Of Keyboards and Beyond — Optimization in Human-Computer Interaction

A dissertation submitted towards the degree Doctor of Engineering of the Faculty of Mathematics and Computer Science of Saarland University

by Maximilian John

Saarbrücken / 2019

Day of Colloquium:	15.01.2020
Dean of the Faculty:	Prof. Dr. Sebastian Hack
Chair of the Committee:	Prof. Dr. Bernd Finkbeiner
Reporters	
First reviewer:	Dr. Andreas Karrenbauer
Second reviewer:	Prof. Dr. Kurt Mehlhorn
Academic Assistant:	Dr. Antonios Antoniadis

Abstract

Abstract In this thesis, we present optimization frameworks in the area of Human-Computer Interaction. At first, we discuss keyboard layout problems with a special focus on a project we participated in, which aimed at designing the new French keyboard standard. The special nature of this national-scale project and its optimization ingredients are discussed in detail; we specifically highlight our algorithmic contribution to this project. Exploiting the special structure of this design problem, we propose an optimization framework that was efficiently computes keyboard layouts and provides very good optimality guarantees in form of tight lower bounds. The optimized layout that we showed to be nearly optimal was the basis of the new French keyboard standard recently published in the National Assembly in Paris. Moreover, we propose a relaxation for the quadratic assignment problem (a generalization of keyboard layouts) that is based on semidefinite programming. In a branch-and-bound framework, this relaxation achieves competitive results compared to commonly used linear programming relaxations for this problem. Finally, we introduce a modeling language for mixed integer programs that especially focuses on the challenges and features that appear in participatory optimization problems similar to the French keyboard design process.

Zusammenfassung Diese Arbeit behandelt Ansätze zu Optimierungsproblemen im Bereich Human-Computer Interaction. Zuerst diskutieren wir Tastaturbelegungsprobleme mit einem besonderen Fokus auf einem Projekt, an dem wir teilgenommen haben: die Erstellung eines neuen Standards für die französische Tastatur. Wir gehen auf die besondere Struktur dieses Problems und unseren algorithmischen Beitrag ein: ein Algorithmus, der mit Optimierungsmethoden die Struktur dieses speziellen Problems ausnutzt. Mithilfe dieses Algorithmus konnten wir effizient Tastaturbelegungen berechnen und die Qualität dieser Belegungen effektiv (in Form von unteren Schranken) nachweisen. Das finale optimierte Layout, welches mit unserer Methode bewiesenermaßen nahezu optimal ist, diente als Grundlage für den kürzlich in der französischen Nationalversammlung veröffentlichten neuen französischen Tastaturstandard. Darüberhinaus beschreiben wir eine Relaxierung für das quadratische Zuweisungsproblem (eine Verallgemeinerung des Tastaturbelegungsproblems), die auf semidefinieter Programmierung basiert. Wir zeigen, dass unser Algorithmus im Vergleich zu üblich genutzten linearen Relaxierung gut abschneidet. Abschließend definieren und diskutieren wir eine Modellierungssprache für gemischt integrale Programme. Diese Sprache ist speziell auf die besonderen Herausforderungen abgestimmt, die bei interaktiven Optimierungsproblemen auftreten, welche einen ähnlichen Charakter haben wie der Prozess des Designs der französischen Tastatur.

Acknowledgments

I would like to thank Andreas for advising me throughout the last 4 years. You did not only introduce me to many interesting topics regarding combinatorial optimization, but you also shaped me personally. Whenever I needed an advice — be it scientifically or personally — I could count on you. I thoroughly enjoyed being part of the Discrete Optimization group together with Ruben and Davis who were not only colleagues but also became friends.

I would like to thank Kurt for being an inspirational role model and creating such a great atmosphere in the institute. Thanks for always having an open ear when I needed it.

Furthermore, I would like to thank Anna, Antti and Mathieu. I had a great time working with you towards the new French keyboard standard. It has been quite a journey and I am proud of what we have achieved together.

I would like to thank my family who have always supported me in any way possible in pursuing my dream carreer. I am happy and grateful that you have walked this path together with me.

Finally, I would like to thank my wonderful wife Laura. Your love and your emotional support helped me even through the hardest times. Thank you for always having my back even if this meant cutting down your own interests for the sake of my work. I would have not accomplished this without you.

Contents

CHAPTER 1

Introduction

Keyboards are one of the earliest forms of interaction between human and computer. Although many new human-computer interaction (HCI) models (e.g., graphical user interfaces or voice commands) have been developed in the last decades, keyboards still play an important role in this area. Many official keyboard layouts have changed only very subtly when compared to the very first layouts that have been produced; despite their poor performance regarding ergonomics or time-to-type. Recent advances in mathematical optimization theory have opened up new possibilities for the optimization of keyboard layouts.

This thesis describes my contributions to optimization problems in HCI, with a special focus on keyboard layout problems. We discuss two algorithms that not only compute good keyboard layouts but also provide quality guarantees in form of tight lower bounds. One of these algorithms has been dedicated to the French keyboard problem, which has a very special structure that can be exploited by our framework. In 2016, the French government started an initiative to design a new keyboard layout that should contain all characters to write proper French and at the same time be optimized with respect to different quality measures. We participated in this large national-scale project by contributing a dedicated optimization framework that could provide good optimality guarantees in a very small time frame. In 2019, this project successfully concluded with the official publication of the new French keyboard standard. The optimized layout that we proposed and proved to be nearly optimal served as a basis for this new standard. We can proudly claim that this standard could have not been realized in its current form without the help of these dedicated optimization methods.

The first chapters provide a technical and historical classification of keyboard optimization in the context of HCI. Moreover, we describe the circumstances and processes of the French keyboard problem. A manuscript, which has been accepted for a publication in the Communications of the ACM [1], serves as a basis for this chapter. Chapter 4 provides general technical details with a focus on optimization theory; we introduce basic definitions, popular solution approaches, and theoretical hardness of keyboard layout and related optimization problems.

The rest of the thesis contains more technical details about our practical contributions to optimization in HCI. The specialized framework for the French keyboard problem is discussed in Chapter 5; the corresponding paper [2] has been published in the conference proceedings of *Mathematical Optimization Theory and Operations Research* 2019. We show that our approach worked well in the whole process of the French keyboard design. Throughout the several years of development, we solved multiple different keyboard layout instances that arose during discussions with the stakeholders of this project. Using our algorithm, we were able to provide very good optimality guarantees (often $< 5\%$) for many instances within only a few minutes. Additionally, our algorithm is very resource-

efficient allowing the user to run it on a standard laptop instead of a powerful compute server. These features of our approach open new possibilities for optimization methods in participatory design processes, where the optimization model has to quickly react to changes of the input data or other relevant optimization parameters.

An alternative approach for quadratic assignment problems — the generalization of keyboard layout problems — is discussed in Chapter 6. While the former algorithm exploited the special structure of the French keyboard optimization problem, this algorithm performs well for more general quadratic assignment problems. It is based on a semidefinite programming relaxation; a relatively young and promising area of research regarding hard optimization problems.

Finally, the last chapter introduces a modeling language for optimization problems; motivated by different challenges that arose during the French keyboard process, this language provides a low entry-hurdle even for non-computer scientists. One of its prominent features is the separation of model and data, which simplifies the use of this modeling language in a scenario where input data frequently changes. The structure of the language, its features and benefits are discussed in Chapter 7. There also exists an implementation of an interpreter for this language as a part of Sören Bund-Becker's Bachelor's thesis that Andreas Karrenbauer and I supervised.

CHAPTER 2

Optimization in Human-Computer Interaction

2.1 Introduction

This chapter introduces the role of combinatorial optimization in the context of keyboards and graphical user interfaces (GUIs). It is based on a survey article on this topic [3], which is a collaborative project of Antti Oulasvirta, Niraj Ramesh Dayama, Morteza Shiripour (all from Aalto Univerisity in Finland), Andreas Karrenbauer and myself. While the scope of the survey paper is much larger, it contains many aspects of GUI designs that are irrelevant for this thesis and are therefore omitted.

The main interaction tools between humans and computers are keyboards, GUIs, gestures and voice commands. In this chapter, we will focus on the former two methods and discuss them in a context of mathematical optimization[1]. Keyboards and GUIs will be shortly referred to as user interfaces (UIs) in the following text.

2.1.1 Keyboard Optimization - A Historic Overview

Many traditional keyboard layouts have only been changed very slightly in the last 100 years. The roots of the American QWERTY layout go back to the 19th century. Christoph Latham Sholes was the first printer who did not sort the characters alphabetically on his typewriter, an image of patent he claimed in 1878 is shown in Figure 2.1. His layout was not supposed to be very ergonomic or to allow fast typing; instead, the main goal was to split up characters that are often used in combination with each other. If those characters are next to each other on a typewriter, pressing them both in quick succession could cause their corresponding type levers to jam.

Today, the keyboard layout is still almost identical despite the drastic changes in hardware and usage of keyboards. Type lever jams are a relict of the past, rendering the whole purpose of this traditional layout irrelevant. The advances in combinatorial optimization over the past decades have initialized many possibilities to compute keyboard layouts that are optimal with respect to certain quality measures. Researchers have proposed alternative layouts, which perform much better regarding typing speed or ergonomics (e.g.,[4]); however, these layouts usually do not reach a public audience mostly because of political reasons and because many people do not want to relearn how to type on a keyboard. This learnability of a new design is a major part of our work towards the new French keyboard layout, which will be discussed in Chapter 3.

In 1975, Pollatschek et al. proposed to compute a keyboard layout by solving a combinatorial optimization problem for the first time. Their approach has been extended

[1] Whenever we discuss optimization in this thesis, it means mathematical optimization

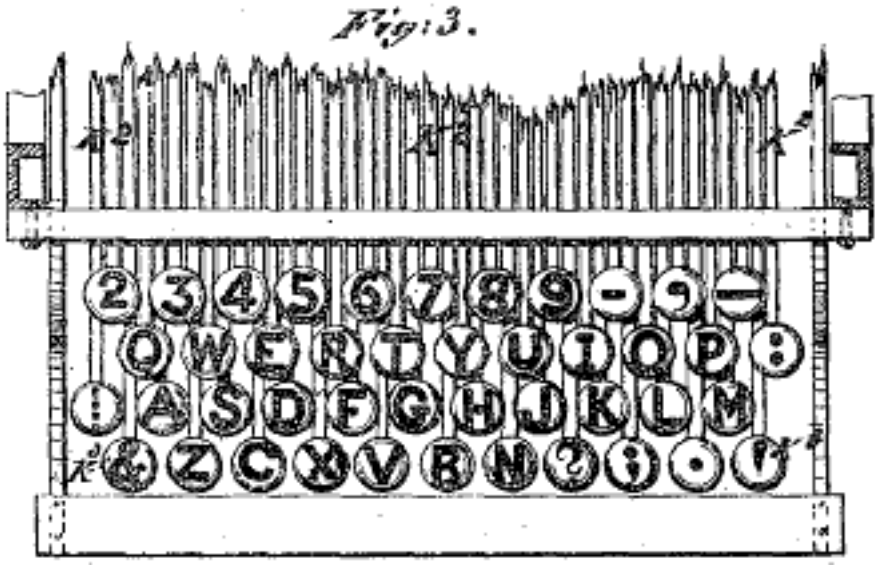

Figure 2.1: The U.S. patent for the QWERTY typewriter layout. Source: U.S. Patent No. 207,559; C.L. Sholes; 1878

by many researchers (e.g., [5]) and is still the basis for many modern approaches: In 2014, Karrenbauer and Oulasvirta proposed integer programming techniques to improve keyboard optimization. In particular, they apply two popular linearization methods to the integer programming formulation of the keyboard layout problem, which led to promising results regarding performance of their algorithm and quality of the resulting keyboards. Their approach, however, was only focused on typing speed and neglected other important quality measures. Moreover, the linearizations that they used do not scale well for keyboard instances with a higher amount of characters and slots (instances with > 40 characters become difficult if the problem structure cannot be exploited). The approaches discussed later in this thesis extend this work and redeem the shortcomings mentioned above.

The design of keyboard layouts and graphical user interfaces show several similarities. Many optimization problems appear in both design processes, the most obvious being the actual layout problem (which character/element should be placed on which key/position), but also, for example, the functionality selection problem which computes an optimal set of functionalities (which characters should be placed on a keyboard or which elements should be available on a GUI) that should be part of the user interface [6]. These similarities inspired us to extend our view of optimization opportunities from keyboard problems to general user interface design problems. A detailed discussion about optimization problems occurring in the design of user interfaces and how to model and solve them is done in our submission [3]. Since this thesis mainly focusses on keyboard layout problems, we will only discuss basic properties of GUIs and their connection to keyboard layout problems.

2.2 Graphical User Interfaces

The following part (until Section 2.3) is inferred from [3]. A GUI presents the state and controls of a computer program visuo-spatially on a display for interaction with a pointing device [7]. Visual presentation serves two functions: firstly, the program state can be conveyed to the user and, secondly, it enables changing the state of the program by interacting with a pointer. Commands are typically carried out by dwelling (e.g., hover-over), clicking, or dragging elements with a pointer. Elements express visually what type of interaction they permit; consider, for example, buttons, widgets, and icons.

The traditional GUI paradigm is known as WIMP: *windows*, *icons*, *menus*, and a *pointing device*. In addition, modern GUIs offer multiple types of *widgets*, such as buttons, entry fields, and choosers. *Containers* of different types are available for media, applications (docks), and documents (folders). *Navigation controls* such as scrollbars, task switchers, search bars, and tabs translate or update views. In a text entry mode, text can be entered also via a virtual or physical keyboard. Keyboard shortcuts can be used to invoke commands without pointing.

Since most software and services have extensive functionality to offer, GUIs are often organized hierarchically. Two principles of hierarchical organization are commonly followed:

- *Visual Containment*: graphically marked containers such as canvases, windows, and boxes can have other containers and elements within them, and

- *Logical Compositionality*: a program can consist of multiple sub-GUIs, such as a settings panel, a drawing canvas, and a dialog. These can be presented in sequence or parallel. For example, the 3D modeling software Maya, which offers 1,346 functions in a menu, arranges them in a hierarchical fashion [6].

A popular way to organize elements in a GUI is the grid layout [8]. It uses grid-lines to organize slots that determine the possible sizes and positions of graphical elements.

Besides purely technical considerations (e.g., software and hardware reliability) and considerations related to marketing and brands, there are end-user-related design objectives in GUI design. They include 1) usefulness; 2) user performance such as speed and accuracy in completing tasks; 3) learnability; and 4) aspects of user experience such as aesthetics, emotions, or perceived value. To understand which objectives are important, companies significantly invest in user research. User research methods include, among others, surveys, online logging, controlled evaluations, and observational studies. Methods like these are used to chart the needs, practices, capabilities, and technical contexts of users. However, it is widely accepted that the quality of design is determined in actual use. This creates a tough challenge for design. A designer must anticipate how well users will perform, and how they will use and experience a design candidate. To this end, designers conventionally rely on design heuristics—well-founded rule-like conventions such as “do not use more than four colors to code information”—, design patterns, empirical evaluation such as usability and A/B testing, and personal experience [9].

2.3 UI Design as Optimization

In the creation process of a graphical user interface, several interdependent design choices have to be made that affect different human and technical factors. These choices are usually made manually by professional designers due to the lack of mathematical definitions of the according design task and the missing notion of an objective function that captures the important aspects of human behaviour. The success of a GUI is often determined by factors like usability, usefulness, learnability, and enjoyability [10]. Most of these qualitiy measures for GUIs can also be directly translated to keyboards, some being more prominent than others (e.g., learnability vs. enjoyability). This chapter describes the power and flexibility of optimization tools in the creation process of these UIs.

What sets combinatorial optimization apart from alternative computational methods? While many optimization and other computational methods allow the exploration of a much larger design space than otherwise humanly possible, mathematical optimization methods also provide a certificate about the quality of presented solutions. An optimizer can prove that a certain solution is within $p\%$ of the best achievable design — a guarantee that many alternative methods in design cannot provide. Moreover, these methods can be easily controlled by the designer through the means of constraints and objective functions; hence making optimization a flexible and transparent tool for design processes (see also [11]). In order to find suitable parameters for optimization models or to translate intuitive ideas of designers into a mathematical notion, machine learning models can be used. In general, however, large training sets are not required for the setup of optimization models.

Despite these advantages a frequent challenge of optimization methods are design tasks to be notoriously ill-defined processes [12]. Classic optimization approaches require an explicit a priori formulation of the model under consideration. The fitness of many user interfaces, however, is best evaluated when in actual use — a value that is hard to anticipate beforehand. Furthermore, designers' work is concerned not only with a concrete layout but also with hard-to-formalize conceptual, structure-based, functional, and aesthetic aspects of design [13]. To this end, designers consider multiple types of constraints when they create, shape, and determine use-oriented qualities [13, 14]. Their success draws on their capabilities in creativity, problem-solving, sensemaking, empathy, and collaboration [13, 15, 16, 17, 18, 19]. They continuously engage in refining the objectives and constraints of design [14, 20]. To explore their ideas, they sketch and implement rapid prototypes [21]. This process is iterative, selective, and corrective: at times, they explore the design space for satisfactory approaches and then switch to in-depth analysis of the problem to identify a hypothetical best design. They alternate between a constructive and a critical stance. Criticality allows them to assess which aspects of a solution belong together and which are compatible with background data. At this stage of the design process, optimization can contribute not only optimal designs, but also surprising alternative solutions, design with particular tradeoffs, designs that find the best compromise between competing objectives, and designs that are robust to variations of the usage conditions [6]. Recent advances in optimization in the field of human-computer interaction also allow the designer to interactively participate in the optimization process. Optimizers can be integrated into design tools to assist designers in sketching, wireframing, and prototyping. Moreover, optimizers can be integrated into

software systems that adapt user interfaces to individual users.

2.3.1 Basic Concepts

User interface design is formulated as algorithmic combination of discrete design decisions utilized with the goal of obtaining an optimal solution defined by an objective function. The following definition of a *design task* extends the definition of the book chapter [22]. Their definition does not include the parametrization of the objective function, which however is crucial for us in the rest of this thesis.

Definition 2.1. Let x be an n-dimensional *design vector*, each dimension describing a *design variable*, X be the set of *candidate designs*. Furthermore, let f be the *objective function*, and θ be a set of parameters defining the *task instance*. We define a *design task* as

$$\max f_\theta(x) \text{ such that } x \in X$$

The design space X contains all design vectors that are appropriate for the design task. For any design vector $x \in X$, x_i denotes a single decision, which can be either integer or continuous. x_i can, for example, decode the size, position, or type of certain elements. Using this formulation and given that X is known a priori, the size of X can be estimated and grows very large for UI design problems, in general. Therefore, a common technique to reduce the complexity of the design space is to generalize certain decisions; e.g., instead of allowing arbitrary placements of items on a GUI, these placement decisions are discretized to a grid.

2.3.2 The Objective Function

What makes a design good or bad? In combinatorial optimization, this evaluative (and often subjective) knowledge must be expressed mathematically.

Formally, an objective function maps a design candidate $x \in X$ to a real number: $f_\theta : X \to \mathbb{R}$. The set of parameters θ may influence the outcome of this function. In many practical examples, f consists of k real-valued objectives ($k > 1$) representing different qualitiy measures. The parameters θ determine then how these k values are aggregated to a single objective function; for example, by calculating the weigthed average of all objectives. In this case, the parameters are part of θ. A more detailed discussion of so-called multi-objective optimization is done later in this section.

While many approaches to express evaluative knowledge as a formal objective function focus on the coverage of relevant factors, these models often come with a performance handicap. Even though they produce the most valid results, they are typically too inefficient for the optimization of large problems. The runtime for state-of-the-art solvers of optimization problems can rise to days or even months depending on the complexity of the model. On the other hand, the simplest linear programming formulations can often be solved within seconds, but lack the modeling power of the more complex formulations. The main challenge when choosing an objective function is the tradeoff among computational performance, model validity, and representational flexibility.

The Keyboard Optimization Problem In anticipation of the upcoming chapters in this thesis, we discuss important cornerstones regarding objective functions of keyboard optimization problems, a prominent optimization problem in HCI.

Feit discusses several objective functions for keyboard optimization, addressing different human factors and types of input devices [23]. Models and heuristics have been proposed to optimize keyboards for one-finger pointing performance, chorded finger movement performance, touch typing performance, reduction in muscle fatigue and strain, and ideal motor complexity. Special emphasis is put on attention, perception, and cognitive aspects of use, such as learning and navigation, as well as experimential aspects like aesthetics.

An important ingredient in many objective functions for keyboard optimization is *Fitt's law* (see e.g. [24]).

Definition 2.2. Let D be the distance between two objects s and t on an input device and W be the width of the target object measured along the axis of the movement. Fitt's law predicts that the time required to move (a finger or a hand) from s to t is

$$a + b \cdot \log_2 \left(\frac{2D}{W} \right)$$

where a and b are parameters dependent on the input device and user-specific factors.

Functions based on Fitt's law are common examples for parameterized objectives. In many cases, these parameters (here: a and b) have to be calibrated for the particular task instance, e.g., by preference elicitation methods [25] or by statistical parameter fitting to some dataset.

Multi-objective Solutions As already outlined for the keyboard optimization problem, GUI design typically involves multiple objectives, comprising different criteria like ergonomics, aesthetics, performance, etc. The resulting objectives f_i for $i = 1, \ldots, k$ may conflict with each other, i.e., the optimal solution for one objective may perform poorly for other objectives. This is a well-known topic in the field of combinatorial optimization but relatively little considered in this application area.

A common technique is to combine the objectives into a single objective expression. This can be done in many ways – by using weighted sums, the lexicographic method, goal programming methods, etc. [26]. The resulting problem can then be solved via well-established algorithms [27]. In the lexicographic method, we define a hierarchical ordering of the objective functions. The optimizer then iteratively finds an optimal solution to one objective function while not degrading higher-ordered objectives. Recent generations of MIP solvers have introduced the specification of multiple objectives so that the user does not have to manually handle the various techniques discussed above. If, however, there are no preferences between the objective functions and it might then be unreasonable to assign weights or a hierarchy to them, so-called *No-preference methods* provide an alternative strategy. The goal-programming method, for example, calculates the ideal objective vector $z_i^* = \inf_{x \in X} f_i(x)$ for $i = 1, \ldots, k$ and scalarizes the objective function to

$$\min_{x \in X} ||f(x) - z^*||_1$$

Alternatively, one can choose one function f_j as the main objective function and bound the values of the other objective functions in the constraints as $f_i(x) \leq \varepsilon_i$. This method is usually referred to as the ε-constraint method [28]. Other approaches involve meta-heuristic algorithms [29]. Here, the problem is solved with respect to all objectives and then the Pareto front of all solutions is computed [30, 31]. Commonly used algorithms for this approach include Non-dominated Sorting Genetic Algorithm II (NSGA-II) [32] and Strength Pareto Evolutionary Algorithm 2 (SPEA2) [33]. The most common approach in GUI computation has been the weighted sum approach, which is sensitive to objective weights.

2.3.3 The Task Instance

A design task may entail many task instances depending on the specific parameter values that are chosen. The *task instantiation* refers to the defining these parameter values that describe the specific design problem at hand. This includes the concrete set of elements that are considered for a design, the weights of multiple objective functions, empirical parameters (like in Fitt's law) or solver-specific tuning parameters.

Task instantiation is often a problem in UI design because the task cannot be precisely defined at the outset. If multiple stakeholders participate in the design process, they might have competing views on which elements to include in the design or which values to assign to the parameters. Additionally, the impact of parameter changes is not obvious before the optimization process although the outcome may be very sensitive to small variations of the input. Finally, tuning parameters of the solver require a deeper understanding of the solver's functionality, which is often out of the scope of the UI designer.

Regarding task instantiation, the literature distinguishes several approaches, which can be considered complementary. While we only shortly mention the most important approaches for the sake of completeness, we want to specifically emphasize the field of interactive optimization because it will be relevant for the rest of this thesis.

In *Design Mining*, parameters, constraints, and objective functions are learned from a data set [34, 35, 36, 37, 38].

Online Learning discovers parameter values experimentally. Different values for key parameters are evaluated by a group of experimental users and the outcome quality is measured. See [39] for the definition of parameter values by Bayesian optimization and [40] for crowd-based acquiry of parameters.

Robust optimization assumes that input values are either changing or uncertain [41, 42, 43]. A designer can indicate the uncertainty of the input in form of, for example, probability distributions. These distributions are used to generate multiple task instances, resulting in diverse candidate solutions that reflect the most probable interpretations of the designer's input and are robust to small changes in the input [6].

Interactive optimization allows human intervention in the optimization process [44, 45, 46, 47]. Interactivity is beneficial when the design problem cannot be precisely articulated. When incorporated into a design tool, an optimizer can assist in creative problem-solving and potentially nudge novice designers toward better, more usable designs [48].

Meignan et al. [46] offer a comprehensive review of interactive optimization approaches. *Search-oriented* interaction lets the designer provide additional information during the optimization process. The user can take any of several roles in this process: In the *assisting* role, the designer modifies solutions. In the *guiding* role, the designer steers exploration by affecting decision variables. In the *tuning* role, the designer sets solver-specific parameters. *Model-oriented* interaction is an alternative to search-oriented. It allows the designer to modify and refine the optimization model during the optimization process. This, in turn, is divided into two approaches. The *adjusting* designer can change parameters in the objective function or constraints when the objective is not fully known at the beginning of the process. The *enriching* designer can add objectives or constraints. The initial model is assumed to be incomplete, and solutions are invalidated or validated in light of interactive feedback. The literature has not provided guidelines for choosing between approaches; their usability is heavily dependent on the concrete optimization task.

A subset of these techniques has been applied to graphical UI design. Techniques for defining input include control panels [47], preference elicitation [25], constraint editing [49], and storyboarding [50]. Techniques for interacting with optimizer outputs include interactive example galleries [51, 52, 53], pareto front visualizations [54], localized suggestions on a design canvas [54], and localized critique of design outputs [25]. During search, one can steer an optimizer by selecting promising designs (biasing the local search) [53], optimizing part-solutions, visualizing optimization landscapes for steering, and locking part-solutions that are ready [54, 55].

CHAPTER 3

The French Keyboard Problem

The following chapter describes a joint project of Anna Feit, Antti Oulasvirta, Mathieu Nancel, Daryl Weirl, Andreas Karrenbauer and myself. In this work, we design, model and optimize a new keyboard tailored to the French language. My major contribution is an integer programming based algorithm that solves keyboard optimization problems and is dedicated to the special structure of the French keyboard problem. A manuscript reporting our experiences and insights from this national-scale project is currently reviewed for the Communications of the ACM [1]. Parts of this chapter are infered from this manuscript without further annotation.

The French language uses accents (*é*, *à*, *î*, etc.), ligatures (*œ* and *æ*), and specific apostrophes and quotation marks (*’ « » “ ”*). While the most commonly used special characters are available on the traditional French keyboard — the so called Azerty, which is depicted in Figure 3.1 – many letters are awkward to reach or cannot be typed at all. Consider the following sentence. `« À l’évidence, l’œnologie est plus qu’un ‘hobby’. »` (“Evidently, wine-making is more than a ‘hobby’.”). The underlined characters are not present, including non-breaking spaces and curved apostrophes. Users who want to write correct French have to rely on software-driven autocompletion or insert rarely used characters via Alt codes or by copy-pasting them from other documents. Since the French ministry of culture was concerned that the current keyboard hinders the proper use of the French language, they wrote an open letter to the French Parliament in 2015 [56]. The letter criticizes the existence of many unofficial French keyboard layouts, which all lack special characters for typing correct French. Some French people were even taught that accents on capital latters (e.g.,*É*) are optional because they are not present on the Azerty keyboard. One year later, AFNOR — the national organization for standardization — was tasked with designing a new keyboard standard [57], a project we joined as HCI and optimization experts. An official standardization committee was

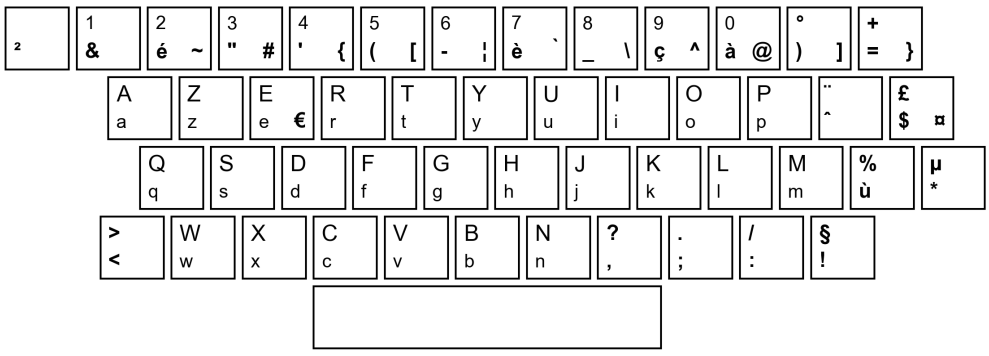

Figure 3.1: The Azerty keyboard standard

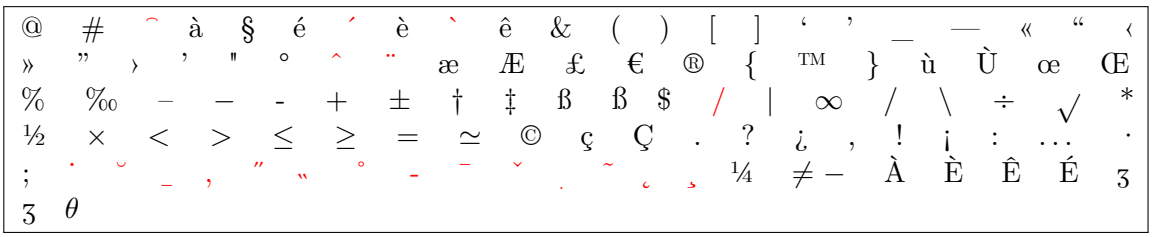

Figure 3.2: Example set of special characters (107). In red are diacritic marks; entered via dead keys

founded — composed of Ministry of Culture representatives and experts in ergonomics, typography, HCI, linguistics, and keyboard manufacturing. A typical standardization process involves meetings to iterate over each aspect of the standard and its wording. Also, AFNOR requires final drafts to be open to public comment.

3.1 The Design Task

The first question to be answered is ”What is a good keyboard?”. While the initial idea might be a keyboard where I can type fast, there are many other (often competing) concepts of good layouts: familiarity to other layouts, discoverability of characters, and support of an expanded character set. Additionally, everyday language varies widely from programming tasks or social media usage; hence, it has to be discussed which user the keyboard wants to appeal to or if we can find a good compromise for all of them.

Secondly, which uses of language to support is tricky to know in advance and, as we learned, a politically loaded question. To decide where to put #, we must weigh the importance of social-media-type language against “correct” literary French, in which that character is rare. Decisions on character positions mean trading off many such factors for a large range of users and typing tasks.

Our task was to develop an improved special-character layout that enables accessing all characters used in the French language and its dialects [1], reflects modern computer use (especially programming and social media), and supports scientific and mathematical characters (e.g., Greek letters) alongside major currency symbols and all characters in Europe’s other Latin-alphabet languages.

There were several challenging requirements. Despite having to add many new characters, the goal was to assign the special characters to the available keyslots such that the keyboard is easy to use and typing French is fast and ergonomic. The process saw the set of characters change frequently; shown in Figure 3.2 is the set in the final layout (the last 24 characters displayed were not part of the optimization problem but added later). The physical layout follows the alphanumeric section of the [58] standard. Each key can hold up to four characters, using combinations of the Shift and AltGr modifiers. For non-accented letters (“AZERTYUIOP...”), digits, and the space, the layout had to remain as in traditional AZERTY, leaving 129 keyslots. The only characters that could be added or moved were the *special characters* described in Figure 3.2; their number,

[1] See `http://www.culturecommunication.gouv.fr/Thematiques/Langue-francaise-et-langues-de-France/Politiques-de-la-langue/Langues-de-France`.

up to 122, depended on the design case. Note that diacritical characters are entered via "dead keys": visible output is only produced upon subsequent input of a letter.

3.2 The Timeline of the Project

When we joined the project the committee was debating each character in its hand-crafted layouts. The subjective explanations, like *"ê is **frequent**, so I gave it direct access because it's **faster**"*, often contained a hidden mathematical background. They were based on intuition, though measurement (of frequency, speed, etc.) was possible. Our first step towards a mathematical model was to turn these ideas into quantifiable objective functions. An additional challenge was the fact that many initial proposals that were constructed by the committee focused on improving a certain objective while compromising others. Their approach inherited from a greedy nature: they first fixed the placements that they considered the most important and then assigned the rest of the characters to remaining slots. The outcome of such a procedure can vary greatly depending on the greedy assignment order, which lead to huge variations of the proposals of different stakeholders.

How can combinatorial optimization assist this complex, mulit-objective problem? We discussed several advantages, but also challenges in the previous chapter. First of all, formalizing the problem as a multi-obecjtive optimization model allows algorithms to compute a solution that considers all objectives at once. Priorities of different stakeholders and compromises between different objectives can be expressed as weights in the formal objective function. Additionally, the impact of manual changes or the quality of ad-hoc designs can be directly reflected using the objective functions. At the beginning of our participation in this project, we evaluated the first hand-crafted layouts and showed that — compared to an optimized keyboard — they showed a severe deficiency in typing performance and ergonomics. The final optimized solution, for example, which formed the basis for the new keyboard standard, outperformed the typing speed and ergonomics of one of the early ad-hoc designs by almost 50%.

After convincing the committee that combinatorial optimization is an important and helpful tool to assist the design of the new keyboard layout, we iteratively defined and adjusted the optimization model to match the committee's intuitions and expectations.

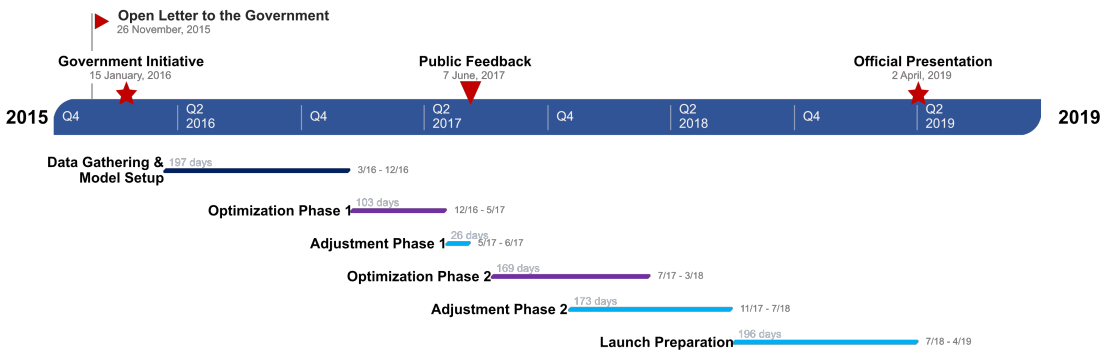

Figure 3.3: Project timeline: Computational methods were involved in all phases but the public comment, governed by interactions with stakeholders.

When we agreed upon an initial model, and the optimizer computed a layout, which was very close to optimal (usually $< 5\%$ optimality gap), the stakeholders requested manual adjustments on this layout. We explained the impact of different constraints and objective parameters to the presented solutions, which lead to a proposal of several parameter changes, the addition/removal of certain characters or the addition/removal of special constraints; e.g., keeping capital and lowercase letters on the same key vs allowing an arbitrary placement of upper- and lowercase special characters. In every iteration, we optimized this new model, discussed a new near-optimal solution, and repeated the process of discussing adjustments and altering the model parameters. Over the course of nine months, the model evolved into a first concensus. After this first *Optimization* phase, final adjustments to this layout have been made by the experts. These adjustments were mandatory according to the experts because they captured exceptions to the optimization model that could not be captured during the optimization phase, like cultural norms or frequently changing character-specific political decisions. Again, the optimizer assisted this *Adjustment* phase by evaluating the impact of manual changes to the objective functions. By utilizing our mathematical model, the committee members could make better-informed decisions and achieve a better tradeoff between individual preferences and the impact to performance or any other objective function.

After a compromise had been found, the first release candidate was presented to the public in June 2017 (see also the timeline in Figure 3.3). According to AFNOR's standardization procedure, the proposed keyboard was publicly available for one month and people could provide feedback. We received over 3,700 suggestions and comments, which were strongly divided on certain aspects of the layout. The placement of programming-related and accentuated characters or the decision of whether or not digits are on shifted keys, for example, appeared to be very controversial. The general consensus of the feedback was then incorporated into the optimization model: we added new constraints, changed weights and used different character sets.

A second cycle of optimization and adjustment phases began until the final layout was agreed on. This final keyboard achieves good scores in objective and subjective criteria, defined and constantly evolved by the committee and the public comments. In April 2019, this layout was approved and presented as an official French keyboard standard in the National Assembly in Paris [59] (see [60] for the English translation).

In conclusion, dedicated optimization tools assisted the design process in not only computing good layouts, but also understanding the underlying optimization problem and the impact of changes to the model or its solution. These tools enabled stakeholders to monitor the effects of their ideas in a quantifiable fashion, leading to an overall more transparent design process.

3.3 The Optimization Model

We model the keyboard optimization problem as an ILP. To this end, we introduce binary decision variables x_{ik} for every character $i \in [N]$ and key slot $k \in [M]$. This variable is 1 if and only if this character i is assigned to the key k.

Every character has to be assigned to a key, which is reflected in the following

constraints for every $i \in [N]$.

$$\sum_{k=1}^{M} x_{ik} = 1 \tag{3.1}$$

Additionally, keys can not hold more than one character. We introduce the constraints for every $k \in [M]$

$$\sum_{i=1}^{N} x_{ik} \leq 1 \tag{3.2}$$

This differentiation between these two types of constraints happens because there are more keyslots on the keyboard than special characters in this case; hence, some slots remain empty.

The design problem was formulated as an integer program (IP), which lets us use effective solvers that provide intermediate solutions with bounds on their distance to optimality. Every feasible binary solution corresponds to a keyboard layout.

3.3.1 Objective Criteria

An objective function measures the goodness of each layout according to each of the criteria. Our first main challenge was to translate intuitive goals such as "facilitate typing and learning" into quantifiable objective functions. We consider the following four different objective criteria that are aggregated by weights w to a single objective function [2].

Performance The performance objective rewards if frequent special characters can be quickly entered in combination with the fixed letters. It is quantified by computing the average time to type a special character before or after any of the regular letters (T_{ck}, T_{kc}), weighted by the special-character–regular-letter pair (p_{ci}, p_{ic}). The corresponding data was gathered in a crowdsourcing-based study.

$$\sum_{i=1}^{N} \sum_{k=1}^{M} \sum_{c=1}^{27} (p_{ci} T_{ck} + p_{ic} T_{kc}) x_{ik} \tag{3.3}$$

Ergonomics This objective criterion penalizes keyslots that require extreme movements putting strain on tendons and joints, which are empirically associated with repetitive strain injuries [61]: extreme outward or inward movements of the wrist ($W_k \in \{0, 1\}$), extreme extension of fingers ($F_k \in \{0, 1\}$), and use of one or two modifier keys ($M_k \in \{0, 1, 2\}$). The score is weighted by the frequency (p_i) of the character assigned to the keyslot.

$$\sum_{i=1}^{N} \sum_{k=1}^{M} p_i (W_k + F_k + M_k) x_{ik} \tag{3.4}$$

[2] See the discussion of multi-criteria optimization in Chapter 2.

Familiarity The familiarity objective models the placement of frequent characters near the position of them in the traditional AZERTY, to facilitate visual search with the new layout [62]. $D_{k\mathcal{A}(i)}$ quantifies the distance between the keyslot k assigned to the character i and its AZERTY position $\mathcal{A}(i)$, weighted by that character's frequency (p_i).

$$\sum_{i=1}^{N}\sum_{k=1}^{M} p_i D_{k\mathcal{A}(i)} x_{ik} \tag{3.5}$$

Intuitiveness We minimize the distance between similar special characters (D_{kl}) and between special characters and similar letters (D_{kc}), to facilitate discovery and learning [63]. This similarity can be syntactic or semantic and is captured by the scores s_{ij}, s_{ic}. All characters are considered equally important for grouping. Note that so far every objective function only contained linear terms. The relation between similar special characters in this objective, however, is quadratic. Although the similarity matrix is very sparse (about 2% of all pairs of special characters are considered similar), this quadratic term makes the problem significantly harder to solve.

$$\sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{k=1}^{M}\sum_{l=1}^{M} s_{ij} D_{kl} x_{ik} x_{jl} + \sum_{i=1}^{N}\sum_{k=1}^{M}\sum_{c=1}^{27} s_{ic} D_{kc} x_{ik} \tag{3.6}$$

3.3.2 The Full Model

During the standardization process, different additional, but simple constraints arose; for example, requiring to place capitalized special characters on the shifted key of their corresponding uncapitalized versions. This family of constraints can be written as equality of two decision variables, which is internally simplified by ILP solvers: one of those variables is projected out. More details on additional constraints are discussed in [23].

The parameters, constraints, and objectives of the integer program reflect the standardization committee's goals: facilitate typing of correct French, enable the input of certain characters not supported by the current keyboard, and minimize learning time by guaranteeing an intuitive-to-use keyboard that is sufficiently similar to the previous AZERTY. For a more detailed discussion of each criterion we refer to [23, 60].

The complete integer programming formulation for our keyboard optimization problem is depicted in Formulation (3.7). For the instance that led to the standardized layout ($N = 85$, $M = 129$), the following weights were chosen: $w_P = 0.3$, $w_E = 0.25$, $w_I = 0.35$, $w_F = 0.1$. This formulation is a special case of the quadratic assignment problem, firstly modelled by Pollatschek in 1975 [64], two years later extended by Burkard and Offermann [5] and later used in several other keyboard related works [23, 65]. The theoretical foundation of QAPs is discussed in Section 4.2.

$$
\begin{aligned}
\min \quad & w_P \sum_{i=1}^{N}\sum_{k=1}^{M}\sum_{c=1}^{27}(p_{ci}T_{ck} + p_{ic}T_{kc})x_{ik} \\
+ \quad & w_E \sum_{i=1}^{N}\sum_{k=1}^{M} p_i(W_k + F_k + M_k)x_{ik} \\
+ \quad & w_F \sum_{i=1}^{N}\sum_{k=1}^{M} p_i D_{k\mathcal{A}(i)}x_{ik} \\
+ \quad & w_I \left(\sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{k=1}^{M}\sum_{l=1}^{M} s_{ij}D_{kl}x_{ik}x_{jl} + \sum_{i=1}^{N}\sum_{k=1}^{M}\sum_{c=1}^{27} s_{ic}D_{kc}x_{ik} \right) \qquad (3.7)
\end{aligned}
$$

subject to

$$
\begin{aligned}
& \sum_{k=1}^{M} x_{ik} = 1 \quad \forall i \in \{1, \dots N\} \\
& \sum_{i=1}^{N} x_{ik} \leq 1 \quad \forall k \in \{1, \dots M\} \\
& x_{ik} \in \{0, 1\} \quad \forall i \in \{1, \dots N\}, k \in \{1, \dots M\}
\end{aligned}
$$

All objective functions discussed above rely on real-world input data that reflect the typing of French users. Therefore, our colleagues (see [23, 66]) gathered large text corpora, with varied topics and writing styles, and weighted them in accordance with the committee's requests. The corpora can be categorized into three types. **Formal** text is curated text with correct French and proper use of special characters. Sources include the French Wikipedia, official policy documents, and professionally transcribed radio shows. **Informal** text has lower standards of orthographic, grammar, and typographic correctness. This includes text in social-media or personal communication. The material we used in this project includes anonymized email and popular accounts' Facebook posts and Tweets. The **Programming** corpora consist of publicly available programming projects using the most commonly used programming and description languages: Python, C++, Java, JavaScript, HTML, and CSS (comments removed). ELDA[3] provided many of the corpora that were classified Formal and Informal [66]. The Programming corpora were extracted from open source platforms like github. Frequencies were computed by corpus, then averaged per character and class, and finally assigned weights subject to committee discussion (Formal: 0.7, Informal: 0.15, Programming: 0.15). Since I did not participate in this early data gathering stage of the project, I only present a high-level overview of this part. More details on the impact of these corpora and the impact of these weights can be found in [60], which is the English translation of [66]. Table 3.1 shows the most common characters in each category. As expected, the frequencies show various differences among the different corpora categories. The characters # and @, for example, appear in the table only for the Informal class, which stems from the fact that they are almost exclusively used in internet-related texts. The common accented letters *é*, *à*, *è*, and *ê* are appear in the same relative order in both Formal and Informal columns; however, they are used much more frequently in Formal corpora, which contains only

[3] The Evaluations and Language resources Distribution Agency; see `http://www.elra.info/en/about/elda/`.

Formal		Informal		Programming	
Char.	Freq. (%)	Char.	Freq. (%)	Char.	Freq. (%)
é	1.883	#	1.139	.	1.584
,	0.896	é	1.074	-	1.315
.	0.796	/	0.895	(	1.310
'	0.765	.	0.805	)	1.309
à	0.332	!	0.712	;	1.158
-	0.262	@	0.648	=	1.035
è	0.241	:	0.497	_	1.002
)	0.156	'	0.457	,	0.926
(	0.141	,	0.447	:	0.922
:	0.135	à	0.269	"	0.918
'	0.118	-	0.209	>	0.527
ê	0.098	"	0.185	/	0.459
/	0.078	è	0.155	<	0.445
!	0.075	'	0.129	{	0.444
;	0.058	ê	0.099	}	0.443
"	0.047	_	0.079	'	0.292
»	0.041	;	0.075	[	0.186
ç	0.041	&	0.068	]	0.186
«	0.041	)	0.063	%	0.150
?	0.040	«	0.059	+	0.144

Table 3.1: The highest-frequency special characters, by category of French text.

curated text. Interestingly, / is present in all three columns, because of its wide range of uses.

In order to predict the typing performance, we extracted key-to-key typing durations from an extensive dataset gathered in a crowdsurcing study with over 900 participants [23, 60]. Special emphasis was put on the time it takes to access a special character keyslot before and after a regular letter. To this end, time data for all combinations (7,560 pairs) of regular and special character key slots were gathered.

For the Intuitiveness objective, we defined a similarity score between characters as a scalar in the range $[0, 1]$, depending on visual proximity (R and ®, ^{o} and $^\circ$, etc.), semantic proximity (e.g., $\times$ and $*$, or $\div$ and /), inclusion of other letters ($ç$ and c, $æ$ and o, etc.), practice (n and ~, e and ´, etc.), or use-based criteria such as lowercase/uppercase and opening/closing character pairs. These intuitiveness values were extensively discussed with the committee and frequently updated during the optimization phases of the project.

3.4 The new French Keyboard Standard

The new French keyboard standard is shown in Figure 3.4; it provides a larger set of characters than the AZERTY keyboard and thus simplifies typing of correct French. Despite the problem's computational complexity [4], we could show that the solution our optimization tools computed is at most 1.98% worse than the best achievable design for the stated problem. This optimized solution was the basis of the final outcome, to which the committee added 24 further, rarer characters. These new character placements were evaluated using the optimizer's objective functions, allowing the committee to locally optimize the layout's intuitiveness.

[4]Complexity statements and other theoretical discussions are done in Chapter 4

Figure 3.4: The new AZERTY layout [59]. The characters included in the design problem are in boldface and color. Marked in red are dead keys.

The new layout enables direct input of more than 190 special characters, a significant increase from the 47 of the current AZERTY[5]. As a consequence, it is possible to access all characters used in French without relying on software-side corrections. Frequently used French characters are accessible without any modifier (*é*, *à*, *«*, *»*, etc.) or intuitively positioned where users would expect them (e.g., *œ* on the *o* key). All accented capital letters (*À*, *É*, etc.) can be entered directly or using a dead key. The main layout offers almost 60 characters not available in AZERTY for entering symbols used in math, linguistics, economics, programming, and other fields. Some programming characters, often having other uses too, were given more prominent slots; for instance, / became accessible without modifiers, and \ is placed on the same key but in a shifted slot. Although the new layout contains many more characters than the traditional AZERTY keyboard, the performance and ergonomics of typing those special characters that are already present in the old AZERTY are improved by 18.4% and 8.4 %, respectively.

The keyboard offers three additional layers accessed via special mode keys. These are dedicated to European characters not used in French (via the Eu key from Alt+H in Fig. 3.4), currency symbols (via ¤ with Alt+F), and Greek letters (via Alt+G's µ), more than 80 additional characters in all. Their placement was beyond the scope of the optimization process, being inconsequential to typing regular French.

One goal of the optimization process was to maintain the similarity of the new layout to the traditional AZERTY, making the transition for users easy. Of the 45 special characters previously available, eight retained their original location and 12 moved by less than three keys. In particular, frequently used characters were kept near their original position. For instance, the most common special character (*é*) is not in the fastest spot to access on average. It stayed at its original spot for similarity and maintaining good performance. Many punctuation characters were moved slightly by the optimizer to better reflect character and character-pair frequencies (see Table 3.1) while remaining in the expected area of the keyboard. If characters were moved by a large margin, the

[5] Not including accented characters that can be created using dead keys, such as ^ + I = Î.

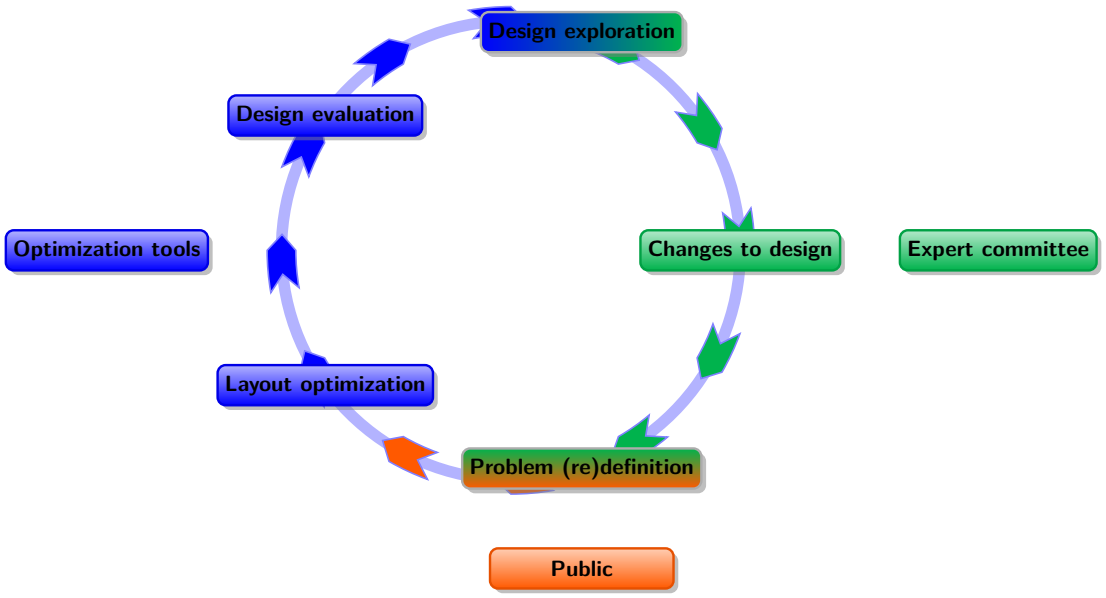

Figure 3.5: The participatory optimization process of the standardization procedure.

advantages in performance, ergonomics and intuitiveness outperformed the similarity measure so much that the move is justified. For instance, opening and closing brackets were moved by many keys in order to bring them close together — a direct result of the public feedback.

Finally, substantial effort was devoted to forming semantic regions for characters, such as mathematical characters, common currency symbols, or quotation marks. Many of these groupings emerged during the optimization process, thanks to the intuitiveness objective. Others resulted from manual changes when the committee decided to prioritize semantic grouping over performance or ergonomics (e.g., following a calculator metaphor for mathematical characters). The Intuitiveness score improved more than fourfold (434.4%) relative to traditional AZERTY.

We published a full visual comparison of the old and new keyboard layout online [6], which also contains explanations (in terms of the objective functions) for all characters' placements.

3.5 Participatory Optimization

In the beginning of the project, we assumed that we could apply standard optimization tools and just compute the best solution in one shot; however, we quickly realized that the nature of the problem forbids this method. This problem was inherently ill-defined and constantly evolving: definitions and objectives changed, and decisions often hinged on subjective opinions, public feedback, or cultural norms, rendering them hard to express mathematically — a typical challenge that optimization in HCI often faces as already discussed in Chapter 2. Over the course of the project, our approach evolved toward

[6] See `http://www.norme-azerty.fr`.

something one could call *participatory optimization*. This is inspired by participatory design, which originated with labor unions and was developed as a co-design method aimed at democratic inclusion of stakeholders [67]. Equal representation and resolving conflicts beign two key aims, the optimization tools have to fulfill the additional purpose to create a level playing field for the stakeholders. If the stakeholders are able to inform and influence each other on the basis of an interactive optimizer, a good solution can be found collaboratively.

The diagram in Figure 3.5 shows the interactions between our optimization tools, the stakeholders and the public (after the feedback phase). On the one side, the optimizer computed solutions, evaluated alternative layouts or manual changes, helping the committee to explore the design space and take well-informed decisions. On the other side, the committee manipulated constraints, objective parameters, and adjusted the character sets. They used the proposed optimized solutions to explore the design space and better understand the impact of certain parts of the optimization model. Simultaneously, both sides were informed by comments from the public, whose expectations and wishes led the experts to question their assumptions and criteria. That led directly to several changes in the weight and constraint definitions within the optimization model. The final design was the outcome of this interaction and could not have been achieved with either side working alone.

In summary, it was impossible to rigorously define the optimization model in the beginning and run classical one-shot optimization methods until an optimal solution is found. There is growing interest in optimization research employing methods that actively include the user in the process, such as *interactive optimization*, which was already discussed in Section 2.3.3.

The notion of participatory optimization goes even beyond this. It focuses particularly on including stakeholders at every step in the process, for which state-of-the-art optimization methods provide limited support. We identified different challenges to address in future work in order to enable active participation of stakeholders *and* optimizer in design processes supported by optimization methods.

Fast (re)definition of the problem In ill-defined design problems, the problem definition is constantly evolving. In most cases where the optimization model changes, state-of-the-art solvers dismiss all information collected during the optimization of the previous model. The ability to reuse this information about previously explored solutions, such as pruning of certain branches in a branch-and-bound tree, could massively increase the solver's performance. The main challenge here is to create an adaptive data structure that not only stores the information of a branch-and-bound tree, but also identifies those subtrees that are affected by certain model changes and have to be reevaluated.

Exploring and understanding the design space An important step in creating a design is the exploration of the design space. Learning the impact of certain constraints or local changes of the model to its outcome is essential to better understand the design problem. Ideally, optimization tools need to provide interfaces that allow stakeholders to propose changes to solutions or input parameters. Moreover, the resulting consequences of these changes need to be communicated in a human-readable format.

Learning subjective optimization criteria When the stakeholders in our project made manual adjustments to a proposed solution, they often applied tacit criteria such as assumptions about users' habits, cultural specificities, subjective preferences, and political agendas. Subsequently, we formalized these subjective criteria to quantifiable functions whenever possible. If optimizers offered an interface, which allows users to propose subjectively better solutions than the ones that are computed by the optimizer, this information could be used to learn a new *subjective function* that could be included and iteratively adjusted in the optimization model.

Justifications for design choices When we proposed a layout to the committee or to the public, a frequently asked question was why certain characters were placed at their proposed key slots or what would happen if two characters were exchanged. The answer that the proposed solution was optimal and an exchange would lead to a decrease in the objective functions might be mathematically correct, but not satisfactory for the general audience. Effective visualization tools that show the impact of manual adjustments regarding the different performance measures could help users to understand and accept the proposed solutions or serve as an argument to redefine certain parameters of the model.

3.6 Conclusion

This chapter discussed how combinatorial optimization can aid the design of a new keyboard in a multi-stakeholder project on a national scale. While this project was dedicated to the French language, our experiences and results can be easily transferred to other languages. Most keyboard layouts have evolved incrementally by adding upcoming characters to empty keyslots, resulting in a layout that is far from optimal or even misses important special characters. Additionally, many languages — even some of the world's most spoken ones[7] — lack any official keyboard standard. The procedures and frameworks that resulted from this collaboration can play an important role in guiding regulators to improve the quality or (re)define their keyboard design.

However, our experiences with this project also showed that much potential of computational methods remain unexploited, which mostly originates from the participatory nature of these design problems. Many mainstream algorithms are purely focused on problem-solving and neglect the exploration of the design space, which leaves this equally important part to human intuition or trial-and-error. We believe that, when designed from a participatory perspective, algorithms could more directly support not only problem-solving but also considering multiple perspectives, making refinements, and learning about a problem.

[7] For example, Pubjabi (10th), Telegu (15th), and Marathi (19th)

CHAPTER 4

Assignment Problems

Before we discuss the theoretical and algorithmic contributions to the French keyboard problem in the upcoming chapters, the following sections introduce the basic concepts of optimization theory used in this thesis and provide the theoretic foundation of assignment problems.

4.1 Basics of Optimization Theory

This section defines the basic structure of optimization theory. The notation and definitions are taken from [68].

4.1.1 Polyhedral theory

Let $\alpha \in \mathbb{R}^n$ be a vector and $\beta \in \mathbb{R}$ be a scalar. The set $H := \{x \in \mathbb{R}^n : \alpha^T x \leq \beta\}$ is called a *halfspace*. A polyhedron P is the intersection of finitely many halfspaces

$$P = \bigcap_{i=1}^{m} H_i := \{x \in \mathbb{R}^n : \alpha_i^T x \leq \beta_i\} \tag{4.1}$$

Formally, we can define the matrix $A \in \mathbb{R}^{m \times n}$ with the i-th row of A being α_i^T and $b \in \mathbb{R}^m$ with the i-th entry being β_i. Then, P is equivalently defined as

$$P = \{x \in \mathbb{R}^n : Ax \leq b\} \tag{4.2}$$

4.1.2 Linear Programming Theory

Let $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$, P the corresponding polyhedron, and $c \in \mathbb{R}^n$, then a linear program is defined as $\min_{x \in P} c^T x$ or equivalently

$$\begin{aligned} \min \quad & c^T x \\ \text{subject to} \quad & Ax \leq b \end{aligned} \tag{4.3}$$

In many practical applications, the so-called standard form of linear programs is used, which is defined as

$$\begin{aligned} \min \quad & c^T x \\ \text{subject to} \quad & Ax = b \\ & x \geq 0 \end{aligned} \tag{4.4}$$

Quadratic programs allow products of variables in the objective function. Formally, let $Q \in \mathbb{R}^{n \times n}$ be a matrix. A quadratic program is formulated as

$$\begin{aligned} \min \quad & x^T Q x + c^T x \\ \text{subject to} \quad & Ax = b \\ & x \geq 0 \end{aligned} \tag{4.5}$$

In mixed integer programs (MIP), variables can be constrained to take only integer values. Let $B \subseteq [n]$ be a subset of indices and let $P' := P \cap \{x \in \mathbb{R}^n : x_i \in \mathbb{Z} \ \forall i \in B\}$ Then, a MIP can be formulated as $\min_{x \in P'} c^T x$

4.2 The Assignment Problem

Assignment problems aim at finding a one-to-one correspondence between n items and n locations. An integer programming formulation of the linear assignment problem (LAP) is depicted in Formulation (4.6)

$$\begin{aligned} \min \quad & \sum_{i,k=1}^{n} c_{ik} x_{ik} & \\ \text{subject to} \quad & \sum_{i=1}^{n} x_{ik} = 1 & \forall k \in [n] \\ & \sum_{k=1}^{n} x_{ik} = 1 & \forall i \in [n] \\ & x_{ik} \in \{0,1\} & \forall i,k \in [n] \end{aligned} \tag{4.6}$$

Intuitively, the constraints decode the property that every item is assigned to a unique location. From a combinatorics perspective, if we consider the x-variables as a matrix $X \in \mathbb{R}^{n \times n}$, then these constraints force X to be doubly stochastic[1]. For further reference, we define

$$\Pi_n := \left\{ X \in \{0,1\}^{n \times n} : \sum_{i=1}^{n} x_{ik} = 1 \, \forall k \in [n] \text{ and } \sum_{k=1}^{n} x_{ik} = 1 \, \forall i \in [n] \right\} \tag{4.7}$$

The polytope in Equation (4.6) is often referred to as the assignment polytope or as the Birkhoff polytope. Birkhoff discovered in 1946 [69] that the set of $n \times n$ doubly stochastic matrices is a convex polytope with $n!$ many vertices — one for every permutation of n items. This last statement is known as the Birkhoff-von Neumann theorem. Alternatively, the polytope can be interpreted as the set of perfect matchings on the complete bipartite graph with n vertices $K_{n,n}$. These observation lead to algorithms solving the LAP in polynomial time. In 1955, Kuhn developed the Hungarian algorithm [70], which solves LAPs in $\mathcal{O}(n^3)$ time. More recently, Duan and Su proposed a bipartite matching algorithm [71], which computes a perfect matching in $\mathcal{O}(n^{\frac{5}{2}} \log C)$ time for integer costs of at most C.

While the LAP is very well understood today, the objective function can only model the effect of an assignment of one item to one location (c_{ik} in (4.6)). However, pairwise

[1] A matrix is doubly stochastic if the values of every row and every column sum up to 1.

dependencies between items and locations are impossible to capture in an LAP. Therefore, Koopmans and Beckman [72] investigated a quadratic variant of the assignment problem by adding quadratic terms to the objective function. The complete formulation is shown in (4.8).

$$\begin{aligned} \min \quad & \sum_{i,k=1}^{n} c_{ik}x_{ik} + \sum_{i,j,k,\ell=1}^{n} q_{ijk\ell}x_{ik}x_{j\ell} & \\ \text{subject to} \quad & \sum_{i=1}^{n} x_{ik} = 1 & \forall k \in [n] \\ & \sum_{k=1}^{n} x_{ik} = 1 & \forall i \in [n] \\ & x_{ik} \in \{0,1\} & \forall i,k \in [n] \end{aligned} \tag{4.8}$$

In their formulation, the quadratic cost term $q_{ijk\ell}$ factors into dependencies between items and dependencies between locations, i.e., $q_{ijk\ell} = f_{ij} \cdot d_{k\ell}$ or $Q = F \otimes D$ using the Kronecker product. Typically, f_{ij} is interpreted as flow between the items i and j and $d_{k\ell}$ denotes the distance between the locations k and ℓ. If F is a 0-1-matrix and D is nonnegative, an alternative combinatorial interpretation of the Koopmans-Beckman formulation is to treat F as an incidence matrix of a graph G_F and D as edge weights for a complete weighted graph G_D. In this case, the QAP corresponds to finding a minimal weight subgraph of G_D that is isomorphic to G_F. If the weights in D satisfy the triangle inequality $d_{ik} \leq d_{ij} + d_{jk}$, the underlying QAP is classified as a *metric QAP*. Even with this restriction on the quadratic cost terms, this variant of the QAP occurs in many practical applications: the facility location problem [73], the traveling salesman problem [74], the wiring problem [75], the hospital layout problem [76, 77], and finally also the keyboard layout problem, which was first considered as a QAP by Pollatschek et al. [64].

The QAP proved to be a notirously hard combinatorial optimization problem. Even if the cost can be factorized to a symmetric block-diagonal matrix and the distance matrix is restricted to a line metric, it is NP-hard to approximate the QAP within any constant factor [78]. Regarding experimental evaluation, the famous QAPLIB benchmark [79] still contains decade old unsolved instances with only 30 items. Other benchmark instances have been solved just recently by dedicated techniques that exploit certain structures of the certain problems [80] or by the application of massive computational power [81].

4.3 Tractable Cases of QAPs

Despite the already mentioned hardness of QAPs, there exist special cases where polynomial time approximation algorithms exist. An algorithm is called an *α-approximation* for an optimization problem $\mathcal{P}$ if for every instance the algorithm's solution value S and the optimal solution value OPT satisfy

$$\frac{S}{OPT} \leq \alpha$$

Note that for minimization problems, it holds that $\alpha \geq 1$ and for maximization problems $\alpha \leq 1$. We say that the approximation ratio α is tight under a certain complexity assumption (like $P \neq NP$) if an asymptotically smaller ratio would disprove this assumption.

Many of the following special cases of the QAP hold a combinatorial structure and can be written in the Koopmans-Beckmans variant with the interpretation as graphs. When describing the problem, we will discuss the structure of the two matrices F and D and the consequences to the corresponding graphs G_F and G_D.

The *Traveling Salesman Problem (TSP)* aims at finding a minimal tour in a complete weighted graph G that visits each node exactly once and returns to the start node in the end. We can model the TSP as QAP by the matrix F that contains a 1 for every entry $\{(i, i+1) : i \in [n-1]\} \cup \{(n, 1)\}$. and 0 elsewhere. We set D as the matrix of all edge weights in G. If additionally D satisfies the triangle inequality, i.e., we consider the metric TSP, then Christofides proposed a $\frac{3}{2}$-approximation algorithm [82]. The more special case of the metric k-TSP, which requires the tour to visit only k nodes, admits a 2-approximation [83].

A close relative to the TSP is the Hamiltonian Path Problem. The only difference between these two optimization problems is that the latter one does not require the path to return to the start node after visiting every node in the graph. Christofides' algorithm is transferable to the Hamiltonian Path Problem and also provides a $\frac{3}{2}$-approximation. If the start and end nodes of the Hamiltonian path are fixed, Hoogeveen shows that a modified version of Christofides' algorithm has a worst case approximation ratio of $\frac{5}{3}$ and also shows that this bound is tight.

The Linear Arrangement Problem consists of labelling the nodes in a graph G with the labels $\{1, \ldots, n\}$ such that the total sum of edge weights is minimized. The weight of an edge (i, j) is computed as the label difference of its corresponding nodes i and j. We can model this as QAP by setting F to the incidence matrix of G and defining D as $d_{ij} = |i - j|$. In 2007, Feige et al. proposed a $\mathcal{O}(\sqrt{\log n} \log \log n)$-approximation algorithm for this problem [84].

The following special cases of QAPs all assume D to satisfy the triangle inequality and F to be an incidence matrix of a graph. The results differ in the special graph structure that is imposed on G_F.

If G_F is a bounded degree (Δ) tree, there exists a $\mathcal{O}(\Delta \log n)$-approximation [85]. A spider is a tree with at most one vertex of degree ≥ 3. In the case of G_F being a spider, there exists a $(7 + 2\sqrt{6})$-approximation. If all paths from the root to the leaves of the tree have equal length, the approximation ratio can be improved to 3 [85]. The same authors also propose a 3-approximation for the Wheel QAP where G_F is a wheel graph. Given a graph with an even number n of vertices, a wheel is a Hamiltonian tour $\{(i, i+1 \mod n) | i \in [n]\}$ together with the edges $\{(i, i + \frac{n}{2} | i \in [\frac{n}{2}]\}$. Furthermore, they investigate the Double Tour QAP. A double tour consists of the edges of a tour $\{(i, i+1 \mod n) | i \in [n]\}$ plus their shortcuts $\{(i, i+2 \mod n) | i \in [n]\}$. For this case, Christofides' algorithm can be modified to a 2.25-approximation.

Usually, QAPs are stated as a minimization problem, which also applies to all special cases mentioned so far. The maximization version of the metric QAP seems a lot easier since it admits a $\frac{1}{4}$-approximation [86]. Without the metric property, the approximation factor increases to $\mathcal{O}(\sqrt{n} \log^2 n)$ [87].

4.4 Approaches to solve QAPs

A general approach to solve integer linear problems is to drop the integrality constraints, solve this relaxed version and find the optimal integer solution in a branch-and-bound framework. Solving the QAP requires one more step of relaxations. More specifically, a common procedure is to linearize the quadratic terms and then apply the usual techniques for integer linear programs. Recently, semidefinite relaxations for quadratic programs were proposed [88, 89]. In the following paragraphs, we will introduce important linear relaxation techniques first and then shortly discuss the basics of semidefinite programming approaches.

The Gilmore-Lawler Bound One of the earliest published lower bounds is the Gilmore-Lawler bound [90]. In a first step, the QAP is linearized by exchanging every product $x_{ik}x_{j\ell}$ by a new binary variable $y_{ijk\ell}$. These newly introduced y-variables are connected to the original assignment variables with the constraints of the following LP.

$$\begin{aligned} \min \quad & \sum_{ijk\ell=1}^{n} c_{ijk\ell} y_{ijk\ell} \\ \text{subject to} \quad & (x_{ik}) \in \Pi_n \\ & \sum_{ijk\ell=1}^{n} y_{ijk\ell} = n^2 \\ & x_{ik} + x_{j\ell} - 2y_{ijk\ell} \geq 0 \quad \forall i,j,k,\ell \in [n] \\ & y_{ijk\ell} \in \{0,1\} \quad \forall i,j,k,\ell \in [n] \end{aligned} \tag{4.9}$$

Now, by rearranging the sums in the objective function, we end up with n^2 independent linear assignment problems of the form

$$\begin{aligned} \ell_{ik} = \min \quad & \sum_{j\ell=1}^{n} c_{ijk\ell} y_{ijk\ell} \\ \text{subject to} \quad & \sum_{j=1}^{n} y_{ijk\ell} = 1 \quad \forall \ell \in [n] \\ & \sum_{\ell=1}^{n} y_{ijk\ell} = 1 \quad \forall j \in [n] \\ & y_{iikk} = 1 \\ & y_{ijk\ell} \in \{0,1\} \quad \forall j,\ell \in [n] \end{aligned} \tag{4.10}$$

Intuitively, these LAPs compute the minimum cost we have to pay assuming that we chose to assign i to k. These lower bounds on the cost-to-pay can then be fed back into the remaining global problem yielding another LAP of the form

$$\begin{aligned} \min \quad & \sum_{ik=1}^{n} l_{ik} x_{ik} \\ \text{subject to} \quad & (x_{ik}) \in \Pi_n \end{aligned} \tag{4.11}$$

We can derive conditions for when the computed solution is even optimal. Let $Y^{(ik)}$ be the solutions to the n^2 subproblems (4.10) and X^* be the optimal solution of (4.11).

We define $Y^* = \left(y^*_{ijkl}\right) = \left(x^*_{ik} y^{(ik)}_{jl}\right)$. Now, if it holds that

$$\frac{1}{n} \sum_{ik=1}^{n} x^*_{ik} Y^{(ik)} \in \Pi_n$$

then Y^* is a Kronecker product of two permutation matrices of dimension n. Hence, it is optimal for the QAP under consideration.

Li et al. showed that this bound deteriorates quickly for instances with increasing size [91].

Kaufman and Broeckx relaxation For this relaxation, let us assume w.l.o.g. that all costs are nonnegative (if not we can add a large constant to every coefficient without changing the optimal solution). Kaufman and Broeckx [92] proposed to rearrange the quadratic objective function and introduce n^2 new variables. As a first step, they rewrite the objective to

$$\sum_{ik=1}^{n} x_{ik} \left(\sum_{j\ell=1}^{n} c_{ijk\ell} x_{j\ell} \right) \tag{4.12}$$

Now they introduce the terms

$$w_{ik} = x_{ik} \sum_{j\ell=1}^{n} c_{ijk\ell} x_{j\ell} \tag{4.13}$$

$$c_{ik} = \sum_{j\ell=1}^{n} c_{ijk\ell} \tag{4.14}$$

and replace the products in the objective function by the new variable w. For every w_{ik}, one additional constraint has to be added to the formulation in order to express the connection between w and the old objective function. The complete linearization is shown in Equation (4.15).

$$\begin{aligned} \text{minimize} \quad & \sum_{ik=1}^{n} w_{ik} \\ \text{subject to} \quad & c_{ik} x_{ik} + \sum_{j\ell=1}^{n} c_{ijk\ell} x_{j\ell} - w_{ik} \leq c_{ik} \quad \forall i,k \in [n] \\ & (x_{ik}) \in \Pi_n \\ & w_{ik} \geq 0 \qquad \forall i,k \in [n] \end{aligned} \tag{4.15}$$

This integer program is shown to be equivalent to the QAP and its linear relaxation can be computed very efficiently.

Xia and Yuan The approach of Xia and Yuan [93] combines the ideas of Kaufman-Broeckx and Gilmore-Lawler. In their first step, the objective function is reordered equivalently to the first step of Kaufman-Broeckx linearization stated in Equation (4.12).

They further define the lower and upper bounds

$$\ell_{ik} := \min_{x \in \Pi_n} \sum_{j,\ell=1}^{n} c_{ijk\ell} x_{j\ell} \tag{4.16}$$

$$u_{ik} := \max_{x \in \Pi_n} \sum_{j,\ell=1}^{n} c_{ijk\ell} x_{j\ell} \tag{4.17}$$

and show that the following inequality holds for the terms in the objective function:

$$x_{ik} \left(\sum_{j\ell=1}^{n} c_{ijk\ell} x_{j\ell} \right) \geq \max \left\{ \ell_{ik} x_{ik}, u_{ik} x_{ik} - u_{ik} + \sum_{j,\ell=1}^{n} c_{ijk\ell} x_{j\ell} \right\} \tag{4.18}$$

The correctness of this inequality follows from the definition of the lower bound term ℓ_{ik} on the left side of the maximum. The right side containing the upper bound term u_{ik} follows directly from Kaufman and Broeckx [92]. The final Xia-Yuan formulation is depicted in Equation (4.19).

$$\begin{aligned} \text{minimize} \quad & \sum_{ik=1}^{n} w_{ik} \\ \text{subject to} \quad & w_{ik} \geq \ell_{ik} x_{ik} \\ & u_{ik} x_{ik} + \sum_{j\ell=1}^{n} c_{ijk\ell} x_{j\ell} - w_{ik} \leq u_{ik} \quad \forall i, k \in [n] \\ & (x_{ik}) \in \Pi_n \end{aligned} \tag{4.19}$$

The size of this linearization is similar to the Kaufman-Broeckx linearization; however, it is easy to see that the Xia-Yuan formulation dominates the one of Kaufman and Broeckx and that they coincide if all lower bound terms ℓ_{ik} evaluate to 0.

Zhang, Beltran-Royo and Ma The linearization described in this paragraph is another part of the Kaufman-Broeckx family of linearization developed by Zhang, Beltran-Royo and Ma [94]. It extends the ideas of Xia and Yuan. Using the same notation as the previous approaches, (4.20) describes the complete formulation.

$$\begin{aligned} \text{minimize} \quad & \sum_{ik=1}^{n} w_{ik} + (c_{iikk} + \ell_{ik})\, x_{ik} \\ \text{subject to} \quad & u_{ik} x_{ik} + \sum_{j\ell=1}^{n} c_{ijk\ell} x_{j\ell} - (\ell_{ik} + c_{iikk})\, x_{ik} - w_{ik} \leq u_{ik} \quad \forall i, k \in [n] \\ & w_{ik} \geq 0 \\ & (x_{ik}) \in \Pi_n \end{aligned} \tag{4.20}$$

All linearization techniques discussed so far can be considered light-weight relaxations for the QAP because they do not increase the asymptotical size of the integer programming model. As a consequence, these techniques are applicable to problems of practically relevant input size. Moreover, primal heuristics of state-of-the-art LP solvers work very well with this formulation. The lower bounds however obtained by relaxing the integrality constraints of this formulation are very weak such that even in a branch-and-bound

framework they often do not even surpass the trivial lower bound of

$$\sum_{i,j=1}^{n} \min\{c_{ijk\ell} : k, \ell \in [n]\} \tag{4.21}$$

in reasonable time. This issue makes it impractical to use these relaxations alone to close the gap between upper and lower bounds in a branch-and-bound process.

Reformulation Linearization Technique In the first place, Frieze and Yadegar [95] developed this linearization of the QAP where again every product $x_{ik}x_{jl}$ of binary variables is replaced by a new variable y_{ijkl}. They connected the new y-variables with the original assignment variables by summing over one of y's indices and requiring that a particular x-variable needs to result. The complete linear programming formulation follows now.

$$\begin{aligned}
\min \quad & \sum_{ijkl=1}^{n} c_{ijkl} y_{ijkl} & \\
\text{subject to} \quad & \sum_{i=1}^{n} y_{ijkl} = x_{jl} & \forall j,k,l \in [n] \\
& \sum_{k=1}^{n} y_{ijkl} = x_{jl} & \forall i,j,l \in [n] \\
& \sum_{j=1}^{n} y_{ijkl} = x_{ik} & \forall i,k,l \in [n] \\
& \sum_{l=1}^{n} y_{ijkl} = x_{ik} & \forall i,j,k \in [n] \\
& y_{iikk} = x_{ik} & \forall i,k \in [n] \\
& y_{ijkl} \in [0,1] & \forall i,j,k,l \in [n] \\
& (x_{ik}) \in \Pi_n &
\end{aligned} \tag{4.22}$$

This formulation yields very tight lower bounds, however its computation is very inefficient due to the large dimension of the variable space.

RLT hierarchy Sherali and Adams [96] discovered a hierarchy of reformulations and linearizations that reductively lead to the integer hull of a polyhedron. They recognized that in the special case of the QAP the first level of their hierarchy coincides with the linearization of Frieze and Yadegar. But since this hierarchy is far more general and applicable to any mixed integer program, we discuss it here, too. The following notation and explanations are taken from [97].

We will discuss the level d of the reformulation-linearization technique(RLT) now which we will also refer to as RLT-d. Suppose we are given a mixed integer program with n binary variables $x_1, \ldots, x_n$ and m continuous variables $y_1, \ldots, y_m$ as well as a set of constraints defining the feasible region X. As a first step, we choose a set of y-free constraints - i.e., the constraints have to contain binary variables only - such that the degree of those constraints' product is exactly d. In more detail, we choose constraints of the form $x_i \geq 0$, $1 - x_i \geq 0$ or $a_i^T x - b \geq 0$ and multiply them to a degree d polynomial. The complete RLT-d formulation contains all polynomials with any possible combination of y-free constraints. Note that any subset of polynomials already yields a lower bound to the original problem; however, generating the full set of polynomials will eventually

yield the integer hull of the polyhedron. In the next step, we multiply every of the given polynomials with all of the existing constraints which obviously yields new valid inequalities. The reformulation step is done, now the linearization takes place. First of all, we substitute any square occurrences of x_i by itself, i.e., replace $x_i^2 = x_i$. Moreover, for any subset $J \subseteq [n]$ of indices such that a product of binary variables occurs in one of the constraints, we introduce a new variable w_J or v_{Jk} depending on whether y_k also occurs or not. More formally, we introduce

$$w_J = \prod_{j \in J} x_j \tag{4.23}$$

$$v_{Jk} = y_k \prod_{j \in J} x_j \tag{4.24}$$

Hereby, we eliminated all non-linear terms and hence obtain a linear program again. We call this new set of constraints X_d. The authors show that if we require x to be binary and v, w, y to be continuous, then $X = X_d$

Furthermore, the authors show that the RLT-n formulation yields the integer hull of the polyhedron. They also show the existence of a hierarchy in the levels of the RLT formulations. More formally, if X_{P_d} is the projection of X_d to the original variable space (x, y), then

$$X \supseteq X_{P_1} \supseteq X_{P_2} \supseteq \cdots \supseteq X_{P_n} = conv.hull(X) \tag{4.25}$$

Due to complexity issues, it is very rarely practical to compute the RLT-n formulation because you get an exponential number of variables and constraints in the worst case. This is either caused by the construction itself or by the projection method. However, even the RLT-1 formulation already yields promising results in terms of the quality of lower bounds during a branch-and-bound process.

SDP relaxations Recently, several relaxations of the QAP as a semidefinite program (SDP) have been proposed. For example, SDP-relaxations for the non-convex constraint $Y = X \otimes X$ were introduced in [88, 89]. Instead of forcing Y to be of rank 1, these approaches require Y to be positive semidefinite. Recent approaches (e.g., [98]) have shown that these approaches can often efficiently produce good lower bounds for the QAP and beat common linear relaxations. Although, we describe an SDP approach to the QAP in Section 6, the underlying model in our approach is different and the related SDP approaches mentioned above are not used in this thesis any more.

CHAPTER 5

Dynamic Sparsification for Quadratic Assignment Problems

The work described in this section is a collaboration of Andreas Karrenbauer and myself published in the conference proceedings of MOTOR 2019 [2]. Parts of the following text are paraphrased or incorporated verbatim from this paper without further annotation. The algorithm described in this section was used to compute lower bounds for the keyboard optimization instances that occurred during the standardization process of the new French keyboard, which is described in the Chapter 3.

The goal of this framework is to efficiently provide optimality guarantees for large, but spare quadratic assignment problems. We discussed the hardness and popular approaches to this problem in Chapter 4. The RLT 1 approach showed to be especially powerful in the computation of lower bounds, but at the same time suffers from a huge increase in model size, making the linearization unusable for problems of realistic size. We aim at applying the RLT1 approach to QAPs while simultaneously controlling the growth of the problem size. Our algorithm dynamically generates the *most relevant* quadratic terms of the QAP, linearizes these new terms and computes an integer optimal solution. In contrast to a classic column-generation approach our algorithm guarantees a sequence of non-decreasing lower bounds in every step instead of non-increasing upper bounds. Additionally, every iteration produces a heuristic primal solution for the initial problem. This iterative framework produces a $(1+\varepsilon)$-approximation for the QAP for any $\varepsilon \geq 0$ without providing a polynomial runtime guarantee. Any polynomial time approximation scheme (PTAS) for the QAP would imply that $P = NP$ [78]. We evaluate our framework on real-world instances generated during the design process of the new French keyboard standard. The lower bounds computed by our algorithm showed very small optimality gaps within a few minutes for sparse QAPs with over 100 items and 130 locations. Considering the dynamic nature of this standardization process, an important main feature of our framework is the possibility to provide almost real-time feedback with very limited resources, e.g., a laptop.

5.1 Algorithm

The starting point of our algorithm is the integer programming formulation of the QAP. Linear relaxations for quadratic programs have been extensively studied over the last decades and are still a good starting point for many new ideas. However, the disadvantage of standalone linear relaxations is either high space complexity or an inefficient bound generation. A more detailed discussion of advantages and disadvantages of commonly used linearizations can be found in Section 4.4. We aim to use a linearization with promising lower bounds while overcoming the complexity issues for QAPs with sparse

quadratic objectives.

To this end, let $\mathcal{S} \subseteq [n]^4$ be a set of indices. We define the following subproblem of (4.8).

$$\begin{aligned}
\min \quad & \sum_{i,k=1}^{n} c_{ik} x_{ik} + \sum_{(i,j,k,\ell)\in\mathcal{S}} q_{ijk\ell} y_{ijk\ell} && && \text{(5.1a)}\\
\text{subject to} \quad & \sum_{i=1}^{n} x_{ik} = 1 && \forall k \in [n] &\\
& \sum_{k=1}^{n} x_{ik} = 1 && \forall i \in [n] &\\
& \sum_{j:(i,j,k,\ell)\in\mathcal{S}} y_{ijk\ell} \leq x_{ik} && \forall i,k,\ell \in [n] & \text{(5.1b)}\\
& \sum_{\ell:(i,j,k,\ell)\in\mathcal{S}} y_{ijk\ell} \leq x_{ik} && \forall i,j,k \in [n] & \text{(5.1c)}\\
& \sum_{i:(i,j,k,\ell)\in\mathcal{S}} y_{ijk\ell} \leq x_{j\ell} && \forall j,k,\ell \in [n] & \text{(5.1d)}\\
& \sum_{k:(i,j,k,\ell)\in\mathcal{S}} y_{ijk\ell} \leq x_{j\ell} && \forall i,j,\ell \in [n] & \text{(5.1e)}\\
& x_{ik} + x_{j\ell} \leq 1 + y_{ijk\ell} && \forall (i,j,k,\ell) \in \mathcal{S} & \text{(5.1f)}\\
& y_{ijk\ell} \in [0,1] && \forall (i,j,k,\ell) \in \mathcal{S} &\\
& x_{ik} \in \{0,1\} && \forall i,k \in [n] &
\end{aligned}$$

Note that it is feasible to add symmetry constraints for the y-variables of the form $y_{ijk\ell} = y_{ji\ell k}$ inspired by the Adams-Johnson formulation because they simulate the commutative multiplication of x_{ik} and $x_{j\ell}$, however, we could not observe any performance gain, and thus, omit them.

We first show that the proposed formulation is exact in the boundary case $\mathcal{S} = [n]^4$.

Lemma 5.1. *Let $\mathcal{S} = [n]^4$ and let $z^{(1)} = x^{(1)}$, $z^{(2)} = (x^{(2)}, y^{(2)})$ be optimal solutions of (4.8) and (5.1), respectively.*

Then $cost(z^{(1)}) = cost(z^{(2)})$.

Proof. Let $(i,j,k,\ell) \in [n]^4$ and consider the linear inequalities (5.1b)-(5.1f). If one of $x^{(2)}_{ik}$ and $x^{(2)}_{j\ell}$ is 0, then at least one of the inequalities (5.1b) to (5.1e) forces $y^{(2)}_{ijk\ell}$ to 0. On the other hand, if both $x^{(2)}_{ik} = x^{(2)}_{j\ell} = 1$, constraint (5.1f) sets $y^{(2)}_{ijk\ell}$ to 1. Therefore, and because $x^{(2)}$ is a binary variable, we can interpret $y^{(2)}_{ijk\ell}$ as the product $x^{(2)}_{ik} \cdot x^{(2)}_{j\ell}$.

Since $\mathcal{S} = [n]^4$, this observation holds for all variables and the formulations (4.8) and (5.1) coincide. □ □

Despite this result, we emphasize that using $\mathcal{S} = [n]^4$ leads to an intractable problem size for most practical input instances, e.g., more than 100 000 000 variables in our application with over 100 items. Even for sparse problems, reducing $\mathcal{S}$ to all the indices with nonzero contribution to the quadratic objective term may not suffice as, e.g., still about 2 000 000 variables remain in our case. To overcome this issue, we select an increasing

sequence of subsets $\mathcal{S}$, with each subset being significantly smaller than $[n]^4$. Lemma 5.2 explains why it is beneficial to do so.

Lemma 5.2. *Let $\mathcal{S} \subset [n]^4$ and z^* be the optimal solution of (5.1). Then cost(z^*) is a lower bound for (4.8).*

Proof. Let $(P, c, q), (P', c', q')$ be the polytopes and objective functions of (5.1) defined over $\mathcal{S}$ and $[n]^4$, respectively. Clearly, the set of constraints of P form a subset of the constraints of P'. Hence, every feasible solution in P' is also feasible in P, i.e., $P' \subseteq P$.

Setting $q_{ijk\ell} = 0$ for all $(i, j, k, \ell) \notin \mathcal{S}$, we can write (5.1a) as

$$\sum_{i,k=1}^{n} c_{ik} x_{ik} + \sum_{(i,j,k,\ell)\in[n]^4} q_{ijk\ell} y_{ijk\ell}.$$

Since all terms in the objective functions are assumed to be non-negative, it holds for every $(i, j, k, \ell) \in [n]^4$ that $q_{ijk\ell} \leq q'_{ijk\ell}$, which concludes the proof. □

Lemma 5.2 shows that dynamically increasing $\mathcal{S}$ yields a hierarchy of integer linear programs with increasing bounds for the original QAP. The proposed iterative algorithm later in this section is a natural consequence of Lemma 5.2. It remains to show how to initialize and update the set $\mathcal{S}$. We remark here that it is advisable to solve the integer linear programs close to optimality instead of considering their linear programming relaxations. Although dropping the integrality constraints drastically reduces the computation time with growing $\mathcal{S}$, the resulting lower bounds have shown to be significantly worse than the ones obtained by solving the integral versions for the same amount of time.

We now present two variants of the algorithm, which differ only in the procedure on how to grow $\mathcal{S}$.

5.1.1 Variant 1: conservative growth

First, we choose an arbitrary $\varepsilon \geq 0$. We will show later that the algorithm then produces a $(1 + \varepsilon)$-approximation of the optimal assignment. Note, however, that our algorithm allows to choose $\varepsilon = 0$, then computing an optimal solution. Assume that for a given index set $\mathcal{S}$, we are given an optimal binary solution (x^*, y^*) with objective value V. We build the candidate set

$$C := \Big\{(i, j, k, \ell) \notin \mathcal{S} : x^*_{ik} = x^*_{j\ell} = 1 \text{ and } q_{ijk\ell} > 0\Big\} \tag{5.2}$$

and sort C in an ascending order with respect to $q_{ijk\ell}$. Note that $|C| \leq n^2$. Formally, we define the function $\pi : [|C|] \to [n]^4$ such that for every $i < j \in \{1, \ldots, |C|\}$, it holds $q_{\pi(i)} \leq q_{\pi(j)}$. Let s be the index that satisfies the following equation.

$$s = \max\left\{t \in \{0, \ldots, |C|\} : \sum_{\alpha=1}^{t} q_{\pi(\alpha)} \leq \varepsilon \cdot V\right\} \tag{5.3}$$

Intuitively, we skip the s smallest positive cost values that sum up to a certain threshold (which is depending on ε) and add the rest of the indices to our active set $\mathcal{S}$.

The complete algorithm is presented in Algorithm 5.1. Theorem 5.3 shows that the update step eventually yields a $(1+\varepsilon)$-approximation.

Input : number of items/locations n, linear cost c, quadratic cost q, precision parameter ε
Result : Optimal assignment or upper/lower bound if aborted

1 $\mathcal{S} \leftarrow \emptyset$
do
2 $(x^*, y^*) \leftarrow$ opt. sol. of (5.1) with $\mathcal{S}$
$V \leftarrow$ evaluate x^* at (4.8)
C, π, s as in equations (5.2)-(5.3)
$\mathcal{S} \leftarrow \mathcal{S} \cup \{\pi(i)\}_{i=s+1}^{|C|}$
3 **while** $\mathcal{S}$ *changed in line 2*;
4 **return** x^*

Algorithm 5.1: The complete algorithm (conservative version)

Theorem 5.3. *Let $\varepsilon \geq 0$. If line 2 of Algorithm 5.1 does not add any index to $\mathcal{S}$, then the x-part of the current solution (x^*, y^*) is $(1+\varepsilon)$-optimal for problem* (4.8).

Before we prove this theorem, let us introduce some notation that increases the readability of the proof.

Definition 5.4. Let $M \subseteq [n]^4$, we define

$$\mathcal{C}_M(x) = \sum_{(i,j,k,\ell)\in M} q_{ijk\ell} x_{ik} x_{j\ell} \tag{5.4}$$

$$\mathcal{L}(x) = \sum_{i,k=1}^{n} c_{ik} x_{ik} \tag{5.5}$$

$$\mathcal{C}_M(x,y) = \sum_{i,k=1}^{n} c_{ik} x_{ik} + \sum_{(i,j,k,\ell)\in M} q_{ijk\ell} y_{ijk\ell} \tag{5.6}$$

We will write $\mathcal{C}(x)$ shortly for $\mathcal{C}_{[n]^4}(x)$.

Proof. Since $C \cap \mathcal{S} = \emptyset$ by definition, the only reason why $\mathcal{S}$ did not change is that $s = |C|$. In particular, this means that

$$\sum_{\alpha\in C} q_\alpha \leq \varepsilon \cdot (\mathcal{C}_{\mathcal{S}}(x^*, y^*))$$

We evaluate (x^*, y^*) on the complete objective function of the QAP and interpret $y^*_{ijk\ell} = x^*_{ik} x^*_{k\ell}$, which is a valid assumption already shown in the proof of Lemma 5.1. We hence

have to show that $\sum_{i,k=1}^{n} c_{ik}x^*_{ik} + \sum_{(i,j,k,\ell)\in[n]^4} q_{ijk\ell}x^*_{ik}x^*_{jk\ell} \leq (1+\varepsilon)OPT$.

$$\begin{aligned}
\sum_{i,k=1}^{n} c_{ik}x^*_{ik} + \sum_{(i,j,k,\ell)\in[n]^4} q_{ijk\ell}x^*_{ik}x^*_{jk\ell} &= \mathcal{L}(x^*) + \mathcal{C}(x^*) \\
&= \mathcal{L}(x^*) + \mathcal{C}_{\mathcal{S}}(x^*) + \mathcal{C}_{[n]^4\setminus\mathcal{S}}(x^*) \\
&\leq \mathcal{L}(x^*) + \mathcal{C}_{\mathcal{S}}(x^*) + \varepsilon \cdot \mathcal{C}_{\mathcal{S}}(x^*) \\
&\leq \mathcal{L}(x^*) + \mathcal{C}_{\mathcal{S}}(x^*) + \varepsilon \cdot (\mathcal{L}(x^*) + \mathcal{C}_{\mathcal{S}}(x^*)) \\
&= (1+\varepsilon)\,(\mathcal{L}(x^*) + \mathcal{C}_{\mathcal{S}}(x^*)) \\
&\leq (1+\varepsilon)\,OPT
\end{aligned}$$

The last inequality follows from Lemma 5.2. □

Corollary 5.5. *Consider any index set C' that contains C as defined in Equation (5.2), i.e. $C \subseteq C'$. If we replace C by C' in Algorithm 5.1, then Theorem 5.3 still holds.*

Proof. We use the notation of the proof of Theorem 5.3. Let $(i,j,k\ell) \in C' \setminus C$. This means that x^*_{ik} and $x^*_{j\ell}$ cannot both be 1, i.e.,

$$x^*_{ik} \cdot x^*_{j\ell} = 0. \tag{5.7}$$

Let $\mathcal{S}$ be the index set corresponding to C and $\mathcal{S}'$ the one corresponding to C'. Then $\mathcal{C}_{\mathcal{S}}(x^*) = \mathcal{C}_{\mathcal{S}'}(x^*)$ due to (5.7); hence, the proof of Theorem 5.3 applies. □

5.1.2 Variant 2: progressive growth

We change the definition of the candidate set C in Equation (5.2) to

$$C' := \left\{(i,j,k,\ell) \notin \mathcal{S} : x^*_{ik} = 1 \vee x^*_{j\ell} = 1 \text{ and } q_{ijk\ell} > 0\right\}. \tag{5.8}$$

This means we consider a tuple as a candidate if at least one of the corresponding x-variables were set to 1 in the previous optimal solution (instead of requiring both variables to be 1). The rest of the algorithm remains the same. In this second variant, $|C'| \leq n^3$, i.e., we potentially add more terms to the model. This can improve the evolution of lower bounds because we consider a more substantial portion of the model more quickly. As a trade-off, we potentially add more irrelevant terms than the conservative variant and, additionally, we could quickly arrive at a model of a size that exceeds the resources of the computer used to run the algorithm. Note that Theorem 5.3 also holds for this variant of the algorithm due to Corollary 5.5.

5.1.3 Variant 3: Hybrid strategy

To achieve a balance between the fast evolution of lower bounds in variant 2 and the moderate growth of model size in variant 1, we propose the hybrid strategy that kick-starts with the progressive variant 2 and switches to the conservative variant 1 before the model size grows too large. The evaluation shows that this strategy is indeed superior to both standalone variants. For all instances, there is a critical point where the amount of generated quadratic terms would grow so large that an MIP solver cannot compute the

integer optimal solution within a reasonable time. Therefore, we switch to the conservative variant at this critical point, which grows the model more slowly while still steadily improving the lower bound.

5.2 Evaluation

We applied our algorithm within several stages of the French keyboard standardization process. The instances consist of over 100 special characters and 130 keys (in order to achieve the classic QAP formulation, one can generate dummy characters symbolizing that a key is left empty), and the objective function consists of a conic combination of a sparse quadratic and dense linear cost terms. The quadratic term can be factorized into a sparse matrix F, which describes the association score (similarity) of two different special characters, and a dense matrix D, describing the distances between two key slots. The weight of the quadratic part ranges between 30% and 50%. Additionally, some instances fix few characters like punctuation symbols to fixed slots or require that the capital versions of special characters are placed on the shifted slot of the same letter (e.g., È is placed on the shifted slot of è) whereas other instances also allow them to be on the Alt-Shift or Alt version of this slot.

We evaluated the instances on a single Intel(R) Xeon(R) CPU E5-2680 v3 @ 2.50GHz processor core with 16GB of RAM. We compare our algorithm to the formulation of Xia and Yuan [93] as a state-of-the-art lightweight linearization for the QAP. As already mentioned before, stronger formulations like RLT1 could not compute any lower bound within the given resources, often because the model size already exceeded the available RAM. We use Gurobi version 8.1 [99] as the underlying solver for both approaches.

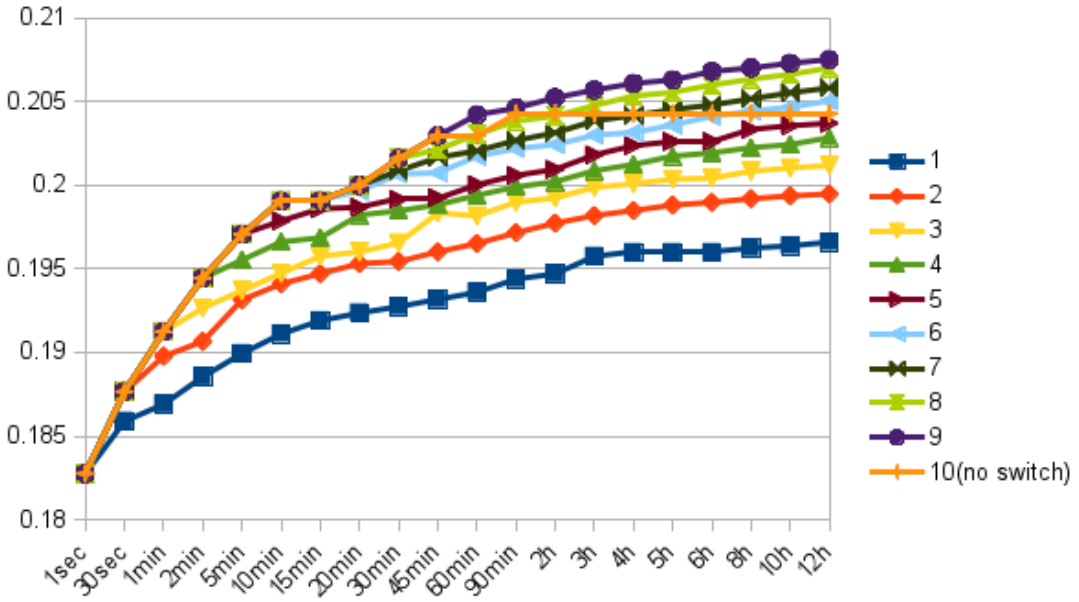

Figure 5.1: The evolution of the lower bound when switching variants for instance N50s. The numbers in the legend describe the iteration at which the switch was triggered.

We first discuss the impact of the variant choice on one example instance. More specifically, we test the hybrid strategy and the effect of the switch from progressive to conservative at different iterations τ. Figure 5.1 shows the evolution of lower bounds for $\tau = 1, \ldots, 10$. Since naturally the bound evolves faster during the first seconds and minutes, the graph shows a more detailed view on the evolution within this first period. Setting $\tau = 1$ leads to using the conservative variant from the beginning while setting $\tau = 10$ implies that the strategy switch does not occur within the given time window of 12 hours because the model of the last iteration is already too large to be solved efficiently. We observe that as long as the model size is moderately low, the progressive variant achieves better results at every time stamp. However, after roughly 45 minutes, the model size for this variant already grows notably large so that the next iterations takes quite a long time. After yet another size increase, the ILP solver could not compute an optimal solution within the remaining 10 hours. Note that depending on the available resources (time limit and hardware) as well as the particular instance (dimension and sparsity), the critical point at which a switch from the progressive to the conservative variant is valuable varies. Since all our instances are of similar size and sparsity, the critical point for this evaluation is at the 9th iteration.

Figure 5.2 compares the evolution of the lower bounds of our algorithm using only variant 2, switching after 9 iterations, and the formulation of Xia and Yuan within a total time period of 12 hours for the same example instance. It is important to note that the setup time for the Xia-Yuan formulation is around 25 minutes for every instance because over 10 000 linear assignment problems are solved beforehand. Therefore, the first bound for the original QAP is only produced after 25 minutes.

To avoid visual clutter in the following figures, we only depict the results of the hybrid algorithm switching at the 9th iteration. The lower and upper bounds for all QAP instances are shown in Figure 5.3. We ran our hybrid algorithm for one hour and compare it against the formulation of Xia and Yuan after one and 12 hours of computation time.

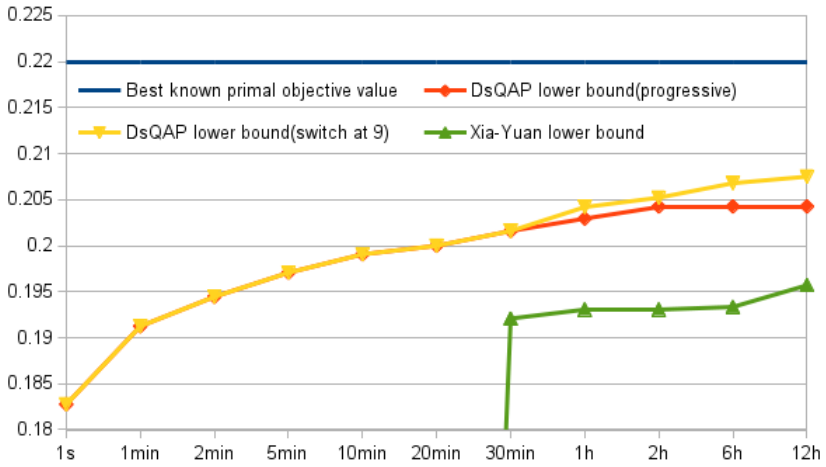

Figure 5.2: The evolution of the lower bounds within 12 hours of computation time for instance N50s.

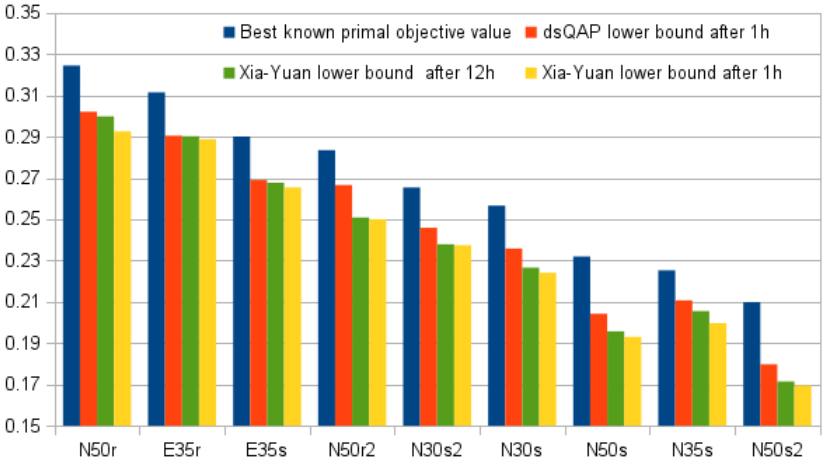

Figure 5.3: Lower and upper bounds for the QAP instances

The naming of the test instances is as follows: the first letter describes the set of additional constraints (N for no additional constraints, and E for fixed punctuation symbols and the fixed symbols è, é, ê, à, and €). The number in the middle describes the weight (in percent) of the quadratic term in the objective function, and the following letter describes if the capitalized letter of a special character has to be placed on the shifted slot (s) or on any alternative of this slot (r). Note that almost every instance uses a slightly different set of characters because this set constantly changed in committee meetings. The full description of all the different character sets and further details about the data gathering is beyond the scope of this thesis and can be found in [23]. Therefore, it occurs that two instances are equally named although they slightly differ in the character set used. In this case, one of the instance names ends with 2 for better differentiation.

We can see that within one hour, we outperform the formulation of Xia and Yuan for every instance independent of its time limit being one hour or 12 hours. Although we slightly improved the lower bounds of all instances, this is not the true benefit of our framework. What we really want to emphasize here is how fast we achieve high-quality lower bounds, which is especially important in the practical application of our algorithm. In this highly interactive environment with countless model updates and changes, receiving valuable feedback of an optimization method after only several minutes can greatly improve the dynamics of an expert committee that discusses different proposals and has to decide the next steps towards a final keyboard standard.

We measure the time our algorithm needs to exceed the lower bounds that the Xia-Yuan formulation produces after 1 hour and 12 hours, respectively. Figure 5.4 shows that we achieve this goal within several minutes for all instances. In the worst case, it takes 20 minutes to exceed the 12 hours bound of Xia-Yuan. Hence, for every instance, we achieve superior lower bounds within the setup time of 25 minutes that is needed for the creation of the Xia-Yuan linearization.

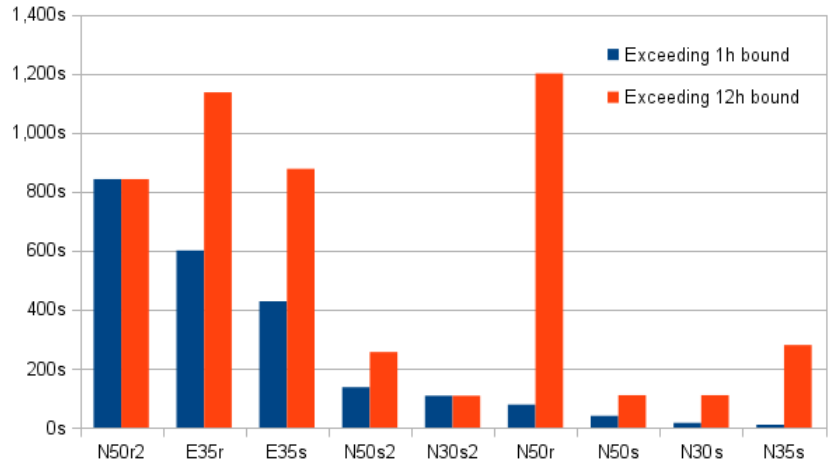

Figure 5.4: The time we need to exceed the bounds of Xia and Yuan after $1h$ and $12h$ (in seconds)

5.3 Robustness Analysis

To analyze the robustness of our approach, we vary the nonzero values of the quadratic cost matrix with additive noise generated by a normal distribution with 0 mean and standard deviation σ.

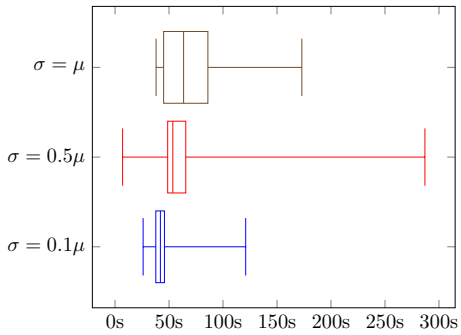

Figure 5.5: Boxplots of the time (in seconds) until our approach exceeded the 12 hours Xia-Yuan bound

Recall that the quadratic matrix Q is the Kronecker product of the dense matrix D containing the distances between the keys and the sparse matrix F encoding the similarity between the special characters. We only add noise to the entries in F while keeping its entries non-negative. More specifically, consider $f_{ij} > 0$ and $\delta_{ij} \sim N(0, \sigma)$, then we set $f'_{ij} = f_{ij} + \delta_{ij}$ if $f'_{ij} > 0$, otherwise we recompute δ_{ij}. Let μ be the average value of all nonzero entries in the association matrix A, then we set σ to 10%, 50%, and 100% of μ. For this evaluation, we use the instance N35s as a base instance and generate 20 randomly variated instances for each of the three variance values. Note that the evaluation of these 60 instances requires at least 720 hours of total computing time in order to compute the 12 hour bound of Xia-Yuan for every instance. Moreover, the

structure of all instances is considerably similar; Figure 5.3 also backs up this claim. This leads to the conclusion that every of the 9 different keyboard instances is a good representative for this robustness analysis.

Figure 5.5 shows the boxplots of the time (in seconds) our approach needed to exceed the bound that the Xia-Yuan formulation achieves after 12 hours. In every of the 60 instances in total, we exceeded said bound after at most five minutes. Note that the Xia-Yuan formulation has a setup time for around 25 minutes for instances of this size. This means we can consistently produce high quality bounds during the setup time of the competing approach.

Moreover, Figure 5.6 shows the lower and upper bounds for the 20 runs each with $\sigma \in \{0.1\mu, 0.5\mu, \mu\}$, respectively. We observe that the results of these randomized instances are very consistent with the results of the original evaluation, independent of the variance.

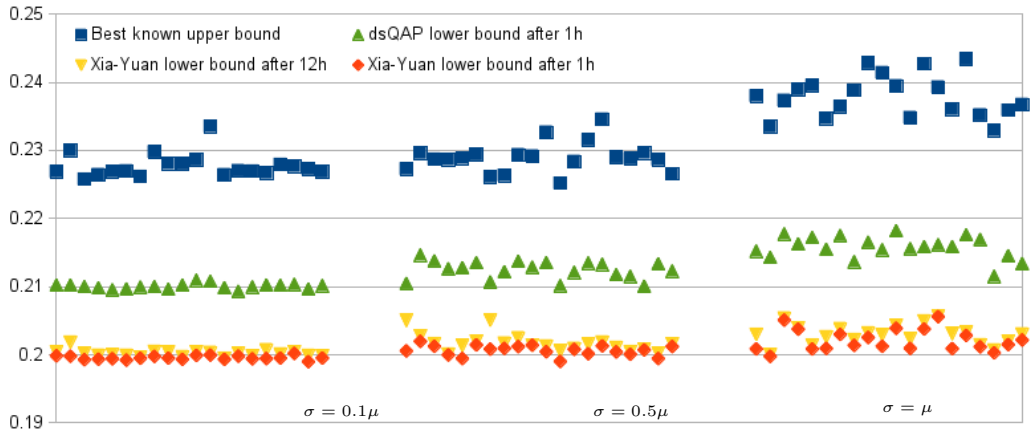

Figure 5.6: Bounds for 60 randomly variated instances (20 each) with variance σ

5.4 Conclusion

We presented a lightweight framework for sparse quadratic assignment problems that combines powerful linearization techniques and ideas from column-generation. It is lightweight in a sense that it can generate good bounds for sparse QAPs of huge size (over 100 items) on a normal laptop. Our algorithm was used in the process of defining the new French keyboard standard. The evaluation, which is based on real data gathered during this standardization process, showed that we can compete with state-of-the-art linearization techniques. We showed that we can produce high quality lower bounds within several minutes, which serves the purpose of almost real-time feedback in such a dynamic interactive optimization process.

CHAPTER 6

Cut Pseudo Bases for Quadratic Assignment Problems

Part of the work described in this section is published at ISCO 2016 [100], which subsequently became my master's thesis. Since the paper was published, the work on this framework has still been extended. Although this paper is hence not a contribution to this dissertation, this section cites and summarizes parts of it for the sake of completeness and self-containment.

In this chapter, we describe a semidefinite program derived from a lower-bound-preserving transformation of a QAP instance to an auxiliary quadratic minimization problem with only $\mathcal{O}(n \log n)$ variables. SDP relaxations with so few variables can be solved efficiently with modern interior point methods for conic optimization problems. Moreover, we also show how to integrate our relaxation in a branch-and-bound framework. While branching on single assignment variables typically results in very unbalanced branch-and-bound trees, our approach avoids this by design. To this end, we introduce the concept of cut pseudo bases, which had not been used — to the best of our knowledge — in this context before. Our goal was to develop an approach that still works with limited computational resources, e.g., on a laptop, for the cases when the lower bounds provided by lightweight linearization techniques are too weak and when it is already infeasible to solve the LP relaxation of more powerful linearizations like RLT1. Furthermore, we present experimental results for instances with $n \geq 25$ in which we outperform both lower bounds mentioned above in terms of efficiency and effectiveness. The bounds produced by our SDP always exceed — just by construction — the trivial lower bound, which is discussed in Chapter 4(see Equation (4.21).

6.1 An SDP-Based Lower Bound

We assume for the sake of presentation that the number of items and locations n is a power of 2. This is not a restriction because we can pad n with dummy items and locations. Moreover, the dummy items can be projected out easily in an implementation so that this also does not harm its performance.

The derivation of this model was done in two steps. First, we designed a new quadratic program that uses fewer binary variables and that we proved to be a lower bound for the original QAP. Moreover, we showed that this new formulations allows for a balanced branching tree. In the second step, we relaxed the new formulation to an SDP.

Concerning the goal of achieving balanced branching trees, let us revisit the well-known problem of branching on single assignment variables. Setting the binary variable x_{ik} to 1 means fixing item i to location j, which is a very strong decision that affects all other variables in the i-th row or k-th column, forcing them to 0. On the other

hand, if we set x_{ik} to 0, we just decide not to fix i to j. However, there are still $n-1$ other possible locations for i, so we basically did not decide much. This yields highly imbalanced branching trees as it is much more likely to prune in the 1-branches of a branch-and-bound process. This undesirable effect can be avoided by the well-known idea of generalized upper-bound branching (see, e.g., Section 7 of [101]). Inspired by this, we considered a similar approach illustrated in the following IP formulation with n auxiliary z-variables:

$$
\begin{aligned}
\text{minimize} \quad & \sum_{i,j,k,\ell=1}^{n} c_{ijk\ell} x_{ik} x_{jl} & & \\
\text{s.t.} \quad & \sum_{i=1}^{n} x_{ik} = 1 & & \forall k \in [n] \\
& \sum_{k=1}^{n/2} x_{ik} = z_i & \sum_{k=n/2+1}^{n} x_{ik} = 1 - z_i \qquad & \forall i \in [n] \\
& x_{ik} \in \{0,1\} & & \forall i,k \in [n] \\
& z_i \in \{0,1\} & & \forall i \in [n].
\end{aligned}
$$

If we branch on the z-variables instead of the assignment variables, our branching tree is much more likely to be balanced because either choice is equally strong. However, it is not sufficient to branch only on these z-variables because too many degrees of freedom still remain open even after all z-variables are set. If we want to completely determine the x-variables and thus be able to project them out, we should introduce further binary z-variables. To this end, we defined *cut pseudo bases*.

6.2 Introduction of Cut Pseudo Bases

The key idea of cut pseudo bases is the usage of cuts in the complete graph with the locations as nodes. Consider a balanced subset of the nodes, i.e., one of size $\frac{n}{2}$. Instead of assigning an item to a certain location, we now assign it to one of the "halves" of the location space. We repeat this cutting of the location space until we reach a state where — after a finite number of assignments — every item can be uniquely mapped to a single location. Moreover, we cut the space in a balanced way, i.e., we require that each side of the cut is equally large. Let us formalize these requirements.

Definition 6.1. A set of cuts over the location space such that

- all cuts are balanced,
- all singleton locations can be expressed by a linear combination of cuts, and
- it is inclusion-wise minimal

is called a *cut pseudo base*.

Clearly, the size of a cut pseudo base is $\log_2 n$ when n is a power of 2 and thus $\lceil \log_2 n \rceil$ in general by the padding argument. To illustrate the concept of a pseudo base, consider the following example.

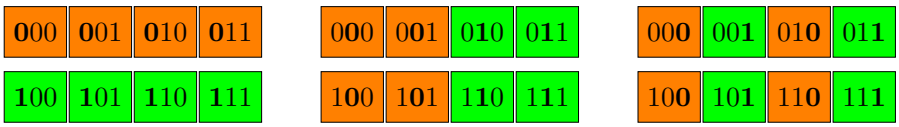

Figure 6.1: Example: the binary decomposition cut pseudo base with $8 = 2^3$ locations

Example 6.2. Enumerate the $n = 2^k$ locations by $0, \dots, n-1$, and consider the binary decomposition of these numbers. For every bit $b = 0, \dots, k-1$, we define a cut that separates all locations with numbers differing in the b-th bit. Then, this collection of cuts forms a cut pseudo base. Figure 6.1 depicts such a cut pseudo base for $n = 8$. There are three cuts; the orange tiles represent the 0-side and the green tiles the 1-side of the cuts.

Note that any arbitrary cut pseudo base can be transformed to the binary decomposition pseudo base by a permutation of the locations. Hence, we will employ this cut pseudo base as a reference throughout this chapter for the sake of presentation and simplicity.

6.2.1 Exchanging Aassignment Variables by Cut Variables

The cut pseudo bases introduced in the previous subsection are balanced by definition, meaning that assigning an item to one side of a cut in the pseudo base is just as effective as assigning it to the other side. However, the decision of whether a particular item should be assigned to a fixed location is highly unbalanced, as we have already discussed. Hence, we aim to remove the classical assignment variables and introduce the cut variables instead. This also benefits the number of binary variables, which decreases from n^2 assignment variables to $n \log_2 n$ cut variables. Let $(S_b)_{b \in [B]}$ be an arbitrary but fixed cut pseudo base. We introduce the variables $z_i^b \in \{0, 1\}$ for every cut S_b, indicating whether i is assigned to the 0- or 1-side of the cut S_b. We relate them to the assignment variables in the following manner.

$$x_{ij} = 1 \Leftrightarrow \forall b \in [B] \;\; j \in z_i^b\text{-side of cut } S_b \tag{6.1}$$

We consider an arbitrary cut b in the following and omit the superscript b to simplify the notation and thereby improve readability. Observe that

$$z_i z_j + z_i(1 - z_j) + (1 - z_i)z_j + (1 - z_i)(1 - z_j) = 1 \tag{6.2}$$

holds for any $z_i, z_j \in \mathbb{R}$ and that for a binary solution exactly one of the four terms is 1, and the others vanish.

Thus, we obtain for any assignment x and the corresponding binary z-variables that $\sum_{i,j,k,\ell=1}^{n} c_{ijk\ell} x_{ik} x_{j\ell}$

$$
\begin{aligned}
&= \sum_{i,j=1}^{n} (z_i z_j + z_i(1-z_j) + (1-z_i)z_j + (1-z_i)(1-z_j)) \sum_{k,\ell=1}^{n} c_{ijk\ell} x_{ik} x_{j\ell} \\
&\geq \sum_{i,j=1}^{n} z_i z_j \cdot \min\{\sum_{k,\ell=1}^{n} c_{ijk\ell} x_{ik} x_{j\ell} : x \in \Pi_{ij}^{(11)}\} \\
&+ \sum_{i,j=1}^{n} z_i(1-z_j) \cdot \min\{\sum_{k,\ell=1}^{n} c_{ijk\ell} x_{ik} x_{j\ell} : x \in \Pi_{ij}^{(10)}\} \\
&+ \sum_{i,j=1}^{n} (1-z_i)z_j \cdot \min\{\sum_{k,\ell=1}^{n} c_{ijk\ell} x_{ik} x_{j\ell} : x \in \Pi_{ij}^{(01)}\} \\
&+ \sum_{i,j=1}^{n} (1-z_i)(1-z_j) \cdot \min\{\sum_{k,\ell=1}^{n} c_{ijk\ell} x_{ik} x_{j\ell} : x \in \Pi_{ij}^{(00)}\}
\end{aligned}
$$

where $\Pi_{ij}^{(\alpha\beta)}$ for $\alpha, \beta \in \{0,1\}$ denotes the set of all assignments in which i is assigned to the α-side and j to the β-side of the cut. In the following, we argue that this is indeed a valid lower bound. To this end, let $c_{ij}^{(\alpha\beta)} := \min\{\sum_{k,\ell} c_{ijk\ell} x_{ik} x_{j\ell} : x \in \Pi_{ij}^{(\alpha\beta)}\}$ denote the optimum objective values of the corresponding optimization problems, and observe that $c_{ij}^{(\alpha\beta)}$ only contributes to the right-hand side if $z_i = \alpha$ and $z_j = \beta$. This yields an objective function that is free of x-variables. Furthermore, the minimum of the original objective taken over all $x \in \Pi$ is bounded from below by the minimum over all z that determine an assignment. At first glance, it seems that we have to solve $4n^2$ QAPs to compute the coefficients for the new objective function. However, a close inspection of the subproblems reveals that $c_{ij}^{(\alpha\beta)}$ is determined by the minimum $c_{ijk\ell}$ over all k, ℓ such that the b-th bits of k and ℓ are α and β, respectively. This can be computed efficiently for each pair ij by a single scan over all $c_{ijk\ell}$. Note that in the Koopmans-Beckmann variant of a QAP, we have $c_{ijk\ell} = f_{ij} \cdot d_{k\ell}$, and thus, it suffices to scan over the distance pairs $d_{k\ell}$ of the locations k and ℓ. Furthermore, such a single scan can also take additional constraints into account, e.g., excluded pairs due to a branching process. Hence, the lower bound of our approach is self-tightening in a branch-and-bound process. In every branching step, we can update our cost estimation for this particular setting of excluded pairs.

6.3 Towards an SDP

Before obtaining the desired SDP relaxation, we applied the typical transformation to map $\{0,1\}$-variables to $\{-1,1\}$-variables. That is, we used the linear transformation $z_i = \frac{1+y_i}{2}$. This implies that $1 - z_i = \frac{1-y_i}{2}$. Plugging this into

$$
c_{ij}^{(11)} z_i z_j + c_{ij}^{(10)} z_i(1-z_j) + c_{ij}^{(01)} (1-z_i) z_j + c_{ij}^{(00)} (1-z_i)(1-z_j)
$$

yields

$$
\begin{aligned}
\sum_{\alpha,\beta=0}^{1} c_{ij}^{(\alpha\beta)} \cdot \tfrac{1-(-1)^{\alpha} y_i}{2} \cdot \tfrac{1-(-1)^{\beta} y_j}{2} &= \sum_{\alpha,\beta=0}^{1} c_{ij}^{(\alpha\beta)} \cdot \tfrac{1-(-1)^{\alpha} y_i - (-1)^{\beta} y_j + (-1)^{\alpha+\beta} y_i y_j}{4} \\
&= \tfrac{c_{ij}^{(11)} + c_{ij}^{(10)} + c_{ij}^{(01)} + c_{ij}^{(00)}}{4} + \tfrac{c_{ij}^{(11)} + c_{ij}^{(10)} - c_{ij}^{(01)} - c_{ij}^{(00)}}{4} \cdot y_i \\
&+ \tfrac{c_{ij}^{(11)} - c_{ij}^{(10)} + c_{ij}^{(01)} - c_{ij}^{(00)}}{4} \cdot y_j + \tfrac{c_{ij}^{(11)} - c_{ij}^{(10)} - c_{ij}^{(01)} + c_{ij}^{(00)}}{4} \cdot y_i y_j.
\end{aligned}
$$

We separate and symmetrize the constant, linear, and quadratic terms such that we can write the total sum over all i, j in matrix-vector notation as

$$y^T C y + c^T y + \gamma$$

with

$$\begin{aligned} C_{ij} &:= \frac{c_{ij}^{(11)} + c_{ji}^{(11)} - c_{ij}^{(10)} - c_{ji}^{(10)} - c_{ij}^{(01)} - c_{ji}^{(01)} + c_{ij}^{(00)} + c_{ji}^{(00)}}{8} \\ c_i &:= \frac{1}{4} \sum_{j=1}^{n} c_{ij}^{(11)} + c_{ji}^{(11)} + c_{ij}^{(10)} - c_{ji}^{(10)} - c_{ij}^{(01)} + c_{ji}^{(01)} - c_{ij}^{(00)} - c_{ji}^{(00)} \\ \gamma &:= \frac{1}{4} \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij}^{(11)} + c_{ij}^{(10)} + c_{ij}^{(01)} + c_{ij}^{(00)}. \end{aligned}$$

To relax the quadratic part in the objective using a semidefinite matrix, we use a standard fact about the trace, i.e., $y^T C y = \mathrm{tr}(y^T C y) = \mathrm{tr}(C y y^T)$. Thus, we replace the quadratic term $y^T C y$ in the objective function by the Frobenius product[1] $C \bullet Y$ and require $Y = yy^T$. However, this rank-1-constraint is not convex, and we relax it to $Y \succcurlyeq yy^T$, which means that $Y - yy^T$ is positive semi-definite. Using the notion of the Schur complement, this condition is equivalent to

$$\begin{pmatrix} 1 & y^T \\ y & Y \end{pmatrix} \succcurlyeq 0.$$

To accomplish this, we could augment the matrix Y by a 0-th row and column. Alternatively, we could also use one of the dummy items that we initially introduced to increase the number of items to a power of 2. These dummy items are already fixed w.r.t. the side of the cut b under consideration. Independently if we use an artificial $0th$ item or one of the dummy items, the idea is that if $y_i = 1$ is already fix for some item i, we may require $Y_{ij} = y_j$ for all j and $Y \succcurlyeq 0$. This constraint can be written as $e_i e_j^T \bullet Y - e_j^T y = 0$ for $y_i = 1$. For $y_i = -1$, we obtain the constraints $e_i e_j^T \bullet Y + e_j^T y = 0$ for all items j instead.

In the following, we list further constraints that we added to the SDP that tighten our formulation and hence improve the strength of the lower bound on the QAP. Recall that for the sake of readability we omitted any superscripts to identify the cut under consideration. In the following however, we will argue with the complete cut pseudo base, so we use Y^b for the matrix corresponding to cut b and y^b to identify the linear terms corresponding to this cut. Similarly, we shall use C^b, c^b, and γ^b to denote the corresponding parts in the objective function. We emphasize again that the cut pseudo base in use is fixed from now on and contains $B = \lceil \log 2(n) \rceil$ many cuts.

6.3.1 Domain of Y

We make sure that every $y_i^b \in \{-1, 1\}$. For the linear variables, we relax this constraint to $y_i^b \in [-1, 1]$, but in the SDP, we can require something stronger. By using the fact that

[1] The Frobenius product $A \bullet B := \mathrm{tr}(A^T B) = \sum_{i,j} a_{ij} b_{ij}$ is the standard inner product on the space of $n \times n$ matrices used in semi-definite programming.

$\left(y_i^b\right)^2 = 1$, we can add the constraint $Y_{ii}^b = 1$ for all $b \in [B], i \in [n]$. Formally, we do this by the SDP constraint $E_i \bullet Y^b = 1$ where E_i has a 1 on index (i,i) and 0s everywhere else.

6.3.2 Injectivity of the Assignment

We ensure that the assignment is injective, i.e., that no two different keys are assigned to the same spot. In terms of y variables, we require for all distinct i and j that y_i^b be different from y_j^b for at least one b. We have $y_i^b = y_j^b$ if and only if the corresponding entry in Y, namely Y_{ij}^b, is 1. Hence, we add the constraint

$$\sum_{b=1}^{B} Y_{ij}^b \leq B - 1 \;\Leftrightarrow\; \left(\sum_{b=1}^{B} \frac{1}{2} Y_{ij}^b + \frac{1}{2} Y_{ji}^b\right) \leq B - 1.$$

Note that in an integer optimal solution, the constraints above already ensure all the properties, we want to have. However, we have found that it is beneficial for the relaxed SDP to add the following constraint.

6.3.3 Zero Row Sums

In the original formulation, injectivity implies that the number of keys assigned to one side of a cut is as large as the number of keys assigned to the opposite side. Recall that we are assuming $n = 2^k$, and we have a cut pseudo base. Hence, the implication above indeed holds. In terms of y variables, this can be modeled as the constraint

$$\sum_{j=1}^{n} y_j^b = 0$$

or as

$$\sum_{j=1}^{n} Y_{ij}^b = \sum_{j=1}^{n} y_i^b y_j^b = y_b^i \cdot \sum_{j=1}^{n} y_j^b \stackrel{!}{=} 0$$

in the SDP for an arbitrary fixed $i \in [n]$. Hence, taking the row sum of Y^b in this case yields the term we are looking for.

6.3.4 Total Entry Sum

We have observed that we can condense the zero-sum-constraints to a single one by exploiting the positive semidefiniteness of Y.

Lemma 6.3. *Let $Y \in \mathbb{R}^{n \times n}$ be positive semidefinite. If $\mathbb{1}\mathbb{1}^T \bullet Y = 0$, then for any $i \in [n]$, it holds that $\sum_{j=1}^{n} Y_{ij}^b = 0$.*

Proof. Observe that

$$0 = \mathbb{1}\mathbb{1}^T \bullet Y = \mathbb{1}^T Y \mathbb{1}.$$

Since Y is positive semidefinite, $\mathbb{1}$ is an eigenvector of Y with eigenvalue 0, which implies that $Y\mathbb{1} = 0\mathbb{1} = \vec{0}$ and proves the claim. □

Hence, instead of imposing n constraints for every single row of Y, we have shown that one constraint is enough to fix all row sums to 0.

This concludes the initial SDP formulation we proposed in [100]. The following additional features (the only exception being the discussion of the objective function in Section 6.3.7) have been examined after the publication of this paper.

6.3.5 Optional constraints

We found several further families of constraints that might be useful to strengthen the lower bound of the SDP. However, you will see that the number of those constraints is too high to include them all in our problem. Therefore, we need to apply techniques like dynamic constraint activation or the spectral bundle method [102].

Fixation Constraints As soon as we encounter a situation during branching that forbids a particular placement, we can add further constraints to enforce the desired behavior. Given a fixed pair of items i, j and a cut b, consider a situation in which it is not allowed to place i to the α-side of cut b and, at the same time, j to the β side of the same cut (because former decisions forbid this behavior). In this case, we add the following constraints

$$(-1)^\alpha + y_i^b + (-1)^\beta y_j^b = 1 + (-1)^{\alpha+\beta} Y_{ij}^b \tag{6.3}$$

Depending on the actual values of α and β, the constraint takes one of the following forms.

$$\begin{aligned}
\alpha = 0, \beta = 0: &\quad y_i^b + y_j^b - Y_{ij}^b &= 1 \\
\alpha = 0, \beta = 1: &\quad y_i^b - y_j^b + Y_{ij}^b &= 1 \\
\alpha = 1, \beta = 0: &\quad -y_i^b + y_j^b + Y_{ij}^b &= 1 \\
\alpha = 1, \beta = 1: &\quad -y_i^b - y_j^b - Y_{ij}^b &= 1
\end{aligned} \tag{6.4}$$

Triangle Inequalities Let $i, j, k \in [n]$ be items and let us fix a cut b. The following inequalities are cutting planes.

$$Y_{ij}^b + Y_{ik}^b + Y_{jk}^b \leq -1 \tag{6.5}$$

$$Y_{ij}^b - Y_{ik}^b - Y_{jk}^b \leq -1 \tag{6.6}$$

As you can see, some permutations of the indices in (6.6) yield other constraints whereas the constraint (6.5) is permutation-invariant. Enumerating all subsets of the form $\{i, j, k\} \subseteq [n]$ and creating one constraint (6.5) and 3 distinct constraints (6.6) yields an amount of $4\binom{n}{3}$ constraints. Note that there are permutations of i, j, k, which do not create new constraints due to the symmetry of the Y matrix (for example, the orderings i, j, k and j, i, k lead to the same constraint).

6.3.6 Branching in the SDP

We want to use the SDP relaxation as a generator of lower bounds for the QAP formulation that uses the cut variables. Branching in this QAP means setting one of the y_i^b to either 1 or -1. How can we transfer this to the SDP where we only have products of variables in the variable matrix Y? Let us first restrict to the case where we only consider one cut b. We can partition the variables into the three sets L_b, R_b and F_b where $L_b = \{i|y_i^b = 1\}$, $R_b = \{i|y_i^b = -1\}$ and F_b contains the rest of the variables (the *free* variables). Using this partition, we can fix entries in Y^b in order to ensure the correct values for the fixed variables in the original problem.

$$\forall (i,j) \in L_b \times L_b: \quad Y_{i,j}^b = \quad 1 \tag{6.7}$$

$$\forall (i,j) \in R_b \times R_b: \quad Y_{i,j}^b = \quad 1 \tag{6.8}$$

$$\forall (i,j) \in L_b \times R_b: \quad Y_{i,j}^b = \quad -1 \tag{6.9}$$

At a first glance, this only fixes the variables of L_b and R_b to different values, but in general not to 1 for all those in L_b and -1 for those in R_b. We also allow the assignment to be the other way around. However, the following Lemma shows that this is irrelevant for our model.

Lemma 6.4. *Let $\hat{y}$ be an $\{-1,1\}$-assignment to the objective function*

$$\sum_{b=1}^{B} \sum_{i,j=1}^{n} M_{ij}^b \left(1 - y_i^b y_j^b\right)$$

Let $B^* \subseteq [B]$ be arbitrary and define $\tilde{y}^b = \hat{y}^b$ for $b \notin B^*$ and $\tilde{y}^b = -\hat{y}^b$ for $b \in B^*$. Then

$$\sum_{b=1}^{B} \sum_{i,j=1}^{n} M_{ij}^b \left(1 - \hat{y}_i^b \hat{y}_j^b\right) = \sum_{b=1}^{B} \sum_{i,j=1}^{n} M_{ij}^b \left(1 - \tilde{y}_i^b \tilde{y}_j^b\right)$$

Proof.

$$\sum_{b=1}^{B} \sum_{i,j=1}^{n} M_{ij}^b \left(1 - \hat{y}_i^b \hat{y}_j^b\right) \tag{6.10}$$

$$= \sum_{b \in [B] \setminus B^*} \sum_{i,j=1}^{n} M_{ij}^b \left(1 - \hat{y}_i^b \hat{y}_j^b\right) + \sum_{b \in B^*} \sum_{i,j=1}^{n} M_{ij}^b \left(1 - \hat{y}_i^b \hat{y}_j^b\right) \tag{6.11}$$

$$= \sum_{b \in [B] \setminus B^*} \sum_{i,j=1}^{n} M_{ij}^b \left(1 - \tilde{y}_i^b \tilde{y}_j^b\right) + \sum_{b \in B^*} \sum_{i,j=1}^{n} M_{ij}^b \underbrace{\left(1 - \left(-\tilde{y}_i^b\right)\left(-\tilde{y}_j^b\right)\right)}_{=1-\tilde{y}_i^b \tilde{y}_j^b} \tag{6.12}$$

$$= \sum_{b=1}^{B} \sum_{i,j=1}^{n} M_{ij}^b \left(1 - \tilde{y}_i^b \tilde{y}_j^b\right) \tag{6.13}$$

□

This shows that we can – independently of b – assign 1 or -1 to L_b and the corresponding opposite value to R_b, we just have to maintain the inner consistency of every set which is exactly what is guaranteed by the constraints stated above.

The bottleneck of these branching constraints appears to be that we need to add many constraints even for a single branch if we are deep in the branching tree. Whenever we enter or leave a branch, it seems we have to touch $\mathcal{O}(n)$ many constraints. However, we can simplify this procedure and modify only a single constraints for every branch. The following lemma explains this simplification.

Lemma 6.5. *Let $Y \in \mathbb{R}^{n\times n}$ be positive semidefinite with diagonal $\mathbb{1}$. Further, let $i, j, k \in [n]$ be arbitrary indices. The following statements hold:*

(1) If $Y_{ij} = 1 = Y_{ik}$, then $Y_{jk} = 1$.

(2) If $Y_{ij} = -1 = Y_{ik}$, then $Y_{jk} = 1$.

(3) If $Y_{ij} = 1$ and $Y_{ik} = -1$, then $Y_{jk} = -1$.

Proof. First of all, Y is positive semidefinite, i.e. there is a $Q \in \mathbb{R}^{n\times n}$ such that $Y = Q^TQ$. Let Q_i be the i-th column of Q, then

$$Y_{ij} = \sum_{k=1}^{n} q_{ki}q_{kj} = Q_i^TQ_j \tag{6.14}$$

Since $Y_{ii} = Y_{jj} = Y_{kk} = 1$, it holds that $\|Q_s\|_2 = 1$ for all $s \in [n]$. We will show the 3rd statement as an example. The other cases follow completely analogously. $Y_{ij} = 1$ means that

$$1 = Q_i^TQ_j = \|Q_i\|_2\|Q_j\|_2 \cos\sphericalangle(Q_i, Q_j) = \cos\sphericalangle(Q_i, Q_j) \tag{6.15}$$

i.e., $Q_i = Q_j$. Analogously, from $Y_{ik} = -1$ it follows that $Q_i = -Q_k$. Hence in total, we get that $Q_j = -Q_k$, i.e., $Y_{jk} = -1$ □

This lemma is very useful for the branching process in the SDP. If we keep one single reference variable (say key 1), it is enough to add branching constraints only with respect to this reference. This means, in practice we only constrain one row (and due to symmetry also one column) of each block in Y and have to modify only one constraint in every branching step.

6.3.7 Alternative Objective Functions for the SDP

In the previous subsection, we first fixed some cut b and then derived a lower bound on the minimum QAP objective value by minimizing an SDP relaxation. That is, we obtain a valid lower bound by solving an SDP with the objective function $C^b \bullet Y^b + (c^b)^T y^b + \gamma^b$, subject to the constraints mentioned above. However, considering only one cut of the pseudo base in the objective could be weak because costs could be evaded by charging them to the other cuts of the pseudo base that are not accounted for in the objective.

$$
\begin{aligned}
\min \quad & \tfrac{1}{B}\sum_{b=1}^{B} C^b \bullet Y^b + (c^b)^T y^b + \gamma^b \\
\text{subject to} \quad & E_i \bullet Y^b = 1 && \forall i \in [n], b \in [B] \\
& \left(\sum_{b=1}^{B} \tfrac{1}{2} Y_{ij}^b + \tfrac{1}{2} Y_{ji}^b\right) \leq B - 1 && \forall i \neq j \in [n] \\
& \mathbb{1}\mathbb{1}^T \bullet Y = 0 \\
& Y \succeq 0
\end{aligned}
$$

Figure 6.2: The complete SDP formulation

Averaging over the cut pseudo base

Since the lower bound holds for arbitrary cuts b, it also holds for the average over all cuts in the cut pseudo base, i.e., the objective becomes

$$\frac{1}{B}\sum_{b=1}^{B} C^b \bullet Y^b + (c^b)^T y^b + \gamma^b.$$

There is no need to add further auxiliary variables or constraints that may harm the numeric stability of an SDP-solver.

Taking the maximum

An even stronger lower bound is obtained by taking the maximum over the cuts of the pseudo base because the arithmetic mean never exceeds the maximum. The standard way to model the maximum over the cut pseudo base is to introduce a new linear variable - say z - and add $\log_2 n$ many constraints, ensuring that z is at least the cost of each cut in the pseudo base. However, one of the solvers we used, more specifically, the Mosek solver (v7.1.0.53), often stalled with this objective function due to numerical instabilities, in contrast to the averaging objective.

We conclude this section by summarizing the model that was discussed in the previous pages in Figure 6.2.

6.3.8 The dual SDP

We consider the general form of the primal SDP

$$
\begin{aligned}
\min \quad & C \bullet X \\
\text{s.t.} \quad & A_i \bullet X = b_i \quad \forall i = 1, \ldots, m \\
& X \succeq 0
\end{aligned}
\tag{6.16}
$$

where C and A_i are defined in the previous sections. The standard dual transformation has the form

$$
\begin{aligned}
\max \quad & b^T y \\
\text{s.t.} \quad & \sum_{i=1}^{m} A_i y_i \preceq C
\end{aligned}
\tag{6.17}
$$

or, equivalently,

$$\begin{aligned} \max \quad & b^T y \\ \text{s.t.} \quad & \sum_{i=1}^{m} A_i y_i + S = C \\ & S \succeq 0 \end{aligned} \quad . \tag{6.18}$$

In our concrete case, the constraint matrices of the form $E_i \bullet Y^b = 1$ ensure that we can make the diagonal of S arbitrarily large (by setting the corresponding dual variable $y_i \ll 0$). Hence, any dual has a strictly feasible solution, and moreover, the primal-dual SDP-pair has the strong duality property [103].

6.4 Using the Right Solver - Discussion and Consequences

So far, we described the modeling of a new lower bound approach for the QAP. A concrete implementation depends on the underlying solver, together with the branch-and-bound realization. Both the SDP solvers and branching strategies are orthogonal research areas that we do not address here. However, it is necessary to discuss the pros and cons of the different configurations we have used in our implementation.

6.4.1 Mosek - a Commercial off-the-shelf SDP Solver

Recent benchmarks[2] suggest that Mosek belongs to the fastest SDP solvers. Its accessibility across several programming languages and its extensive documentation make Mosek the SDP solver of our choice. The drawback, however, is the missing implementation of a branch-and-bound procedure for integral SDPs. Hence, we implemented our own branch-and-bound framework and examined different simple branching strategies. One advantage of using our own framework is its ability to handle integral solutions differently than fractional ones. We have already mentioned that we transform all integral solutions to solutions of the original QAP using the 1-to-1 correspondence between the variables in use. This means we can evaluate integral solutions with the QAP cost and thus obtain an optimal solution to the QAP at the end of the optimization. Moreover, we can alter the objective function at any node in order to make use of our self-tightening cost estimation. The bottleneck of a self-implemented framework still remains the branching. Many man-years have been invested in the development of good branching strategies and heuristics. We observed that even with our simple branching strategies our evaluation yielded promising results. This suggests the assumption that our framework would strongly benefit from an advanced branch-and-bound framework and further improve its results. A minor drawback of Mosek is the aforementioned numerical instability of the maximum objective function. Therefore, we chose to investigate a further approach.

6.4.2 Gurobi - a Commercial off-the-shelf MIP Solver

Gurobi, one of the leading and fastest MIP solvers, provides a highly advanced branch-and-bound procedure that is customizable via callbacks. It is, however, not capable of semidefiniteness constraints. If we simply dropped these constraints, we would still obtain

[2] See the benchmarks by Hans Mittelmann: `http://plato.asu.edu/bench.html`

a formulation with a valid lower bound for the SDP (and so for the QAP). However, this bound would be too weak if we did not compensate the missing semidefiniteness at all. Therefore, we simulate the semidefiniteness constraints by a weak membership oracle, i.e., we separate negative eigenvalues by adding lazy constraints in a callback. More specifically, consider the solution X at the current node in the branching tree, and let λ be its smallest eigenvalue together with the eigenvector v. If λ is negative (allowing a small tolerance at 0), we add the constraint $v^T X v \geq 0$ in order to separate this eigenvalue. Recall that X is positive semidefinite if and only if $v^T X v \geq 0$ for every $v \in \mathbb{R}^n$. Note that this is a linear inequality in the variables x_{ij}. We cannot enforce this constraint for every possible vector v since those would be uncountably many. Instead, we seek good candidates during the optimization and add them in a lazy fashion. This has proven a good trade-off between the semidefiniteness precision and the number of constraints being added.

Another point we need to mention here is the objective function. Recall that we handle integral and fractional solutions in a different fashion and update the objective function as soon as our cost estimation got tighter with our own branching framework, while Gurobi does not support any of those features. We can, however, circumvent them by substituting the objective function with an additional variable ζ. Moreover, we add the constraints that ζ is (1) at least the original QAP objective, and (2) at least the SDP objective. We have observed, indeed, that the SDP objective dominates the value of ζ for fractional solutions, while for integral solutions the QAP objective does. This comes at the expense of introducing the n^2 original QAP variables, connecting them to the cut pseudo base variables, and creating the mentioned quadratic ζ-constraint together with the assignment constraints. Note, however, that these variables are not declared binary since they are bound to the binary cut pseudo base variables. Moreover, this structure allows to choose between the average and maximum SDP objective function at zero expense. Gurobi does not run into numerical issues with the maximum SDP objective function.

If we want to avoid the quadratic constraint, we can choose any tight linearization of the QAP objective, e.g., the Kaufman-Broeckx linearization. The advantage of this particular linearization is its low dimension. Its inclusion does not increase the asymptotic complexity of our formulation. Additionally, it is tight even if it is weak for fractional solutions. Our framework levels out this weakness with the SDP constraints. This means, the QAP linearization's only purpose is to evaluate the integral solutions to the correct objective value. Whether we decide to use the original QAP objective as a quadratic constraint or the linearized Kaufman-Broeckx version, we will end up with a formulation using only $\mathcal{O}(n^2)$ many variables, of which only $\mathcal{O}(n \log(n))$ are integral. This favorably contrasts our formulation with every other QAP relaxation that we are aware of. The resulting positive effect on the scalability, while still providing good lower bounds, will be shown and discussed in Section 6.6.

It remains to implement the updates to the cost function whenever our cost estimates get tighter. Recall that we already have the constraint that ζ is at least the SDP objective at the root relaxation. Instead of altering this particular constraint — which is not supported by Gurobi — we add a new locally valid constraint containing the updated SDP objective function. In order to ensure that this tighter SDP objective constraint is only active when we are at the correct branching subtree, we use a standard penalization

method. Let I_+ be the set of all indices i such that y_i is set to $+1$ in the current branch and let, accordingly, I_- be the set of indices with y_i being branched to -1.

$$\zeta \geq C_{SDP} - M \left(\sum_{i \in I_+} 1 - y_i + \sum_{i \in I_-} 1 + y_i \right)$$

The large constant M acts as a deactivator for the whole constraint. If all the y-assignments are correct, the sum term is 0, and the constraint is $\zeta \geq C_{SDP}$. As soon as at least one assignment differs from the current branch, we obtain the constraint $\zeta \geq C_{SDP} - \alpha M$ for some integer α and a large constant M (dependent on the concrete problem instance). Thus, the constraint becomes redundant and ineffective for any differing branch.

The concluding drawback is the introduction of $n^2 + 1$ linear variables and the usage of one quadratic constraint. Moreover, we need to add constraints instead of being able to alter the objective function.

Both Mosek and Gurobi have their clear advantages and disadvantages, and it is not obvious which option to prefer. Therefore, we have implemented and evaluated both versions and report the results in Section 6.6.

6.5 Comparison to the Gilmore-Lawler bound

Our relaxation can be compared to the Gilmore-Lawler bound [90], to some extent. For every pair of a single item and location i and k, their approach computes linear assignment problems as independent subproblems. Intuitively, the solutions of those subproblems describe the cost that you have to at least pay if you decide to assign item i to location k. This is similar to our cost estimation $c_{ij}^{(\alpha\beta)}$ described in Section 6.2.1. The difference is that the Gilmore-Lawler approach reduces to a single linear assignment problem, whereas we still have to solve an SDP. Then again, we can easily read off the values for $c_{ij}^{(\alpha\beta)}$ and do not have to solve n^2 many linear subproblems.

A question that naturally arises is if our approach dominates the Gilmore-Lawler bound or vice versa. It turns out that neither dominates. Consider the following small 3×3 example.

$$F = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \qquad D = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

The Gilmore-Lawler bound is 0 here, whereas our SDP approach reports 0.5 as the root relaxation value. The reason is that the independent subproblems can evade costs using always the third location. Since the constraints of our formulation provide a stricter connection between items and locations, costs cannot be evaded as easily in the SDP.

6.6 Evaluation

We compare our approaches to two classic linearizations, the Kaufman-Broeckx linearization [92] and the first level of the Reformulation Linearization Technique (RLT1) [95].

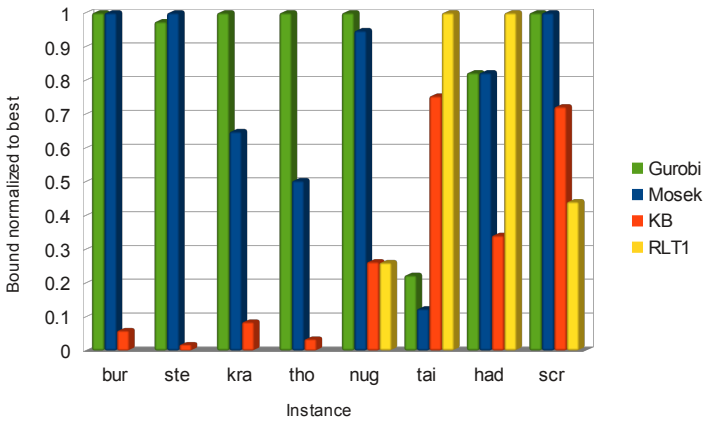

Figure 6.3: Averaged QAPLIB instances after one hour of computation

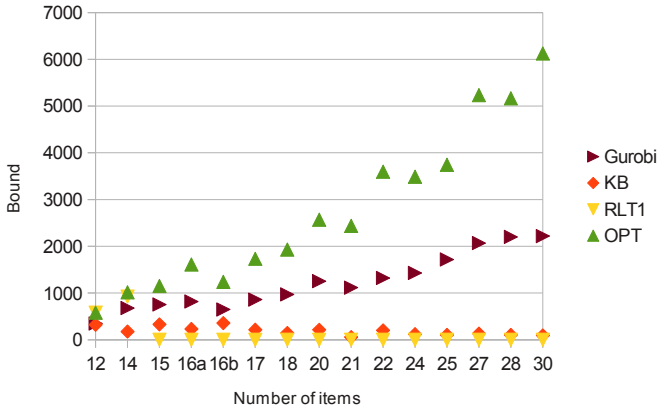

Figure 6.4: The Nugent instances with 12 to 30 items after one hour of computation, RLT1 fails to solve $n \geq 15$ within this time and resource limit.

For the sake of fair comparison, we solve the integral versions of the linearizations and use the commercial state-of-the-art solver Gurobi (v6.5.1) [99] to solve them. As already discussed in Section 6.4 we implemented both frameworks, one using Mosek (v7.1.0.53) [104] and the other one using Gurobi. Note that the paper [100] only contains the Mosek implementation. The discussion and implementation of the Gurobi version has been done after this publication; hence, the evaluation has also been extended. We will report the best known lower bound produced by running the branch-and-bound process for one hour.

We ran experiments on a single Intel (R) Xeon (R) E5-2680 2.50GHz core with 8 GB RAM, running Debian GNU/Linux 7 with kernel 3.18.35.1. The code was compiled with gcc version 4.7.2 using the `-O3` flag.

The instances were taken from the QAPLIB homepage [79]. The names of the instances are formed by the name of the author (first three letters), the number of items, followed by a single letter identifier. The test instances cover a wide range of QAP applications including keyboard assignment, hospital layout, and several further graph problems.

Figure 6.3 shows the average lower bound of the different approaches after one hour

of computation time. We decided to average the lower bounds of a certain instance set because the single test cases within that set were similar and all approaches behaved consistently there. One can see that RLT1 performs quite well if we have enough computation power to compute bounds there (see `tai` or `had`, for example). However, many test instances are too large for our computing resources to compute even the RLT1 root relaxation. In these *hard* cases, our approaches outperform both linear relaxations by several orders of magnitude. The reason Gurobi performs much better than Mosek in some cases is that Gurobi is capable of producing non-trivial lower bounds for much larger problem instances. This results in higher green bars in Figure 6.3 for all instance sets that include instances of size 35 and higher. The largest instances for which Mosek and Gurobi could produce a non-trivial lower bound consisted of 36 and 80 items, respectively. We see the reason for this immense improvement in scalability in the advanced branch-and-bound framework that Gurobi offers. We point out that for instances where at least the root relaxation can be solved by both solvers, the resulting lower bounds are similar with a really small advantage for Gurobi in many cases. This advantage is not consistent throughout all the instances, i.e., there are instances where our Mosek approach produces a better lower bound than the Gurobi approach.

The `nug` instances are a special instance set because this set contains test cases of consistently increasing size. Therefore, the behavior of the approaches varies throughout the different test cases, and the average reported in Figure 6.3 cannot reflect the overall behavior for `nug`. To this end, we report a detailed description of the whole `nug` test set in Figure 6.4. This also shows how our framework scales with increasing n. We decided to only show the performance of our Gurobi framework and omit the Mosek results since the Gurobi results were consistently up to 10% better, but the graphic representation is not able to show that at this scale. The full numerical table is available online for the interested reader[3].

At the beginning, for small n, RLT1 can solve the instances to optimality within one hour, which meets our expectations. For instances of small size, the advantages of our approach in efficiency are too small to make up for the loss of precision caused by the distance approximations. The trend changes as soon as n grows above 16. The RLT1 formulations are already too large to solve the root relaxation after one hour of computation with a single thread, and the bounds of the Kaufman-Broeckx relaxation are lower than ours, which confirms the use-case that we proposed in the beginning of this chapter.

[3] http://resources.mpi-inf.mpg.de/keyboardoptimization/ISCO2016/fulleval.csv

CHAPTER 7

A Markup Language for Optimization Problems

7.1 Introduction

This chapter describes a new modeling language for mixed integer linear programs based on the HTML standard. This project was motivated by the French keyboard problem and its participatory optimization structure that we discussed earlier. Our main goal is to create a platform where domain experts without programming skills can model optimization problems that appear in their fields. The committee in the French keyboard problem, for example, consisted of experts from various domains like keyboard manufacturers or linguists; they have very valuable knowledge for the design of keybaords, but could not directly transfer it to optimization models with current state-of-the-art techniques. While providing users with the full power of linear optimization solvers, we aim to create a modeling language that lowers this entry hurdle for people working in not purely mathematical fields, but who still frequently encounter optimization problems. We aim to achieve this simplicity by providing an HTML extension that allows the modeling of complex optimization problems via a graphical user interface with drag-and-drop features. Moreover, we want to provide a concise, human-readable and easily expandable description. The separation of model and data has shown to be relevant in related modeling languages and is also supported by our framework. In participatory optimization problems, optimization parameters and input data frequently change. By separating model and data, the optimization model expressed in our language can remain unchanged whenever changes are made to the input data of the problem. There exists an implementation of an interpreter for this language as a part of Sören Bund-Becker's Bachelor's thesis [105] that Andreas and I supervised.

7.1.1 Combinatorial Optimization Outside of Computer Science

Optimization problems do not only occur in mathematics or computer science, but also, for example, in fields related to medicine, economics, or biology.

Mitsos et al. performed experiments on drug effects [106], and the data fitting is done by an integer linear programming formulation.

Process optimization is a big research area in the field of economics. Popular challenges in this area are, for example, workflow optimization [107], and resource optimization [108], which can be formulated as linear programs. Often, these problems do not only require an optimal solution, but also a solution that is robust to changes (delay in workflow, decay of resources, etc.), which can be accounted for in optimization problems.

Bioinformatics is a young research field that contains many NP-hard optimization problems, many of which can be tackled by ILP techniques [109]. Recently, Karrenbauer

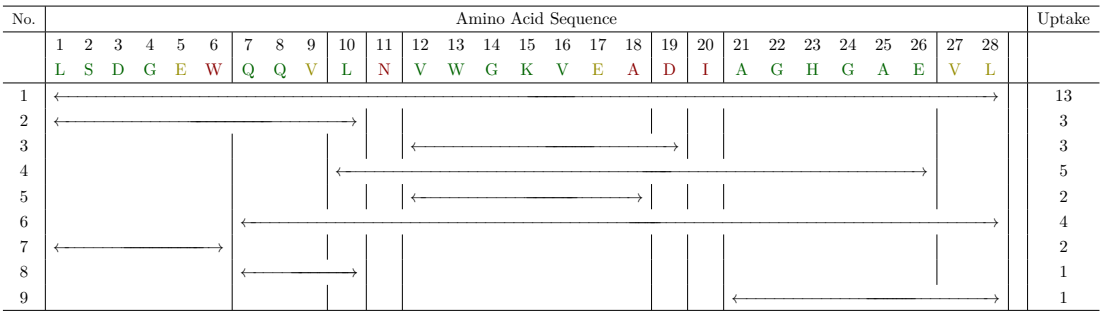

Figure 7.1: Example of the measurements for an HDX experiment; taken from Optimization class 2019

and Wöll [110] have proposed a mixed integer programming approach to track blinking molecules in single molecule microscopy. Their approach optimizes the likelihood of the connection between molecule positions even if a fluorescent molecule is blinking, i.e, is dark for several recoreded frames.

Another optimization challenge appears in the measurement of hydrogen-deuterium exchange(HDX) speeds of single residues. In these experiments, protein chains are put into heavy water, leading to an exchange of Deuterium atoms with Hydrogen atoms of the protein residues. The resulting difference in mass can be measured by mass-spectometry, which only works for random fragements of the protein chain and not for single residues. Moreover, these mass measurements are prone to small errors so that the exact sequence of residue changes cannot be restored. It is possible, however, to recover a sequence that minimizes the total deviation of the measured data over all fragments by an LP approach that was proposed by Althaus et al. [111] We discuss this LP formulation in more detail in the following section and show later in this chapter how to model this rather complex real-world problem in our new modeling language.

7.1.2 The HDX Experiments

Let n be the number of residues in the protein chain and m the number of fragments that we measure a mass difference. $\mathcal{F}$ denotes the set of all these m fragments and the ith fragment is defined by its first and last residue (ℓ_i, r_i) that is contained in it. Furthermore, the number b_i denotes the total number of residues that have changed in fragment i. An example of this scenario is depicted in Figure 7.1. We introduce binary variables x_i for each residue $i \in [n]$.

$$x_i = \begin{cases} 1, & \text{if residue } x_i \text{ has been changed} \\ 0, & \text{otherwise} \end{cases} \tag{7.1}$$

The objective function minimizes the total deviation from the measurements, i.e.,

$$\min \sum_{i \in \mathcal{F}} \left| b_i - \sum_{k=\ell_i}^{r_i} x_k \right| \tag{7.2}$$

For better readability, we introduce the matrix $A \in \{0,1\}^{m \times n}$, which is

$$A_{ij} = \begin{cases} 1 & \text{, if } s_i \leq j \leq t_i \\ 0 & \text{, otherwise} \end{cases} \tag{7.3}$$

leading to the objective function being $|b - Ax|$.

Furthermore, we linearize the absolute value by introducing two slack vectors $s^+, s^- \in \mathbb{R}^m_{\geq 0}$ such that $s_i^+ - s_i^- = b_i - (Ax)_i$ and $s_i^+ + s_i^- = |b_i - (Ax)_i|$. The resulting linear programming formulation is as follows[1].

$$\begin{aligned} \min \quad & \mathbf{1}^T s^+ + \mathbf{1}^T s^- \\ \text{s.t.} \quad & Ax + s^+ - s^- = b \\ & 0 \leq x_i \leq 1, \quad \forall i \in [n] \\ & s_j^+, s_j^- \geq 0, \quad \forall j \in [m] \end{aligned} \tag{7.4}$$

7.1.3 Why HTML?

The main reason for the choice of our modeling language being based on HTML is its great accessibility. The markup language provides not only great structure, but also an easy way to separate model and data. Around 84% of all websites are based on HTML according to a recent online study of `https://w3techs.com`. This means that many users of our target group probably have programmed or at least seen HTML code. Furthermore, HTML offers a wide variation of visualization via stylesheets. We can use the available functionality of browsers if we offer our modeling tool as an HTML extension, which simplifies the user's installation process and hence increases the availability of our framework. Using only the browser capabilities, we can provide a graphical user interface that allows dragging and dropping of model parts and mathematical expressions. For more complex functionalities, we can even integrate Javascript code into our modeling language without much effort. More details on the Javascript integration are discussed at the end of this chapter. Regarding input formats, we can use existing tools to read JSON or similar formats that are commonly used in the context of HTML; hence, utilizing already existing data as input for our models. Moreover, there exist fully automatic validation tools that check the correctness of written HTML code.

7.1.4 Related Work

Many academic and commercial LP solvers offer interfaces to commonly used programming languages, allowing the user to create their model and access solver-specific parameters. In order to simplify the process of modeling optimization problems, changing common parameters, and accessing information about the outcome of the solution process.

[1] The x-variables are defined as continuous because the constraint matrix is totally unimodular. In this case, every extreme point of the corresponding polyhedron is already integer.

Open Solver Interface The Open Solver Interface (OSI) is a part of the open source framework COIN-OR [112] that offers a C++ interface to model an optimization problem and access the most common solver parameters. The special feature of OSI is that it provides a generic interface that is not bound to a particular solver. A single command can translate your OSI model to the solver of your choice. This is especially useful if you want to compare the performance of different competing solvers without needing to code your model multiple times.

Algebraic Modelling Languages There exist several modeling languages that are based on an algebraic language structure. A Mathematical Programming Language (AMPL) [113] is one such example. Its mathematical syntax is combined with descriptive parts improving the readability of the constructed models. The General Algebraic Modeling System (GAMS) [114] follows a similar approach. Both of these mathematical modeling languages provide a full built-in support for all expected commercial LP solvers. The Zuse Institute Mathematical Programming Language (ZIMPL) [115] also proposes a modeling language with a similar syntax. A major difference of this framework to its mentioned alternatives is the creation of an intermediate model file which can then be used as an input to most LP solvers. One important key feature that all of these approaches share is the separation of model and data. In many commercial applications, the same optimization problem repeatedly appears with varying input data. Therefore, it is convenient to reuse the same model file and only alter the critical data.

7.1.5 Our Contribution

We present a markup language for the design of linear optimization problems that is

(1) **simple**: users without deep knowledge in math or computer science can model complex optimization problems by drag-and-dropping tags in a web interface;

(2) **concise**: aggregation of terms allows a short and human-readable representation of complex optimization problems;

(3) **expressive**: every linear program can be modeled with our language;

(4) **extendable**: the modular structure of our implementation encourages the extension of features or the connection to other ILP solvers.

Similar to the approaches mentioned above, we emphasize the possibility to separate model and data in our language. This does not only allow an easier reusability of models, but also improves the readability and maintainability of more complex optimization models. Furthermore, we provide an implementation of an interpreter for this language, which parses the HTML code, translates it to an internal representation of an optimization problem and then offers the choice to either directly translate the problem to a solver's interface or to produce an LP file, which can then be used as an input for the LP solver of your choice. Our implementation offers the connection to the Gurobi interface, which serves as an example and shows the simplicity of connecting an arbitrary solver that provides a C/C++ interface.

7.2 Grammar and Features

We introduce the grammar of our modeling language incrementally. First, we present a minimal language $\mathbb{L}_{\text{min}}$ that enables an arbitrary linear optimization problem to be modelled by $\mathbb{L}_{\text{min}}$. This minimal language allows us to prove the LP-completeness, of optXML; i.e., every linear programming problem can be expressed. Similar to an axiomatic definition, every subsequent additional feature that we introduce in the rest of this chapter can be reduced to a block of tags in $\mathbb{L}_{\text{min}}$ (axioms) and thus does not change the expressive power of the enhanced language. Note that the HTML standard demands a dash within any custom tag. We solve this by requiring the prefix "opt-" in front of every tag; however, for the sake of readability, we omit all prefixes in the definition and discussion of tags and examples in this chapter. In order to interpret the examples with our framework, all tags have to include the prefix, e.g., <constraint> has to be replaced by <opt-constraint>.

7.2.1 A Minimal LP-Complete Language

⟨*instance*⟩ ::= `<instance>` ⟨*model*⟩ ⟨*data*⟩ `</instance>`

⟨*model*⟩ ::= `<model>` ⟨*objectives*⟩ ⟨*constraints*⟩ ⟨*variables*⟩ `</model>`

⟨*objectives*⟩ ::= | `<objectives>` ⟨*objective*⟩ `</objectives>`

⟨*objective*⟩ ::= | `<objective>` ⟨*statement*⟩ `</objective>` ⟨*objective*⟩

⟨*variables*⟩ ::= `<variables>` ⟨*vardef*⟩ `</variables>`

⟨*vardef*⟩ ::= | `<var/>` ⟨*vardef*⟩

⟨*constraints*⟩ ::= `<constraints>` ⟨*constraint*⟩ `</constraints>`

⟨*constraint*⟩ ::= | `<constraint>` ⟨*lhs*⟩ ⟨*rhs*⟩ `</constraint>` ⟨*constraint*⟩

⟨*lhs*⟩ ::= `<lhs>` ⟨*statement*⟩ `</lhs>`

⟨*rhs*⟩ ::= `<rhs>` ⟨*statement*⟩ `</rhs>`

⟨*statement*⟩ ::= | ⟨*add*⟩ | ⟨*mul*⟩ | ⟨*var*⟩ | ⟨*constant*⟩

⟨*add*⟩ ::= | `<add>` ⟨*statement*⟩ `</add>` ⟨*add*⟩

⟨*mul*⟩ ::= | `<mul>` ⟨*statement*⟩ `</mul>` ⟨*mul*⟩

⟨*var*⟩ ::= `<var/>`

⟨*constant*⟩ ::= `<constant/>`

With the definition of this grammar, we only served one of the four key principles we promised so far: the *expressive power*. It is easy to show that an arbitrary linear program can be transformed to an equivalent optXML problem using only the structure introduced above. However, the complexity of such a problem would be at least as large as the one of *.lp*-files, which is exactly one of the issues we wanted to improve. In order to allow for a more *concise* representation of optXML problems, we extend the grammar by aggregation tags and macros in the following section.

7.2.2 Tag-Specific Parameters

So far, we only presented the grammar, which serves as the basic structure of our language. In order to provide crucial information about the optimization model, we allow different parameters for every tag.

Structural tags The tags `instance`, `model`, `data`, `objectives`, `constraints`, and `vardef` serve a structuring role in the HTML tree. The former `instance` tag is the root node of the tree, which has to exist according to the HTML standard. The last three tags are responsible to bundle all objectives, constraints and variable definitions in order to improve the structure and readability of the model.

Although these tags do not have any special parameters due to their purely structural role, one could add parameters to the `instance` tag that encode solver-specific parameters; for example, the choice of the solving method, the focus on lower or upper bounds, or the amount of presolving done before the actual optimization.

```
<objective sense="minimize">
  ...
</objective>
```

(a) A single-objective minimization problem

```
<objective sense="minimize" weight="0.4">
  ...
</objective>
```

(b) A multi-objective optimization problem applying the weighted average

```
<objectives>
        <objective sense="minimize" rank="1">
          ...
        </objective>
        <objective sense="maximize" rank="2">
          ...
        </objective>
        <objective sense="minimize" rank="3">
          ...
        </objective>
</objectives>
```

(c) A multi-objective optimization problem applying the lexicographic method

Figure 7.2: Three possible use-cases for the objective tag.

Objective The `objective` tag has multiple parameters. The first and mandatory parameter is the objective sense, which can be either *maximize* or *minimize*. In the scenario of mulit-objective optimization, the tag obtains two additional parameters, *weight* and *rank*. The *weight* attribute indicates that the multiple objectives are linearized to a weighted average, whereas the *rank* attribute indicates the use of the lexicographic method. Figure 7.2 shows three examples for the use-cases mentioned above.

```
<constraint relation="<=">
  <lhs>
    ...
  </lhs>
  <rhs>
     ...
  </rhs>
</constraint>
```

(a) A constraint of the form $lhs \leq rhs$

```
<constraint relation="<=" rhs="1">
  <lhs>
    ...
  </lhs>
</constraint>
```

(b) The right-hand side is contained in the attribute of the constraint-tag

```
<constraint expression="x + y <= 1"/>
```

(c) The full constraint expression is stated in JavaScript format.

Figure 7.3: Three possible use-cases for the constraint tag.

Constraint The constraint tag has one mandatory attribute: its relation ($=$, $\leq$, or $\geq$). Additional optional parameters allow the user to specify simple expressions as right hand sides or even as the full constraints. Figure 7.3 shows three examples of simple constraint structures.

```
<var name="x" index="(1,2)"/>
```

(a) The variable tag for $x_{1,2}$

```
<var name="x" index="1" type="binary"/>
```

(b) Defining the binary variable x_1

```
<var name="z" type="integer" lowerbound="2" upperbound="10"/>
```

(c) Defining the integer variable z with the additional constraint $2 \leq z \leq 10$

Figure 7.4: Three possible use-cases for the variable tag.

Variable Variables must have a name, which is defined as a parameter in the variable tag. An optional attribute is an index, which can be a natural number or a tuple of numbers. Using this index attribute, we can express variables of the form x_i or $x_{i,j}$. The usage of indices will be especially important when we introduce aggregation tags or macros. If we define a variable in the `vardef` section of our model, additional parameters are enabled; e.g., the type of variable (binary, integer, or continuous) and lower/upper bounds. Figure 7.4 shows an example for using a variable in an expression and two additional examples for defining variables in the `vardef` section.

```
<constant number="5"/>
```

(a) The constant tag for a concrete number

```
<constant name="c" index="(i,j)"/>
```

(b) The constant tag for the not yet specified term $c_{i,j}$

Figure 7.5: Two possibilities to use the constant tag in an expression.

Constant Constants can either be hard-coded numbers or have a name and possibly also an index similar to variables. This latter case plays an important role in the separation of model and data that we discussed before. We can model a general optimization problem without specifying actual weights; instead, we use placeholders for these constant expressions, which are replaced by constant numbers during the instantiation of a concrete optimization instance.

```
<mult>
  <constant number="5"/>
  <var name="x"/>
</mult>
```

(a) The expression $5 \cdot x$

```
<var name="x" coefficient="5"/>
```

(b) The equivalent simplified expression.

```
<add>
  <var name="x"/>
  <var name="y" coefficient="2"/>
</add>
```

(c) The expression $x + 2y$

Figure 7.6: Adding and multiplying constants and variables

Mathematical operators We allow addition and multiplication as mathematical operations. These tags do not have any additional parameters. While we only solve mixed integer linear problems at the moment, the structure of our language easily allows quadratic or higher-order terms of variables. Simple expressions like the multiplication of a variable with a constant number can be simplified by enhancing the variable tag with a coefficient parameter. Figure 7.6 show both alternative versions.

7.3 Additional Features

Many expressions in linear programs can be simplified using the $\sum$- or the $\forall$-operators. These aggregation operators allow for more concise representations of even complex optimization problems. We introduce the <each> tag, which mimics both these operators depending on its position in the HTML tree.

```
<each item="i" family="[m:n]" >
  <add>
    <var name="x" index="i">
  </add>
</each>
```

(a) The expression $\sum_{i=m}^{n} x_i$

```
<each item="(a,b)" family="F">
  <add>
    <var name="x" index="a">
    <var name="x" index="b">
  </add>
</each>
```

(b) The expression $\sum_{i \in F} (x_a + x_b)$

```
<each item="i" family="[n]">
  <constraint>
    ...
  </constraint>
</each>
```

(c) Defining a constraint $\forall i \in [n]$

Figure 7.7: Using each to express sums or ∀-operators.

It indicates the iteration over a given family F. For simplicity, the set $\{1, \ldots, n\}$ can also be written as $[n]$, and the set $\{m, \ldots, n\}$ for $m \leq n$ can be written as $[m : n]$. The children of an each-tag can access the current element via the *item* parameter (i in Figure 7.7). Note that, depending on the structure of F, an item can also be a tuple or vector, which can be written as *item="(a,b)"*.

The linear programming formulation for the HDX experiment inspired us to consider an additional use-case. The constraint $Ax + s^+ - s^- = b$ in the linear programming formulation (7.4)) contains the fragments (ℓ_i, r_i) for all $i \in \mathcal{F}$ and the number of changes b_i in this fragment. The full expression is

$$\forall i \in \mathcal{F} : s_i^+ - s_i^- + \sum_{k=\ell_i}^{r_i} x_k = b_i \tag{7.5}$$

The inherent challenge of modeling this constraint is that the data points ℓ_i and r_i are used to define the set of indices of x-variables in the summation term again. At the same time, we need to know the fragment number i in order to add the variables s_i^+ and s_i^- to the correct constraint. Assuming that we are given the fragments as a vector containing the tuples (ℓ_i, r_i), a naive solution to this challenge would be to enhance the tuple to (ℓ_i, r_i, i). Although this would allow us to easily access all the information we need to model this problem, it is quite tedious to manipulate data with trivial information. Instead, we propose to introduce the *index* feature of the each tag, which can be used alongside the item parameter. If this index parameter is used, every item in a family

```
<each  item="(l,r)" family="F" index="i">
        <constraint relation="=">
                <lhs>
                        <add>
                                <var name="splus" index="i"/>
                                <var name="sminus" index="i" coefficient="-1"/>
                                <each item="k" family="[l,r]" >
                                        <var name="x" index="k"/>
                                </each>
                        </add>
                </lhs>
                <rhs>
                        <add>
                                <constant name="b" index="i"/>
                        </add>
                </rhs>
        </constraint>
</each>
```

Figure 7.8: The index parameter used in the HDX model

```
<add>
        <each item="i" family="F" filter="i%2==0">
                <var name="x" index="i"/>
        </each>
</add>
```

Figure 7.9: Each with a filter: $\sum_{i \in F : i \text{ even}} x_i$

F is labelled with an ID reflecting the item's ordering in F, which can then be used to access the corresponding entry of a different vector. Figure 7.8 shows the use of this new parameter in the HDX model.

Filtering each If the user wants to iterate over a certain family and test the current item before deciding whether or not to process this item (i.e., generate an expression, constraint, variable, ...), we propose the *filter* parameter of the each tag. The filter has to be a valid Javascript expression that evaluates to a boolean; the children of the each tag are only processed for every item that evaluates to TRUE. Figure 7.9 shows an example where we add only the even numbers of a family F to a sum expression.

Macro Programming In order to further simplify the modelling of optimization problems and allow users with more programming expertise to reuse frequently used model parts, we introduce macros. Similar to macros in popular programming languages like Java or C, we allow users to define custom tag structures. Whenever this custom tags appears in the optimization model, it is replaced according to its definition in a preprocessing step. The <macro> tag is used for the macro definition. Its mandatory parameters are the name, which has to be distinct from the tag names we reserved so far and from any other macro names in the model. Additional parameters are specified

in the *parameter* attribute. Figure 7.1 shows the definition of a macro for assignment constraints.

Example 7.1. We first define the following macro for assignment constraints. We declare the parameters xvar and num, which are used inside the definition and are replaced when we call the macro in a model.

```
<macro name="assignment" parameter="xvar,num">
   <each family="[num]" item="i">
        <constraint relation="=">
                <lhs>
                        <add>
                                <each item="j" family="[num]">
                                        <var name="xvar" index="(i,j)"/>
                                </each>
                        </add>
                </lhs>
                <rhs>
                        <constant number="1"/>
                </rhs>
        </constraint>
   </each>
   <each item="j" family="[num]">
        <constraint relation="=">
                <lhs>
                        <add>
                                <each item="i" family="[num]">
                                        <var name="xvar" index="(i,j)"/>
                                </each>
                        </add>
                </lhs>
                <rhs>
                        <constant number="1"/>
                </rhs>
        </constraint>
   </each
</macro>
```

If we want to use this macro in a model, we use it equivalently to a standard tag and define the required parameters.

```
<constraints>
        <assignment xvar="x" num="n"/>
</constraints>
<variables>
        <each item="i" family="[n]">
                <variable name="x" index="i" type="binary"/>
        </each>
</variables>
```

This features can not only be useful for users to create their own custom macros, it also offers the chance for us to create a huge library of macros for commonly used structures in optimization problems. Similar to HTML templates, users could download the macros they need for their problem (e.g., macros for assignment or covering constraints) and include them in their model.

Javascript The integration of Javascript into HTML code is very simple; HTML 5 even includes basic Javascript techniques into its standard [116]. In the context of our modelling language, Javascript code could be useful to generate expressions that require more complex mathematical calculations than can be done in simple filter expression. Experienced users can include <script> tags within the <model> tree. When the HTML tree is traversed and interpreted, the Javascript code in an occurring <script> node is interpreted in a sandbox and the script node is then replaced by the HTML structure generated during the code interpretation. If the parent tags of the script node contain value definitions (for example, when unfolding a tag like *<each family="[n]" item="i">*, i has a value in $[n]$), these values are available to be read by the Javascript code. In order to avoid side effects or undefined behaviour, we do not allow Javascript to manipulate these values; the access is read-only. Exporting HTML code from Javascript to our model tree is done via a predefined function EVALUATE. This Javascript function expects a string that is converted to a HTML tree, which is then added at the current position. An example of constraint generation with Javascript is depicted in Figure 7.10. It also shows an alternative (more convenient) option to add the constraint from Javascript to the model. It is easily possible to define similar evaluation functions (like EVALUATECONSTRAINT, EVALUATEVARIABLE, etc.) that are dedicated for different structures of the optimization model, which supposedly further simplifies the use of Javascript.

7.4 Conclusion

This chapter lays the foundation for a markup language to model optimization problems. While we mainly focused on the design of the language, its structure and features to simplify its use and lower the entry hurdles in this chapter, a major focus of Sören Bund-Becker's Bachelor's thesis, which I supervised with Andreas Karrenbauer, was the implementation of an interpreter for the language including all the features described above. The presented structure of the language together with this implementation allow us to conveniently handle the modeling of the French keyboard problem, which was the motivation that initiated this project. Before this framework can reach a broader audience, there are still some open challenges that have to be resolved. A visually pleasing and user-friendly implementation of GUI, for instance by using stylesheets, enabling drag-and-drop, etc. still has to be done. Having a simple interface for domain experts without programming skills is a crucial step towards increasing the acceptance of this modeling language. Moreover, an integration of our language into commonly used spreadsheet programs like Microsoft Excel that allows to directly pull and manipulate data of spreadsheets would immensely increase the reach of our approach. One of the most important premises of this project was the extendibility of both design and implementation for future students and researchers. I believe, we achieved this goal and proposed a solid initial framework that invites to be extended and further improved in the future.

```
<constraints>
        <each  item="i" family="[4]">
                <script type="text/javascript">
                        var k = Math.sqrt(i);
                        if ( (Math.floor(k) == k) {
                                evaluate(''<constraint expression="x_i+k<=y"/>'');
                        }
                </script>
        </each>
</constraints>
```

(a) Using Javascript to define a constraint $x_i + \sqrt{i} \leq y$ if and only if $\sqrt{i} \in \mathbb{N}$

```
<constraints>
        <each item="i" family="[4]">
                <script type="text/javascript">
                        var k = Math.sqrt(i);
                        if ( (Math.floor(k) == k) {
                                evaluateConstraint(''x_i + k<= y'');
                        }
                </script>
        </each>
</constraints>
```

(b) An equivalent formulation with a more convenient evaluation function

```
<constraints>
        <constraint expression="x_1+1<=y"/>
        <constraint expression="x_4+2<=y"/>
</constraints>
```

(c) The HTML code generated by the Javascript interpretation

Figure 7.10: The modeling capabilities of Javascript

List of Figures

Bibliography

[1] A. Feit, M. Nancel, M. John, A. Karrenbauer, D. Weir, and A. Oulasvirta, "Azerty amélioré: Computational design on a national scale," *accepted in Communications of the ACM*, 2020.

[2] M. John and A. Karrenbauer, "Dynamic sparsification for quadratic assignment problems," in *Mathematical Optimization Theory and Operations Research* (M. Khachay, Y. Kochetov, and P. Pardalos, eds.), (Cham), pp. 232–246, Springer International Publishing, 2019.

[3] A. Oulasvirta, N. R. Dayama, M. Shiripour, M. John, and A. Karrenbauer, "Combinatorial optimization of graphical user interface designs," *Proceedings of the IEEE*, 2020, doi=10.1109/JPROC.2020.2969687.

[4] I. S. MacKenzie and S. X. Zhang, "The design and evaluation of a high-performance soft keyboard," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '99, (New York, NY, USA), pp. 25–31, ACM, 1999.

[5] R. Burkard and J. Offermann, "Entwurf von Schreibmaschinentastaturen mittels quadratischer Zuordnungsprobleme," *Zeitschrift für Operations Research*, vol. 21, pp. 121–132, 1977.

[6] A. Oulasvirta, A. Feit, P. Lähteenlahti, and A. Karrenbauer, "Computational support for functionality selection in interaction design," *ACM Transactions on Computer-Human Interaction (TOCHI)*, vol. 24, no. 5, p. 34, 2017.

[7] A. M. Memon, M. L. Soffa, and M. E. Pollack, "Coverage criteria for gui testing," *ACM SIGSOFT Software Engineering Notes*, vol. 26, no. 5, pp. 256–267, 2001.

[8] S. K. Feiner, "A grid-based approach to automating display layout," in *Proceedings on Graphics Interface '88*, (Toronto, Ont., Canada, Canada), pp. 192–197, Canadian Information Processing Society, 1988.

[9] W. O. Galitz, *The essential guide to user interface design: an introduction to GUI design principles and techniques.* John Wiley & Sons, 2007.

[10] W. R. King and J. He, "A meta-analysis of the technology acceptance model," *Information & management*, vol. 43, no. 6, pp. 740–755, 2006.

[11] A. Oulasvirta, X. Bi, and A. Howes, *Computational Interaction.* Oxford University Press, 2018.

[12] N. Cross, *Designerly ways of knowing.* Springer, 2006.

[13] J. Löwgren and E. Stolterman, *Thoughtful interaction design: A design perspective on information technology.* The MIT press, 2004.

[14] A. Chevalier and M. Y. Ivory, "Web site designs: Influences of designer's expertise and design constraints," *International Journal of Human-Computer Studies*, vol. 58, no. 1, pp. 57–87, 2003.

[15] A. Dix, *Human-computer interaction*. Springer, 2009.

[16] L. Hallnäs and J. Redström, *Interaction design: foundations, experiments*. Textile Research Centre, Swedish School of Textiles, Unversity College of Borås and Interactive Institute, 2006.

[17] J. Preece, H. Sharp, and Y. Rogers, *Interaction Design-beyond human-computer interaction*. John Wiley & Sons, 2015.

[18] D. Saffer, *Designing for interaction: creating innovative applications and devices*. New Riders, 2010.

[19] T. Winograd, "The design of interaction," in *Beyond calculation*, pp. 149–161, Springer, 1997.

[20] K. Dorst and N. Cross, "Creativity in the design process: co-evolution of problem–solution," *Design studies*, vol. 22, no. 5, pp. 425–437, 2001.

[21] B. Buxton, *Sketching user experiences: getting the design right and the right design: getting the design right and the right design*. Morgan Kaufmann, 2010.

[22] A. Oulasvirta and A. Karrenbauer, "Combinatorial optimization for UI design," in *Computational Interaction* (A. Oulasvirta, P. O. Kristensson, X. Bi, and A. Howes, eds.), pp. 97–120, Oxford, UK: Oxford University Press, 2018.

[23] A. M. Feit, *Assignment Problems for Optimizing Text Input*. G5 artikkeliväitöskirja, 2018.

[24] J. Accot and S. Zhai, "Refining fitts' law models for bivariate pointing," in *Proceedings of the SIGCHI conference on Human factors in computing systems*, pp. 193–200, ACM, 2003.

[25] K. Gajos and D. S. Weld, "Preference elicitation for interface optimization," in *Proceedings of the 18th annual ACM symposium on User interface software and technology*, pp. 173–182, ACM, 2005.

[26] R. T. Marler and J. S. Arora, "Survey of multi-objective optimization methods for engineering," *Structural and multidisciplinary optimization*, vol. 26, no. 6, pp. 369–395, 2004.

[27] F. Hillier and G. Lieberman, *Introduction to Operations Research*. No. v. 1 in Introduction to Operations Research, McGraw-Hill, 2001.

[28] M. Kaisa, *Nonlinear Multiobjective Optimization*, vol. 12 of *International Series in Operations Research & Management Science*. Boston, USA: Kluwer Academic Publishers, 1999.

[29] K.-L. Du, M. Swamy, *et al.*, "Search and optimization by metaheuristics," *Birkhaüser) July*, 2016.

[30] E.-G. Talbi, *Metaheuristics: from design to implementation*, vol. 74. John Wiley & Sons, 2009.

[31] S. Chand and M. Wagner, "Evolutionary many-objective optimization: A quick-start guide," *Surveys in Operations Research and Management Science*, vol. 20, no. 2, pp. 35–42, 2015.

[32] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: Nsga-ii," *IEEE transactions on evolutionary computation*, vol. 6, no. 2, pp. 182–197, 2002.

[33] E. Zitzler, M. Laumanns, and L. Thiele, "Spea2: Improving the strength pareto evolutionary algorithm," *TIK-report*, vol. 103, 2001.

[34] S. Lok and S. Feiner, "A survey of automated layout techniques for information presentations," *Proceedings of SmartGraphics*, vol. 2001, pp. 61–68, 2001.

[35] R. Kennard and R. Steele, "Application of software mining to automatic user interface generation," in *International Conference on Software Methods and Tools*, IOS Press, 2008.

[36] R. Kumar, A. Satyanarayan, C. Torres, M. Lim, S. Ahmad, S. R. Klemmer, and J. O. Talton, "Webzeitgeist: design mining the web," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 3083–3092, ACM, 2013.

[37] P. O'Donovan, A. Agarwala, and A. Hertzmann, "Learning layouts for single-pagegraphic designs," *IEEE Transactions on Visualization and Computer Graphics*, vol. 20, pp. 1200–1213, Aug 2014.

[38] J. Talton, L. Yang, R. Kumar, M. Lim, N. Goodman, and R. Měch, "Learning design patterns with bayesian grammar induction," in *Proceedings of the 25th Annual ACM Symposium on User Interface Software and Technology*, UIST '12, (New York, NY, USA), pp. 63–74, ACM, 2012.

[39] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. De Freitas, "Taking the human out of the loop: A review of bayesian optimization," *Proceedings of the IEEE*, vol. 104, no. 1, pp. 148–175, 2016.

[40] Y. Koyama and T. Igarashi, "Computational design with crowds," *Computational Interaction*, p. 153, 2018.

[41] H.-G. Beyer and B. Sendhoff, "Robust optimization–a comprehensive survey," *Computer methods in applied mechanics and engineering*, vol. 196, no. 33-34, pp. 3190–3218, 2007.

[42] B. L. Gorissen, İhsan Yanıkoğlu, and D. den Hertog, "A practical guide to robust optimization," *Omega*, vol. 53, pp. 124 – 137, 2015.

[43] N. V. Sahinidis, "Optimization under uncertainty: state-of-the-art and opportunities," *Computers and Chemical Engineering*, vol. 28, no. 6, pp. 971 – 983, 2004. FOCAPO 2003 Special issue.

[44] M. Fisher, "Interactive optimization," *Annals of Operations Research*, vol. 5, no. 3, pp. 539–556, 1985.

[45] G. W. Klau, N. Lesh, J. Marks, M. Mitzenmacher, and G. T. Schafer, "The hugs platform: A toolkit for interactive optimization," in *Proceedings of the working conference on advanced visual interfaces*, pp. 324–330, ACM, 2002.

[46] D. Meignan, S. Knust, J.-M. Frayret, G. Pesant, and N. Gaud, "A review and taxonomy of interactive optimization methods in operations research," *ACM Transactions on Interactive Intelligent Systems (TiiS)*, vol. 5, no. 3, p. 17, 2015.

[47] A. Sears, "Aide: A step toward metric-based interface development tools," in *Proceedings of the 8th annual ACM symposium on User interface and software technology*, pp. 101–110, ACM, 1995.

[48] B. Myers, S. E. Hudson, and R. Pausch, "Past, present, and future of user interface software tools," *ACM Transactions on Computer-Human Interaction (TOCHI)*, vol. 7, no. 1, pp. 3–28, 2000.

[49] A. Swearngin, A. J. Ko, and J. Fogarty, "Scout: Mixed-initiative exploration of design variations through high-level design constraints," in *The 31st Annual ACM Symposium on User Interface Software and Technology Adjunct Proceedings*, UIST '18 Adjunct, (New York, NY, USA), pp. 134–136, ACM, 2018.

[50] M. R. Frank and J. D. Foley, "Model-based user interface design by example and by interview," in *Proceedings of the 6th Annual ACM Symposium on User Interface Software and Technology*, UIST '93, (New York, NY, USA), pp. 129–137, ACM, 1993.

[51] B. Lee, S. Srivastava, R. Kumar, R. Brafman, and S. R. Klemmer, "Designing with interactive example galleries," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 2257–2266, ACM, 2010.

[52] P. O'Donovan, A. Agarwala, and A. Hertzmann, "Designscape: Design with interactive layout suggestions," in *Proceedings of the 33rd annual ACM conference on human factors in computing systems*, pp. 1221–1224, ACM, 2015.

[53] K. Todi, D. Weir, and A. Oulasvirta, "Sketchplore: Sketch and explore with a layout optimiser," in *Proceedings of the 2016 ACM Conference on Designing Interactive Systems*, pp. 543–555, ACM, 2016.

[54] G. Bailly, A. Oulasvirta, T. Kötzing, and S. Hoppe, "Menuoptimizer: Interactive optimization of menu systems," in *Proceedings of the 26th annual ACM symposium on User interface software and technology*, pp. 331–342, ACM, 2013.

[55] N. D. Ramesh, K. Todi, and A. Oulasvirta, "Interactive grid layout design with integer programming," in *submitted for review.*

[56] DGLFLF, "Rapport au Parlement sur l'emploi de la langue française." Government Report, 2015. From the Délégation générale à la langue française et aux langues de France of the Ministère de la Culture et de la Communication. In French.

[57] DGLFLF, "Vers une norme française pour les claviers informatiques." Government Publication, 2016. From the Délégation générale à la langue française et aux langues de France of the Ministère de la Culture et de la Communication. In French.

[58] "ISO/IEC 9995-1:2009 Information technology – Keyboard layouts for text and office systems – Part 1: General principles governing keyboard layouts," standard, International Organization for Standardization, Geneva, CH, Oct. 2009.

[59] AFNOR, "Interfaces utilisateurs - Dispositions de clavier bureautique français, NF Z71-300 Avril 2019. La Plaine Saint-Denis: AFNOR, Version de 2019-04-P, 85 p.."

[60] AFNOR, "User interfaces - French keyboard layouts for office, NF Z71-300 Avril 2019. La Plaine Saint-Denis: AFNOR, Version de 2019-04-P, 85 p.."

[61] A. Yassi, "Repetitive strain injuries," *The Lancet*, vol. 349, pp. 943–947, mar 1997.

[62] J. P. P. Jokinen, S. Sarcar, A. Oulasvirta, C. Silpasuwanchai, Z. Wang, and X. Ren, "Modelling Learning of New Keyboard Layouts," in *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems - CHI '17*, (New York, New York, USA), pp. 4203–4215, ACM Press, 2017.

[63] P. U.-J. Lee and S. Zhai, "Top-down learning strategies: can they facilitate stylus keyboard learning?," *International Journal of Human-Computer Studies*, vol. 60, pp. 585–598, may 2004.

[64] M. Pollatschek, M. Gershoni, and Y. Tadday, "Improving the hebrew typewriter," *Report Technion Haifa*, 1975.

[65] A. Karrenbauer and A. Oulasvirta, "Improvements to keyboard optimization with integer programming," in *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology*, UIST '14, (New York, NY, USA), pp. 621–626, ACM, 2014.

[66] A. M. Feit, M. Nancel, D. Weir, G. Bailly, M. John, A. Karrenbauer, and A. Oulasvirta, "Élaboration de la disposition AZERTY modernisée," tech. rep., June 2018.

[67] J. Simonsen and T. Robertson, *Routledge international handbook of participatory design*. Routledge, 2012.

[68] D. Bertsimas and J. Tsitsiklis, *Introduction to Linear Optimization*. Athena Scientific, 1st ed., 1997.

[69] D. Birkhoff, "Tres observaciones sobre el algebra lineal," *Universidad Nacional de Tucuman Revista , Serie A*, vol. 5, pp. 147–151, 1946.

[70] H. W. Kuhn, "The hungarian method for the assignment problem," *Naval Research Logistics Quarterly*, vol. 2, pp. 83–97, 1955.

[71] R. Duan and H.-H. Su, "A scaling algorithm for maximum weight matching in bipartite graphs," in *Proceedings of the Twenty-third Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '12, pp. 1413–1424, SIAM, 2012.

[72] T. Koopmans and M. J. Beckmann, "Assignment Problems and the Location of Economic Activities," Cowles Foundation Discussion Papers 4, Cowles Foundation for Research in Economics, Yale University, 1955.

[73] C. Nugent, T. Vollman, and J. Ruml, "An Experimental Comparison of Techniques for the Assignment of Facilities to Locations," *Operations Research*, vol. 16, no. 1, pp. 150–173, 1968.

[74] R. E. Burkard, E. Çela, P. M. Pardalos, and L. S. Pitsoulis, *The Quadratic Assignment Problem*, pp. 1713–1809. Boston, MA: Springer US, 1998.

[75] L. Steinberg, "The Backboard Wiring Problem: A Placement Algorithm," *SIAM Review*, vol. 3, no. 1, pp. 37–50, 1961.

[76] J. Krarup and P. M. Pruzan, *Mathematical Programming in Use*, ch. Computer-aided layout design, pp. 75–94. Berlin, Heidelberg: Springer Berlin Heidelberg, 1978.

[77] A. N. Elshafei, "Hospital layout as a quadratic assignment problem," *Operational Research Quarterly (1970-1977)*, vol. 28, no. 1, pp. 167–179, 1977.

[78] M. Queyranne, "Performance Ratio of Polynomial Heuristics for Triangle Inequality Quadratic Assignment Problems," *Operations Research Letters*, vol. 4, no. 5, pp. 231 – 234, 1986.

[79] R. E. Burkard, S. E. Karisch, and F. Rendl, "Qaplib - a quadratic assignment problemlibrary," *J. of Global Optimization*, vol. 10, pp. 391–403, June 1997.

[80] Z. Drezner, "Finding a cluster of points and the grey pattern quadratic assignment problem," *OR Spectrum*, vol. 28, pp. 417–436, 07 2006.

[81] K. Anstreicher, N. Brixius, J.-P. Goux, and J. Linderoth, "Solving large quadratic assignment problems on computational grids," *Mathematical Programming*, vol. 91, no. 3, pp. 563–588, 2014.

[82] N. Christofides, "Worst-case analysis of a new heuristic for the travelling salesman problem," tech. rep., DTIC Document, 1976.

[83] M. K. Anstreicher, "Recent advances in the solution of quadratic assignment problems," *Mathematical Programming*, vol. 97, no. 1, pp. 27–42, 2003.

[84] U. Feige and J. R. Lee, "An improved approximation ratio for the minimum linear arrangement problem," *Inf. Process. Lett.*, vol. 101, no. 1, pp. 26–29, 2007.

[85] R. Hassin, A. Levin, and M. Sviridenko, "Approximating the minimum quadratic assignment problems," *ACM Trans. Algorithms*, vol. 6, pp. 18:1–18:10, Dec. 2009.

[86] E. M. Arkin, R. Hassin, and M. Sviridenko, "Approximating the maximum quadratic assignment problem," *Information Processing Letters*, vol. 77, no. 1, pp. 13 – 16, 2001.

[87] V. Nagarajan and M. Sviridenko, "On the maximum quadratic assignment problem," *Mathematics of Operations Research*, vol. 34, no. 4, pp. 859–868, 2009.

[88] Q. Zhao, S. E. Karisch, F. Rendl, and H. Wolkowicz, "Semidefinite Programming Relaxations for the Quadratic Assignment Problem," *Journal of Combinatorial Optimization*, vol. 2, no. 1, pp. 71–109, 1998.

[89] J. Povh and F. Rendl, "Copositive and Semidefinite Relaxations of the Quadratic Assignment Problem," *Discret. Optim.*, vol. 6, pp. 231–241, Aug. 2009.

[90] P. C. Gilmore, "Optimal and Suboptimal Algorithms for the Quadratic Assignment Problem," *SIAM J. Appl. Math.*, vol. 10, pp. 305–313, 1962.

[91] Y. Li, P. M. Pardalos, K. G. Ramakrishnan, and M. G. C. Resende, "Lower bounds for the quadratic assignment problem," *Annals of Operations Research*, vol. 50, no. 1, pp. 387–410, 1994.

[92] L. Kaufman and F. Broeckx, "An Algorithm for the Quadratic Assignment Problem Using Benders' Decomposition," *European Journal of Operational Research*, vol. 2, no. 3, pp. 207 – 211, 1978.

[93] Y. Xia and Y.-X. Yuan, "A new Linearization Method for Quadratic Assignment Problems," *Optimization Methods and Software*, vol. 21, no. 5, pp. 805–818, 2006.

[94] H. Zhang, C. Beltran-Royo, and L. Ma, "Solving the Quadratic Assignment Problem by Means of General Purpose Mixed Integer Linear Programming Solvers," *Annals OR*, vol. 207, pp. 261–278, 2013.

[95] A. Frieze and J. Yadegar, "On the quadratic assignment problem," *Discrete Applied Mathematics*, vol. 5, no. 1, pp. 89 – 98, 1983.

[96] H. D. Sherali and W. P. Adams, "A Hierarchy of Relaxations and Convex Hull Characterizations for Mixed-Integer Zero—One Programming Problems," *Discrete Applied Mathematics*, vol. 52, no. 1, pp. 83 – 106, 1994.

[97] H. D. Sherali and W. P. Adams, *Handbook of Combinatorial Optimization*, ch. Reformulation–Linearization Techniques for Discrete Optimization Problems, pp. 2849–2896. New York, NY: Springer New York, 2013.

[98] J. Peng, H. Mittelmann, and X. Li, "A new Relaxation Framework for Quadratic Assignment Problems Based on Matrix Splitting," *Mathematical Programming Computation*, vol. 2, no. 1, pp. 59–77, 2010.

[99] L. Gurobi Optimization, "Gurobi Optimizer Version 8.1," 2019.

[100] M. John and A. Karrenbauer, *A Novel SDP Relaxation for the Quadratic Assignment Problem Using Cut Pseudo Bases*, pp. 414–425. Cham: Springer International Publishing, 2016.

[101] L. A. Wolsey, *Integer programming*. Wiley-Interscience series in discrete mathematics and optimization, New York (N.Y.), Chichester, Weinheim: J. Wiley & sons, 1998. A Wiley-Interscience publication.

[102] C. Helmberg and F. Rendl, "A spectral bundle method for semidefinite programming," *SIAM Journal on Optimization*, vol. 10, no. 3, pp. 673–696, 2000.

[103] M. V. Ramana, L. Tunçel, and H. Wolkowicz, "Strong duality for semidefinite programming," vol. 7, pp. 641–662, Aug. 1997.

[104] MOSEK ApS, *The MOSEK C optimizer API manual Version 7.1 (Revision 52)*, 2016.

[105] S. Bund-Becker, "Creating a markup language for mixed integer linear programs," 2019, Bachelor's thesis, Saarland University.

[106] A. Mitsos, I. N. Melas, P. Siminelakis, A. D. Chairakaki, J. Saez-Rodriguez, and L. G. Alexopoulos, "Identifying drug effects via pathway alterations using an integer linear programming optimization formulation on phosphoproteomic data," *PLOS Computational Biology*, vol. 5, pp. 1–11, 12 2009.

[107] D. Schuller, A. Miede, J. Eckert, U. Lampe, A. Papageorgiou, and R. Steinmetz, "Qos-based optimization of service compositions for complex workflows," in *International Conference on Service-Oriented Computing*, pp. 641–648, Springer, 2010.

[108] Y. Y. Haimes, W. A. Hall, and H. T. Freedman, *Multiobjective optimization in water resources systems: the surrogate worth trade-off method*, vol. 3. Elsevier, 2011.

[109] E. Althaus, G. Klau, O. Kohlbacher, H.-P. Lenhof, and R. Knut, "Integer linear programming in computational biology," vol. 5760, pp. 199–218, 01 2009.

[110] A. Karrenbauer and D. Wöll, "Blinking molecule tracking," *CoRR*, vol. abs/1212.5877, 2012.

[111] E. Althaus, S. Canzar, M. R. Emmett, A. Karrenbauer, A. G. Marshall, A. Meyer-Baese, and H. Zhang, "Computing h/d-exchange speeds of single residues fromdata of peptic fragments," in *The 2008 ACM symposium on Applied computing-SAC'08*, pp. 1273–1277, 2008.

[112] R. Lougee, "The coin-or initiative: Open-source software accelerates operations research progress," *ORMS Today*, vol. 28, pp. 20–22, 01 2001.

[113] R. Fourer, D. M. Gay, and B. Kernighan, "Algorithms and model formulations in mathematical programming," ch. AMPL: A Mathematical Programming Language, pp. 150–151, Berlin, Heidelberg: Springer-Verlag, 1989.

[114] G. D. Corporation, "General algebraic modeling system (gams) release 27.1.0," 2019.

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

[116] S. O'Hara, P. Aas, S. Dixit, B. Lawson, X. Wu, T. Eden, and S. Moon, "HTML 5.3," W3C working draft, W3C, Oct. 2018. https://www.w3.org/TR/2018/WD-html53-20181018/.