrum* 43 (6), 2006, pp. 32–37.

[PS85] F. P. Preparata and M. I. Shamos. *Computational Geometry. An Introduction.* Texts and Monographs in Computer Science. Springer, New York, 1985.

[PS98] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity.* Corrected republication. Dover Publications, Mineola, New York, 1998.

[PSQL05] *PostgreSQL 8.1.11 Documentation.* The PostgreSQL Global Development Group. 2005. URL: `http://www.postgresql.org/docs/8.1/static/index.html`.

[Roc72] R. T. Rockafellar. *Convex Analysis.* Princeton University Press, Princeton, New Jersey, 1972.

[Sch86] A. Schrijver. *Theory of Linear and Integer Programming.* Wiley, Chichester, 1986.

[SD97] J. Z. Su and W. W. Dai. “Post-route optimization for improved yield using a rubber-band wiring model”. In: *1997 IEEE/ACM International Conference on Computer-Aided Design, Digest of Technical Papers.* 1997, pp. 700–706.

[SIA07] Semiconductor Industry Association. *International Technology Roadmap for Semiconductors (ITRS): 2007 Edition.* 2007. URL: `http://www.itrs.net/Links/2007ITRS/Home2007.htm`.

[SM06] K. Sundaresan and N. R. Mahapatra. “Value-Based Bit Ordering for Energy Optimization of On-Chip Global Signal Buses”. In: *Proceedings of Design, Automation and Test in Europe, 2006. DATE ’06.* Vol. 1. 2006, pp. 1–2.

[SS01] Y. Shin and T. Sakurai. “Coupling-driven bus design for low-power application-specific systems”. In: *DAC ’01: Proceedings of the 38th conference on Design automation.* ACM Press, New York, 2001, pp. 750–753.

[Sun06] *Java Platform, Standard Edition 6 API Specification.* Sun Microsystems Inc. Santa Clara, 2006. URL: `http://java.sun.com/javase/6/docs/api/`.

[Sup57] F. Supnick. “Extreme Hamiltonian lines”. In: *Annals of Mathematics* 66, 1957, pp. 179–201.

[SW70] J. Stoer and C. Witzgall. *Convexity and Optimization in Finite Dimensions I.* Springer, Heidelberg, 1970.

[Tan08] Till Tantau. *The TikZ and PGF Packages. Manual for version 2.00.* Lübeck, 2008. URL: `http://sourceforge.net/projects/pgf`.

[VW62a] A. F. Veinott and H. M. Wagner. “Optimal Capacity Scheduling I”. In: *Operations Research* 10 (4), 1962, pp. 518–532.

[VW62b] A. F. Veinott and H. M. Wagner. “Optimal Capacity Scheduling II”. In: *Operations Research* 10 (4), 1962, pp. 533 –546.

[Vyg04] J. Vygen. “Near-Optimum Global Routing with Coupling, Delay Bounds, and Power Consumption”. In: *Integer Programming and Combinatorial Optimization.* Ed. by D. Bienstock and G. Nemhauser. Vol. 3064. Lecture Notes in Computer Science. Springer, 2004, pp. 308–324.

[Wel95] D. J. A. Welsh. “Matroids: Fundamental Concepts”. In: *Handbook of Combinatorics.* Ed. by R. Graham, M. Grötschel, and L. Lovász. Vol. I. Elsevier, Amsterdam, 1995. Chap. 9, pp. 481–526.

[Whi35] H. Whitney. “On the Abstract Properties of Linear Dependence”. In: *American Journal of Mathematics* 57, 1935, pp. 509–533.

[Win08] C. Windeck. “Transistor-Catwalk. International Solid-State Circuits Conference 2008”. In: *c’t* 5, Feb. 2008, pp. 44–45.

[Woe03] G. J. Woeginger. “Computational Problems without Computation”. In: *Nieuw Archief voor Wiskunde* 5/4 (2), 2003, pp. 140–147.

[WZS02] A. Windschiegl, P. Zuber, and W. Stechele. “Exploiting Metal Layer Characteristics For Low-Power Routing”. In: *Integrated Circuit Design. Power and Timing Modeling, Optimization and Simulation.* Ed. by Bertrand Hochet, Antonio J. Acosta, and Manuel J. Bellido. Vol. 2451. Lecture Notes in Computer Science. Springer, 2002, pp. 55–64.

[YK99] J.S. Yim and C.M. Kyung. “Reducing cross-coupling among interconnect wires in deep-submicron datapath design”. In: *Proceedings of the 36th ACM/IEEE conference on Design automation*, 1999, pp. 485–490.

[YTO91] N. E. Young, R. E. Tarjan, and J. B. Orlin. “Faster Parametric Shortest Path and Minimu-Balance Algorithms”. In: *Networks* 21 (2), Mar. 1991, pp. 205–221.

[ZBIRS08] P. Zuber, O. Bahlous, T. Ilnseher, M. Ritter, and W. Stechele. “Wire topology optimization for low power CMOS”. In: *IEEE Transactions on VLSI Systems*, 2008+ (accepted).

[ZGRS05] P. Zuber, P. Gritzmann, M. Ritter, and W. Stechele. “The optimal wire order for low power CMOS”. In: *Integrated Circuit and System Design.* Vol. 3728. Lecture Notes in Computer Science. Springer, 2005, pp. 664–683.

[Zub07] P. Zuber. “Wire topology optimisation for low power CMOS”. PhD thesis. Technische Universität München, 2007. URL: `http://nbn-resolving.de/urn/resolver.pl?urn:nbn:de:bvb:91-diss-20070802-618152-1-6`.

Index

This index lists all important occurrences of the terms collected here. Bold numbers indicate the page where an item is properly defined or explained. For mathematical symbols and notation, only the place of their definition or first use appears here; a short description of the mathematical symbols can also be found in the list of symbols.