CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

# A Constraint Programming Approach to Finding Stable Matchings within Airline Manpower Planning

Master's thesis in Computer Science—Algorithms, Languages & Logic

JAKOB JARMAR

FABIAN SÖRENSSON

Department of Computer Science & Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2017

Master's thesis

# A Constraint Programming Approach to Finding Stable Matchings within Airline Manpower Planning

Jakob Jarmar

Fabian Sörensson

Department of Computer Science & Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2017

A Constraint Programming Approach to Finding Stable Matchings within Airline Manpower Planning

JAKOB JARMAR
FABIAN SÖRENSSON

Technical advisor: Björn Thalén, Jeppesen Systems AB
Supervisor: Birgit Grohe, Department of Computer Science & Engineering
Examiner: Richard Johansson, Department of Computer Science & Engineering

Department of Computer Science & Engineering
Chalmers University of Technology
University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2017

A Constraint Programming Approach to Finding Stable Matchings within Airline Manpower Planning

JAKOB JARMAR
FABIAN SÖRENSSON
Department of Computer Science & Engineering
Chalmers University of Technology
University of Gothenburg

## Abstract

The objective of airline manpower planning is to have the right number of pilots with the right qualifications at the right time. To accomplish this, one has to solve the subproblem of assigning pilots to promotion courses such that pilots' seniority ranks and preferences are taken into account: no senior pilot should be able to find a junior pilot with a promotion that the senior pilot would have preferred. We call this subproblem the airline promotion assignment problem (APA). The objective of this thesis is to develop an efficient model for APA.

We show how APA can be modelled as a *stable matching* problem, and more specifically how it can be formulated as an instance of the hospitals/residents problem with ties and forbidden pairs. A constraint satisfaction problem model for APA is presented, which we have implemented in a constraint programming system. We also present a model for an extension to APA, which we call the airline promotion assignment problem with detailed preferences (APA-D), and which involves additional rules used within a specific airline. We show results from running our constraint programming implementation on different types of test data derived from real airline data. The thesis is concluded with a discussion of our work and some remarks on how the problem could be modelled and solved differently.

Keywords: stable matching, hospitals/residents problem, stable marriage problem, constraint satisfaction, constraint programming, airlines, manpower planning

# Acknowledgements

During the course of this project, Jeppesen has kindly provided us with an office and the resources needed to make this thesis possible, which we are grateful for. We thank our supervisor at Jeppesen, Björn Thalén, who has continuously given us valuable feedback and help. We would also like to thank Staffan Nilsson, Mattias Grönkvist and Kristoffer Ptasinski at Jeppesen for their help, along with everyone else at the Jeppesen office.

We thank our supervisor at Chalmers, Birgit Grohe, for the helpful discussions and all the great feedback on our writing. Also at Chalmers, we would like to thank our examiner Richard Johansson, and our fellow students Daniel Toom, David Grankvist and Andreas Hallberg.

Finally, we would like to thank everyone in Chalmersspexet Bob, for all the joy and hard work we have shared, and for bringing us together to do this thesis.

Jakob Jarmar, Gothenburg, March 2017
Fabian Sörensson, Gothenburg, March 2017

# Contents

# List of Figures

# List of Tables

# Glossary

**APA** airline promotion assignment problem. v, ix, xi, xiii, 1, 2, 3, 19, 20, 22, 23, 25, 27, 28, 29, 30, 31, 32, 33, 34, 43, 44, 45, 46

**APA-D** airline promotion assignment problem with detailed preferences. v, ix, xi, xiii, 2, 19, 22, 23, 25, 30, 31, 32, 33, 34, 44, 45

**CP** constraint programming. v, 1, 3, 8, 9, 11, 14, 16, 32, 44, 45

**CSP** constraint satisfaction problem. v, 1, 2, 3, 8, 9, 11, 13, 14, 16, 25, 27, 28, 29, 30, 31, 32, 44, 45

**HR** hospitals/residents problem. xi, 3, 4, 5, 6, 7, 8, 14, 20, 25, 27, 43, 44, 45, 46

**HRTF** hospitals/residents problem with ties and forbidden pairs. v, 1, 2, 19, 20, 23, 44

**SM** stable marriage problem. 3, 4, 5, 6, 7, 8, 43, 44, 45

# 1 Introduction

The airline industry, with its large operational scale, multiple cost factors such as fuel and personnel, and many planning problems, is an interesting context for applying optimization methods. As such, the industry has attracted much attention from the operations research community [63]. In 2015, SAS employed over 11 000 people, had more than 800 departures daily and over 150 aircraft in service [50]. With SAS just being a mid-sized airline, it is easy to see how crucial optimization can be to stay competitive.

One planning problem in the airline industry relates to *manpower planning*: the airline manpower problem involves predicting the supply and demand of pilots and then closing the gap between the two [24]. This is achieved by planning for example hiring, transitioning, vacation times and training times. The airline manpower problem has been the subject of some prior research. Yu, Dugan, and Argüello [64] have written one of the first papers to thoroughly analyze the problem, and they present a decision support system which uses specialized heuristics to help find optimized solutions. Other solution methods, such as integer programming [46, 65, 54] and stochastic algorithms [54], have been proposed and utilized. Different methods are typically used for different subproblems in the larger planning problem.

Central to airline manpower planning is a promotion bidding system often employed by airlines: pilots are allowed to list promotions by preference, and who is assigned which promotion is primarily based on pilot seniority [46]. Due to union rules, a senior pilot must never be able to find a junior pilot with a promotion that the senior pilot would have preferred [56]. In order to get a promotion, a pilot must take a training course. We call the subproblem of assigning pilots to promotion training courses according to their bid preferences and seniority rankings the airline promotion assignment problem (APA), which is the focus of this thesis.

The objective of this thesis is to develop an efficient model for APA. We approach APA by showing how the problem can be naturally described as a *stable matching* problem: more specifically as an instance of the hospitals/residents problem with ties and forbidden pairs (HRTF). We are to our knowledge the first to make this connection between airline manpower planning and stable matching problems. By mapping APA to HRTF, we have been able to utilize the existing body of research in stable matching problems. Mainly taking inspiration from [44], we formulate APA as a constraint satisfaction problem (CSP), and use constraint programming (CP) to solve it.

## 1.1 Jeppesen

This master's thesis has been conducted at Jeppesen, which has provided us with this problem, a supervisor and a workplace, as well as necessary software and hardware. Jeppesen is a subsidiary of The Boeing Company, and specializes in aviation information systems, with customers including Delta Airlines, Lufthansa, Qatar Airways and SAS. Their Gothenburg office, with around 300 employees, focuses on planning, scheduling and optimization software.

Multiple master's theses dealing with pilot transitioning have been conducted at Jeppesen prior to this thesis. Holm's thesis from 2008 deals with a number of subproblems within airline manpower planning, and includes mathematical models for the different problems, one of which is the problem of planning training and vacation for pilots [24]. This problem includes ensuring that the demands for different positions are met, and thus planning for when pilots should take courses in order to be promoted. Holm used a mixed integer programming solver to solve the problem. Her model, however, does not deal with *whether* pilots should transition, only *when* they should do so. The theses of Thalén [56] and Morén [41] deal with the larger staffing and transitioning problem, and their solutions use tabu search and branch and bound methods respectively.

Jeppesen has a commercially available solver under continuous improvement for the airline manpower problem, which uses a combination of heuristics and mathematical programming to find good plans. The objective of this thesis is to develop a model for APA, and since APA is a subproblem of the larger airline manpower problem, such a model could be effectively used as a subroutine within a larger manpower planner.

## 1.2 Outline

In Chapter 2, we explain the necessary background in stable matching problems and constraint programming needed to understand how we model and solve APA. In Chapter 3, a comprehensive description of APA is presented, together with a mapping of the problem to an instance of HRTF. Also explained is an extension to APA which we call APA-D, and which comprises additional rules for a specific airline. Chapter 4 contains our constraint satisfaction problem (CSP) models for APA and APA-D, with proofs of correctness, and a short description of the model software we have implemented, including a pre-processing step. In Chapter 5, we present test data derived from some different airlines, and performance results for our model on these data sets. Finally, the thesis is concluded in Chapter 6 with a discussion of our solution methods and results, and some suggestions for possible future work.

# 2
# Background

This chapter describes the necessary theoretical background to understand how we model and solve the airline promotion assignment problem (APA). In Section 2.1, we present variations of stable matching problems, including solution methods. We model APA as a constraint satisfaction problem (CSP) and solve it using constraint programming (CP); thus, in Section 2.2, we describe how CSPs can be modelled and what solution methods are typically used by CP solvers.

## 2.1 Stable matchings

In this section, we present the stable marriage problem (SM) as well as the hospitals/residents problem (HR), and variations of these problems that are relevant to this thesis. Also presented are solution methods for the computationally harder variants of SM and HR.

In their seminal paper from 1962, Gale and Shapley present two stable matching problems: the well-known and well-studied SM, and the more general HR (under the name college admissions problem), along with polynomial-time algorithms to solve both [16]. In short, HR considers a set of hospitals, each with a maximal quota of residents (medicine student interns), and a set of applicants seeking residency at the hospitals. Hospitals rank applicants by preference, omitting only those they would not accept under any circumstances, and applicants similarly rank acceptable hospitals by preference. The goal is to assign residents to hospitals in such a manner that the assignment is *stable*:

**Definitions**

A matching of residents to hospitals is **stable** if there is no resident $r$ and hospital $H$, such that:

- *either* $r$ is unassigned and finds $H$ acceptable
- *or* $r$ is assigned to another hospital but prefers $H$

*and*

- *either* $H$ still has unfilled positions and finds $r$ acceptable
- *or* $H$ prefers $r$ to its least preferred assigned applicant.

If such a pair $\langle r, H\rangle$ exists, we call it a **blocking pair**.

$$\begin{array}{c||ccc} r_1 & H_1 & H_2 & \\ r_2 & H_2 & H_1 & \\ r_3 & H_1 & H_2 & \\ & & & \\ H_1 & r_1 & r_2 & r_3 \\ H_2 & r_1 & r_3 & r_2 \end{array}$$

$$\text{quota}(H_1) = 2, \quad \text{quota}(H_2) = 1$$

Example instance of HR. On the left hand side are the residents and the hospitals, on the right hand side are their respective preference lists. Hospital quotas are also given. All preference lists are complete in this example, although this is not a requirement in HR.

$$\begin{array}{c||ccc} r_1 & \underline{H_1} & H_2 & \\ r_2 & \underline{H_2} & H_1 & \\ r_3 & \underline{H_1} & H_2 & \\ & & & \\ H_1 & \underline{r_1} & r_2 & \underline{r_3} \\ H_2 & r_1 & r_3 & \underline{r_2} \end{array}$$

**(a)** Stable assignment.

$$\begin{array}{c||ccc} r_1 & \underline{H_1} & H_2 & \\ r_2 & H_2 & \underline{H_1} & \\ r_3 & H_1 & \underline{H_2} & \\ & & & \\ H_1 & \underline{r_1} & \underline{r_2} & r_3 \\ H_2 & r_1 & \underline{r_3} & r_2 \end{array}$$

**(b)** Stable assignment.

$$\begin{array}{c||ccc} r_1 & \enclose{circle}{H_1} & \underline{H_2} & \\ r_2 & H_2 & \underline{H_1} & \\ r_3 & \underline{H_1} & H_2 & \\ & & & \\ H_1 & \enclose{circle}{r_1} & \underline{r_2} & \underline{r_3} \\ H_2 & \underline{r_1} & r_3 & r_2 \end{array}$$

**(c)** Unstable assignment.

**Figure 2.1:** Example instance of the hospitals/residents problem (HR), along with all possible complete assignments. Assignment (a) and (b) constitute legal solutions; however, assignment (c) contains the blocking pair $\langle r_1, H_1 \rangle$ and is thus unstable.

Figure 2.1 shows an example of a HR instance and all its possible complete assignments: two stable assignments (a) and (b) and an unstable assignment (c). Assignment (a) is optimal for the residents: they are assigned as good hospitals as they can get, which in this case are their most preferred hospitals. It is easy to see that this assignment is stable, since all residents get their most preferred option. Assignment (b) is optimal for the hospitals: they are assigned as good residents as they can get. Hospital $H_2$ did not get its most preferred resident $r_1$; no legal solution would allow this assignment, since $r_1$ and $H_1$ top each other's preference list. In assignment (c), $r_1$ and $H_2$ are assigned to each other: this makes $\langle r_1, H_1 \rangle$ a blocking pair, since $r_1$ prefers $H_1$ over $H_2$ and $H_1$ prefers $r_1$ to its least preferred assigned resident $r_3$. Thus, assignment (c) is not stable.

The stable marriage problem (SM), while typically formulated in terms of *men* and *women*, is the special-case where all hospital quotas are equal to 1, and where all preference lists are complete [16]. The variation of SM where preference lists may be incomplete is known as the stable marriage problem with incomplete lists [17, 39]. It is worth noting that any instance of HR can be formulated as an instance of SM with incomplete lists using a technique called *cloning* [42].

HR was studied further by Roth in a paper from 1984 [48]. He analyzed the method used in the National Resident Matching Program, which matches American residents to hospitals, and showed that the algorithm used was equivalent to the one provided by Gale and Shapley for HR—despite being developed more than 10 years prior to the publication of Gale's and Shapley's paper. This is when the problem got the name hospitals/residents problem. Later, in 2012, Gale and Roth received *The Sveriges Riksbank Prize in Economic Sciences in Memory of Alfred Nobel* for their work on stable matchings [7].

$$\begin{array}{r||lll}
r_1 & (H_1 & H_2) & H_3 \\
r_2 & H_1 & (H_2 & H_3) \\
r_3 & H_3 & & \\
 & & & \\
H_1 & r_1 & r_2 & \\
H_2 & r_2 & r_1 & \\
H_3 & r_1 & (r_2 & r_3)
\end{array}$$

$$\text{quota}(H_1) = 1, \quad \text{quota}(H_2) = 1 \quad \text{quota}(H_3) = 1$$

Example instance of HR with ties. See Figure 2.1 for a syntax explanation. Hospitals and residents within parentheses in the preference lists are tied.

$$\begin{array}{r||lll}
r_1 & (\underline{H_1} & H_2) & H_3 \\
r_2 & H_1 & (\underline{H_2} & H_3) \\
r_3 & \underline{H_3} & & \\
 & & & \\
H_1 & \underline{r_1} & r_2 & \\
H_2 & \underline{r_2} & r_1 & \\
H_3 & r_1 & (r_2 & \underline{r_3})
\end{array}$$

**(a)** Stable, maximum assignment.

$$\begin{array}{r||lll}
r_1 & (H_1 & \underline{H_2}) & H_3 \\
r_2 & \underline{H_1} & (H_2 & H_3) \\
r_3 & \underline{H_3} & & \\
 & & & \\
H_1 & r_1 & \underline{r_2} & \\
H_2 & r_2 & \underline{r_1} & \\
H_3 & r_1 & (r_2 & \underline{r_3})
\end{array}$$

**(b)** Stable, maximum assignment.

$$\begin{array}{r||lll}
r_1 & (\underline{H_1} & H_2) & H_3 \\
r_2 & H_1 & (H_2 & \underline{H_3}) \\
r_3 & H_3 & & \\
 & & & \\
H_1 & \underline{r_1} & r_2 & \\
H_2 & r_2 & r_1 & \\
H_3 & r_1 & (\underline{r_2} & r_3)
\end{array}$$

**(c)** Stable assignment.

**Figure 2.2:** Example instance of HR with ties, and some possible assignments. All given assignments are stable. However, assignment (c) is smaller than (a) and (b). With ties in preference lists, solutions may be of varying sizes.

### 2.1.1 Variations on stable matching problems

Further variations of SM and HR include SM with ties and incomplete lists [30] and HR with ties [29]. These variations do not require preferences to be strictly ordered: ties are allowed in the preference lists. SM with ties (and complete lists) has also been studied, and a polynomial time algorithm for it is presented in [18]; however, when both ties and incomplete lists are allowed, the problems become harder: stable matchings may be of varying sizes, and it has been proven that the problem of finding a minimum or maximum cardinality solution for SM with ties and incomplete lists or HR with ties is NP-hard [30, 29]. This is true even for restricted variants, such as when ties are one-sided [39].

With ties in preference lists, one can have different definitions of stability. *Weak* stability is similar to the previously defined notion of stability, and requires both parties in a blocking pair to strictly prefer each other to their assignments. *Strong* stability requires only one party to strictly prefer the other, while the second party may be indifferent to the first party compared to its assignment. *Super-strong* stability allows blocking pairs to form even when both parties are indifferent to each other compared to their assignments (*"the grass is always greener on the other side..."*) [27]. For this thesis, we are only interested in weak stability, and will hence refer to it simply as *stability* in the remainder of the report.

An example instance of HR with ties together with some solution assignments to the example are given in Figure 2.2. Both assignment (a) and (b) are maximum cardinality solutions, since no larger assignments are possible. Note however that assignment (c) lacks assignments for resident $r_3$ and hospital $H_2$; still, the assignment is stable, since there are no blocking pairs. Ties and incomplete lists make this a possibility: had $r_2$ not been indifferent to $H_2$ and $H_3$, or if $r_3$ had considered hospital $H_2$, then assignment (c) would not have been possible.

Dias et al. [13] introduced the concepts of *forced* and *forbidden* pairs to SM. In these variants, some pairs of men and women either must be in the final solution, or cannot be in the final solution—but the solution must still be stable with regard to these pairs. In other words, the participants of a forced couple may still be part of other blocking pairs, and a forbidden couple may also form a blocking pair. For this thesis, the concept of forbidden pairs is of special interest.

An example instance of HR with forbidden pairs, and its possible assignments, is given in Figure 2.3. The resident-optimal assignment (a) is stable and is therefore a legal solution; however, assignment (b) is not stable, since $\langle r_1, H_2 \rangle$ makes a blocking pair. According to preference lists, the two would prefer to be assigned to each other, but because the pair is also forbidden, the assignment is illegal.

### 2.1.2 Solution methods for stable matching problems

For the basic versions of SM and HR, Gale and Shapley provide a polynomial-time algorithm, which produces an assignment of residents to hospitals that is not only stable, but also *optimal* for the residents: every resident is at least as well off as under any other stable assignment. The Gale-Shapley algorithm is a simple greedy algorithm: all unassigned residents apply to their most preferred hospitals, and hospitals temporarily take in their most preferred residents among the applying ones, such that the hospital quotas are not exceeded. If a resident is not assigned to the hospital he/she applies to, that hospital is removed from the resident's preference list. This process is repeated until all residents are assigned to a hospital, or all hospital quotas are filled up. The Gale-Shapley algorithm has a time complexity of $\mathcal{O}(nm)$, where $n$ is the number of men/residents and $m$ is the number of women/hospitals [16].

Simple polynomial-time variations of the Gale-Shapley algorithm have been pre-

| | | |
|---|---|---|
| $r_1$ | $H_1$ | $\cancel{H_2}$ |
| $r_2$ | $H_2$ | $H_1$ |
| | | |
| $H_1$ | $r_2$ | $r_1$ |
| $H_2$ | $\cancel{r_1}$ | $r_2$ |

quota$(H_1) = 1$, quota$(H_2) = 1$, forbidden pairs: $\langle r_1, H_2 \rangle$

Example instance of HR with forbidden pairs. See Figure 2.1 for a syntax explanation. Forbidden pairs are crossed out.

| | | |
|---|---|---|
| $r_1$ | $\underline{H_1}$ | $\cancel{H_2}$ |
| $r_2$ | $\underline{H_2}$ | $H_1$ |
| | | |
| $H_1$ | $r_2$ | $\underline{r_1}$ |
| $H_2$ | $\cancel{r_1}$ | $\underline{r_2}$ |

**(a)** Stable assignment.

| | | |
|---|---|---|
| $r_1$ | $H_1$ | $\boxed{\cancel{H_2}}$ |
| $r_2$ | $H_2$ | $\underline{H_1}$ |
| | | |
| $H_1$ | $\underline{r_2}$ | $r_1$ |
| $H_2$ | $\boxed{\cancel{r_1}}$ | $r_2$ |

**(b)** Unstable assignment.

**Figure 2.3:** Example instance of HR with forbidden pairs, and its possible assignments. Both assignments follow preference lists correctly. However, in the unstable assignment, $\langle r_1, H_2 \rangle$ constitutes a blocking pair, since they would prefer to be assigned to each other over being unassigned. Since $\langle r_1, H_2 \rangle$ is a forbidden pair, they cannot be assigned to each other.

sented for both SM with incomplete lists [17] and SM with ties [18]. When both ties and incomplete lists are allowed, however, the situation is different; due to the NP-hardness of finding maximum cardinality solutions for these problems, another approach must be taken. Below, we list some solution methods for these harder variants of SM and HR.

A number of polynomial-time approximation algorithms have been proposed for SM with ties and incomplete lists. The simplest one, which has been shown to yield a 2-approximation to the maximum cardinality problem, consists of arbitrarily breaking all ties and then using the standard Gale-Shapley algorithm [39]. The best solution today uses a modification of the Gale-Shapley algorithm, and gives a 3/2-approximation for the general maximum cardinality problem [33]. There are however algorithms that yield even better approximations for special instances of the problem. One such example is the 22/15-approximation algorithm for SM with one-sided ties and incomplete lists, which also uses a modification of the Gale-Shapley algorithm, and which is presented in [25].

Specialized heuristics for HR with one-sided ties (on the hospital side) and incomplete lists have also been devised. By employing the Gale-Shapley algorithm, but doing more sophisticated tie-breaking than arbitrary, good results can be achieved. Two examples of such algorithms, and an empirical study of their use on both real and generated data, can be found in [28].

There are many general approaches to solving combinatorial problems. Approaches that have been studied and used for SM with ties and incomplete lists and HR with ties include integer programming [35], SAT [22], and answer set programming [10]. Local search methods have also been employed to solve SM with ties and incomplete lists [19]. A method using adaptive search has been shown to be very effective in finding solutions efficiently [43]. While local search heuristics typically use randomization, the methods used have experimentally been shown to almost always find maximum solutions when used on generated data [19, 43]. The adaptive search method has also been shown to work particularly well with parallelization, and has been extended to solving HR with ties [42].

One of the more well-studied solution methods for different variants of SM and HR is constraint programming (CP). CP allows for intuitive modelling of problems that can be expressed as a set of variables and a set of constraints (see Section 2.2 below), and the stable matching community has shown growing interest in CP solution methods [37]. Various CP models and encodings for different stable matching variants have been proposed, such as for SM with and without incomplete lists [21], SM with ties and incomplete lists [20], HR [38], HR with ties [44] and HR with forced and forbidden pairs [53]. Encodings based on binary constraints as well as specialised constraints have been considered [38, 58].

## 2.2 Constraint programming

Constraint programming (CP) is a declarative programming paradigm where a program is stated as a constraint satisfaction problem (CSP) [1]. CSP is a general type of problem formulated as a set of variables with corresponding domains, and a set of constraints on these variables. A solution to a CSP is an instantiation of the variables such that all constraints are satisfied [47].

A more mathematical definition, adapted from [47], follows.

**Definitions**

A **CSP** is defined by a triple $\mathcal{P} = \langle X, D, C \rangle$, where:

- $X$ is an $n$-tuple of variables $X = \langle x_1, ..., x_n \rangle$,
- $D$ is an $n$-tuple of corresponding domain sets $D = \langle dom(x_1), ..., dom(x_n) \rangle$, such that $x_i \in dom(x_i)$, and
- $C$ is a $t$-tuple of constraints $C = \langle C_1, ..., C_t \rangle$.

Each constraint $C_j$ is a pair $\langle R_{S_j}, S_j \rangle$, where:

- $S_j$ is a *scope*, a set containing a subset of the variables in $X$, and
- $R_{S_j}$ is a relation on the variables in the scope $S_j$.

Thus, $R_{S_j}$ is a subset of the Cartesian product of the domains of the variables in $S_j$. With this formulation, a solution to $\mathcal{P}$ is an $n$-tuple $A = \langle a_1, ..., a_n \rangle$, where $a_i \in dom(x_i)$ and each constraint $C_j \in C$ is satisfied, meaning that relation $R_{S_j}$ holds on the projection of $A$ on $S_j$.

In subsequent formulations of CSPs, we do not define all these sets explicitly, and we write constraints simply as relations, with scopes implicit.

With this broad definition, many tasks can be formulated as CSPs [34]. When formulated in a CP language or framework, the problem can be solved—or deemed unsolvable—by a general CP solver. Thus, in CP, the developer's focus is typically put on problem *modelling*, rather than finding a solution method. A relevant quote:

> Constraint programming represents one of the closest approaches computer science has yet made to the Holy Grail of programming: the user states the problem, the computer solves it. (Freuder, [15])

One of the earliest systems to utilize a CSP formulation is the ground-breaking *Sketchpad* system, presented by Sutherland in [55]. Sketchpad uses constraints to reason about geometric objects, and a specialized constraint solver to ensure the geometrical correctness of images. Since then, CSP has become more formalized, and much has been written about CSP properties such as *local consistency* and the related *constraint propagation* methods [36] (more on this in Section 2.2.2).

In this section, we will describe different aspects of CP. First, we will give a typical example of a CSP. Then, we will describe the general solution methods used by CP solvers, and delve further into modelling concerns. Finally, we will give a quick overview of the systems used by today's CP community.

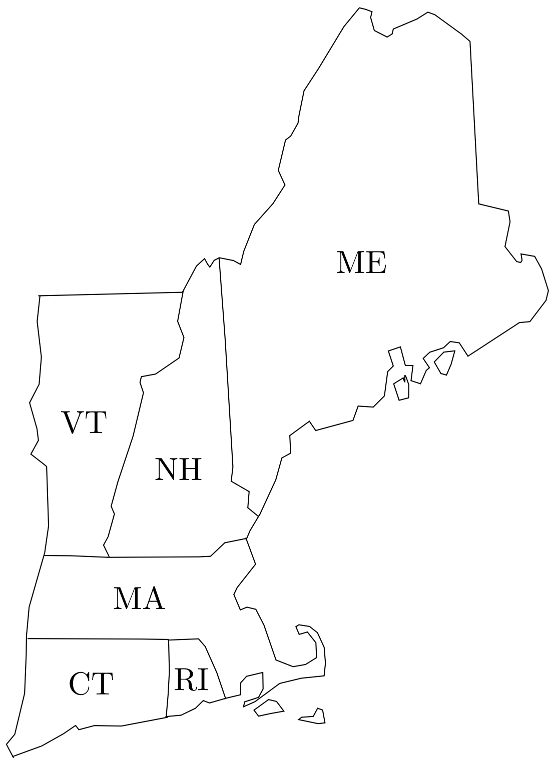
### 2.2.1 Constraint satisfaction problem example

A simple example of a problem that naturally can be formulated as a CSP is the *map-coloring problem.* The map-coloring problem is a well-known combinatorial puzzle, which asks the question: given a map and a finite set of colors, can the regions of the map be colored, such that no two adjacent regions have the same color? More generally, the problem is often called the *graph-coloring problem*, and the general graph-coloring problem is known to be NP-complete [32].

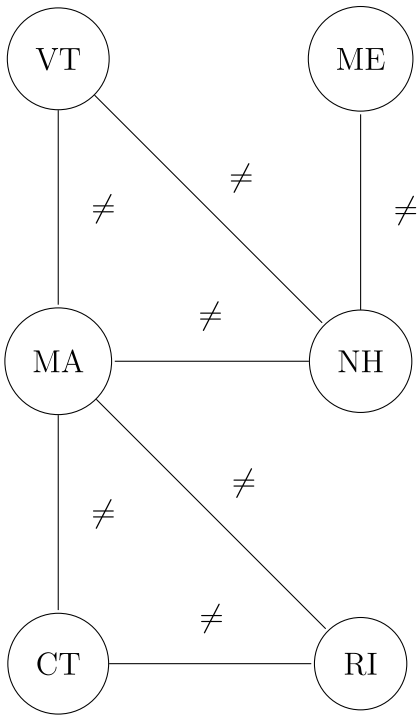
Let's look at an instance of the map-coloring problem. In Figure 2.4a, we see a map of New England, which should be colored with the colors red, green and blue. To make this problem instance more interesting, we have added two additional requirements: Maine should not be colored green, and Massachusetts must be colored red. We formulate this problem instance as a CSP. First, the variables with their common domain:

$$\mathrm{ME}, \mathrm{NH}, \mathrm{VT}, \mathrm{MA}, \mathrm{CT}, \mathrm{RI} \in \{\mathrm{red}, \mathrm{green}, \mathrm{blue}\}$$

In the problem, neighboring regions are not allowed to be of the same color. We



**(a)** A map depicting the states of New England. (ME is Maine, NH is New Hampshire, VT is Vermont, MA is Massachusetts, CT is Connecticut and RI is Rhode Island.)



**(b)** A multigraph representation of the corresponding map-coloring problem. Nodes are variables and edges are binary constraints.

**Figure 2.4:** An instance of the map-coloring problem for the states of New England.

formulate this as a list of inequality constraints:

$$
\begin{aligned}
\text{ME} &\neq \text{NH} \\
\text{NH} &\neq \text{VT} \\
\text{NH} &\neq \text{MA} \\
\text{VT} &\neq \text{MA} \\
\text{MA} &\neq \text{CT} \\
\text{MA} &\neq \text{RI} \\
\text{CT} &\neq \text{RI}
\end{aligned}
$$

Also, as stated above, Maine should not be colored green, and Massachusetts must be colored red:

$$
\begin{aligned}
\text{ME} &\neq \text{green} \\
\text{MA} &= \text{red}
\end{aligned}
$$

From this formulation, a CP solver would be able to find a solution using general techniques. An example of a solution is:

$$\begin{aligned}
\text{ME} &= \text{red}\\
\text{NH} &= \text{green}\\
\text{VT} &= \text{blue}\\
\text{MA} &= \text{red}\\
\text{CT} &= \text{green}\\
\text{RI} &= \text{blue}
\end{aligned}$$

### 2.2.2 Solution techniques

The two most common methods for solving CSPs are *search* and *constraint propagation* [1]. While they can be used by themselves, the most typical approach is to use both methods together. In short, this approach uses constraint propagation to reduce the variable domains by removing values inconsistent with the given constraints. One may vary the amount of propagation performed. If the domains after constraint propagation only contain single values, a solution is found; if any domain contains multiple values, the problem is decomposed into simpler subproblems, effectively creating a search tree. Both search and constraint propagation are explained in the upcoming subsections.

### 2.2.3 Search

The search method usually employed in CP systems is called *backtracking search*. Backtracking search is a form of depth-first search. At each iteration, the domain of a variable is reduced: either the domain is reduced to a single value (i.e. the variable is instantiated to that value) or multiple values remain (typically the domain is split in half). Then, all constraints involving that variable are checked: if any constraint is violated by the domain reduction, the algorithm backtracks and eliminates that domain reduction from the search tree [34]. When all variables' domains are down to a single value, and no constraints are violated, a solution is found. The method is *complete*, meaning it is guaranteed to find a solution if there is one; or if there is no solution, the method can determine that [47]. This is one of the strengths of CP, compared to incomplete heuristics. However, the time complexity for the method is exponential, but by interweaving variable instantiations with constraint propagation, the run-time can be reduced for many problem instances [34].

One problem that may still occur is *thrashing*. Thrashing is when the search keeps failing in different parts of the search tree, but for the same reason. Enforcing local consistency can reduce the amount of thrashing [34], but another important factor is the order in which variables are selected for instantiation/domain reduction, and the order in which values are instantiated/the way the domain is reduced.

*Variable orderings* and *value orderings* can be crucial for solving a problem efficiently [47], and can be selected by the user in many modern CP solvers, such as Gecode [51] and Google or-tools [26]. Examples of variable ordering heuristics include using the input order, assigning variables with the smallest domain first, or

| | | Fixed variables | | | | | | Violated | Backtrack |
|---|---|---|---|---|---|---|---|---|---|
| **Step** | Action | ME | NH | VT | MA | CT | RI | constraint | to step |
| **1** | ME := r | r | | | | | | | |
| **2** | NH := r | r | r | | | | | ME $\neq$ NH | 1 |
| **3** | NH := g | r | g | | | | | | |
| **4** | VT := r | r | g | r | | | | | |
| **5** | MA := r | r | g | r | r | | | VT $\neq$ MA | 4 |
| **6** | MA := g | r | g | r | g | | | MA = r | 4 |
| **7** | MA := b | r | g | r | b | | | MA = r | 3 |
| **8** | VT := g | r | g | g | | | | NH $\neq$ VT | 3 |
| **9** | VT := b | r | g | b | | | | | |
| **10** | MA := r | r | g | b | r | | | | |
| **11** | CT := r | r | g | b | r | r | | MA $\neq$ CT | 10 |
| **12** | CT := g | r | g | b | r | g | | | |
| **13** | RI := r | r | g | b | r | g | r | MA $\neq$ RI | 12 |
| **14** | RI := g | r | g | b | r | g | g | CT $\neq$ RI | 12 |
| **15** | RI := b | r | g | b | r | g | b | | |

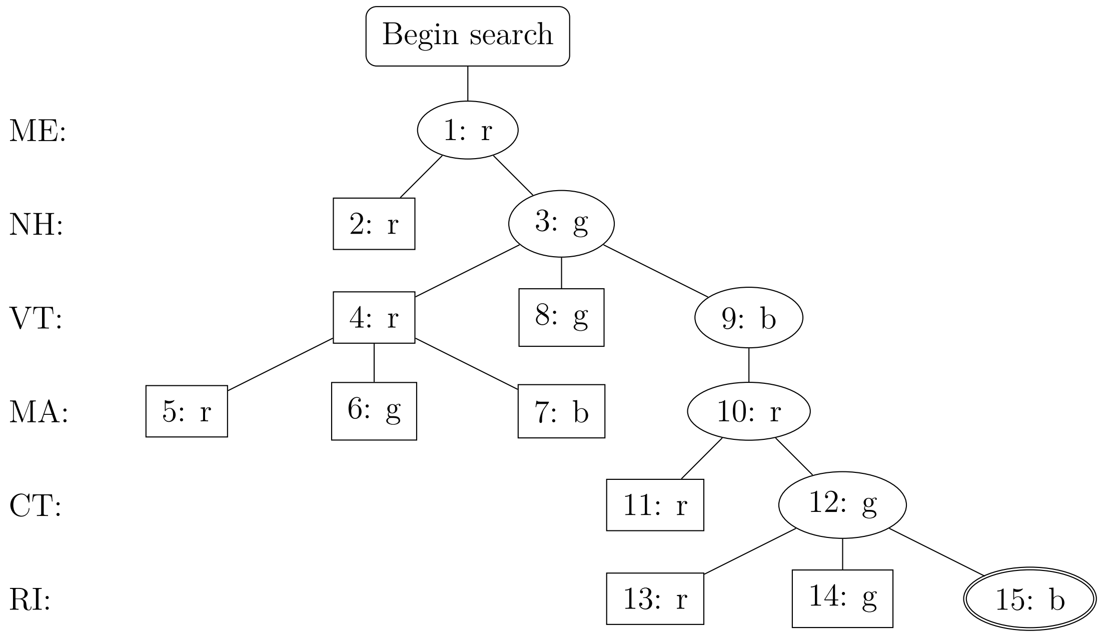


**Figure 2.5:** Backtracking search example without constraint propagation. Note that color values are abbreviated. The levels of the search tree correspond to the variable ordering, and each node indicates which value instantiation was attempted at a given step. The children of each node follow the value ordering $\langle r, g, b\rangle$. Rectangles represent instantiations which were backtracked from due to constraint violations, while ellipses represent successful instantiations.

assigning variables involved in the fewest or most constraints first [47] [52]. Examples of value ordering heuristics include using the input order, assigning the smallest value first (only for integers), or splitting the domain in half and trying the lower values first (also only for integers) [51].

Let's look at an example of using backtracking search, without constraint propagation, in order to solve the map-coloring problem from Section 2.2.1. We use the simple `input order` heuristic as both variable and value ordering, where the variable input order is $\langle \mathrm{ME}, \mathrm{NH}, \mathrm{VT}, \mathrm{MA}, \mathrm{CT}, \mathrm{RI} \rangle$, and the value input order for each variable domain is $\langle \mathrm{red}, \mathrm{green}, \mathrm{blue} \rangle$. The steps performed and the corresponding search tree are shown in Figure 2.5. While this approach works, it can be made much more efficient by performing constraint propagation before each search step, which is shown in an example in the next section.

### 2.2.4 Constraint propagation

Constraint propagation can be described as domain reduction by processing one constraint at a time. By looking at only one constraint and its involved variables, one can view this as solving a small part of the problem without worrying about the rest of the problem, that is, with a local view. The goal of constraint propagation is to achieve some degree of *local consistency*, which will be exemplified below.

Constraints may be categorized by their arity (how many variables they affect). A unary constraint simply restricts the domain of its variable, and a binary constraint defines a relation between two variables. While $n$-ary constraints ($n > 2$) are also well defined, classic CSP literature [40, 36] focuses on unary and binary constraints, since they are easier to reason about, and every $n$-ary constraint may be reduced into a set of binary constraints if auxiliary variables are introduced [49]. This makes it possible to visualize a CSP as a multigraph, where nodes represent variables and arcs represent constraints (possibly multiple between a pair of nodes, if there are multiple constraints involving that pair). An example of such a graph is given in Figure 2.4b, which shows the binary constraints of the example problem in the previous section. From this type of visualization, the terms *node-consistency*, *arc-consistency* and *path-consistency*—all forms of local consistency—were derived [23].

A CSP instance is node-consistent if all values in each variable's domain are legal with regard to all unary constraints. Arc-consistency ensures that each legal value of any variable has a legal match in any other variable's domain, with regard to binary constraints. Path-consistency takes this one step further: for any legal instantiation of two variables there is a legal instantiation of any third variable [11]. When only taking into account unary and binary constraints, node-consistency, arc-consistency and path-consistency may also be called 1-, 2- and 3-consistency, respectively. This reasoning can be generalized to what is called $i$-consistency, meaning that any consistent instantiation of $i - 1$ variables can legally be extended to any other $i$'th variable. When a CSP is $i$-consistent for every $i \leq n$, where $n$ is equal to the number of variables in the problem, the problem is said to be *globally consistent*, and is essentially solved [11].

By using constraint propagation to remove inconsistent values from variable domains, the search space of the subsequently used search algorithm can be reduced.

Higher degrees of consistency remove more values, thus reducing the search space further. However, there is a trade-off between the level of consistency achieved and the computational complexity of the algorithm that achieves it [11]. Algorithms that achieve $i$-consistency typically have a time and space complexity exponential in $i$ [11].

Research in the CSP community has mainly been focused on arc-consistency algorithms, since they yield a good trade-off. A number of algorithms to enforce arc-consistency have been presented and discussed in the CSP literature. The basic `AC-1`, which essentially just uses brute-force to remove inconsistent values, has a time complexity of $\mathcal{O}(enk^2)$, where $e$ is the number of binary constraints, $n$ the number of variables and $k$ the maximum domain size [11]. `AC-3`, an algorithm utilizing a priority queue, has a time complexity of $\mathcal{O}(ek^3)$. `AC-4` has a time complexity of $\mathcal{O}(ek^2)$, but with a worse best-case performance than both `AC-1` and `AC-3` [11]. Many later arc-consistency algorithms are based on either `AC-3` [6, 14] or `AC-4` [3, 5, 2] and these two families of arc-consistency algorithms are often called *coarse-grained* and *fine-grained*, respectively [6].

A modern arc-consistency algorithm, called `AC-2001/3.1`, is presented in [6]. `AC-2001/3.1` is a coarse-grained algorithm with a time-complexity of $\mathcal{O}(ek^2)$, and is the algorithm used for user-added constraints in the CP solver Gecode [52]. A state-of-the-art method for achieving arc-consistency, called `Compact-Table`, is described in [12]; this is the method used in the CP solvers Google or-tools [26] and OsCaR [45].

In addition to letting users define their own constraints, CP solvers provide specialized constraints with corresponding specialized, and oftentimes very efficient, propagation algorithms. These constraints are commonly known as *global constraints*, since they may define relations between an arbitrary number of variables. A commonly used example is the `all_different` constraint, which defines that a set of variables all must be pairwise distinct. These global constraints not only make modelling easier, but often also help make the solving process more efficient [47].

Many CP solvers also make it possible for the user to define their own specialized constraints with corresponding propagation algorithms (called *propagators*). A propagator for a constraint is defined by what should happen when a variable affected by that constraint has its domain reduced. With specialized propagators, it is sometimes possible to achieve stronger and more efficient propagation. Examples of specialized constraints and propagators for HR are given in [38, 58].

Let's look again at the map-coloring instance from Section 2.2.1 as an example of performing constraint propagation. Our aim is to make the problem node- and arc-consistent with regard to all constraints. The steps performed and the variable domains after each step are shown in Figure 2.6.

After these steps, no more propagation can be performed, and the problem is arc-consistent. In order to solve the problem, search must be performed. Between each search step, further propagation might be possible. The process of solving the problem to completion using both search and propagation is illustrated in Figure 2.7. The number of search nodes and backtracking steps are considerably lower than in the example without propagation shown in Figure 2.5, illustrating the need to use propagation for efficient search.

| Step | ME | NH | VT | MA | CT | RI | Propagated constraint |
|---|---|---|---|---|---|---|---|
| | Variable domains | | | | | | |
| **Start** | {r, g, b} | {r, g, b} | {r, g, b} | {r, g, b} | {r, g, b} | {r, g, b} | |
| **1** | {r, b} | {r, g, b} | {r, g, b} | {r, g, b} | {r, g, b} | {r, g, b} | ME $\neq$ g |
| **2** | {r, b} | {r, g, b} | {r, g, b} | {r} | {r, g, b} | {r, g, b} | MA = r |
| **3** | {r, b} | {g, b} | {r, g, b} | {r} | {r, g, b} | {r, g, b} | NH $\neq$ MA |
| **4** | {r, b} | {g, b} | {g, b} | {r} | {r, g, b} | {r, g, b} | VT $\neq$ MA |
| **5** | {r, b} | {g, b} | {g, b} | {r} | {g, b} | {r, g, b} | MA $\neq$ CT |
| **6** | {r, b} | {g, b} | {g, b} | {r} | {g, b} | {g, b} | MA $\neq$ RI |

1. We enforce node-consistency with regard to constraint ME $\neq$ green, thus removing *green* from *dom*(ME).

2. We enforce node-consistency with regard to constraint MA = red, thus removing *green* and *blue* from *dom*(MA).

3 - 6. We enforce arc-consistency with regard to constraints NH $\neq$ MA, VT $\neq$ MA, MA $\neq$ CT, MA $\neq$ RI, thus removing *red* from *dom*(NH), *dom*(VT), *dom*(CT) and *dom*(RI).

**Figure 2.6:** Performing initial constraint propagation on the domains of the map-coloring example. Note that color values are abbreviated.

| Step | Action | ME | NH | VT | MA | CT | RI | Propagated constraints |
|---|---|---|---|---|---|---|---|---|
| | | Variable domains | | | | | | |
| **Start** | | {r, b} | {g, b} | {g, b} | {r} | {g, b} | {g, b} | |
| **1** | ME := r | {r} | {g, b} | {g, b} | {r} | {g, b} | {g, b} | |
| **2** | NH := g | {r} | {g} | {b} | {r} | {g, b} | {g, b} | NH $\neq$ VT |
| **3** | CT := g | {r} | {g} | {b} | {r} | {g} | {b} | CT $\neq$ RI |

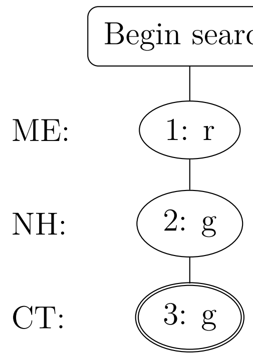


**Figure 2.7:** Backtracking search example with constraint propagation. The same variable and value orderings as in Figure 2.5 are used. The initial propagation is explained in Figure 2.6. After steps 2 and 3, further propagation is performed.

### 2.2.5 Modelling

For a developer interested in solving a combinatorial problem using CP, the main focus is not on the solving techniques used, but how the problem is *modelled* as a CSP.

There may be many different ways to model a problem, and different models may yield the same solution but vary greatly in efficiency. A common concern is deciding what should be represented by variables, and what should be represented by values. In some cases auxiliary variables may be helpful for a good formulation, and constraints can often be described in different ways [1]. Apt notes that choosing a representation "requires proper understanding of the underlying alternatives and occasionally some good insights. Even then it is difficult to draw hard conclusions."[1] He still proposes some rules of thumb, such as using less variables and not using disjunctive constraints. Also recommended is to prefer using global constraints, since their special constraint propagation algorithms typically are more efficient and yield better propagation than general consistency algorithms [1]. However, that there are no hard and fast rules to a good CSP formulation is a shared belief in the research community:

> Sadly, for a method with so many advantages, there is very little we can say about reformulation because there is no fully general technique known. [...] There is a wonderful research opening for the discovery of general techniques [...], moving the area of reformulation from a black art to a science where questions were on tradeoffs and implementation issues, rather than the need for magical insights. (Rossi, Van Beek, and Walsh, [47])

### 2.2.6 Constraint programming systems

There are a number of different CP systems available. The classic approach to CP has been to extend the Prolog programming language, with systems such as CHIP [59], ECL$^i$PS$^e$ [61] and SWI-Prolog [62]. Most systems today, however, interface through a programming language library, which provides facilities for creating variables with respective domains, defining constraints on these variables and finding solutions with a built-in solver. CP systems often feature a number of global constraints, and the possibility to choose which variable and value orderings to use. Many CP systems also allow use of MiniZinc, which is a high-level, solver-independent language for modelling CSPs. MiniZinc compiles into a lower-level language called FlatZinc, which is used as input for the respective CP solver [8].

A competition for CP solvers, called the MiniZinc Challenge, is held yearly [9]. The aim of the challenge is to compare solvers using the same problem set, which is defined in MiniZinc. A number of different CP systems, such as Gecode, Opturion CPX and JaCoP, have performed well in the challenge. In the latest MiniZinc Challenge (2016), Google or-tools won in the `fixed` category, in which the solver must follow a specified search heuristic [9]. Google or-tools is a software suite for combinatorial optimization developed internally at Google, and which includes a CP solver. It is implemented in C++, but with interfaces for a variety of programming languages. Google or-tools is open-source and free, and released under the permissive Apache License 2.0 [26].

# 3 Problem

The objective of this thesis is to develop a model for the airline promotion assignment problem (APA). In this chapter, in Section 3.1, a formal definition of APA is given. In Section 3.2, we present a mapping from APA to a known stable matching problem, the hospitals/residents problem with ties and forbidden pairs (HRTF). Finally, in Section 3.3, we describe an extension to APA, which we call the airline promotion assignment problem with detailed preferences (APA-D).

## 3.1 The airline promotion assignment problem

The airline promotion assignment problem (APA) is concerned with assigning pilots to promotion courses, such that all course *demands* are met, the pilots' *seniority ranks* and *preferences* are respected and all assignments are *legal*. Definitions of important terms follow:

**position** A pilot's *position* within an airline is determined by factors such as the pilot's rank (whether they are a captain or a first officer), qualification (what types of airplane they are allowed to fly) and home base (where they are stationed) [46].

**promotion** A *promotion* is defined as a transfer to a certain position at a certain date or date range.

**course** A *course* lets a pilot get a promotion. Not every pilot is allowed to take every course: courses are sometimes only for pilots transitioning from one certain position to another.

**seniority** *Seniority* is in essence a measure of how long a pilot has worked within an airline. However, it also encompasses other rules regarding precedence. One example is so-called lock-in periods, meaning that a recently promoted pilot cannot be promoted to certain positions for a period of time, but might for example still be eligible for a promotion within the same aircraft type (i.e. from first officer to captain). Thus, seniority is dependent on the combination $\langle$*pilot*, *course*$\rangle$. The seniority rank for each pilot is unique within a course. Not every pilot must have a seniority rank for every course, since not every pilot is eligible for every course.

**preference** Each pilot has a list of all courses they are interested in, ordered by *preference*. This list contains a subset of all courses, and may contain ties.

**legality** Under certain conditions, a pilot may have a course on their preference list, and a seniority rank for that course, but is still not allowed to take that course. We then refer to that assignment as *illegal* (see example below).

**demand** Each course has a *demand* that must be met, that is, the number of pilots that must be assigned to that course.

Due to union rules, when two pilots are interested in the same promotion, the company has to pick the more senior pilot [56]. This gives us an important constraint on our solution, which we will call the **seniority criterion**: for a solution to be valid all senior pilots must be content, meaning that no pilot should be able to find a junior pilot who received a promotion they would have preferred to what they got.

Note that the seniority criterion holds even when a senior pilot $p$ would prefer to be assigned to a course $c$, but the assignment $\langle p, c\rangle$ is illegal: this means that no pilots junior to $p$ may be assigned to course $c$, even if pilot $p$ cannot legally be assigned to $c$. As a motivation for this, consider the case where a junior and a senior pilot are interested in the same position, but the time period under consideration only contains the course relevant for the junior pilot, and not the course relevant for the senior pilot. In this case, and under the seniority criterion, it would be unfair to assign the course to the junior pilot.

A more formal description of the problem parameters follows. The input given consists of:

- a set of pilots $P = \{p_1, \ldots, p_n\}$,
- a set of courses $C = \{c_1, \ldots, c_m\}$,
- for each pilot $p_i \in P$ a preference list containing a subset of $C$, possibly including ties between courses $p_i$ prefers equally,
- for each course $c_j \in C$ a seniority list strictly ranking a subset of $P$,
- for each course $c_j \in C$ an integer $d_j$ indicating its demand, and
- for each possible assignment $\langle p_i, c_j\rangle$ a boolean indicating its legality.

The task is to find a total assignment of pilots to courses such that all assignments are legal, all course demands are met, and the seniority criterion holds. An example of an APA instance, along with a solution, is given in Figure 3.1.

One should note that the rules described are more due to policy, rather than inherent to the theoretical problem. They may therefore be subject to change, making a flexible solution method important. Also, different airlines may have slightly different rules; one such problem variation is described in Section 3.3.

## 3.2 Stable matching model

APA appears natural to describe as a stable matching problem, and there is a straightforward mapping between APA and the hospitals/residents problem with ties and forbidden pairs (HRTF). For the relevant background on stable matching problems, see Section 2.1.

$$
\begin{array}{c||lll}
p_1 & (c_1 & c_2) & c_3 \\
p_2 & \cancel{c_1} & (c_2 & c_3) \\
p_3 & c_1 & c_3 & \\
\\
c_1 & p_1 & \cancel{p_2} & p_3 \\
c_2 & p_2 & p_1 & \\
c_3 & p_1 & p_2 & p_3
\end{array}
$$

$$\text{demand}(c_1) = 1, \quad \text{demand}(c_2) = 1, \quad \text{demand}(c_3) = 1$$

Illegal pairs: $\langle p_2, c_1 \rangle$

An example APA problem instance, with an example solution. Syntax is similar to the HR examples in Section 2.1.

$$
\begin{array}{c||lll}
p_1 & (\underline{c_1} & c_2) & c_3 \\
p_2 & \cancel{c_1} & (\underline{c_2} & c_3) \\
p_3 & c_1 & \underline{c_3} & \\
\\
c_1 & \underline{p_1} & \cancel{p_2} & p_3 \\
c_2 & \underline{p_2} & p_1 & \\
c_3 & p_1 & p_2 & \underline{p_3}
\end{array}
$$

**(a)** Solution example.

$$
\begin{array}{c||lll}
p_1 & (c_1 & \underline{c_2)} & c_3 \\
p_2 & \enclose{circle}{\cancel{c_1}} & (c_2 & c_3) \\
p_3 & \underline{c_1} & c_3 & \\
\\
c_1 & p_1 & \enclose{circle}{\cancel{p_2}} & \underline{p_3} \\
c_2 & p_2 & \underline{p_1} & \\
c_3 & p_1 & p_2 & p_3
\end{array}
$$

**(b)** Invalid assignment.

**Figure 3.1:** An example APA problem instance, with an example solution. Note the similarity to HR examples in Section 2.1. Here, however, ties are *one-sided*: they can only occur in pilots' preference lists. Note also that if pilot $p_1$ was assigned to course $c_2$, as in assignment (b), then pilot $p_3$ could not have been assigned to course $c_1$, since $\langle p_2, c_1 \rangle$ would form a blocking pair. This is despite $\langle p_2, c_1 \rangle$ being an illegal assignment.

In a mapping between APA and HRTF, pilots correspond to residents, courses to hospitals, pilot preferences to resident preferences, seniority ranks to hospital preferences, illegal assignments to forbidden pairs and the seniority criterion to stability. Course demands are mapped to hospital quotas. For a given instance of APA, a complete solution (meaning that all hospital slots are filled) to the corresponding HRTF instance could be directly translated back to a solution to APA.

This mapping also allows us to define *blocking pairs* for the seniority criterion in APA: a pair $\langle p_i, c_j \rangle$ is a blocking pair if $p_i$ is senior to $c_j$'s least senior assigned pilot, and $p_i$ is either unassigned or assigned to a course it strictly prefers less than $c_j$ (note that we can safely assume that all course slots are filled, since that is needed for all course demands to be met). With this definition, the seniority criterion holds as long as there are no blocking pairs in the assignment.

## 3.3 Extension: The airline promotion assignment problem with detailed preferences

The problem described in Section 3.1 above is the basic variant of APA. Since different airlines may have different rules, and rules may be subject to change, our model should also be able to handle extensions of APA.

One airline, which we call *Airline Blue*, has some additional rules regarding promotions. Airline Blue's rules involve so-called *preference groups* and *detailed preferences*, and we call the problem with these added rules airline promotion assignment problem with detailed preferences (APA-D). We introduce the following definitions:

**preference group** Each pilot may belong to (at most) one *preference group*.

**detailed preference** Each pilot that belongs to a preference group has, in addition to their default preference list, a *detailed preference* list. The detailed preference list must contain the same courses as the pilot's default preference list. It must also have the same *total preorder* as the default preference list, meaning that the detailed preference list may be stricter or looser than the default preference list, but may not have a reversed ordering between any pair of courses. Formally, let $rank_i(c)$ be the rank of course $c$ in pilot $p_i$'s default preference list (tied courses have the same rank), and equivalently define $rank_i^{det}(c)$ for pilot $p_i$'s detailed preference list. Then, for each pair of courses $c_j$ and $c_l$ in $p_i$'s preference lists,

$$rank_i(c_j) \leq rank_i(c_l) \;\Leftrightarrow\; rank_i^{det}(c_j) \leq rank_i^{det}(c_l)$$

All rules in basic APA are still present in APA-D. However, in addition to the seniority criterion, APA-D involves a **preference group criterion**. It is defined as follows: for each pair of pilots $\langle p_1, p_2 \rangle$ belonging to the same preference group, where $p_1$ is senior to $p_2$, $p_2$ may not be assigned to a course which is better ranked on $p_1$'s detailed preference list than the course $p_1$ is assigned to. This is equivalent to the seniority criterion, except it is only enforced for pairs of pilots within the same preference group, and their detailed preference lists are used instead of their

$$\begin{array}{r|c||ccc} p_1 & g_1 & ([c_1 & c_2] & c_3) \\ p_2 & g_1 & (\cancel{c_1} & [c_2 & c_3]) \\ p_3 & g_2 & c_1 & c_3 & \\ & & & & \\ & c_1 & p_1 & \cancel{p_2} & p_3 \\ & c_2 & p_2 & p_1 & \\ & c_3 & p_1 & p_2 & p_3 \end{array}$$

$$\text{demand}(c_1) = 1, \quad \text{demand}(c_2) = 1, \quad \text{demand}(c_3) = 1$$

Illegal pairs: $\langle p_2, c_1 \rangle$

An example APA-D problem instance. Here, $g_1$ and $g_2$ refer to different preference groups, and brackets indicate detailed preferences.

$$\begin{array}{r|c||ccc} p_1 & g_1 & ([\underline{c_1} & c_2] & c_3) \\ p_2 & g_1 & (\cancel{c_1} & [\underline{c_2} & c_3]) \\ p_3 & g_2 & c_1 & \underline{c_3} & \\ & & & & \\ & c_1 & \underline{p_1} & \cancel{p_2} & p_3 \\ & c_2 & \underline{p_2} & p_1 & \\ & c_3 & p_1 & p_2 & \underline{p_3} \end{array}$$

**(a)** Solution example.

$$\begin{array}{r|c||ccc} p_1 & g_1 & ([c_1 & \underline{c_2}] & c_3) \\ p_2 & g_1 & (\cancel{c_1} & [c_2 & \underline{c_3}]) \\ p_3 & g_2 & \underline{c_1} & c_3 & \\ & & & & \\ & c_1 & p_1 & \cancel{p_2} & \underline{p_3} \\ & c_2 & p_2 & \underline{p_1} & \\ & c_3 & p_1 & \underline{p_2} & p_3 \end{array}$$

**(b)** Solution example.

**Figure 3.2:** An example APA-D problem instance, with two example solutions. Note the similarity to the APA example given in Figure 3.1: the seniority lists are the same, and the detailed preferences here are equivalent to the default preferences in the Figure 3.1. However, in this example, $p_1$'s and $p_2$'s default preferences consist entirely of ties, and $p_3$ does not belong to the same preference group as $p_1$ and $p_2$. As a consequence, it is possible to assign $p_3$ to $c_1$, as in solution example (b).

default preference lists. As a motivating example for the preference group criterion, take a situation where a senior pilot should have priority over pilots with similar background (i.e. pilots belonging to the same preference group) but not over other pilots; this could be modelled by letting the pilot's detailed preference list be stricter than their default preference list.

Thus, the task of APA-D is to find a total assignment of pilots to courses such that all assignments are legal, all course demands are met, the seniority criterion holds and the preference group criterion holds. An example of an APA-D instance, together with a solution, is given in Figure 3.2.

Unlike APA, which corresponds to HRTF, no established stable matching problem directly corresponds to APA-D. However, since the preference group criterion is similar to the seniority criterion, it is also similar to the notion of stability in stable matching problems. Thus, we can define *blocking pairs* in the context of the preference group criterion in APA-D: a pair $\langle p_i, c_j \rangle$ is a blocking pair if $p_i$ is senior to $c_j$'s least senior assigned pilot belonging to the same preference group as $p_i$, and $p_i$ is either unassigned or assigned to a course $c_l$, where $rank_i^{det}(c_l) < rank_i^{det}(c_j)$. With this definition, the preference group criterion holds as long as there are no blocking pairs in the assignment.

# 4
# Model

This chapter describes the model we have developed for the airline promotion assignment problem (APA). In Section 4.1, we describe our constraint satisfaction problem (CSP) model for APA, adapted from the HR with ties model in [44], and prove its equivalency to APA. In Section 4.2, we present a constraint for the APA-D extension. Finally, in Section 4.3, we shortly describe our model implementation, including a symmetry-reducing method we used for data pre-processing.

## 4.1 CSP model for APA

Our CSP model is adapted from O'Malley in [44].

### Sets

$$\begin{aligned} P &= \text{available pilots} \\ C &= \text{courses} \\ F &= \text{illegal pilot assignments} \quad (F \subseteq P \times C) \\ A(p_i) &= \text{courses found acceptable by pilot } p_i \\ B(c_j) &= \text{pilots ranked by course } c_j \end{aligned}$$

Note that $c_j \in A(p_i) \Leftrightarrow p_i \in B(c_j)$. Further, note that $|A(p_i)|$ is the length of $p_i$'s preference list, and that similarily $|B(c_j)|$ is the length of $c_j$'s seniority list. Also, note that we require that all course slots are filled, so $|P| \geq \sum_C d_j$ (with $d_j$ defined below).

### Parameters

$$\begin{aligned} d_j &= \text{the demand for course } c_j \\ rank_i(c_j) &= \text{the rank of course } c_j \text{ in pilot } p_i\text{'s preference list} \\ pref_i(c_j) &= \text{the } \textit{position} \text{ of course } c_j \text{ in pilot } p_i\text{'s linearized preference list} \\ sen_j(p_i) &= \text{the position of pilot } p_i \text{ in course } c_j\text{'s seniority list} \end{aligned}$$

The preference and seniority lists are ordered such that items that occur earlier (i.e. with lower position) are more preferred or more senior, respectively. Note that both $pref_i(c)$ and $sen_j(p)$ are bijective, and thus have well-defined inverse functions, which we call $pref_i^{-1}(a)$ and $sen_j^{-1}(b)$ respectively; we use these in our constraint definitions below.

## Variables

$$x_i \text{ corresponds to the course assigned to pilot } p_i \qquad p_i \in P$$

$$y_{j,k} \text{ corresponds to the pilot assigned to slot } k \text{ of course } c_j \qquad \begin{array}{l} c_j \in C \\ k \in \{1, \dots, d_j\} \end{array}$$

## Domains

$$dom(x_i) = \{|C| + 1\} \cup \{1, \dots, |A(p_i)|\} \setminus \{pref_i(c_j) : \langle p_i, c_j \rangle \in F\} \quad p_i \in P$$

$$dom(y_{j,k}) = \{1, \dots, |B(c_j)|\} \setminus \{sen_j(p_i) : \langle p_i, c_j \rangle \in F\} \qquad \begin{array}{l} c_j \in C \\ k \in \{1, \dots, d_j\} \end{array}$$

The domains of the variables refer to the *positions* of courses and pilots, respectively, in the preference and seniority lists. E.g., $x_2 = 5$ encodes the case where pilot $p_2$ is assigned the course in the fifth position on their linearized preference list. The special value $|C| + 1$ refers to the case where a pilot is not assigned to a course.

Note that illegal assignments are removed from the domains. This is equivalent to using the inequality constraint for forbidden pairs given in [58], which would perform the same domain reduction.

## Constraints

**Course slots are filled in increasing order:**

$$y_{j,k} < y_{j,k+1} \qquad \begin{array}{l} c_j \in C \\ k \in \{1, \dots, d_j - 1\} \end{array} \tag{4.1}$$

**If $p_i$ is assigned a course strictly less preferred than $c_j$, then all slots of $c_j$ should be filled by pilots senior to $p_i$:**

$$x_i > a^+ \Rightarrow y_{j,k} < sen_j(p_i) \qquad \begin{array}{l} p_i \in P \\ a \in \{1, \dots, |A(p_i)|\} \\ k \in \{1, \dots, d_j\} \end{array} \tag{4.2}$$

*where*

$$c_j = pref_i^{-1}(a)$$

$$a^+ = \max\Big\{k \in \{1, \dots, |A(p_i)|\} \ : \ rank_i(pref_i^{-1}(k)) = rank_i(pref_i^{-1}(a))\Big\}$$

In other words, $a^+$ is the position of the last course tied with the course at position $a$ in pilot $p_i$'s preference list.

**If $p_i$ is not assigned to $c_j$, no slots of $c_j$ should contain $p_i$:**

$$x_i \neq a \Rightarrow y_{j,k} \neq sen_j(p_i) \qquad \begin{array}{l} p_i \in P \\ a \in \{1, \ldots, |A(p_i)|\} \\ k \in \{1, \ldots, d_j\} \end{array} \tag{4.3}$$

$$\begin{array}{l} where \\ \quad c_j = pref_i^{-1}(a) \end{array}$$

**For each slot $s$ on $c_j$, if $p_i$ is assigned to $c_j$ and all slots previous to $s$ are assigned to pilots senior to $p_i$, slot $s$ should be filled by a pilot at least as good as $p_i$:**

First, the special case for the first slot:

$$x_i = a \Rightarrow y_{j,1} \leq sen_j(p_i) \qquad \begin{array}{l} p_i \in P \\ a \in \{1, \ldots, |A(p_i)|\} \end{array}$$

All remaining slots:

$$(x_i = a \wedge y_{j,k-1} < sen_j(p_i)) \Rightarrow y_{j,k} \leq sen_j(p_i) \qquad \begin{array}{l} p_i \in P \\ a \in \{1, \ldots, |A(p_i)|\} \\ k \in \{2, \ldots, d_j\} \end{array} \tag{4.4}$$

$$\begin{array}{l} where \\ \quad c_j = pref_i^{-1}(a) \end{array}$$

**If the last slot of $c_j$ is filled by a pilot senior to $p_i$, $p_i$ is not assigned to $c_j$:**

$$y_{j,d_j} < b \Rightarrow x_i \neq pref_i(c_j) \qquad \begin{array}{l} c_j \in C \\ b \in \{d_j, \ldots, |B(c_j)|\} \end{array} \tag{4.5}$$

$$\begin{array}{l} where \\ \quad p_i = sen_j^{-1}(b) \end{array}$$

Now, a short motivation for the constraints is given. Constraint 4.1 ensures that course slots are assigned in seniority order: this reduces symmetry in the solution space, and is a prerequisite for the formulations of the other constraints. Constraint 4.2 encodes the seniority criterion. Constraint 4.3, 4.4 and 4.5 all ensure that course and pilot variables have consistent assignments. Constraint 4.3 says that if pilot $p_i$ stops considering course $c_j$ (i.e., if $pref_i(c_j)$ is removed from the domain of $x_i$), then course $c_j$ should stop considering pilot $p_i$ (and thus remove $sen_j(p_i)$ from all its course variables' domains). Constraint 4.4 and Constraint 4.5 together ensure that if a pilot $p_i$ is assigned a course $c_j$, then one of $c_j$'s slots will contain $p_i$. Constraint 4.5 states the reverse of Constraint 4.3: if course $c_j$ stops considering pilot $p_i$ (if the last of $c_j$'s slot variables, which must refer to the least senior of $c_j$'s assignments, only has pilots more senior than $p_i$ in its domain), then pilot $p_i$ should stop considering course $c_j$ (and thus remove $pref_i(c_j)$ from the domain of $x_i$).

### 4.1.1 Correctness of the model

Now, we will argue that our CSP model is equivalent to APA. With Lemma 1, we show that a solution to this CSP is a solution to APA, and with Lemma 2 that a solution to APA is a solution to this CSP. These two lemmas together support Theorem 1, which says that a complete algorithm looking for a solution to an APA instance encoded with this CSP model is guaranteed to find one if one exists. Our argument is similar to the one given by O'Malley in [44] as to why his HR with ties model yields all stable matchings.

**Lemma 1.** *Let $I$ be an instance of APA and let $P$ be a translation of $I$ to a CSP using the model above. Then, a solution to $P$ is also a solution to $I$.*

*Proof.* To show this, we will argue that in the solution to $P$, all demands are met, no illegal assignments are included, the course and pilot assignments are consistent and the seniority criterion holds.

We begin by showing that all course demands are met in the solution to $P$. First, note that for each course, the number of course slot variables is equal to that course's demand. While the domains of the pilots' assignment variables each include a sentinel value corresponding to that pilot not being assigned a course, the course slot variables do not. As such, each course slot variable must be assigned a value corresponding to a pilot, meaning that all course demands must be met.

Now, we argue that there are no illegal assignments in the solution to $P$. By the definitions of the domains, illegal assignments are not included. Thus, no illegal assignments can be present in a solution to our CSP encoding.

We will now show that the assignments must be consistent. First, we show that if pilot $p_i$ is assigned course $c_j$, then exactly one of $c_j$'s slots is assigned $p_i$. By Constraint 4.1, all course slots must follow strict seniority ordering, meaning that no two slots in the same course can be assigned pilots with equal seniority rank. Since the seniority rank of a pilot is unique within each course, this means that a course cannot be assigned the same pilot more than once. Now, we use proof by contradiction to show that if pilot $p_i$ is assigned course $c_j$, then one of course $c_j$'s slots is assigned $p_i$. Assume that pilot $p_i$'s assignment variable $x_i = pref_i(c_j)$, but all of course $c_j$'s slot variables $y_{j,k} \neq sen_j(p_i)$, $k \in \{1, ..., d_j\}$. According to Constraint 4.4, if $p_i$ is assigned $c_j$, then either there is a slot $c_j$ which is assigned $p_i$, or all of $c_j$'s slots are assigned pilots senior to $p_i$. By our assumption, no slot is assigned $p_i$, thus each slot must be assigned a senior pilot, meaning that $y_{j,k} < sen_j(p_i)$, $k \in \{1, ..., d_j\}$. However, by the contrapositive of Constraint 4.5, $x_i = pref_i(c_j) \Rightarrow y_{j,d_j} \geq sen(p_i)$, we see that $c_j$'s last slot must be assigned $p_i$ or a pilot junior to $p_i$—this contradicts the statement that all slots are assigned senior pilots. Thus, if pilot $p_i$ is assigned course $c_j$, exactly one of $c_j$'s slots must be assigned $p_i$.

Now, the converse: if one of course $c_j$'s slots is assigned pilot $p_i$, then pilot $p_i$ is assigned course $c_j$. This means that if one of course $c_j$'s slot variables $y_{j,k} = sen_j(p_i)$, then pilot $p_i$'s assignment variable $x_i = pref_i(c_j)$. This is shown by the contrapositive of Constraint 4.3, $y_{j,k} = sen_j(p_i) \Rightarrow x_i = pref(c_j)$. Thus, the assignments in the solution to $P$ are consistent.

Finally, we want to show that the seniority criterion holds in a solution to our CSP. Assume that we have a blocking pair (with the definition for APA from

Section 3.2) $\langle p_i, c_j \rangle$ in our solution. This means that either $p_i$ is unmatched and $x_i = |C| + 1$, or $p_i$ is assigned to a course it prefers strictly less than $c_j$: either way, $x_i > a^+$, with $a^+$ being the position of the last course tied with $c_j$ in $p_i$'s preference list, as defined above. Thus, from Constraint 4.2, we get that $y_{j,k} < sen_j(p_i)$, $k \in \{1, ..., d_j\}$—this contradicts that $\langle p_i, c_j \rangle$ is a blocking pair. As such, the solution to $P$ will not contain any blocking pairs, and therefore the seniority criterion must hold.

Thus, to conclude, a solution to the CSP instance $P$ must be a solution to the APA instance $I$. □

We have shown that a solution to our CSP is a solution to APA: now, to ensure that our model does not risk missing any solution, we must show that an arbitrary solution to APA is also a solution to our CSP.

**Lemma 2.** *Let $I$ be an instance of APA and $P$ be the encoding of $I$ using our CSP model. Then, a solution to $I$ is also a solution to $P$.*

*Proof.* We instantiate the variables like this: For each pilot $p_i$, if $p_i$ is assigned to a course $c_j$, then $x_i = pref_i(c_j)$, and if $p_i$ is unassigned, then $x_i = |C| + 1$. For each course $c_j$ assigned to a set of pilots $\{p_{i_k} : k \in \{1, ..., d_j\}\}$, such that $sen_j(p_{i_k}) < sen_j(p_{i_{k+1}})$, $k \in \{1, ..., d_j - 1\}$, we instantiate corresponding slot variables according to this seniority order, $y_{j,k} = sen_j(p_{i_k})$, $k \in \{1, ..., d_j\}$. This gives us an assignment in $P$ equivalent to the solution to $I$. We will now show that all constraints in $P$ hold for this assignment.

First, consider Constraint 4.1. Since course slot variables were instantiated in seniority order, this constraint must hold.

Next, consider Constraint 4.2. Assume that the left-hand side, $x_i > a^+$, is true for some $p_i$ and $a = pref_i(c_j)$. This means that there is a course $c_j$ that pilot $p_i$ would strictly prefer to their assignment. Since the seniority criterion must hold in the solution to $I$, we know that $c_j$ must be assigned $d_j$ pilots that are all senior to $p_i$: by the instantiation, this means that $y_{j,k} < sen_j(p_i)$, $k \in \{1, ..., d_j\}$—which is the right-hand side of Constraint 4.2. Thus, Constraint 4.2 must hold.

Now, consider Constraint 4.3. Assume that the left-hand side, $x_i \neq a$, is true for some $p_i$ and $a = pref_i(c_j)$. This means that pilot $p_i$ is not assigned course $c_j$. Thus, since the solution to $I$ by definition must be consistent, $c_j$ is not assigned pilot $p_i$: by the instantiation, this means that $y_{j,k} \neq sen_j(p_i)$, $k \in \{1, ..., d_j\}$—which is the right-hand side of Constraint 4.3. Thus, Constraint 4.3 must hold.

Consider the special case for the first slot in Constraint 4.4. Assume that the left-hand side, $x_i = a$, is true for some $p_i$ and $a = pref_i(c_j)$. This means that pilot $p_i$ is assigned course $c_j$. Since the solution to $I$ by definition must be consistent, we know that $c_j$ is assigned pilot $p_i$: by the instantiation, this means that some slot variable $y_{j,v} = sen_j(p_i)$, $\exists v \in \{1, ..., d_j\}$. We know that slot variables are instantiated in seniority order: thus, if pilot $p_i$ is the most senior pilot, then $y_{j,1} = sen_j(p_i)$; otherwise, $y_{j,1} < sen_j(p_i)$. Either way, $y_{j,1} \leq sen_j(p_i)$—which is the right-hand side of the special case of the constraint. Thus, the special case of Constraint 4.4 must hold.

Now, we consider the general case for Constraint 4.4. Assume that the left-hand side, $x_i = a \wedge y_{j,k-1} < sen_j(p_i)$, is true for some $p_i$ and $c_j$, where $a = pref_i(c_j)$, and some value $k \in \{2, \ldots, d_j\}$. This means that pilot $p_i$ is assigned course $c_j$, and (since slots are instantiated in seniority order) that there are $k-1$ pilots assigned to $c_j$ which are senior to $p_i$. Since the solution to $I$ by definition is consistent, we know that there is some slot $y_{j,v} = sen_j(p_i)$, $\exists v \in \{1, ..., d_j\}$, and since there are $k-1$ pilots senior to $p_i$, we know that $v \geq k$. If $v = k$, then $y_{j,k} = sen_j(p_i)$; if $v > k$, then $y_{j,k} < sen_j(p_i)$. Either way, $y_{j,k} \leq sen_j(p_i)$—which is the right-hand side of the general case of the constraint. Thus, Constraint 4.4 must hold.

Finally, consider Constraint 4.5. Assume that the left-hand side, $y_{j,d_j} < b$, is true for some $c_j$ and $b = sen_j(p_i)$. This means that the least senior pilot assigned to course $c_j$ is more senior than pilot $p_i$. By the seniority criterion, we then know that $p_i$ cannot be assigned $c_j$: by the instantiation, this means that $x_i \neq pref_i(c_j)$—which is the right-hand side of the constraint. Thus, Constraint 4.5 must hold.

Since all constraints of $P$ hold for the assignment in $P$, the assignment is a correct solution. Thus, a solution to an APA instance $I$ must be a solution to the corresponding CSP instance $P$. □

Lemma 1 and Lemma 2 lead to the following theorem:

**Theorem 1.** *Let $I$ be an instance of APA and $P$ be the corresponding CSP instance encoded by our CSP model. Then, if a solution to $P$ is found, that solution corresponds to a solution to $I$. If no solution to $P$ is found when using a complete algorithm, then no solution to $I$ exists.*

## 4.2 A constraint for APA-D

In order to solve the airline promotion assignment problem with detailed preferences (APA-D) (as described in Section 3.3), we use the same CSP model as described for APA above, but with an added constraint to handle the preference group criterion. For this constraint, we must first define some parameters:

$$\begin{aligned} rank_i^{det}(c_j) &= \text{the rank of course } c_j \text{ in pilot } p_i\text{'s detailed preference list} \\ group(p_i) &= \text{the set containing all pilots in the same preference group as } p_i \\ &\phantom{=} \text{(an empty set if } p_i \text{ does not belong to a preference group)} \end{aligned}$$

**If $p_i$ is assigned a course with strictly worse rank in $p_i$'s detailed preference list than $c_j$, then all pilots in $p_i$'s preference group assigned a slot in $c_j$ must be senior to $p_i$:**

$$(x_i > a_d^+ \land p_s \in group(p_i)) \Rightarrow y_{j,k} < sen_j(p_i) \qquad \begin{array}{l} p_i \in P \\ a \in \{1, \ldots, |A(p_i)|\} \\ k \in \{1, \ldots, d_j\} \end{array} \tag{4.6}$$

*where*

$$c_j = pref_i^{-1}(a)$$
$$p_s = sen_j^{-1}(y_{j,k})$$
$$a_d^+ = \max\Big\{k \in \{1,\ldots,|A(p_i)|\} \;:\; rank_i^{det}(pref_i^{-1}(k)) = rank_i^{det}(pref_i^{-1}(a))\Big\}$$

In other words, $a_d^+$ is the position of the last course tied with the course at position $a$ in pilot $i$'s detailed preference list. Note that function $pref_i(c)$ and its inverse $pref_i^{-1}(a)$ work equivalently for $p_i$'s detailed preference list as they do for $p_i$'s default preference list: since the two lists have the same total preorder (as explained in Section 3.3), we can use the same linearization for the two lists.

## 4.2.1 Correctness of the model

Similar to our argument for APA in the last section, let us now show that our model with the added Constraint 4.6 is equivalent to APA-D. With Lemma 3, we show that a solution to this CSP is a solution to APA-D, and with Lemma 4 that a solution to APA-D is a solution to this CSP. The two lemmas together support Theorem 2, which says that a complete algorithm searching for a solution to our CSP encoding of APA-D is guaranteed to find one if one exists.

**Lemma 3.** *Let $I$ be an instance of APA-D and $P$ be the encoding of $I$ using our CSP with the added Constraint 4.6. Then, a solution to $P$ is also a solution to $I$.*

*Proof.* For a solution to $P$ to be a solution to $I$, all course demands must be met, there can be no illegal assignments, the course and pilot assignments must be consistent and the seniority criterion must hold. All these criteria have been been proven to hold for APA with Lemma 1: this proof applies to APA-D too. For APA-D, however, the preference group criterion must also hold.

To show that Constraint 4.6 ensures that the preference group criterion is met, we make an argument similar to the one for the seniority criterion and Constraint 4.2 in the proof to Lemma 1. Assume that we have a blocking pair (according to the definition for APA-D in Section 3.3) $\langle p_i, c_j\rangle$ in a solution to $P$. This means that either $p_i$ is unmatched and $x_i = |C| + 1$, or $p_i$ is assigned a course $c_l$, $rank_i^{det}(c_l) < rank_i^{det}(c_j)$: either way, $x_i > a_d^+$, with $a_d^+$ being the position of the last course tied with $c_j$ in $p_i$'s detailed preference list, as defined above. Now, if there is any pilot $p_s$ assigned to one of $c_j$'s slots, meaning $p_s = sen_j^{-1}(y_{j,k})$, $\exists k \in \{1,...,d_j\}$, who also belongs to the same preference group as $p_i$, that is $p_s \in group(p_i)$, it is clear from Constraint 4.6 that any such $p_s$ must be senior to $p_i$—this means that $\langle p_i, c_j\rangle$ cannot be a blocking pair, giving us a contradiction. Thus, there can be no blocking pairs in the solution, and Constraint 4.6 guarantees that the preference group criterion is met.

Thus, we can conclude that a solution to $P$ must also be a solution to $I$. □

As in the last section, we will now show that any solution to APA-D is also a solution to the CSP with Constraint 4.6 included.

**Lemma 4.** *Let $I$ be an instance of APA-D and $P$ be the encoding of $I$ using our CSP model with Constraint 4.6 included. Then, a solution to $I$ is also a solution to $P$.*

*Proof.* For a solution to $I$ to also be a solution to $P$, all constraints from Section 4.1 must hold. Again, we can reuse our arguments from one of our previous lemmas: the conclusions from Lemma 2 apply for the case of APA-D too. Now, all we have to prove is that after translating the solution to $I$ to a solution to $P$, Constraint 4.6 will hold.

First, we make an instantiation of the variables using the same procedure as in the proof to Lemma 2. Now, consider Constraint 4.6. Assume that the left-hand side, $x_i > a_d^+ \wedge p_s \in \mathit{group}(p_i)$, with $a_d^+$ defined as above and $p_s = \mathit{sen}_j^{-1}(y_{j,k})$, is true for some $p_i$, $c_j$ and $k \in \{1, ..., d_j\}$. Since the preference group criterion holds in the solution to $I$, and $p_i$ ranks $c_j$ higher in their detailed preference list than their assignment, we know that any pilot assigned to $c_j$ who is also in the same preference group as $p_i$ must be senior to $p_i$ with regard to $c_j$. Since pilot $p_s$ is assigned to $c_j$ and is in the same preference group as $p_i$, this means that $\mathit{sen}_j(p_s) = \mathit{sen}_j(y_{j,k}) < \mathit{sen}_j(p_i)$—which is the right-hand side of Constraint 4.6. As such, Constraint 4.6 must hold.

Thus, we can conclude that a solution to $I$ must also be a solution to $P$. □

Lemma 3 and Lemma 4 support the following theorem:

**Theorem 2.** *Let $I$ be an instance of APA-D and $P$ be the corresponding CSP instance encoded by our CSP model, including Constraint 4.6. Then, if a solution to $P$ is found, that solution corresponds to a solution to $I$. If no solution to $P$ is found when using a complete algorithm, then no solution to $I$ exists.*

## 4.3 Model implementation

Our model for APA was implemented in C++11. It uses the CP system Google or-tools (see Section 2.2.6) for solving an APA instance encoded using our CSP model: we have more or less directly translated the variables, domains and constraints to corresponding CP code. Our model also contains a pre-processing step, which merges similar courses: see Section 4.3.1 below. In addition, our system contains classes that model problem instances, courses and pilots; parsers for reading datasets; systems for verifying input data and solution correctness; a text-based user interface; and a thorough unit test suite.

### 4.3.1 Data pre-processing

Our model performs some pre-processing steps in order to simplify the problem. In particular, it is possible to eliminate a great deal of symmetry from some of our test cases by identifying and merging courses that are structurally equivalent. That is, a set $G$ of courses can be replaced by a single replacement course, with demand equal to the sum of the demands of its constituent courses, if there exists a course $c_*$ such that:

$$\begin{aligned} &\forall c_j \in G. \quad B(c_j) = B(c_*) \\ \forall p_i \in B(c_*) \quad &\forall c_j \in G. \quad \mathit{sen}_j(p_i) = \mathit{sen}_*(p_i) \\ \forall p_i \in B(c_*) \quad &\forall c_j \in G. \quad \mathit{rank}_i(c_j) = \mathit{rank}_i(c_*) \end{aligned}$$

# 5
# Results

In this chapter, we present results from running our APA model on problem instances based on data from two different airlines. First, in Section 5.1 we present our test data. In Section 5.2 we present the different solver configurations used for achieving our results, such as different variable and value ordering heuristics, comparing their performance. Finally, in Section 5.3, our main results are presented.

## 5.1 Test case data

In order to evaluate our model, Jeppesen has provided us with data derived from two different airlines: *Airline Red* and *Airline Blue* (their real names have been removed). We have two datasets per airline, which we call `red-a` & `red-b`, and `blue-a` & `blue-b`, respectively. The `b` datasets are generally larger than the `a` datasets. Each airline dataset contains course and pilot data, including preference and seniority lists, and lists of illegal assignments. The airline datasets define instances of APA/APA-D, except for course demands, which are given separately in what we call *demand items*. Each demand item defines a subset of courses and their corresponding demands.

For each airline dataset, Jeppesen has provided us with two sets of computer-generated demand items, with different characteristics: `good` and `typical`:

**`good`** Generated from near-optimal solutions to the larger manpower problem. Are known to have stable assignments.

**`typical`** Typical examples of demands that would be used in an integrated process. Some have stable solutions and the rest are in general "almost" solvable.

The combination of an airline dataset and a demand item forms a problem instance. The problem instances are summarized in Table 5.1. As can be seen, the characteristics of the Airline Red and Airline Blue problems are very different. First and foremost, all Airline Blue problems are instances of APA-D, while Airline Red problems are instances of APA. Airline Blue problem instances are also generally larger in terms of number of pilots, courses and course slots, and the differences in seniority and preference list lengths are very large. Also, note the differences in average *tie density*. Tie density is a measure of the prevalence and size of ties the preference lists: it is defined as the probability that a course is tied with the next course in a pilot's preference list. While the detailed preference lists of Airline Blue are comparable with regard to tie density to the preference lists of Airline Red, the default preference lists in the Airline Blue instances generally contain very large ties.

| Demand item set | APA-D? | Pilots | Courses | Slots | Seniority list length | Preference list length | Illegal assignments | Tie density | Tie density (detailed) | # of instances |
|---|---|---|---|---|---|---|---|---|---|---|
| `red-a-good` | No | 1100 | 30 | 100 | 174 | 4 | 6% | 29% | — | 24 |
| `red-a-typical` | No | 1100 | 30 | 100 | 188 | 4 | 6% | 32% | — | 100 |
| `red-b-good` | No | 1600 | 50 | 200 | 180 | 5 | 1% | 13% | — | 31 |
| `red-b-typical` | No | 1600 | 60 | 300 | 183 | 6 | 1% | 20% | — | 100 |
| `blue-a-good` | Yes | 1600 | 190 | 400 | 658 | 76 | 74% | 93% | 8% | 60 |
| `blue-a-typical` | Yes | 1600 | 210 | 400 | 665 | 86 | 73% | 94% | 8% | 100 |
| `blue-b-good` | Yes | 1900 | 200 | 600 | 752 | 80 | 84% | 89% | 18% | 35 |
| `blue-b-typical` | Yes | 1900 | 250 | 600 | 798 | 102 | 84% | 90% | 22% | 100 |
| | | | | | | | | | **Total** | 550 |

**Table 5.1:** Summary of problem instances used for evaluation. Figures given are averages within the demand item set. The numbers for pilot, course and slot counts have been rounded off for data anonymization reasons. Airline Blue contains detailed preferences and preference groups, while Airline Red does not, so all Airline Blue problems are instances of APA-D, while Airline Red problems are APA instances.

## 5.2 Solver settings

We have run our problem instances with a number of different solver configurations. Mainly, we have tested different heuristics for variable and value orderings, which we present here. We also present some additional settings that we have tested.

### 5.2.1 Search configurations

We ran tests in order to evaluate a number of different variable and value ordering heuristics, and whether assigning pilot variables or course slot variables performs better. The `red-b-good` and `red-b-typical` datasets were used for this purpose. Below, we list the variable and value orderings we used.

#### Variable orderings

**`average seniority`** Only when assigning courses to pilots. Sort pilots according to their average positions in seniority lists. Select variables corresponding to pilots in that order.

**`average preference`** Only when assigning courses to pilots. Sort courses according to their average positions in preference lists. Select slot variables (starting with the slot for the most senior pilot) corresponding to courses in that order.

**`min size-lowest min`** Select the variable with smallest domain size: in case of a tie, select the variable with the lowest minimum value in its domain.

**`min size-highest min`** Select the variable with smallest domain size: in case of a tie, select the variable with the highest minimum value in its domain.

**`lowest min`** Select the variable with the lowest minimum value in its domain.

**`max size`** Select the variable with the largest domain size.

#### Value orderings

**`assign min`** Assign the minimum value in the domain (corresponding to the most senior pilot/most preferred course).

**`assign max`** Assign the minimum value in the domain (corresponding to the least senior pilot/least preferred course).

**`split lower`** Split the domain in half, select the lower half (corresponding to the most senior pilots/most preferred courses).

**`split upper`** Split the domain in half, select the upper half (corresponding to the least senior pilots/least preferred courses).

| **Assigning courses to pilots** | `assign min` | `assign max` | `split lower` | `split upper` |
|---|---|---|---|---|
| **`red-b-good`** | | | | |
| `average seniority` | 128135* | 6599 | 127841* | 6593 |
| `min size-lowest min` | 6496 | 790 | 6499 | 790 |
| `min size-highest min` | 96804* | 234973 | 86693* | 234973 |
| `lowest min` | 111822* | 98 | 88631* | 93 |
| **`red-b-typical`** | | | | |
| `average seniority` | 702 | 12 | 702 | 12 |
| `min size-lowest min` | 69 | 12 | 69 | 12 |
| `min size-highest min` | 323 | 25 | 323 | 25 |
| `lowest min` | 748 | 16 | 4223 | 16 |
| **Total average** | | | | |
| `average seniority` | 64418* (10.94 s) | 3305 (0.90 s) | 64271* (11.15 s) | 3302 (0.87 s) |
| `min size-lowest min` | 3282 (1.55 s) | 401 (0.53 s) | 3284 (1.55 s) | 401 (0.53 s) |
| `min size-highest min` | 48563* (30.20 s) | 117499 (6.28 s) | 43508* (30.23 s) | 117499 (6.32 s) |
| `lowest min` | 56285* (20.21 s) | **57 (0.49 s)** | 46427* (26.10 s) | **54 (0.49 s)** |

**Table 5.2:** Average number of branches required to find a solution for different combinations of variable and value orderings, using the assignment of *courses to pilots* as the primary variables. A star indicates that the time limit of five minutes was reached for at least one instance in the set.

The results can be seen in Tables 5.2 and 5.3. Some additional variable and value orderings which performed poorly across the board have been excluded for brevity. It is worth noting that the ordering combinations which give good results when assigning courses to pilots give very poor results when instead assigning pilots to courses, and vice versa. This is further discussed in Chapter 6.

We selected the five most performant configurations (marked bold in Tables 5.2 and 5.3), and ran tests on `blue-a-good` and `blue-b-good`, with a presolve percentage of 75% and a timeout of two minutes. Assigning pilots to courses using `average preference` and `assign min` as variable and value ordering heuristics performed considerably better than the other alternatives, and this setting was used for the remaining test runs.

| **Assigning pilots to courses** | `assign min` | `assign max` | `split lower` | `split upper` |
|---|---|---|---|---|
| **`red-b-good`** | | | | |
| `average preference` | 31 | 172199* | 204 | 107160* |
| `min size-lowest min` | 2474 | 742573* | 2509 | 1106328* |
| `min size-highest min` | 10827 | 621563* | 10843 | 1232136* |
| `lowest min` | 20 | 220764 | 116 | 2894 |
| `max size` | 77 | 583989* | 5172822* | 1786297* |
| **`red-b-typical`** | | | | |
| `average preference` | 8 | 1025 | 42 | 37 |
| `min size-lowest min` | 13 | 1327109* | 28 | 1069499* |
| `min size-highest min` | 84 | 672889* | 94 | 761896* |
| `lowest min` | 9 | 15770 | 52 | 105 |
| `max size` | 5 | 384883* | 295981 | 958178* |
| **Total average** | | | | |
| `average preference` | **19** **(0.47 s)** | 86612* (27.39 s) | 123 (0.47 s) | 53598* (14.36 s) |
| `min size-lowest min` | 1243 (0.65 s) | 1034841* (166.67 s) | 1268 (0.65 s) | 1087913* (165.73 s) |
| `min size-highest min` | 5455 (1.33 s) | 647226* (157.15 s) | 5468 (1.31 s) | 997016* (155.46 s) |
| `lowest min` | **14** **(0.47 s)** | 118267 (13.02 s) | 84 (0.47 s) | 1499 (0.75 s) |
| `max size` | **41** **(0.47 s)** | 484436* (229.23 s) | 2734401* (51.11 s) | 1372237* (191.85 s) |

**Table 5.3:** Average number of branches required to find a solution for different combinations of variable and value orderings, using the assignment of *pilots to courses* as the primary variables. A star indicates that the time limit of five minutes was reached for at least one instance in the set.

### 5.2.2 Additional settings

In addition to using different branching heuristics, these further settings were used for some of our tests.

#### Assigning some pilots beforehand

For test cases based on the `good` demand items, we already have solutions available. By assigning pilots to a subset of course slots before running the solver, we can in a way make these test cases smaller. With this, we can evaluate how our model performs on test cases of different size. We refer to this practice as *presolving*. We may select how many pilots we assign, and the pilots are selected in a deterministic manner based on implementation details.

#### No course merging

As explained in Section 4.3.1, our model merges courses that always appear in the same ties in preference lists, and have equal seniority lists. We evaluate how much this pre-processing step affects problem complexity and solver performance.

## 5.3 Performance results

We have carried out a series of test runs to answer different questions about our model and test data. All test runs were performed on a Xeon E5-2667 v2 CPU with 32 GB of available RAM.

### 5.3.1 Main results

A summary of our main results is given in Table 5.4. We were able to achieve good results for the Airline Red datasets. Over all corresponding demand item sets, the average run time was 338 ms, and no problem instance required more than 540 ms to either be solved or deemed unsolvable.

For Airline Blue, the results were different. Using the solutions available for the `good` demand item sets, we were able to verify that our model accepted those solutions as stable. However, as seen in Table 5.4, our model was overall only able to solve two demand items from `blue-a-good` and none from the other Airline Blue demand item sets. We also tried attempts with very long timeouts (of an hour or above) on some demand items from each demand item set, without achieving any better results. It should however be noted that our model was able to deem some problem instances in `blue-a-typical` and all in `blue-b-typical` infeasible within the time limit.

### 5.3.2 Reduced cases of Airline Blue

As noted in Section 5.3.1, we were unable to get good results for Airline Blue. We therefore attempted to measure how large problems our model is able to handle, by attempting to solve the `blue-a-good` and `blue-b-good` demand item sets with

| Demand item set | % solved | % infeasible | % timeout | Run time (solved) | Run time (infeasible) |
|---|---|---|---|---|---|
| `red-a-good` | 100% | — | 0% | 0.19 s | — |
| `red-a-typical` | 94% | 6% | 0% | 0.20 s | 0.19 s |
| `red-b-good` | 100% | — | 0% | 0.48 s | — |
| `red-b-typical` | 59% | 41% | 0% | 0.50 s | 0.43 s |
| `blue-a-good` | 3% | — | 97% | 1.47 s | — |
| `blue-a-typical` | 0% | 27% | 73% | — | 2.58 s |
| `blue-b-good` | 0% | — | 100% | — | — |
| `blue-b-typical` | 0% | 100% | 0% | — | 2.26 s |

**Table 5.4:** Main results from running our model on all datasets. The solver was configured to assign pilots to courses with variable ordering `average preference` and value ordering `assign min`. The time limit was set to 30 s. Results are divided between solved problems, problems that were deemed infeasible, and problems for which the time limit was reached. Run times are given as averages. Note that the problem instances in the `good` demand item sets by definition are solvable.

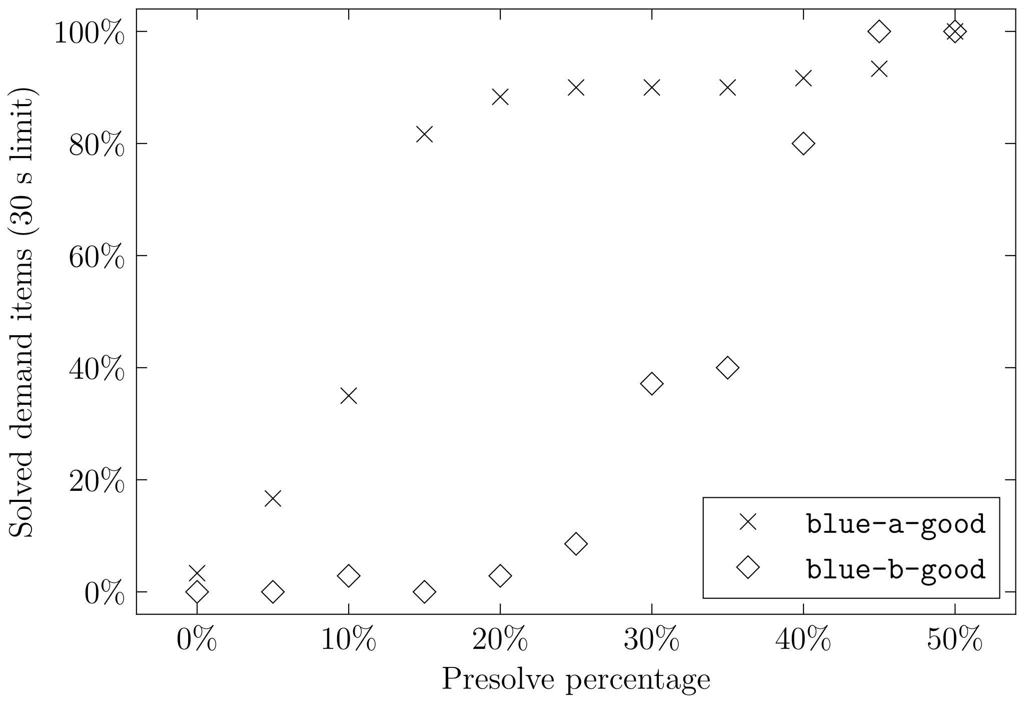


**Figure 5.1:** The percentage of demand items that were solvable in 30 s, for a range of presolving settings. With a presolve setting of 50% (or above), all demand items were solvable for both `blue-a-good` and `blue-b-good`.

varying levels of presolving. The results can be seen in Figure 5.1. Assigning 15% of pilots beforehand was sufficient to be able to solve over 80% of the demand items in `blue-a-good` within 30 s, but achieving the same results for `blue-b-good` required a presolve percentage of 40%.

### 5.3.3 No course merging

To evaluate the effects of merging courses, we ran tests on the Airline Red datasets with merging switched off. The results can be seen in Figure 5.2. Most problem instances had similar run times to those achieved when using course merging (see Table 5.4), but a number of instances timed out despite a high time limit of five minutes. In particular, the model without course merging had issues with determining instances as infeasible.

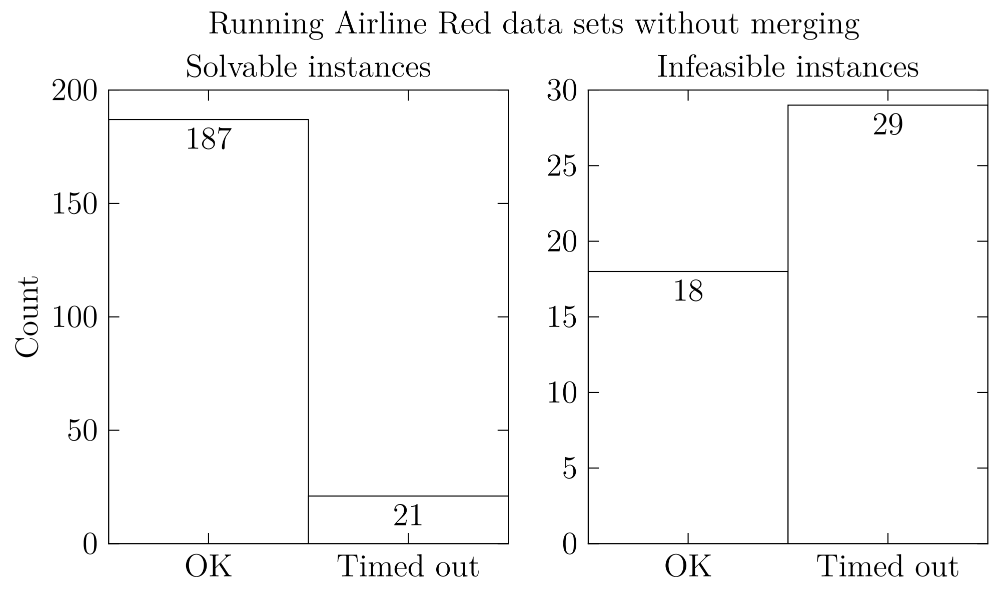


**Figure 5.2:** The results form running Airline Red datasets with course merging switched off, using a time limit of five minutes. 'OK' problem instances were correctly solved or deemed infeasible within the time limit. Note that the model, with course merging turned off, timed out more on problem instances known to be infeasible, suggesting that course merging may be especially efficient for those problems.

# 6
# Discussion and conclusion

In this chapter, we present some final discussions and conclusions. In Section 6.1, we discuss the various results from Chapter 5. This is followed by our conclusions in Section 6.2. We end this final chapter with some suggestions on future work in Section 6.3.

## 6.1 Discussion of results

Running our airline promotion assignment problem (APA) model on the two airline datasets gave different results. We were able to efficiently solve the Airline Red test cases, but not those from Airline Blue. This is however not very surprising: in addition to the presence of detailed preferences in Airline Blue, the data in Table 5.1 shows clear differences between the datasets, especially when it comes to seniority and preference list lengths, the number of illegal assignments, and tie density. Longer seniority and preference lists and higher tie density should increase the size of the search space, and thus it is natural that the Airline Blue problems are harder. Since HR with ties, and thus APA, is NP-hard, it is not very surprising that our model was not able to find any solutions within reasonable time for the harder problem instances. However, in the `typical` testcases, especially `blue-b-typical`, we were able to quickly determine if a problem instance was unsolvable. This means that our model may still be usable for Airline Blue problem instances: even if it does not solve many problem instances, it is still useful to be able to say if problem instances are unsolvable. Further, our results from running reduced versions of the Airline Blue test cases suggest that our model has some potential for being able to solve similar problem instances.

An open problem listed in [44] is to find well-suited variable and value orderings for the presented HR with ties model. In Section 5.2, we present the results of running our model with different search configurations, and we got the best results from assigning pilots to courses using `average preference` and `assign min` as variable and value ordering heuristics. Other similar search configurations, where the most preferred courses are given the most senior pilots, also performed well. However, contrary to our intuition, search configurations where courses were assigned to pilots, and the by average most senior pilots received their most preferred courses first, did not perform well at all. Instead, the opposite, where senior pilots received their worst possible choices first, performed much better. This may seem counterintuitive, but it has been noted for HR and SM problems that there is a relation between one side being assigned their optimal choices, and the other side being assigned their worst possible choices [16]. Since our data is derived from a substep of a larger airline manpower solver, and that solver primarily works from the perspective of courses rather than pilots, this is reasonable.

As explained in Section 5.3.3, we also performed some test runs with our course merging pre-processing step turned off, in order to evaluate how it affects performance. Evidently, this step affects the efficiency of our model greatly, and in our test data it was especially efficient for problem instances known to be infeasible.

## 6.2 Conclusion

The aim of this thesis was to implement an efficient model for APA, a subproblem to the airline manpower problem. We approached this by mapping APA to a stable matching problem, more specifically the hospitals/residents problem with ties and forbidden pairs, and then utilizing existing stable matching literature to find a suitable solution method. The method we chose for our APA model was constraint programming (CP), and the constraint satisfaction problem (CSP) model we developed was an adaptation of the hospitals/residents problem (HR) with ties model from [44]. We believe that we are the first to write about and utilize the connection between the airline manpower problem and stable matching problems. We are also the first to perform an empirical study of the HR with ties model from [44].

We have developed some extensions to the CSP model from [44]. Our CSP model contains support for forbidden pairs (illegal assignments), and we have developed a constraint for the preference group criterion in APA-D.

## 6.3 Future work

### Could a different CSP model yield better performance?

In Section 2.2.5, we discuss the difficulty of coming up with an efficient CSP model for a problem. As mentioned there, there are no fast and easy rules to a good CSP formulation: one simply has to try different formulations. We have only thoroughly evaluated one model for APA, which is an adaptation of O'Malley's CSP encoding for HR with ties in [44]. For better performance, one may want to try alternative models.

Regarding what models could potentially yield a better result, the common consensus of the CP community is to utilize global constraints as often as possible. Our model does however not contain any global constraints. Could one potentially reformulate some (or all) of our constraints into global constraints, such as `all_different`? Or could one potentially keep our constraints, and add redundant global constraints that make the propagation more efficient? The latter approach was tried by Gent and Prosser in [20], an early paper presenting a CSP model for SM with ties and incomplete lists. They added two redundant `all_different` constraints, but found that it did not increase performance. Despite this, it might still be interesting to try reformulating our problem with global constraints.

As mentioned in Section 2.2.4, it is possible to define specialized constraints with corresponding specialized propagators. This approach has been successfully tried for basic HR, and in [38] both a specialized binary constraint and a specialized $n$-ary constraint for HR are presented, both of which performed better than a direct constraint formulation. We have however not been able to find any specialized constraints for HR with ties. Unsworth, in [57], presents some early work on a specialized binary constraint for SM with ties and incomplete lists, and suggests that it should be possible to extend his work and the work in [38] to both binary and $n$-ary constraints for HR with ties. However, while this approach may be more efficient, it should be kept in mind that one of the strong points of CP is the directness of the model formulations. Specialized constraints may lead to a less flexible model, closer to that of a fixed algorithm.

### Could better search heuristics yield better performance?

In Section 5.2.1, we present the results of running our model with a number of different search configurations, and it is clear that some combinations of variable and value orderings are more efficient than others. The most efficient variable ordering we came up with is based on sorting courses according to their average positions in preference lists. It might well be possible to come up with more sophisticated orderings. Since much of the complexity of the problem comes from the presence of ties in pilots' preference lists, perhaps an extension of our sorting with more sophisticated tie-breaking could yield better results. Inspiration for better tie-breaking could perhaps come from heuristics or approximation algorithms for HR with ties and SM with ties and incomplete lists, since many of them are based on sophisticated tie-breaking [28, 33].

It may also be interesting to try other general variable and value orderings. However, our choice of CP system, Google or-tools, does not support a very wide variety of orderings. For example, variable orderings based on the *degrees* of variables, i.e. how many constraints they are involved in [47, p. 105], could be interesting to try. Such variable orderings can for example be found in the CP system Gecode [52].

### Could dynamic constraint satisfaction be utilized?

Our APA model is primarily meant to be used by trying different course demand values while keeping the same underlying pilot and course data. Used this way, the solver tries to solve many similar problem instances in sequence, and thus it is possible that the respective solution spaces also are similar. It would therefore be interesting to try to utilize this in some way. *Dynamic constraint satisfaction problems* are CSPs where the set of constraints or the set of variables may change during runtime [60]. While some different solution methods for dynamic CSPs have been proposed [4, 60], we have not been able to find a modern CP system with any support for dynamic solution methods. The most modern such system we have found is the PaLM system [31] from 2000, and there does not seem to have been much progress made since then.

### Could another method altogether perform better when solving APA?

Many methods have been tried and evaluated to solve HR with ties. However, there is a lack of meta studies that compare different solution methods. We investigated other methods, such as using an integer programming model, but previous work was only able to solve small instances [35]. We chose CP since it is a complete method, which is an advantage when many problem instances may be unsolvable, and since CP models are easy to extend for various rulesets, such as APA-D. However, recent studies show that methods based on local search, such as adaptive search [42], may be very efficient for HR with ties. On the other hand, local search has the drawback of being an incomplete method, and is less flexible compared to CP.

Of course, yet another method may perform better for solving APA. Methods such as SAT and answer set programming [10] have been used for solving HR with ties. Adapting and evaluating those methods for APA could be worthwhile.

# Bibliography

[1] Krzysztof Apt. *Principles of constraint programming.* Cambridge University Press, 2003 (cited on pp. 8, 11, 16).

[2] Marlene Arangú and Miguel Salido. “A fine-grained arc-consistency algorithm for non-normalized constraint satisfaction problems”. In: *International Journal of Applied Mathematics and Computer Science* 21.4 (2011), pp. 733–744 (cited on p. 14).

[3] Christian Bessiere. “Arc-consistency and arc-consistency again”. In: *Artificial intelligence* 65.1 (1994), pp. 179–190 (cited on p. 14).

[4] Christian Bessiere. “Arc-Consistency in Dynamic Constraint Satisfaction Problems.” In: *AAAI.* Vol. 91. 1991, pp. 221–226 (cited on p. 45).

[5] Christian Bessière, Eugene C Freuder, and Jean-Charles Régin. “Using constraint metaknowledge to reduce arc consistency computation”. In: *Artificial Intelligence* 107.1 (1999), pp. 125–148 (cited on p. 14).

[6] Christian Bessière et al. “An optimal coarse-grained arc consistency algorithm”. In: *Artificial Intelligence* 165.2 (2005), pp. 165–185 (cited on p. 14).

[7] Economic Sciences Prize Committee et al. *Stable matching: Theory, Evidence, and Practical Design-The Sveriges Riksbank Prize in Economic Sciences in Memory of Alfred Nobel 2012: Scientific Background.* Tech. rep. Tech. rep. The Nobel Foundation, Stockholm, Sweden, 2012 (cited on p. 5).

[8] Data61, CSIRO. *MiniZinc.* 2016. URL: `http://www.minizinc.org` (visited on 2016-09-09) (cited on p. 16).

[9] CSIRO Data61. *MiniZinc Challenge 2016 Results.* 2016. URL: `http://www.minizinc.org/challenge2016/results2016.html` (visited on 2016-11-01) (cited on p. 16).

[10] Sofie De Clercq et al. “Solving stable matching problems using answer set programming”. In: *arXiv preprint arXiv:1512.05247* (2015) (cited on pp. 8, 46).

[11] Rina Dechter. *Constraint processing.* Morgan Kaufmann, 2003 (cited on pp. 13, 14).

[12] Jordan Demeulenaere et al. “Compact-Table: Efficiently Filtering Table Constraints with Reversible Sparse Bit-Sets”. In: *arXiv preprint arXiv:1604.06641* (2016) (cited on p. 14).

[13] Vânia MF Dias et al. “The stable marriage problem with restricted pairs”. In: *Theoretical Computer Science* 306.1 (2003), pp. 391–405 (cited on p. 6).

[14] Marc RC van Dongen. “AC-3d an efficient arc-consistency algorithm with a low space-complexity”. In: *International Conference on Principles and Practice of Constraint Programming.* Springer. 2002, pp. 755–760 (cited on p. 14).

[15] Eugene C Freuder. “In pursuit of the holy grail”. In: *Constraints* 2.1 (1997), pp. 57–61 (cited on p. 9).

[16] David Gale and Lloyd S Shapley. “College admissions and the stability of marriage”. In: *The American Mathematical Monthly* 69.1 (1962), pp. 9–15 (cited on pp. 3–6, 44).

[17] David Gale and Marilda Sotomayor. “Some remarks on the stable matching problem”. In: *Discrete Applied Mathematics* 11.3 (1985), pp. 223–232 (cited on pp. 4, 7).

[18] Peter Gärdenfors. “Match making: assignments based on bilateral preferences”. In: *Behavioral Science* 20.3 (1975), pp. 166–173 (cited on pp. 5, 7).

[19] Mirco Gelain et al. “Local search for stable marriage problems”. In: *arXiv preprint arXiv:1007.0859* (2010) (cited on p. 8).

[20] Ian P Gent and Patrick Prosser. “An empirical study of the stable marriage problem with ties and incomplete lists”. In: *ECAI.* 2002, pp. 141–145 (cited on pp. 8, 44).

[21] Ian P Gent et al. “A constraint programming approach to the stable marriage problem”. In: *International Conference on Principles and Practice of Constraint Programming.* Springer. 2001, pp. 225–239 (cited on p. 8).

[22] Ian P Gent et al. “SAT encodings of the stable marriage problem with ties and incomplete lists”. In: *SAT* 2002 (2002), pp. 133–140 (cited on p. 8).

[23] Mattias Grönkvist. “The tail assignment problem”. PhD thesis. Chalmers University of Technology and Göteborg University, 2005 (cited on p. 13).

[24] Åsa Holm. “Manpower Planning in Airlines: Modeling and Optimization”. MA thesis. Linköping University, 2008 (cited on pp. 1, 2).

[25] Chien-Chung Huang and Telikepalli Kavitha. “An improved approximation algorithm for the stable marriage problem with one-sided ties”. In: *International Conference on Integer Programming and Combinatorial Optimization.* Springer. 2014, pp. 297–308 (cited on p. 7).

[26] Google Inc. *Google Optimization Tools.* 2016. URL: `https://developers.google.com/optimization/` (visited on 2016-09-09) (cited on pp. 11, 14, 17).

[27] Robert W Irving. “Stable marriage and indifference”. In: *Discrete Applied Mathematics* 48.3 (1994), pp. 261–272 (cited on p. 6).

[28] Robert W Irving and David F Manlove. “Finding large stable matchings”. In: *Journal of Experimental Algorithmics (JEA)* 14 (2009), p. 2 (cited on pp. 8, 45).

[29] Robert W Irving, David F Manlove, and Sandy Scott. “The hospitals/residents problem with ties”. In: *Scandinavian Workshop on Algorithm Theory.* Springer. 2000, pp. 259–271 (cited on pp. 5, 6).

[30] Kazuo Iwama et al. “Stable marriage with incomplete lists and ties”. In: *International Colloquium on Automata, Languages, and Programming.* Springer. 1999, pp. 443–452 (cited on pp. 5, 6).

[31] Narendra Jussien and Vincent Barichard. “The PaLM system: explanation-based constraint programming”. In: *Proceedings of TRICS: Techniques foR Implementing Constraint programming Systems, a post-conference workshop of CP.* Vol. 2000. 2000, pp. 118–133 (cited on p. 45).

[32] Richard M Karp. “Reducibility among combinatorial problems”. In: *Complexity of computer computations.* Springer, 1972, pp. 85–103 (cited on p. 9).

[33] Zoltán Király. “Linear time local approximation algorithm for maximum stable marriage”. In: *Algorithms* 6.3 (2013), pp. 471–484 (cited on pp. 7, 45).

[34] Vipin Kumar. “Algorithms for constraint-satisfaction problems: A survey”. In: *AI magazine* 13.1 (1992), p. 32 (cited on pp. 9, 11).

[35] Augustine Kwanashie and David F Manlove. “An integer programming approach to the hospitals/residents problem with ties”. In: *Operations Research Proceedings 2013.* Springer, 2014, pp. 263–269 (cited on pp. 8, 46).

[36] Alan K Mackworth. “Consistency in networks of relations”. In: *Artificial intelligence* 8.1 (1977), pp. 99–118 (cited on pp. 9, 13).

[37] David F Manlove. *Algorithmics of matching under preferences.* Vol. 2. World Scientific, 2013 (cited on p. 8).

[38] David F Manlove et al. “A constraint programming approach to the hospitals/residents problem”. In: *International Conference on Integration of Artificial Intelligence (AI) and Operations Research (OR) Techniques in Constraint Programming.* Springer. 2007, pp. 155–170 (cited on pp. 8, 14, 45).

[39] David F Manlove et al. “Hard variants of stable marriage”. In: *Theoretical Computer Science* 276.1 (2002), pp. 261–279 (cited on pp. 4, 6, 7).

[40] Ugo Montanari. “Networks of constraints: Fundamental properties and applications to picture processing”. In: *Information sciences* 7 (1974), pp. 95–132 (cited on p. 13).

[41] Björn Morén. “Utilizing problem specic structures in branch and bound methods for manpower planning”. MA thesis. Linköping University, 2012 (cited on p. 2).

[42] Danny Munera et al. “A Local Search Algorithm for SMTI and its extension to HRT Problems”. In: *3rd International Workshop on Matching Under Preferences.* 2015 (cited on pp. 4, 8, 46).

[43] Danny Munera et al. “Solving hard stable matching problems via local search and cooperative parallelization”. In: *29th AAAI Conference on Artificial Intelligence.* 2015 (cited on p. 8).

[44] Gregg O'Malley. "Algorithmic aspects of stable matching problems". PhD thesis. University of Glasgow, 2007 (cited on pp. 1, 8, 25, 28, 43, 44).

[45] OscaR Team. *OscaR: Scala in OR*. 2012. URL: `https://bitbucket.org/oscarlib/oscar` (visited on 2017-01-06) (cited on p. 14).

[46] Xiangtong Qi, Jonathan F. Bard, and Gang Yu. "Class Scheduling for Pilot Training". In: *Operations Research* 52.1 (2004), pp. 148–162 (cited on pp. 1, 19).

[47] Francesca Rossi, Peter Van Beek, and Toby Walsh. *Handbook of constraint programming*. Elsevier, 2006 (cited on pp. 8, 11, 13, 14, 16, 45).

[48] Alvin E Roth. "The evolution of the labor market for medical interns and residents: a case study in game theory". In: *The Journal of Political Economy* (1984), pp. 991–1016 (cited on p. 5).

[49] Staurt J Russell and Peter Norvig. *Artificial Intelligence (A Modern Approach)*. 2010 (cited on p. 13).

[50] *SAS annual report with sustainability review, November 2014–October 2015*. Feb. 2016 (cited on p. 1).

[51] Christian Schulte, Mikael Lagerkvist, and Guido Tack. "Gecode". In: *Software download and online material at the website: http://www.gecode.org* (2006) (cited on pp. 11, 13).

[52] Christian Schulte, Guido Tack, and Mikael Z Lagerkvist. *Modeling and programming with Gecode*. 2016 (cited on pp. 13, 14, 45).

[53] Mohamed Siala and Barry O'Sullivan. "Revisiting two-sided stability constraints". In: *International Conference on AI and OR Techniques in Constriant Programming for Combinatorial Optimization Problems*. Springer. 2016, pp. 342–357 (cited on p. 8).

[54] Milind G Sohoni, Ellis L Johnson, and T Glenn Bailey. "Long-range reserve crew manpower planning". In: *Management Science* 50.6 (2004), pp. 724–739 (cited on p. 1).

[55] Ivan E Sutherland. "Sketchpad: a man-machine graphical communication system". In: *Proceedings of the SHARE design automation workshop*. ACM. 1964, pp. 6–329 (cited on p. 9).

[56] Björn Thalén. "Manpower planning for airline pilots: A tabu search approach". MA thesis. Linköping University, 2010 (cited on pp. 1, 2, 20).

[57] C Unsworth. "A specialised binary constraint for the stable marriage problem with ties and incomplete preference lists". In: *Doctoral program at the 12th International Conference on Principles and Practice of Constraint Programming (CP'06)*. 2006 (cited on p. 45).

[58] Chris Unsworth. "A specialised constraint approach for stable matching problems". PhD thesis. University of Glasgow, 2008 (cited on pp. 8, 14, 26).

[59] Pascal Van Hentenryck. *Constraint satisfaction in logic programming*. Vol. 5. MIT press Cambridge, 1989 (cited on p. 16).

[60] Gérard Verfaillie and Thomas Schiex. "Solution reuse in dynamic constraint satisfaction problems". In: *AAAI*. Vol. 94. 1994, pp. 307–312 (cited on p. 45).

[61] Mark Wallace, Stefano Novello, and Joachim Schimpf. "ECLiPSe: A platform for constraint logic programming". In: *ICL Systems Journal* 12.1 (1997), pp. 159–200 (cited on p. 16).

[62] Jan Wielemaker et al. "Swi-prolog". In: *Theory and Practice of Logic Programming* 12.1-2 (2012), pp. 67–96 (cited on p. 16).

[63] Gang Yu. *Operations research in the airline industry*. Vol. 9. Springer Science & Business Media, 2012 (cited on p. 1).

[64] Gang Yu, Stacy Dugan, and Mike Argüello. "Moving toward an integrated decision support system for manpower planning at Continental Airlines: Optimization of pilot training assignments". In: *Industrial Applications of Combinatorial Optimization*. Springer, 1998, pp. 1–24 (cited on p. 1).

[65] Gang Yu et al. "Optimizing pilot planning and training for continental airlines". In: *Interfaces* 34.4 (2004), pp. 253–264 (cited on p. 1).