CHALMERS | UNIVERSITY OF GOTHENBURG

MASTER'S THESIS

An Implementation of a Constraint Branching Algorithm for Optimally Solving Airline Crew Pairing Problems

Douglas Potter

Department of Mathematical Sciences
CHALMERS UNIVERSITY OF TECHNOLOGY
GOTHENBURG UNIVERSITY
Göteborg, Sweden 2008

Thesis for the Degree of Master of Science

An Implementation of a Constraint Branching Algorithm for Optimally Solving Airline Crew Pairing Problems

Douglas Potter

CHALMERS | UNIVERSITY OF GOTHENBURG

Department of Mathematical Sciences
Chalmers University of Technology and Gothenburg University
SE-412 96 Göteborg, Sweden
Göteborg, November 2008

Department of Mathematical Sciences
Göteborg 2008

Abstract

Competition in the airline industry depends greatly on how efficiently crews are scheduled. Scheduling problems can be modeled as integer programs which can be solved exactly using branch-and-price methods. However, in practice, in order to find a good schedule expediently, the branch-and-price tree is often only partially explored. A constraint branching heuristic called connection fixing often selects branches containing optimal or near-optimal solutions. This thesis investigates utilizing connection fixing in a branch-and-price algorithm to exactly solve airline crew scheduling problems.

We present a mathematical model for optimizing airline crew scheduling that is suitable for the branch-and-price algorithm. Then we present the branch-and-price method for solving integer programs, the connection fixing heuristic, and how this can be integrated into a branch-and-price method. Finally, we evaluate these ideas by implementing a branch-and-price system using connection fixing and use this system to solve exactly several small and medium sized crew scheduling problems. The numerical results suggest that the branch-and-price method with connection fixing is a promising method for exactly solving large-scale crew scheduling problems.

Acknowledgments

I would like thank my supervisor and examiner at Chalmers University of Technology, Ann-Brith Strömberg, for her support during the writing of this thesis. I would like to express my gratitude to many of the Jeppesen AB employees, especially my supervisor at Jeppesen AB, Lennart Bengtsson, for their guidance and insights into branch and price. I am also particularly indebted to Henrik Delin at Jeppesen AB who was very generous with his time, providing much programming advise. Finally, I would like to thank my parents for their support during these years in Sweden.

Contents

1 Introduction

This thesis is a proof of concept project implemented at Jeppesen AB that aims to evaluate the feasibility of utilizing a connection branching branch-and-price (B&P) system within the crew pairing component of crew scheduling. Crew pairing is the creation of shifts or work periods such that the flights are staffed. This differs from crew rostering, which is the assignment of specific individuals to these shifts [3] [7]. An example of this distinction is given in Chapter 2.

Crew costs are the second largest cost in the airline industry after fuel costs [3] [7]. Consequently, optimal or near-optimal crew scheduling is crucial for airline competitiveness. Determining an optimal crew schedule is extremely complex due to the large amount of human resources being managed, the varying and overlapping competences of the crew, the number of ways the crew can be matched to different tasks at different times and locations, governmental regulations, labor rules, worker preferences, and seniority rules [3]. Billions of scheduling combinations can arise from a few thousand flights [5]. Accordingly, for a specific scheduling problem, there may exist millions of solutions of varying cost for which all tasks will be completed while fulfilling all the labor rules and other constraints.

In order to manage the enormous complexity of crew scheduling and other optimization problems common to the transportation sector, many companies employ optimization products developed by Jeppesen AB. The process of solving large-scale scheduling problems at Jeppesen utilizes many optimization technologies. These technologies include algorithms based on the theoretical and applied mathematics of optimization. The *connection fixing* heuristic, which will be explained in Section 5.3, is one algorithm used in generating near-optimal solutions very quickly. Connection fixing is based roughly on the idea of searching for schedules that do or do not include a connection (for the crew) between two flights.

Currently, connection fixing is a part of the system Jeppesen uses to determine high quality crew pairings. Some of these crew pairing problems are intractable when approached with other methods, such as commercial integer programming solvers, due to their very large size. That such huge problems can be tackled at all is due in part to the efficiency of connection fixing. While every feasible solution (a schedule that does not violate any of the constraints) cannot be considered by the current process, it may be possible to utilize connection fixing ideas to exhaustively search small and medium sized pairing problems. Performing an exhaustive search is theoretically possible with traditional B&P approaches. However, the hope is that a B&P system branching on connections, i.e. connection branching, will lead to a time efficient exhaustive search. Time efficient methods for finding the best schedule (i.e. a global optimum) are of great importance since, in practice, the time between when a schedule planning begins and the schedule commences is limited.

In order to evaluate using connection branching as a basis for a B&P system, Jeppesen AB has provided the software and hardware requisite for developing the B&P system. We develop the connection branching B&P system mostly from freely available open source software. The primary software development tools are the GNU C and C++ compilers [14], the Mercurial software code management system [23], and the Eclipse C/C++ Development Tools Project [13]. The connection branching B&P system relies on two commercial products: the primal heuristic PAQS by Jeppesen AB, which is based on work by Wedelin [30], and the linear programming solver and the integer programming solver in XPRESS® by Fair Issac Co. [32]. Additionally, we write Python [25] scripts to analyze and visualize the data. These Python scripts use the following libraries or software packages: igraph [16], matplotlib [22], and Graphviz [15].

Chapter 2 describes the airline crew pairing problem in detail and situates it within airline industry scheduling. Furthermore, Chapter 2 discusses the role of optimization techniques and their applicability to airline industry scheduling problems. Suitable mathematical models for the airline crew pairing problem are developed in Chapter 3. Chapter 4 explains the B&P algorithm which is suitable for solving the models developed in Chapter 3. Chapter 5 explains how B&P is tailored in our implementation for solving the airline crew pairing problem. The results of using this implementation to solve several small and medium test problems are displayed in Chapter 6. We conclude with Chapter 7, summarizing our results and discussing directions for further research and development.

2 Optimization of Airline Crew Scheduling

This chapter aims to situate the crew pairing problem within the wider context of the airline industry as well as expand upon the description of the crew pairing problem presented in the introduction. We begin by describing the usage of optimization or operations research in the airline industry and then focus on the crew pairing problem.

The airline industry is faced with many complex business decisions. As with all industries, making such decisions is a matter of minimizing costs and maximizing profits. These businesses decisions can be modeled mathematically and expressed as mathematical optimization problems. Then the techniques and methods of mathematical optimization can be applied, aiding the decision making process.

The utility of employing optimization methods depends both on how well the mathematical model represents the actual business decisions and their related processes as well as on how well the model can be solved using optimization methods (the tractability of the model). Many factors affect the modeling of business decisions. For example, the complexity of the decisions, the quality of the input data, the interdependencies and the number of uncertainties all affect the modeling. The extensive use of optimization methods within the airline industry attests to the fact that the mathematical models adequately represent the business decisions in the airline industry [3] [7]. The difficulty is with the tractability of these models. Mathematical properties of these models such as discreteness, nonlinearity, and a huge number of variables characterize optimization problems that are difficult to solve [7]. This means that determining an optimal or near-optimal decision in terms of the model can be extremely taxing in terms of the computational resources, and can potentially exceed the available computing resources.

The crucial decisions in the airline industry are related to scheduling. Generally, 'scheduling' is deciding when and where certain people (e.g. pilots) and machines (e.g. aircraft) should carry out certain activities (e.g. piloting an aircraft, performing maintenance on an aircraft). Following the survey by Barnhart et al [7], scheduling in the airline industry can be decomposed into the sequence of four scheduling problems described in Table 2.

Each of these four scheduling problems is not really a separate problem since the input data for each scheduling problem is determined by the solution to the preceding one. For example, 'schedule design' determines which flights will be flown and thus which of the flights 'crew scheduling' needs to staff.

Table 1: Airline Scheduling Components

Scheduling Problem	Scheduling Problem Objective
Schedule Design	Scheduling flights to satisfy passenger demand for specific routes
Fleet Assignment	Scheduling the appropriate aircraft types to these flights
Aircraft Maintenance Routing	Scheduling aircraft so they are at maintenance sites with sufficient time and frequency
Crew Scheduling	Scheduling crew members to ensure that the flights have sufficient staff

Accordingly, the problems should properly be treated as one integrated problem in order to truly schedule optimally. Otherwise, the scheduling can go awry due to the fact that each scheduling problem constrains the following scheduling problem. In an extreme case, aircraft flights could be scheduled in a way that satisfies much of the passenger demand, but forces many crew members to fly many costly ”deadheads” (traveling to an other airport as a passenger, instead of being on active duty). A fully integrated model, integrating the four scheduling problems into a single model, would take account of such interdependencies. Unfortunately, integrated problems are extremely complex and can result in billions of *decision variables* (the variables in a mathematical optimization model that are to be optimally determined and correspond to real world decisions; these variables differ from *help variables* that are used, primarily for modeling purposes). Thus a common framework for airline scheduling is to work separately on the four scheduling problems while allowing for some feedback between them [3]. Feedback between the scheduling problems means that the impact of the schedules on each other is communicated between departments that plan each of the schedules. However, this feedback is not such that it fully integrates the scheduling problems. This thesis focuses on the separate, or sequential, framework.

In this sequential optimization process, crew scheduling comes last and thus crew scheduling is influenced by, but does not impact, schedule design, fleet assignment, and aircraft maintenance routing. Consequently, crew scheduling can focus purely on its own objectives: staffing cabin crews (e.g. flight attendants) and cockpit crews (e.g. pilots) such that all flights are properly staffed and all labor rules are satisfied at the lowest possible cost. An additional important factor not examined in this thesis is determining schedules that are robust. A robust schedule is a schedule that is easier and less expensive to adjust after a perturbation or a change, such as delays at an airport. Consequently, increasing the robustness of a schedule decreases the actual cost of a schedule. Since optimization removes slack and redundancies in the schedule, optimized schedules which do not consider robustness can be more sensitive to minor perturbations [7].

Finally, the crew scheduling problem is further divided into two subproblems: crew pairing and crew rostering. As mentioned in the introduction, crew pairing creates work periods that staff flights (pairing work periods with flights) and crew rostering assigns real individuals to the work periods (manning the crews or assigning people to the roster). For example, a crew pairing could dictate that an anonymous pilot A departs Minneapolis at 9:00 for New York city and departs New York city at 14:00 for Minneapolis and that an anonymous pilot B departs New York city at 8:00 for Minneapolis, departs Minneapolis at 11.30 for Los Angeles and finally departs Los Angeles for New York city at 16:30. Crew rostering could determine that the anonymous pilot A will be 'Jane Doe' and that the anonymous pilot B will be 'Bob Johnson'.

This subdivision means that the chosen crew pairing conditions rostering possibilities. Two reasons for breaking crew scheduling into these two subproblems are complexity reduction and delaying rostering. Considering the subproblems sequentially, solving crew pairing and then crew rostering, reduces the mathematical complexity and thus reduces the computational burden. However, as with dividing the airline scheduling problem into four parts, dividing crew scheduling into pairing and rostering means that not all feasible crew schedules will be considered. Since the integrated crew scheduling problem is more complex and computationally intensive than the rostering problem, splitting the crew scheduling makes it computationally easier. Concretely, this means that solving the crew rostering problem requires less time and planning resources than the integrate crew scheduling problem.

Due to complex pay structures of crews, determining a good crew pairing is not just a matter of scheduling such that all the flights are flown. Each task performed or flight piloted does not have a fixed cost, but instead the cost depends a number of factors such as the time worked during a duty period (a work day), rest time alloted, time away from the home base for the crew, and the duty performed. Furthermore, the salary is often calculated according to several different formulas of which the maximum is used to calculate the actual commensuration for a crew member. This entails that the costs per crew per task can be a complicated nonlinear function [6] [7].

Given the complexity of large crew pairings, finding a good pairing "manually" (that is, without using optimization software) is nearly impossible. Consequently, techniques of mathematical optimization are employed. In order to use such techniques, the pairing problem needs be formulated in terms of the mathematics of optimization.

3 Optimization Formulation of the Crew Pairing problem

Crew pairing is often formulated as a set partitioning problem or a set covering problem [3] [5] [6] [27]. In turn, the set partitioning problem and the set covering problem can be formulated as integer linear programs (ILP) which can be solved with optimization techniques such as B&B (branch and bound) [31] or B&P (branch and price) [5]. Section 3.1 precisely formulates the pairing problem so that it can be modeled mathematically in the subsequent sections. In Section 3.2, the set partitioning problem is formulated and the corresponding ILP formulation is explained. Furthermore, we elucidate how the crew pairing problems fits the set partitioning model. In Section 3.3, the set covering formulation will be explained along with some reasons as to why it is often the preferred formulation in practice. Finally in Section 3.4, a few slight amendments will be made to the set covering model so that it better models the pairing problem.

3.1 The Pairing Problem

In this section, the ground for mathematically modeling the crew pairing problem is cleared by precisely defining the crew pairing problem. This section concludes with Example 3.1 which describes a small pairing instance.

First, there are a number of flights that must be staffed. Each flight starts at a specific time at a specific airport and ends at a specific time at a specific airport. A pairing is a schedule for a crew (or crew member). A pairing consists of a sequence of flights which satisfies the labor rules, other regulations, and airline specific requirements. Furthermore, the sequence of flights in a pairing must start and end at the same airport, referred to as the 'home base'. This means that crew is returned to its home base at the end of the pairing. Every pairing i has a cost c_i. Determining the cost c_i for a particular pairing can be complicated and is outside of the scope of this thesis; some of the relevant factors were discussed at the end of Chapter 2.

Example 3.1 A Pairing Problem

This example shows the input data for a crew pairing problem. The flying times, airports, and the flight network have been chosen for notational simplicity rather than attempting to model an actual crew pairing problem. Table 2 list the flights and Figure 1 depicts the same flights. The pairings listed in Tables 3 and 4 are all the feasible pairings that can be formed with the flights. In order to understand the meaning of the pairings, consider the pairing 17 listed in Table 4: pairing 17 represents a crew (or a crew member) flying from London(LHR) via flight 1 to Gothenburg(GOT), then from Gothenburg(GOT) via flight 7 to Copenhagen(CPH), and finally from Copenhagen(CPH) via flight 5 back to London(LHR). Note also that it may be difficult to gauge the cost of an individual pairing. For example pairing 1 may seem at first overly expensive, but after considering that the working day for pairings 4, 7, and 8 are just as long as that of pairing 1 along with some other unknown cost factor, the cost of pairing 1 no longer seems unreasonable.

Table 2: Flights

Flight	From	To	Depart.	Arrival
1	LHR	GOT	9.30	12.00
2	GOT	LHR	17.00	19.30
3	GOT	CPH	12.30	13.00
4	CPH	GOT	15.30	16.00
5	CPH	LHR	17.00	19.15
6	LHR	CPH	8.00	10.15
7	GOT	CPH	12.30	13.00
8	CPH	GOT	13.30	14.00

Figure 1: Flight Network

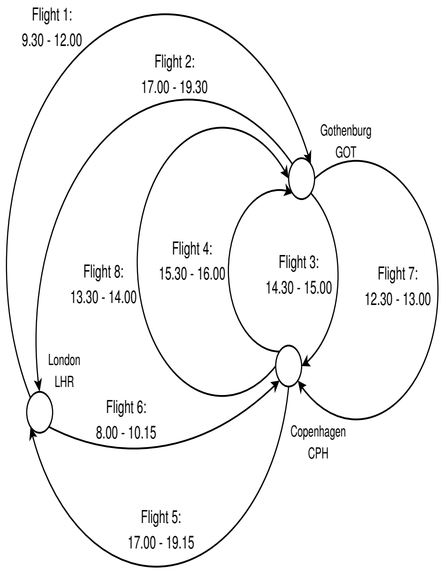


Table 3: Pairings 1-8

Pairing	Cost	Flight order
1	8	1-2
2	7	6-5
3	5	3-4
4	9	1-3-4-2
5	5	1-3-5
6	6	6-4-2
7	8	1-7-8-2
8	8	1-7-4-2

Table 4: Pairings 9-17

Pairing	Cost	Flight order
9	3	3-8
10	3	7-8
11	5	7-4
12	4	6-8-2
13	6	6-8-3-5
14	5	7-8-3-4
15	6	6-8-3-5
16	4	1-7-5
17	7	1-7-8-3-4-2

3.2 Set Partitioning Formulation

A set partitioning problem has two components: the ground set $\tilde{A}$ consisting of elements $\tilde{e}_j$ and a family $\tilde{F}$ of subsets $\tilde{a}_i \subset \tilde{A}$. Each $\tilde{a}_i$ is associated with a cost $c_i \in \Re_+$. The objective of a set partitioning problem is to select a set B of these subsets $\tilde{a}_i \in \tilde{F}$ such that B defines a partition of $\tilde{A}$ and that the sum of the costs c_i of the chosen subsets in B is minimized. The set B of the chosen subsets is a partition of $\tilde{A}$ if and only if $\cup_{\tilde{a}_i \in B} \tilde{a}_i = \tilde{A}$ and $\tilde{a}_i \cap \tilde{a}_j = \emptyset$ for all $\tilde{a}_i, \tilde{a}_j \ \in B$ such that $\tilde{a}_i \neq \tilde{a}_j$. In general, there is not any guarantee that a partition of $\tilde{A}$ exists; we will not consider this issue, since it cannot occur (in practice) with the final model used in this thesis.

We can formulate a finite set partitioning problem compactly as an ILP in the following way: let the ground set $\tilde{A}$ consist of m elements $\tilde{e}_j$ and let the family $\tilde{F}$ consist of n subsets, index the elements $\tilde{e}_j$ of the ground set $\tilde{A}$ with j such that $j = 1, \ldots, m$, index the subsets $\tilde{a}_i \in \tilde{F}$ with i for $i = 1, \ldots, n$ and let c_i, where $i = 1, \ldots, n$, represent the costs corresponding to each subset $\tilde{a}_i$. The decision vector $\mathbf{x}$ is defined to consist of $x_i \in \{0, 1\}$ for $i = 1, \ldots, n$ and indicates which subsets are chosen. If $x_i = 1$ then the subset i is selected, i.e. $i \in B$, and if $x_i = 0$ then the subset i is not selected, i.e. $i \notin B$. The vector $\mathbf{c} \in \Re^n_+$ is defined to consist of the given costs $c_i \in \Re_+5$ for $i = 1, \ldots, n$. We define an $m \times n$ matrix $\mathbf{A}$ that corresponds to the ground set $\tilde{A}$ and the partitioning subsets by

$$A_{ji} = \begin{cases} 1, & \text{if } \ \tilde{e}_j \in \tilde{a}_i, \\ 0, & \text{otherwise,} \end{cases} \qquad j = 1, \ldots, m, \quad i = 1, \ldots, n. \tag{1}$$

This gives the integer linear program

$$\begin{aligned} \min_{\mathbf{x} \in \{0,1\}^n} \ & \mathbf{c}^\mathsf{T} \mathbf{x}, \\ \text{s.t. } & \mathbf{A}\mathbf{x} = \mathbb{1}^m. \end{aligned} \tag{2}$$

Note that *s.t.* means ‘subject to’. Defining $\mathbf{a}_i \in \Re^m$ for $i = 1 \ldots n$ to be the column vectors of the matrix $\mathbf{A}$, we can reformulate (2) in terms of sums,

$$\begin{aligned} \min_{\mathbf{x} \in \{0,1\}^n} \ & \sum_{i=1}^{n} c_i x_i, \\ \text{s.t. } & \sum_{i=1}^{n} \mathbf{a}_i x_i = \mathbb{1}^m. \end{aligned} \tag{3}$$

It should be noted that although, together, the integrality constraints, $\mathbf{x} \in \{0,1\}^n$ and partitioning constraints $\sum_{i=1}^{n} \mathbf{a}_i x_i = \mathbb{1}^m$, of the problem (3) can be considered difficult, the *objective function*, $f(\mathbf{x}) = \sum_{i=1}^{n} c_i x_i$ is a linear function. Optimization models with linear objective functions are generally considered to be easier to handle than ones with nonlinear objective functions [4] [8].

The pairing problem as specified in Section 3.1 can be characterized as a set partitioning problem and thus modeled by the ILP (3). We consider all the flights that need to be staffed as the ground set in a set partitioning problem and note then that the pairings correspond to the subsets $\tilde{a}_i \in \tilde{F}$ and the columns $\mathbf{a}_i$ of the matrix $\mathbf{A}$. In terms of the ILP (2), we obtain that the $m \times n$ matrix $\mathbf{A}$, where m is the number of flights and n is the number of pairings, indicates the flights that each pairing contains. The n dimensional vector $\mathbf{x}$ consists of the decision variables, indicating which pairings are selected. This gives us the following definitions of $\mathbf{x}$ and $\mathbf{A}$:

$$x_i = \begin{cases} 1, & \text{if the pairing } i \text{ is used in the schedule,} \\ 0, & \text{otherwise,} \end{cases} \tag{4}$$

$$A_{ji} = \begin{cases} 1, & \text{if flight } j \text{ is in pairing } i, \\ 0, & \text{otherwise.} \end{cases} \tag{5}$$

Minimizing the ILP (3) with $\mathbf{x}$ and $\mathbf{A}$ defined as in (4) and (5) will then give the minimum cost schedule. This pairing formulation is often used since it avoids bringing the nonlinear crew costs into the objective function [3] [5] [27]. If the decision variable were, for example, the specific assignment of crews to tasks instead of choosing pairings, the objective function would be complicated, nonlinear, and difficult to optimize, see the discussion at the end of Chapter 2. It should be noted that not all the information about the pairings is preserved in the matrix $\mathbf{A}$. For instance, the temporal order of the flights is not contained in $\mathbf{A}$. The process of modeling a crew pairing problem as an ILP is shown in Example 3.2. This example will be further developed in the following chapters and sections in order to clarify various modeling decisions and algorithms.

Example 3.2 Set Partitioning Formulation

Using the definitions (4) and (5), the ILP for the crew pairing problem from Example 3.1 is constructed:

$$\min_{\mathbf{x}\in\{0,1\}^{17}} \begin{bmatrix} 8 & 7 & 5 & 9 & 5 & 6 & 8 & 8 & 3 & 3 & 5 & 4 & 6 & 5 & 6 & 4 & 7 \end{bmatrix}\mathbf{x},$$

$$\begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ . \\ . \\ . \\ . \\ x_{16} \\ x_{17} \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \qquad (6)$$

An optimal solution to this pairing problem is choosing the pairings $3, 12,$ and 16 giving a cost of 13. If the LP relaxation of (6)—replacing the integrality constraint $\mathbf{x} \in \{0,1\}^{17}$ with $\mathbf{x} \in [0,1]^{17}$—is considered instead, an optimal solution is $\mathbf{x} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0.5 & 0 & 0 & 0 & 0 & 0 & 0 & 0.5 & 0 & 0 & 0.5 & 0.5 \end{bmatrix}$ with a cost of 11.5. The solution to the LP relaxation corresponds to choosing half of each of the pairings $6, 13, 16,$ and 17. While in reality, a half pairing cannot be chosen, the LP relaxation solution gives a lower bound on the ILP and, as will be shown, plays an important role in solving the ILPs from crew pairing problems.

Although the set partitioning formulation and its ILP formulation (3) represent the crew pairing problem, these formulations have a computational deficiency. Often during the process of determining an optimal solution to an ILP, the integrality requirements on the decision variables are relaxed—this modification is known as *LP relaxing* and it transforms an ILP into a LP (linear program) known as the *LP relaxation*. In our case, this would mean that the vector $\mathbf{x}$ is allowed to take the values $[0,1]^n$ instead of $\{0,1\}^n$,

$$\begin{aligned} \min_{\mathbf{x}\in[0,1]^n} \ & \sum_{i=1}^{n} c_i x_i, \\ \text{s.t.} \ & \sum_{i=1}^{n} \mathbf{a}_i x_i = \mathbb{1}^m. \end{aligned} \qquad (7)$$

Problematically, the equality constraints make the problem (7) numerically instable and difficult to solve, especially when compared to the set covering formulation [5], which is discussed in the next section and alleviates these difficulties without requiring extra variables.

3.3 Set Covering Formulation

The set covering formulation is almost identical to the set partitioning formulation. The difference is that the requirement that every element in the ground set be in one and only one of the selected subsets is replaced by that every element in the ground set is in at least one of these subsets. This difference is shown in Example 3.3, which formulates the crew pairing problem from Example 3.2 as set covering problem.

We formulate the set covering problem in such a way that the equivalent IP formulation is clear. The finite set covering problem is, given a finite ground set $\bar{A}$ containing elements $\bar{e}_j$ where $j = 1, \ldots, m$, and a family subsets $\bar{a}_i \in \bar{F}$ with costs c_i, where $i = 1, \ldots, n$, such that $\bar{a}_i \subset \bar{A}$, find a minimum cost set $\bar{B}$ of the subsets such that the union of the subsets contains every element in the ground set $\bar{A}$.

As with the set partitioning problem, the decision vector $\mathbf{x}$ is defined to consist of $x_i \in \{0, 1\}$ for $i = 1, \ldots, n$. The vector $\mathbf{x}$ indicates which subsets are selected. If $x_i = 1$ then the subset i is selected and if $x_i = 0$ then the subset i is not selected. Furthermore, we can define a $m \times n$ matrix $\mathbf{A}$ to represent $\bar{A}$ by

$$A_{ji} = \begin{cases} 1, & \text{if } \bar{e}_j \in \bar{a}_i, \\ 0, & \text{otherwise,} \end{cases} \quad j = 1, \ldots, m, \quad i = 1, \ldots, n. \tag{8}$$

Letting $\mathbf{a}_i$ for $i = 1, \ldots, n$ be the column vectors of $\mathbf{A}$, we can formulate the set covering problem as an ILP,

$$\begin{aligned} \min_{\mathbf{x} \in \{0,1\}^n} \; & \sum_{i=1}^{n} c_i x_i, \\ \text{s.t.} \; & \sum_{i=1}^{n} \mathbf{a}_i x_i \geq \mathbb{1}^m. \end{aligned} \tag{9}$$

Since this formulation has inequality constraints instead of equality constraints, it is more stable numerically [5]. It is always trivial to find a feasible solution to the problem (9) and the corresponding LP relaxed problem. A trivial solution can be constructed by selecting pairings that contain uncovered flights (flights which are not in any of the pairings chosen so far) until all flights are covered. A flight that is overcovered, that is, a flight that is in two or more of the selected pairings, indicates where a deadhead is in the schedule. Note that a solution to the set partitioning problem is also a solution to the corresponding set covering problem and that a solution to the set covering problem without overcovers is also a solution to the corresponding set partitioning problem. It may even be possible be transform a solution to the set covering problem with overcovers to a partitioning solution. If the requirement that a covering subset or pairing must constitute a sequence of flights (such that each flight transports pilots to the next flight, starting and ending at the home base) could be discarded, it would be trivial to remove overcovers to create a partitioning solution—but then we would loose the practical applicability to the crew pairing problem.

Example 3.3: Set Covering Formulation

The crew pairing problem described in Example 3.2 can also be modeled as a set covering problem as this Section explains. Stating the corresponding ILP in the same form as (9) yields:

$$\min_{\mathbf{x}\in\{0,1\}^{17}} \begin{bmatrix} 8 & 7 & 5 & 9 & 5 & 6 & 8 & 8 & 3 & 3 & 5 & 4 & 6 & 5 & 6 & 4 & 7 \end{bmatrix}\mathbf{x},$$

$$\begin{bmatrix} 1&0&0&1&1&0&1&1&0&0&0&0&0&0&0&1&1\\ 1&0&0&1&0&1&1&1&0&0&0&1&0&0&0&0&1\\ 0&0&1&1&1&0&0&0&1&0&0&0&1&1&1&0&1\\ 0&0&1&1&0&1&0&1&0&0&1&0&0&1&0&0&1\\ 0&1&0&0&1&0&0&0&0&0&0&0&1&0&1&1&0\\ 0&1&0&0&0&1&0&0&0&0&0&1&1&0&1&0&0\\ 0&0&0&0&0&0&1&1&0&1&1&0&0&1&0&1&1\\ 0&0&0&0&0&0&1&1&1&1&0&1&1&1&1&0&1 \end{bmatrix} \begin{bmatrix} x_1\\ x_2\\ .\\ .\\ .\\ .\\ x_{16}\\ x_{17} \end{bmatrix} \geq \begin{bmatrix} 1\\1\\1\\1\\1\\1\\1\\1 \end{bmatrix}. \qquad (10)$$

An optimal solution to this pairing problem is choosing the pairings $12, 14$ and, 16 giving a cost of 13. This solution has overcovers and would be infeasible in the model (6) from Example 3.2, since the left hand sides of the flight constraints 7 and 8 equal 2—that is for the solution $\mathbf{x} = \begin{bmatrix} 0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,1\,0\,1\,0\,1\,0 \end{bmatrix}$,

$$\begin{bmatrix} 1&0&0&1&1&0&1&1&0&0&0&0&0&0&0&1&1\\ 1&0&0&1&0&1&1&1&0&0&0&1&0&0&0&0&1\\ 0&0&1&1&1&0&0&0&1&0&0&0&1&1&1&0&1\\ 0&0&1&1&0&1&0&1&0&0&1&0&0&1&0&0&1\\ 0&1&0&0&1&0&0&0&0&0&0&0&1&0&1&1&0\\ 0&1&0&0&0&1&0&0&0&0&0&1&1&0&1&0&0\\ 0&0&0&0&0&0&1&1&0&1&1&0&0&1&0&1&1\\ 0&0&0&0&0&0&1&1&1&1&0&1&1&1&1&0&1 \end{bmatrix} \begin{bmatrix} 0\\ \vdots\\ 0\\ 1\\ 0\\ 1\\ 0\\ 1\\ 0 \end{bmatrix} = \begin{bmatrix} 1\\1\\1\\1\\1\\1\\2\\2 \end{bmatrix}. \qquad (11)$$

Consequently, some crew will fly ‘deadhead’ on flights 7 and 8. Note, although the partitioning solution found in Example 3.2 is also optimal in (10), the set covering formulation does not distinguish between overcovering solutions and partitioning solutions. In Section 3.4 we augment the set covering model so that overcovers can be given a cost.

The LP relaxation of the problem (10) has an optimal solution $\mathbf{x} = \begin{bmatrix} 0\,0\,0\,0\,0\,0.5\,0\,0\,0\,0\,0\,0\,0.5\,0\,0\,0.5\,0.5 \end{bmatrix}$ with a cost of 11.5. This solution does not contain any overcovers and is the same solution as that found to the set partitioning formulation in Example 3.2. That these LP relaxed solutions are identical is not unexpected, since the possibility of dividing pairings in the LP relaxation, allows (unrealistically flexible) combinations of pairings so that the constraints are satisfied with equality.

3.4 The Augmented Set Covering Formulation

The set covering formulation (9) represents the pairing problem well and thus when minimized gives a good schedule as long as two modeling assumptions are met: overcovers in the solution are actually feasible and not prohibitively expensive and there are no important interdependencies between the pairings. The nature and source of these interdependencies is explained in this section.

Since overcovers result in costly deadheads (which occupy a passenger seat), we want to avoid them if possible. This is accomplished by penalizing overcovers with a non-negative penalty vector $\mathbf{f} \in \Re^m_+$ where m is the number of flights. This gives the overcover penalized covering problem,

$$\begin{aligned} \min_{\mathbf{x}\in\{0,1\}^n} \quad & \sum_{i=1}^{n} c_i x_i, + \mathbf{f}^{\mathsf{T}} \left(\sum_{i=1}^{n} \mathbf{a}_i x_i - \mathbb{1}^m \right), \\ \text{s.t.} \quad & \sum_{i=1}^{n} \mathbf{a}_i x_i \geq \mathbb{1}^m. \end{aligned} \tag{12}$$

The elements of the vector $\sum_{i=1}^{n} \mathbf{a}_i x_i - \mathbb{1}^m$ indicate how much the corresponding flights (or elements in the ground set) are over-covered. If every flight is covered once, i.e. the set covering is also a set partition, then $\sum_{i=1}^{n} \mathbf{a}_i x_i - \mathbb{1}^m = \mathbf{0}^m$, and thus the overcover cost penalty is not incurred. Example 3.4.1 illustrates penalizing overcovers by augmenting Example 3.3 with overcover penalties.

Example 3.4.1: Penalizing Overcovers

The crew pairing problem described in Example 3.2 can be modeled in several ways. The model in Example 3.3 gives a solution with overcovers. If overcovers incur an additional cost or are undesirable, they can be penalized using an overcover penalized set covering model as shown in (12). In this example, every flight constraint is given an overcover penalty cost of 10. Thus in terms of (12), $\mathbf{f}^{\mathsf{T}} = [\, 10\ 10\ 10\ 10\ 10\ 10\ 10\ 10 \,]$. Recalling that $\mathbf{a}_i$ is a column of the constraint matrix, $\mathbf{f}^{\mathsf{T}} \left(\sum_{i=1}^{17} \mathbf{a}_i x_i - \mathbb{1}^8 \right) = [\, 20\ 20\ 20\ 40\ 30\ 30\ 40\ 50\ 20\ 20\ 20\ 30\ 40\ 40\ 40\ 30\ 60 \,]\mathbf{x} - 80$ is calculated. Adding this term to the objective function in Example 3.3, the ILP is stated in the same form as (12):

$$\min_{\mathbf{x} \in \{0,1\}^{17}} \quad [\, 28\ 27\ 25\ 49\ 35\ 36\ 48\ 58\ 23\ 23\ 25\ 34\ 46\ 45\ 46\ 34\ 67 \,]^{\mathsf{T}} \mathbf{x} - 80,$$

$$\begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 1 \end{bmatrix}^{\mathsf{T}} \begin{bmatrix} x_1 \\ x_2 \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ x_{16} \\ x_{17} \end{bmatrix} \geq \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \tag{13}$$

An optimal solution to this pairing problem is choosing pairings $3, 12$ and 16, which results in no overcovers, and has a cost of 13. This is the same solution as that found for the set partitioning model in Example 3.2. The overcovering solution with pairings $12, 14$, and 16 found in Example 3.3 would now cost 33, which explains why this solution, while still feasible, is not chosen. The LP relaxation of this problem (13) has an optimal $\mathbf{x} = [\, 0\ 0\ 0\ 0\ 0\ 0.5\ 0\ 0\ 0\ 0\ 0\ 0\ 0.5\ 0\ 0\ 0.5\ 0.5 \,]$ with a cost of 11.5. This solution is identical to the solution found to the LP relaxations in Examples 3.2 and 3.3.

Furthermore, there can exist interdependencies that further restrict the combinations of the pairings that may be chosen. Pairings are interdependent due to the resources that some pairings use and the nature of these resources. For example, there often exist minima and maxima on the number of pairings with certain home bases that can be selected since the total staff is limited and often guaranteed a minimum amount of work at a given home base [11].

The restrictions on how the pairings can and cannot be combined in a schedule are known as the *GLC*s (**GL**obal **C**onstraints). Enforcing the GLCs is accomplished by adding *soft constraints*. A definition and discussion of soft constraints is found in [4, Section 1.8]. The following soft constraints are added: $\mathbf{Bx} - \mathbf{y} \leq \mathbf{d}$ where $\mathbf{B}$ is an integer $q \times n$ matrix, q is the number of GLCs, and n is the number of pairings in the original overcover-penalized covering problem; the matrix $\mathbf{B}$ is defined as follows: $B_{ki} =$ quantity of resource k used by pairing i; the vector $\mathbf{d} \in \Re^q_+$ defines the soft maximum usage of resource k for $k = 1, \ldots, q$—that is, the maximum usage allowed without incurring any additional cost; the variable vector $\mathbf{y} \in \Re^q_+$ brings the amount exceeding the maximum usage into the objective function.

Accordingly, the penalty term $\mathbf{p}^\mathsf{T}\mathbf{y} = \sum_{k=1}^{q} p_k y_k$, where $p_k > 0$, is the penalty cost per unit of using additional amount of resource i, is added to the objective function. For notational simplicity, we let $\mathbf{b}_i \in \mathbb{Z}^q$ for $i = 1 \ldots n$ be the column vectors of $\mathbf{B}$. Combining GLCs with the overcover-penalized covering problem, we obtain the following MILP (mixed integer linear program),

$$
\begin{aligned}
\min_{\mathbf{x}\in\{0,1\}^n\ \mathbf{y}\in\Re^q_+} \quad & \sum_{i=1}^{n} c_i x_i + \mathbf{f}^\mathsf{T}\left(\sum_{i=1}^{n} \mathbf{a}_i x_i - \mathbb{1}^m\right) + \sum_{k=1}^{q} p_k y_k, \\
\text{s.t.} \quad & \sum_{i=1}^{n} \mathbf{a}_i x_i \geq \mathbb{1}^m, \\
& \sum_{i=1}^{n} \mathbf{b}_i x_i - \mathbf{y} \leq \mathbf{d}.
\end{aligned}
\tag{14}
$$

Note: It may not always seem natural to model certain restrictions on the ways pairings can be combined with "$\leq$" constraints as shown in (14) (e.g. a minimum number of pairings with a certain home base. However, a single GLC k that would naturally be modeled by $\sum_{i=1}^{n} B_{ki} x_i + y_k \geq d_k$ (e.g. a minimum number of pairings with a certain home base) can be modeled as explained above by multiplying each constant in such a constraint by -1 giving $-\sum_{i=1}^{n} B_{ki} x_i - y_k \leq -d_k$. The role of GLCs is demonstrated in Example 3.4.2.

Example 3.4.2: Adding GLCs

The crew pairing problem described in Example 3.2 is modeled in this example as a penalized set covering problem (see Example 3.4.1) augmented with GLCs. The restriction that no more than two pairings can be chosen without incurring extra cost is imposed. This restriction is modeled by defining that every pairing uses 1 unit of crew and imposing a soft maximum crew supply of 2 and a penalty cost of 20 for every crew unit that exceeds the soft maximum.

In order to adapt the model (13) from Example 3.4.1, we need to determine the penalty terms $\sum_{k=1}^{q} p_k y_k$ and the soft constraints $\sum_{i=1}^{n} \mathbf{b}_i x_i - \mathbf{y} \leq \mathbf{d}$ for the model (14), which are, respectively, $20y_1$ and $[1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1]\mathbf{x} - y_1 \leq 2$. Adding this term to the model from 3.4.1 yields,

$$\begin{aligned}
\min_{\substack{\mathbf{x}\in\{0,1\}^{17} \\ y_1 \geq 0}} \quad & [28\ 27\ 25\ 49\ 35\ 36\ 48\ 58\ 23\ 23\ 25\ 34\ 46\ 45\ 46\ 34\ 67]\,\mathbf{x} + 20y_1 - \quad 80 \\
& \begin{bmatrix}
1&0&0&1&1&0&1&1&0&0&0&0&0&0&0&1&1\\
1&0&0&1&0&1&1&1&0&0&0&1&0&0&0&0&1\\
0&0&1&1&1&0&0&0&1&0&0&0&1&1&1&0&1\\
0&0&1&1&0&1&0&1&0&0&1&0&0&1&0&0&1\\
0&1&0&0&1&0&0&0&0&0&0&0&1&0&1&1&0\\
0&1&0&0&0&1&0&0&0&0&0&1&1&0&1&0&0\\
0&0&0&0&0&0&1&1&0&1&1&0&0&1&0&1&1\\
0&0&0&0&0&0&1&1&1&1&0&1&1&1&1&0&1
\end{bmatrix}
\begin{bmatrix} x_1\\ x_2\\ \cdot\\ \cdot\\ \cdot\\ \cdot\\ x_{16}\\ x_{17} \end{bmatrix}
\geq
\begin{bmatrix} 1\\1\\1\\1\\1\\1\\1\\1 \end{bmatrix} \\
& [1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1]\,\mathbf{x} - y_1 \quad \leq \quad 2\ .
\end{aligned} \tag{15}$$

In this model, the solution with pairings $3, 13,$ and 17 found in Examples 3.2 and 3.4.1 increases in cost from 13 to 23. An optimal solution to the model (15) is choosing the pairings 2 and 17 which gives a cost of 14 with $y_1 = 0$. The LP relaxation of this problem (15) gives the solution $\mathbf{x} = [0\ 0\ 0\ 0\ 0\ 0\ 0.5\ 0\ 0\ 0\ 0\ 0\ 0\ 0.5\ 0\ 0\ 0.5\ 0.5]$ with a cost of 11.5 and $y_1 = 0$.

Formulation (14) is a MILP since it contains variables both with and without the integrality requirements. In this case, $\mathbf{x}$ has the integrality requirement and $\mathbf{y}$ lacks the integrality requirement. Also note that the structure of the GLCs implies that the choice of $\mathbf{x}$ completely determines $\mathbf{y}$ but not vice versa. The vector $\mathbf{x}$ contains the decision variables and the vector $\mathbf{y}$ contains help variables (for a discussion of help variables, or non-decision variables, see [4, Remark 1.4]). The fact that all the decision variables are integral implies that we can treat the MILP (14) as an Integer Program (IP),

$$\begin{aligned}
\min_{\mathbf{x}\in\{0,1\}^n} \quad & g(\mathbf{x}), \\
\text{s.t.} \quad & \sum_{i=1}^{n} \mathbf{a}_i x_i \geq \mathbb{1}^m
\end{aligned} \tag{16}$$

where

$$\begin{aligned} g(\mathbf{x}) := \min_{\mathbf{y}\in\Re^q_+} \quad & \sum_{i=1}^{n} c_i x_i + \mathbf{f}^\mathsf{T} \left(\sum_{i=1}^{n} \mathbf{a}_i x_i - \mathbb{1}^m \right) + \sum_{k=1}^{q} p_k y_k, \\ \text{s.t.} \quad & \sum_{i=1}^{n} \mathbf{b}_i x_i - \mathbf{y} \leq \mathbf{d}. \end{aligned} \tag{17}$$

This formulation, while not linear, is quite simple. Note also that the function $g(\mathbf{x})$ is a convex function. The choice of using formulation (16)–(17) or formulation (14) varies depending on the solution method to be applied. More importantly, the existence of these two formulations makes it logically consistent to discuss the crew pairing model with overcover penalties and GLCs as the MILP (14) or the IP (16)–(17). This is equivalence is important for Section 4.1, which explains B&B algorithm, for the crew pairing problem, in terms an IP.

The problem (14) captures the essential characteristics of the crew pairing problem—that is, it models the essential characteristics of the problem, and given accurate input data, should yield staffed aircraft at a lowest possible cost. However, two crucial issues remain: How can a large integer program be solved? and Which pairings should be considered? This latter question is particularly important since handling all possible pairings explicitly is intractable for many real-world problem instances and considering fewer but good pairings makes solving (14) easier. The next chapter explains the B&P algorithm that can resolve these issues.

4 Solving Integer Programs with Branch and Price

Finding optimal or near-optimal solutions to integer linear programs (ILP) has been the subject of much research since many important decision problems from industry can be formulated as ILPs [29] [31]. While the ability to succinctly model many problems is valuable, it does not itself lead to high quality solutions. The computational resources required to solve one particular ILP depends greatly on the choice of the algorithm.

Some algorithms classified as approximation algorithms run in polynomial time, but can only guarantee near-optimal solutions; other algorithms classified as exact algorithms guarantee optimal solutions but often do not run in polynomial time [19]. Accordingly, the computational complexity of algorithms differ. While the theoretical complexity of algorithms is an important topic, it is beyond the scope of this thesis. The problem at hand is determining a practical algorithm for finding an optimal solution to the pairing problem as formulated in Section 3.4. This being said, it should be noted that the set covering problem is NP-complete and thus should temper expectations for quick solution methods [19].

The exact algorithm for solving ILPs that we will focus on is the B&P algorithm. The appearance of this algorithm in the literature dates back to the 1980s when it was first used to solve routing and scheduling problems [12] [31]. The column generation roots of B&P are found in the early work of Ford and Fulkerson on multi-commodity flows, see [21].

The B&P algorithm combines the ideas from B&B (branch and bound) and column generation. Combining these ideas is not straightforward and a time-efficient implementation is problem specific. We follow a common tact of explaining B&B and column generation algorithms and then explain B&P in terms of these two algorithms. The following exposition assumes a basic understanding of general optimization theory and linear programming. The requisite theory is covered in depth in [4], [10], [18] and [28].

4.1 Branch and Bound

Through algebraic manipulation (as shown in Section 3.4), the final set covering MILP (14) can be equivalently formulated as the following general IP with n variables and m constraints

$$\begin{aligned} \min_{\mathbf{x}\in\mathbb{Z}^n} \quad & g(\mathbf{x}), \\ \text{s.t.} \quad & \sum_{i=1}^{n} \mathbf{a}_i x_i \ \geq \mathbf{h}, \end{aligned} \tag{18}$$

where $\mathbf{a}_i, \mathbf{h} \in \Re^m$ are given. Note that the requirement on the variables, $\mathbf{x} \in \{0,1\}^n$ present in (16), can be incorporated into the constraints $\sum_{i=1}^{n} \mathbf{a}_i x_i \geq \mathbf{h}$ and $\mathbf{x} \in \mathbb{Z}^n$ found in (18). Given that the IP (18) is bounded, we can theoretically determine the optimal solutions by enumerating and evaluating all its feasible solutions (all solutions vectors that satisfy all the constraints of an optimization problem). For most problems, this simple method is clearly not tenable due to the size of the solution space (the set of feasible solutions). Even if the problem of enumerating the solution space is ignored, the remaining part of the algorithm—evaluating the feasible solutions—is not a polynomial-time algorithm since the size of the solution space as function of the number of decision variables, n, grows faster than any polynomial for large values of n [19]. B&B is a method for dealing with the size of the solution space by systematically partitioning the feasible set (set of feasible solutions or the solution space) in such a way that portions can successively be discarded until an optimal solution is found.

B&B begins by relaxing the integrality constraint on $\mathbf{x}$ in the IP (18) and partitioning the solution space of the relaxed problem to create a number of subproblems. For each subproblem, the feasible set is a partition constructed by restricting the solution space in some way—meaning that the feasible set for each subproblem is a subset of the feasible set of the original relaxed problem. Before explaining further, we consider these subproblems: a subproblem can never attain a lower objective value (the value of the objective function for a given solution values) than the original relaxed problem since the feasible set of a subproblem is a subset of the feasible set of the original relaxed problem. Moreover, if the optimal solution to the original problem is also in the feasible set of a subproblem, then the optimal objective value of a subproblem cannot be higher than that of the original problem.

We can discard or prune a subproblem if it is infeasible, if its solution is greater than or equal to a known integer solution to the original problem, if its optimal solution is integral, or if all the subproblem's subproblems have be discarded; these pruning conditions are referred to, respectively, as 'pruned by infeasibility', 'pruned by the bound', 'pruned by optimality', and 'pruned by its children' [31]. The condition referred as 'pruned by optimality' implies that a subproblem can be discarded since if a subproblem has an integer solution, this solution is also a solution to the original IP problem [31]. The latter condition gives rise to the tree structure of a B&B search—this tree of subproblems is referred to as the *search tree*.

By repeating this process of creating subproblems, eventually the optimal solution will be found and all subproblems will be pruned—proving the optimality of the lowest integral solution. This method is called branch and bound since subproblems known as *branches* are created that are cut off by a *bound*—the best integral solution available [31] [24]. In this thesis, the terms branches, subproblems and nodes are used synonymously. A node that is branched upon to create new subproblems, is called a parent. The subproblems created by branching on a parent are called children. The parent of a node, the parent of the parent of a node, etc are considered the ancestors of a node; and the children of a node, and the children of the children of a node, etc are descendants of a node. The node that is the ancestor to all other nodes, i.e. the node whose subproblem that is the LP relaxation of the original IP, is referred to as the root node. Example 4.1 demonstrates how the B&B algorithm can be used to solve the crew pairing model (6) presented in Example 3.2.

Example 4.1: B&B

This example demonstrates how B&B can be used to solve the ILP (6) from Example 3.2:

$$\min_{\mathbf{x}\in\{0,1\}^{17}} \quad [8\ 7\ 5\ 9\ 5\ 6\ 8\ 8\ 3\ 3\ 5\ 4\ 6\ 5\ 6\ 4\ 7]\,\mathbf{x},$$

$$\begin{bmatrix} 1&0&0&1&1&0&1&1&0&0&0&0&0&0&0&1&1\\ 1&0&0&1&0&1&1&1&0&0&0&1&0&0&0&0&1\\ 0&0&1&1&1&0&0&0&1&0&0&0&1&1&1&0&1\\ 0&0&1&1&0&1&0&1&0&0&1&0&0&1&0&0&1\\ 0&1&0&0&1&0&0&0&0&0&0&0&1&0&1&1&0\\ 0&1&0&0&0&1&0&0&0&0&0&1&1&0&1&0&0\\ 0&0&0&0&0&0&1&1&0&1&1&0&0&1&0&1&1\\ 0&0&0&0&0&0&1&1&1&1&0&1&1&1&1&0&1 \end{bmatrix} \begin{bmatrix} x_1\\ x_2\\ .\\ .\\ .\\ .\\ x_{16}\\ x_{17} \end{bmatrix} = \begin{bmatrix} 1\\1\\1\\1\\1\\1\\1\\1 \end{bmatrix}. \qquad (19)$$

This ILP can be solved with a B&B algorithms which branch either on the variables or the constraints. Since this ILP has binary decision variables, branching on variables is simple and is accomplished creating subproblems which successively fix variables to either 0 or 1. An example of this is variable branching scheme is shown in Figure 2. The variables that are fixed in any subproblem can been seen in the figure by following the path up the tree from a subproblem back to the root node.

Branching on the constraints is a bit more complicated. The subproblems created satisfy two constraints, i.e. constraint g and constraint h, in two different ways: in one subproblem only pairings that satisfy both or neither g nor h are considered and in the other subproblem only pairings that only contain only one of g or h or contain neither g nor h. These branchings are denoted respectively by $\mathbf{both(g,h)}$ and $\mathbf{dif(g,h)}$. This branching scheme is demonstrated in Figure 3. For more details concerning this constraint branching scheme see the description of Ryan-Foster branching in Section 4.3.

Figure 2: Variable Branching

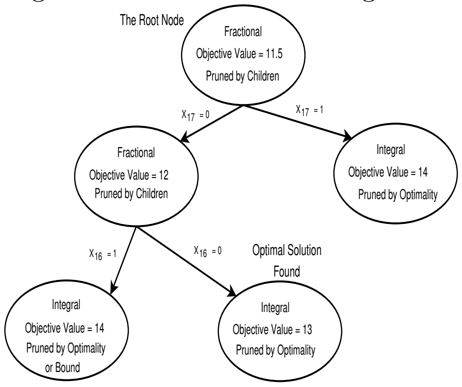
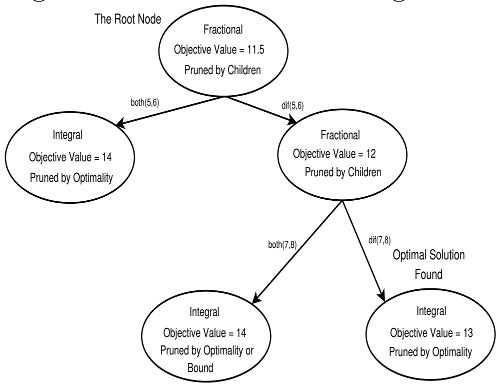


Figure 3: Constraint Branching



It is worthwhile to note that an optimal solution was found relatively early in both branching trees. The similarity in number of nodes required and geometry of the branching trees in the variable and constraint branching are specific to this instance and the branching choices and not to the branching schemes in general.

At first glance, B&B seems simple, but in practice, determining the branches—that is, how the feasible set should be partitioned—is nontrivial. Choosing, for example, whether the B&B algorithm branches over the values the variables take (i.e. variable branching) or over how the constraints are satisfied (i.e. constraint branching), see 'restrictions on subsets' in [5]) can affect both the difficulty of solving the subproblems and how well B&B can work within other frameworks. Example 4.1 demonstrates both of these branching alternatives. It is possible that in pathological cases a B&B algorithm will not be able to cut off any branches until all subproblems have been solved—this means that what we set out to avoid, evaluating every feasible solution, will have occurred [17]. Further, determining the order in which the branches are explored is critical, since good integral solutions may be discovered along the way [1] [9] [31]. These good integral solutions, which give upper bounds on the optimal objective value, let us discard subproblems, thus avoiding unnecessary branching.

Once the above issues are resolved for a particular class of ILPs, B&B can efficiently find the optimal solutions. However B&B does not suffice for the crew pairing problem. This is because there are usually far too many possible pairings which would make the B&B subproblems very large, requiring an unacceptably long time to optimize. Thus, we must select or generate good subsets of the possible pairings in some way. Section 4.2 covers the column generation algorithm which resolves the pairing generation issue.

4.2 Column Generation

This section explains column generation in the context of LP. It also builds on general LP theory, such as the theory of the simplex method. This general LP theory is covered in [4], [10], [18], and [28]. For more details concerning column generation, the classic book by Lasdon [20] is an excellent source. For a review of recent developments in column generation, see Lübbecke and Desrosiers [21]. Although the end goal is to use column generation for solving ILPs within a B&P framework, this section only covers the column generation in terms of LP problems. This suffices since the goal is to explain the role of column generation within B&P algorithms which use column generation to solve the LP relaxations of huge ILPs.

Consider the following LP relaxation of (18) which will be referred to as the *master problem* (MP):

$$\begin{aligned} z_{MP} = \min_{x \in \Re^n} \quad & \sum_{i \in I} c_i x_i, \\ \text{s.t.} \quad & \sum_{i \in I} \mathbf{a}_i x_i \geq \mathbf{h}, \end{aligned} \tag{20}$$

where $\mathbf{a}_i, \mathbf{h} \in \Re^m$. We assume that the feasible set of defined by (20) is bounded. In the MP (20), the index set I indexes costs c_i, variables x_i in the objective function, and the associated constraint columns $\mathbf{a}_i$. The MP (20) may be intractable due to the large number of variables indexed in I. This may happen, for instance, if the algorithm, used to solve (20), requires more memory than available. The way around this impasse is to only consider a subset of the variables and columns. Thus instead of considering I, consider $I' \subset I$. This gives the *restricted master problem* (RMP) to the MP (20):

$$\begin{aligned} z_{RMP} = \min_{x \in \Re^n} \quad & \sum_{i \in I' \subseteq I} c_i x_i, \\ \text{s.t.} \quad & \sum_{i \in I' \subseteq I} \mathbf{a}_i x_i \geq \mathbf{h}, \end{aligned} \tag{21}$$

where $\mathbf{a}_i, \mathbf{h} \in \Re^m$. Given that I' is a sufficiently small subset of I, the RMP is tractable. Of course, the optimal solution to the RMP (20) is not generally optimal in the MP (20). Since every feasible solution in the RMP is also feasible in the MP, it holds that $z_{RMP} \geq z_{MP}$. However, for large-scale problems, the optimal solution to the MP is sparse, or in other words, most of the optimal x_i's for $i \in I$ are equal to zero. This means if we can correctly select I', the optimal solutions to the RMP will be optimal solutions to the MP and the RMP will remain tractable.

The RMP can be successively updated to solve the MP using column generation: first select an initial subset $I' \subseteq I$ such that the RMP is feasible. The RMP is then solved using the simplex method and the optimal dual solution vector $\mathbf{u} \in \Re^m_+$ is saved [20, Chapter 4]. Using the terminology of the revised simplex method, we define the reduced cost vector $\bar{\mathbf{c}} \in \Re^{|I \setminus I'|}$ to consist of the reduced costs $\bar{c}_i = c_i - \mathbf{u}^{\mathsf{T}}\mathbf{a}_i$ for $i \in I \setminus I'$ [28]. Then we solve the *pricing subproblem*,

$$\min_{i \in I \setminus I'} \{\bar{c}_i := c_i - \mathbf{u}^{\mathsf{T}}\mathbf{a}_i\}. \tag{22}$$

This can be solved by enumeration—that is calculating all $\bar{c}_i$ and selecting the minimum. Define $\bar{c}^* := \min\{\bar{c}_i | i \in I \setminus I'\}$. If $\bar{c}^* \geq 0$, then the RMP (21) optimally solves the MP (20) since there are no columns indexed in I that, when added to the RMP, can improve its objective value. On the other hand, if $\bar{c}^* < 0$, then the RMP (21) does not optimally solve the MP (20). Thus, in this case, the RMP is updated by adding to I' some indices $i \in I \setminus I'$ such that $\bar{c}_i < 0$; doing this ensures that the optimal value z_{RMP} of the RMP (21) will decrease, [20, Chapter 4].

Now the RMP (21) is resolved and the *pricing subproblem* is repeated. This process of solving the RMP (21) and the *pricing subproblem* continues until $\bar{c}^* \geq 0$ or, in other words, until the optimal solution to the RMP (21) is also optimal to the MP (20). This method for solving the MP (20) is exact and finite if the implementation of the simplex method does not cycle [21]. Solving the *pricing subproblem* is known as *column generation* since it generates columns $\mathbf{a}_i$. Finally, note that in the context of the crew pairing problem, generating columns corresponds to generating pairings.

Since solving the *pricing subproblem* is computationally less demanding than solving the MP (20) (provided that the columns $\mathbf{a}_i$ are efficiently generated), column generation allows us to tackle large LP problems. This explanation of column generation has omitted many important details such as how the initial set I' is determined, the role of numerical precision, methods for determining the dual vector $\mathbf{u}$, and different methods for selecting a good subset of the columns with negative reduced cost. These and other important details are covered in the references listed at the beginning of this section.

Before proceeding to B&P, a few comments pertaining to the set covering model for crew scheduling developed in Section 3.4 are in order: solving the *pricing problem* by enumeration as presented may be overly burdensome. Perhaps generating $\mathbf{a}_i$ for all $i \in I$ is intractable in itself—determining the feasibility of an $\mathbf{a}_i$ can be complicated, as it is for the crew pairing problem—or solving the *pricing subproblem* is computationally prohibitive. Both of these issues present a serious difficulty when solving medium and large crew scheduling problems. There are various schemes such as partial pricing that can alleviate such difficulties [10]. Moreover, for many *pricing subproblems*, there exists an equivalent compact or combinatorial formulation that, instead of calculating the reduced costs $\bar{\mathbf{c}}$, generates some columns with a negative reduced cost $\bar{c}_i$. Often the compact formulation is much easier to solve than the standard *pricing subproblem* (22) when I is large. This is especially true when the compact formulation is a standard combinatorial optimization problem such as the *knapsack problem* or, in the case of our model for crew pairing, a *constrained shortest path problem.* The connection between the constrained shortest path problem and the crew pairing MP is laid out in [5] and [7].

When deciding which columns to add the RMP, there usually exist many columns $\mathbf{a}_i$ such that $\bar{c}_i < 0$. The traditional least reduced cost pricing rule is to add the column with the least reduced cost $\bar{c}_i$. However, for the LP relaxations of set partitioning and covering problems, the *lambda pricing rule* works particularly well [21]. Assuming positive costs c_i, the lambda pricing rule adds the columns from the set

$$\arg\min_{i \in J} \frac{c_i}{\mathbf{u}^\mathsf{T}\mathbf{a}_i} \tag{23}$$

where

$$J = \left\{ i \in I \middle| \mathbf{u}^\mathsf{T}\mathbf{a}_i > 0 \text{ and } \frac{c_i}{\mathbf{u}^\mathsf{T}\mathbf{a}_i} < 1 \right\}. \tag{24}$$

The lambda pricing rule is used in our implementation and results in about approximately 40% less columns being generated. For both the pricing rules, multiple columns such that $\bar{c}_i < 0$ can be added. Although adding more than one column may add unnecessary columns to the RMP—unnecessarily increasing its size—and making it more difficult to solve, the number of times the RMP and *pricing subproblem* needs to be solved is typically reduced [5]. Consequently, there is a trade-off between increasing the size of the RMP and the number of times the *pricing subproblem* must be solved. The optimal setting depends on the computational cost of solving a large RMP versus the computational cost of solving the *pricing subproblem.*

4.3 Branch and Price

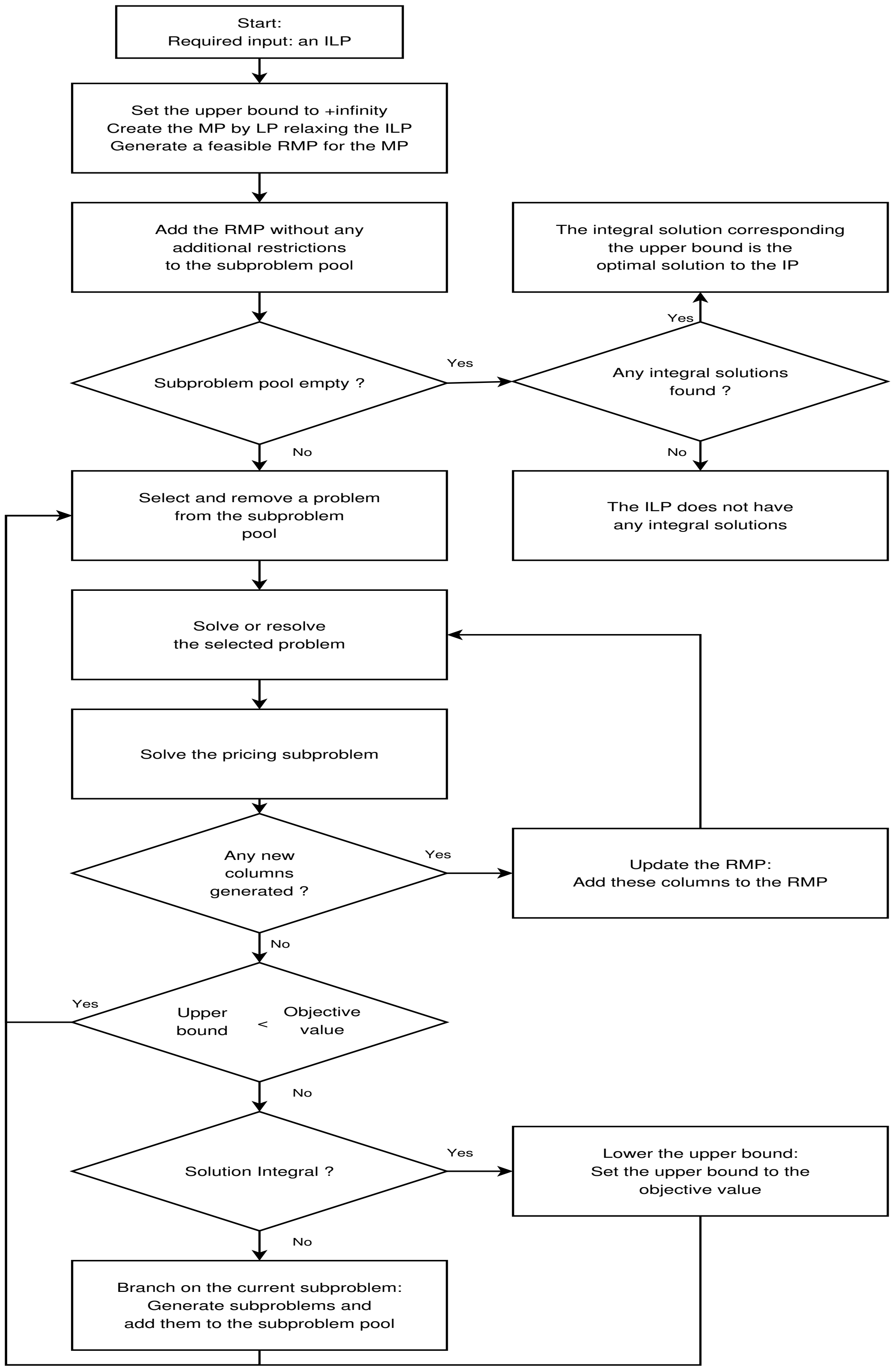
B&P is a combination of B&B and column generation. B&B provides the machinery for determining optimal integral solutions to an ILP. Column generation supplies B&B with the variables and the corresponding constraint matrix columns—or in our case, crew pairings—of the ILP (the master problem) that need to be considered. The branching rule for the B&B algorithm must be chosen such that it can incorporate these new variables and constraint matrix columns as they are generated. This usually means that we forego variable branching, employing constraint branching instead since column generation does not change the constraints—all pairings generated must satisfy the same set of constraints and GLCs.

For instance, given a set partitioning problem with several rows, we can summarize a branching rule suitable for B&P called the Ryan-Foster branching [26]. This rule is based on the fact that if the solution to the LP relaxation is not integral, there exists at least two columns whose associated variables are fractional. Variables that have values that are not integer are known as *fractional variables*. Furthermore, there must exist rows r and s such that two of these *fractional columns*—columns in the constraint matrix corresponding to fractional variables—have a 1 in row s and only one of them contains a 1 in row r. From this fact follows the branching rule: one subproblem should contain all columns (of those already generated) which have a 1 in rows r and s and the other subproblem should contain no column has a 1 in both r and s. Note that some columns with 0 in both rows r and s are in both subproblems. These two subproblems can be branched upon in the same way. Consequently, we can apply the B&B procedure using the Ryan-Foster branching.

Once we have an appropriate branching rule, such as Ryan Foster branching, the B&P algorithm begins by using column generation to solve the LP relaxation of the ILP—this LP relaxation becomes the RMP (21). Then, B&B is used, applying the branching rule, to create a number of subproblems. In order to find the solution to each of these subproblems, we apply column generation. It is crucial that the columns generated comply with the restrictions specific to each subproblem. Furthermore, it is possible that for columns currently in the RMP, the RMP has become infeasible for subproblems. In this case, the definition of solving the *pricing subproblem* is widened to scanning the MP for columns that would restore feasibility. If feasibility is restored, the normal pricing can proceed, otherwise the subproblem is infeasible. Now we return to B&B to decide whether a subproblem should be discarded or branched upon. The entire B&P algorithm is depicted in Figure 4. Note that in Figure 4, one algorithmic detail is ambiguous: whether every subproblem has its own RMP or there is a global RMP shared by all subproblems. Actually, both these design choices are valid and the choice depends on the ILP and the computational environment.

Enforcing the restrictions specific to a branch and its subproblem during column generation is not always trivial since the enforcement may entail examining all the columns generated or, if using a compact formulation, complex modifications to the *pricing subproblem* might need to be made. This means that the branching rule must be compatible with the *pricing subproblem*. Chapter 5, we present a branching rule for our pairing model that is compatible with the constrained shortest path problem [6].

Figure 4: B&P algorithm for minimizing an ILP


Start:
Required input: an ILP
Set the upper bound to +infinity
Create the MP by LP relaxing the ILP
Generate a feasible RMP for the MP
Add the RMP without any additional restrictions to the subproblem pool
Subproblem pool empty ?
Yes
Any integral solutions found ?
Yes
The integral solution corresponding the upper bound is the optimal solution to the IP
No
The ILP does not have any integral solutions
No
Select and remove a problem from the subproblem pool
Solve or resolve the selected problem
Solve the pricing subproblem
Any new columns generated ?
Yes
Update the RMP:
Add these columns to the RMP
No
Upper bound < Objective value
Yes
No
Solution Integral ?
Yes
Lower the upper bound:
Set the upper bound to the objective value
No
Branch on the current subproblem:
Generate subproblems and add them to the subproblem pool

5 Branch and Price Utilizing Connection Branching

This chapter describes the B&P system that we use to solve crew pairing problems. The system exactly solves the crew pairing model (14) presented in Section 3.4. Throughout this chapter, we assume that solutions to the pairing problems and various subproblems do not contain overcovers; this assumption can be enforced, in most cases, by adjusting the overcover penalties, see Section 3.4. While this assumption is not actually necessary for the B&P system to work properly, it greatly simplifies the presentation.

Section 5.1 discusses a branching rule for B&P, referred as *connection branching*, which suits the crew pairing problem (14). Section 5.2, which is more applied, discusses how to resolve some issues that arise in conjunction with actually using the connection branching rule. These first two sections cover details that would need to be addressed in any B&P system using connection branching. Section 5.3, is a slight digression which outlines how the ideas in this chapter can be used to construct a heuristic solver for the paring problem, while Section 5.4 discusses details specific to our implementation.

5.1 Connection Branching: the Branching Rule

This section explains connection branching by first defining a *connection* and introducing some related terminology and then showing how connections can define a constraint branching rule. The notation that is developed in this section is intended to give a precise description of which pairings are allowed in each branch or node during branch and price. This notation is summarized in Table 5. The connection branching rule is compatible with the constrained shortest path *pricing subproblem.* This branching rule is used in our implementation and it is related to the 'branch-on-follow-on' rule used by Vance et al [27].

Table 5: Connection Branching Notation

Symbol	Description
P	the set of all known feasible pairings
$\textsc{pairing}_i$	a pairing, if $\textsc{pairing}_i \in P$ then $\textsc{pairing}_i$ is feasible.
$\textsc{conx}_{gh}$	if $g \neq h$, then flight g follows flight h in some pairing if $g = h$, then flight g is the last flight in some pairing
H_g	the set of all flights that follow flight g in some feasible pairing or g itself if some pairing ends with flight g; or symbolically, $h \in H_g$ if there exists a $\textsc{pairing}_i \in P$ such that $\textsc{conx}_{gh} \in \textsc{pairing}_i$
P_{gh}	the set of all feasible pairings that have $\textsc{conx}_{gh}$; symbolically, $P_{gh} := \{\textsc{pairing}_i \in P \vert \textsc{conx}_{gh} \in \textsc{pairing}_i\}$
B_{gh}	the set of all feasible pairings that cannot be in any solution without overcovers that contains a pairing from P_{gh}
P^+_{gh}	the set of all feasible pairings that have $\textsc{conx}_{gh}$ or do not contain flights g or h; symbolically, $P^+_{gh} := P \setminus B_{gh}$
$\tilde{P}^+_{gh}$	an expanded version of P^+_{gh} for B&P which includes the pairings P^+_{gh} plus any pairings generated during pricing that include a connection $\textsc{conx}_{gk}$ such that $k \notin H_g$

For convenience, the crew pairing problem (14) is restated here in terms of an index set P that indexes all the pairings and their costs, a set J that indexes all the flight constraints and a set K that indexes all the GLC's variables,

$$\begin{aligned} \min_{\mathbf{x}\in\{0,1\}^n\ \mathbf{y}\in\Re^q_+} \quad & \sum_{i\in P} c_i x_i + \mathbf{f}^\mathsf{T}\left(\sum_{i\in P}\mathbf{a}_i x_i - \mathbb{1}^m\right) + \sum_{k\in K} p_k y_k, \\ \text{s.t.} \quad & \sum_{i\in P}\mathbf{a}_i x_i \geq \mathbb{1}^m, \\ & \sum_{i\in P}\mathbf{b}_i x_i - \mathbf{y} \leq \mathbf{d}. \end{aligned} \tag{25}$$

Connection branching is based on the idea that a pairing is composed of a number of connections. A *normal* connection consists of two flights such that in some pairing the start flight (temporally) precedes the end flight without any other flights occurring in between the two—symbolically, this is expressed by $\textsc{conx}_{gh}$ such that $g \neq h$ and flight g immediately precedes flight h in the pairing. This relationship of immediately preceding or following is expressed with the terminology: flight g precedes flight h or flight g is the precedent of flight h; analogously, flight h follows flight g or flight h is the follower of the flight g. The last flight in a pairing is also considered a connection—symbolically written $\textsc{conx}_{gg}$ where flight g is the last flight in the pairing. The purpose of including this additional *last flight* connection will become clear when the branching rule is explained.

Together, these two kinds of connections define how the flights g and h in a pairing i form connections:

$$\textsc{conx}_{gh} \in \textsc{pairing}_i \text{ if } \begin{cases} g \neq h \text{ and } g \text{ precedes } h \text{ in } \textsc{pairing}_i, \text{ or,} \\ g = h \text{ and } g \text{ has no followers in } \textsc{pairing}_i. \end{cases} \tag{26}$$

The connections of a pairing completely specify the flights in a pairing. Connections can specify information not preserved in the constraint matrix A of the pairing problem (14) because connections represent the temporal order of the flights. This can be useful for cyclic schedules in which the connections, $\textsc{conx}_{gh}$ and $\textsc{conx}_{hg}$, both occur. Furthermore, all the flights in a pairing occur exactly in two connections (contained by this pairing), except for the first flight in the pairing which is only in the first connection. Table 6 concretely illustrates what connections are by the adding connection information for each pairing found in Table 3 from Example 3.

Table 6: Pairings 1-8 with Connections

Pairing	Cost	Flight order	Connections
1	8	1-2	$\textsc{conx}_{12}$ $\textsc{conx}_{22}$
2	7	6-5	$\textsc{conx}_{65}$ $\textsc{conx}_{66}$
3	5	3-4	$\textsc{conx}_{34}$ $\textsc{conx}_{44}$
4	9	1-3-4-2	$\textsc{conx}_{13}$ $\textsc{conx}_{34}$ $\textsc{conx}_{42}$ $\textsc{conx}_{22}$
5	5	1-3-5	$\textsc{conx}_{13}$ $\textsc{conx}_{35}$ $\textsc{conx}_{55}$
6	6	6-4-2	$\textsc{conx}_{64}$ $\textsc{conx}_{42}$ $\textsc{conx}_{22}$
7	8	1-7-8-2	$\textsc{conx}_{17}$ $\textsc{conx}_{78}$ $\textsc{conx}_{82}$ $\textsc{conx}_{22}$
8	8	1-7-4-2	$\textsc{conx}_{17}$ $\textsc{conx}_{74}$ $\textsc{conx}_{42}$ $\textsc{conx}_{22}$

In order to understand how connections are used in the branching rule, we consider first an integral solution vector $\mathbf{x}$ to the crew pairing problem (14) without any overcovers. The pairings that are in the solution, i.e. pairings i such that $x_i = 1$, are referred to as the *solution pairings*. Given a flight g that has some followers in some pairings, we define the set of flights H_g to contain these followers plus g itself:

$$H_g := \{j \in J \mid \textsc{conx}_{gj} \in \textsc{pairing}_i \text{for some } \textsc{pairing}_i,\ i \in P\}. \tag{27}$$

The set of pairings that include the $\textsc{conx}_{gh}$ is denoted by P_{gh},

$$P_{gh} := \{i \in P \mid \textsc{conx}_{gh} \in \textsc{pairing}_i\}. \tag{28}$$

Exactly one of $\textsc{conx}_{gh}$ such that $h \in H_g$ can be in the set of solution pairings since otherwise the flight constraint g would be either overcovered or uncovered. Of course, if the flight g does not have any followers in the solution pairings, then $H_g = \{g\}$ and $\textsc{conx}_{gg}$ must be in the solution. We can assume that some flights have followers, since otherwise each pairing would contain only one flight (resulting in a trivial problem). Thus given a fixed flight g and $\{h_1, \ldots, h_n\} = H_g$, then the sets

$$P_{gh_1}, \ldots, P_{gh_n}, \tag{29}$$

represent all the ways the flight constraint g can be satisfied. Here there may arise questions concerning if some $\textsc{pairing}_j$ containing a connection $\textsc{conx}_{fg}$ such that $g \neq f$ have been neglected. However, given such a $\textsc{pairing}_j$, either g is the last flight or g has some followers; in the first case, $\textsc{conx}_{gg} \in \textsc{pairing}_j$ by definition (26) and thus $\textsc{pairing}_j \in P_{gg} \subset \bigcup_{h \in H_g} P_{gh}$ or in the second case, g has a follower l and thus again by definition (26), $\textsc{pairing}_j \in P_{gl} \subset \bigcup_{h \in H_g} P_{gh}$ since $l \in H_g$ holds.

Moreover, the fact that a flight g with some followers can only have one follower $h \in H_g$ per pairing gives the following relationship for a fixed g,

$$P_{gj} \cap P_{gk} = \emptyset, \qquad j \neq k, \quad j, k \in H_g. \tag{30}$$

It is clear that sets like P_{gh} can form part of the basis for the connection branching rule. However, we cannot use P_{gh} alone for grouping the pairings into sets since the union of all P_{gh} for a fixed g is not generally equal to all the pairings, i.e. $\bigcup_{h \in H_g} P_{gh} \neq P$. This is because there are more constraints to satisfy than just for flight g. Consequently, P_{gh} such that $h \in H_g$ for a fixed g groups just a small portion of the feasible pairings. Continuing with the idea of assigning pairings according to connections, we define B_{gh} to be the set of the pairings that *break* a connection $\textsc{conx}_{gh}$,

$$\begin{aligned} B_{gh} := \{i \in P | \; & i \in P_{gj} \text{ for some } j \neq h \quad \text{or} \\ & i \in P_{kh} \text{ for some } k \neq g \quad \text{or} \\ & \textsc{pairing}_i \text{ starts with } h \; \}. \end{aligned} \tag{31}$$

Given a solution without overcovers, if a $\textsc{conx}_{gh}$ is contained in some pairing among the solution pairings, then the pairings in B_{gh} cannot occur in the solutions pairings. Hence, pairings B_{gh} break a solution containing $\textsc{conx}_{gh}$. The definition (31) of B_{gh} implies for a fixed g,

$$\bigcap_{h \in H_g} B_{gh} = \emptyset. \tag{32}$$

Let $P^+_{gh} := P \setminus B_{gh}$. Concretely, the set P^+_{gh} contains the only pairings that permit solutions that include $\textsc{conx}_{gh}$. The definition of P^+_{gh} implies for any flights g and h that

$$P_{gh} \subseteq P^+_{gh} \tag{33}$$

and for a fixed flight g that

$$P_{gj} \not\subseteq P^+_{gh}, \qquad j \neq h, \quad j, h \in H_g \tag{34}$$

Thus for a fixed g,

$$\bigcup_{h \in H_g} P^+_{gh} = \bigcup_{h \in H_g} (P \setminus B_{gh}) = P \setminus \bigcap_{h \in H_g} B_{gh} = P. \tag{35}$$

The relationship (33) implies that for every flight g, the sets of pairings P^+_{gh} for $h \in H_g$ contains the pairings P such that each P_{gh} includes all the pairings that contain connection $\textsc{conx}_{gh}$; the relationship (34) implies that each P_{gh} does not contain any other connections starting with flight g. Finally, the relationship (35) implies that for a fixed g, P^+_{gh} such that $h \in H_g$ groups the feasible pairings into sets such that the solution can only be in one of them.

Armed with P^+_{gh} and the definition of H_g (27), we can state the connection branching rule for the crew pairing problem (25): choose a flight g such that $|H_g| > 1$ and then create a subproblem or branch for each $\textsc{conx}_{gh}$ such that $h \in H_g$ by replacing P with $P \cap P^+_{gh}$. Each subproblem can be further branched upon if $|H_g| > 1$ for some other flight g. Now the set of pairings, P, for each subproblem is some P_{gh}^{+} in terms of the original problem. When branching again on some of the subproblems, P will become an even smaller set. For instance, if for one of the P_{gh} subproblems, flight j is then branched upon creating a (sub)subproblem P_{jk}, then P for this subproblem in terms of the original problem is $P \cap P_{gh} \cap P_{jk}$. In this way, all the ways of satisfying the flight constraints will be explored. Consequently, the pairings considered in any subproblem in the search tree can be characterized as

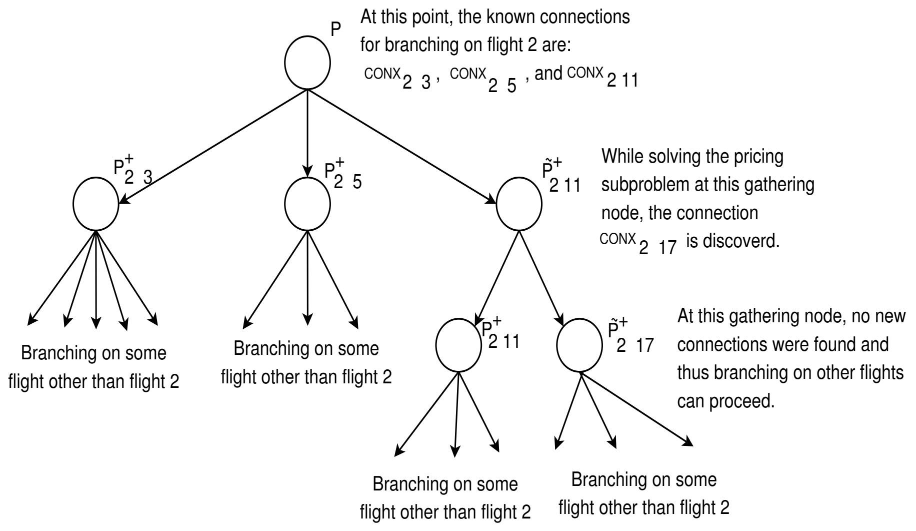
$$P \cap P_{g_1h_1}^{+} \cap P_{g_2h_2}^{+} \cap \ldots \cap P_{g_nh_n}^{+} \neq \emptyset, \tag{36}$$

where $h_1 \in H_{g_1}, h_2 \in H_{g_2}, \ldots, h_n \in H_{g_n}$.

In practice, not all the possible (or relevant) connections for a given crew pairing problem will be known in advance. Usually only some of the possible connections are known during B&P. This situation can be handled in the following manner: when branching on a start flight g, one of the branches is designated as a gathering node or branch. The subproblem for the gathering branch is defined by $\tilde{P}_{gh}^{+}$ which includes the usual P_{gh}^{+} plus all pairings containing any other unknown connections beginning with the start flight g. If we let $\textsc{conx}_{gh}$ such that $h \in H_g$ be the known connections for a start flight g and let $\tilde{H}_g$ contain all the flights h such that $\textsc{conx}_{gh}$ is a connection, we can formally define $\tilde{P}_{gh}^{+} := P_{gh}^{+} \cup \{i \in P | \textsc{pairing}_i \in P_{gk}^{+} \text{ for some } k \in \tilde{H}_g \setminus H_g\}$. Including pairings in $\tilde{P}_{gh}^{+}$ lets pricing implicitly consider unknown connections. In fact, if not all the possible connections are known, every branching must include one gathering node.

Properly handling gathering nodes, illustrated in Figure 5, is simple. If while solving the *pricing subproblem* for a gathering node for which the subproblem is defined by $\tilde{P}_{gh}^{+}$, some new connections $\textsc{conx}_{gi_1}, \ldots \textsc{conx}_{gi_n}$ are discovered and the node is not pruned, then we choose the node that has branched on multiple connections. The branching for this node is, accordingly, to create nodes for each of the discovered connections $\textsc{conx}_{gi_1}$, $\ldots$, $\textsc{conx}_{gi_n}$ as well as the original $\textsc{conx}_{gh}$, and then to designate one of these nodes as a gathering node.

Figure 5: Gathering Node Example


P
At this point, the known connections for branching on flight 2 are: CONX 2 3, CONX 2 5, and CONX 2 11
P+ 2 3
P+ 2 5
P̃+ 2 11
While solving the pricing subproblem at this gathering node, the connection CONX 2 17 is discoverd.
P+ 2 11
P̃+ 2 17
At this gathering node, no new connections were found and thus branching on other flights can proceed.
Branching on some flight other than flight 2
Branching on some flight other than flight 2
Branching on some flight other than flight 2
Branching on some flight other than flight 2

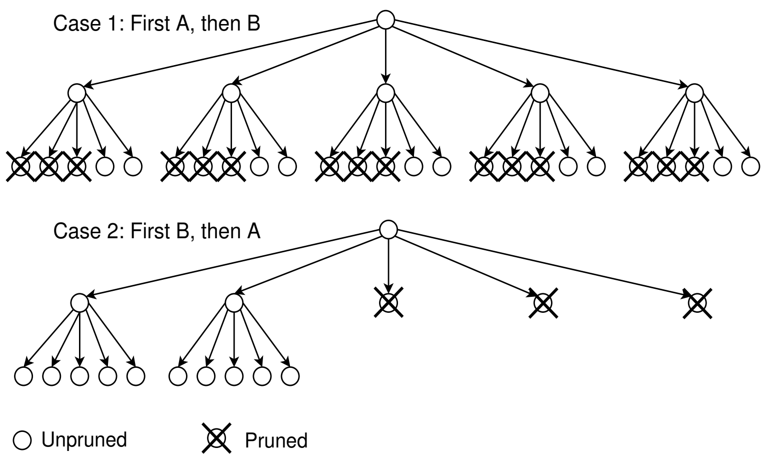
The discussion above concerning gathering nodes opens up the possibility of grouping together already known connections CONX_{gh} such that $h \in H_g$ for a given flight g. For example, the subproblems $P^+_{gh_2}$ and $P^+_{gh_4}$ of a branching on subproblems $P^+_{gh_1}$, $P^+_{gh_2}$, $P^+_{gh_3}$, $P^+_{gh_4}$, and $\tilde{P}^+_{gh_5}$ could be combined to produce the following subproblems: $P^+_{gh_1}$, $P^+_{gh_2} \cup P^+_{gh_4}$, $P^+_{gh_3}$, and $\tilde{P}^+_{gh_5}$. If the subproblem $P^+_{gh_2} \cup P^+_{gh_4}$ is not pruned, then it will be branched upon to produce the subproblems $P^+_{gh_2}$ and $P^+_{gh_4}$. Whether or when combining subproblems is advantageous is a heuristic decision that is related to the selection and ordering of branches discussed in Section 5.2. Some issues related to combining branches are explicitly considered in Section 5.4.

5.2 Connection Branching: Selection and Ordering

The connection branching rule, developed in Section 5.1, supplies a branching rule that is closely related to the structure of the crew pairing problem. While such a close relationship is often prerequisite for a successful B&P system, there remain two main unresolved issues: determining which branches should be formed for an unpruned node, i.e. a branch or node selection rule, and determining the order in which branches should be explored, i.e. a branch or node ordering rule. This section presents some approaches for resolving these issues. Some of the approaches presented have a firm mathematical foundation, while others are rooted in praxis.

A branch selection rule is needed since for most crew pairing problems, the inequality $|H_g| > 1$ usually holds for several choices of g. Thus there are several branching choices for every unpruned branch. Selection rules, together with the branching rule, dictate how the space of feasible pairings is partitioned, i.e. defines the feasible set for each subproblem. In turn, how the space of feasible pairings is partitioned determines both when good integral solutions are found, and the total number of subproblems that must be solved to guarantee that the optimal solution has been found [1] [9] [31]. A reduction of the total number of subproblems can be achieved by branching such that many of the branches will be pruned by the upper bound when solved.

Figure 6: Extreme Effects of Branch Selection



In order to understand how selection rules affect the number of subproblems, consider the following example illustrated in Figure 6. For a hypothetical ILP, there exist two different branching choices, A and B, which can be chosen independently of each other and both the branching choices A and B result in five new subproblems. Thus, if no pruning occurs, applying A and then B or applying B and then A both result in 30 subproblems. However, in practice, the pruning of branches makes branch selection noncommutative. Now consider two hypothetical cases using A and B. In the first case, a problem is branched using A into five subproblems and none of these five subproblems is pruned; then each of these subproblems is branched using B into five new (sub)subproblems of which three are pruned. This leaves ten unpruned subproblems. In the second case, the branching choices are reversed: a problem is branched using B into five subproblems and three of these are pruned and then for each of the two remaining subproblems, five new branches are created using A. None of these new branches are pruned so the same ten unpruned subproblems that case one finds remain. In the first case, 30 subproblems were solved but in the second case, only 15—yet, both cases make the same amount of progress. While branch selection decisions do not always lead to such a dramatic reduction in the number of the branches, the effect is usually palpable [1] [9].

Determining the order in which the branches are explored is crucial since an apt ordering explores more nodes with good integral solutions sooner. In turn, good integral solutions provide good bounds and thus make it possible to prune branches sooner—further accelerating the search for the optimal solution and proving optimality [9] [31].

Developing selection and ordering rules for connection branching requires a measure upon which such rules can be based. A measure called *connection weight*, denoted for a connection $\textsc{conx}_{gh}$ by $\mathbf{W}_{gh}$ is related to the inclusion of a given connection in a solution. A connection weight $\mathbf{W}_{gh}$ is defined in terms of the LP relaxation of problem (25),

$$\begin{aligned} \min_{\mathbf{x}\in[0,1]^n\ \mathbf{y}\in\Re^q_+} \quad & \sum_{i\in P} c_i x_i + \mathbf{f}^{\mathsf{T}}\left(\sum_{i\in P}\mathbf{a}_i x_i - \mathbb{1}^m\right) + \sum_{k\in K} p_k y_k, \\ \text{s.t.} \quad & \sum_{i\in P}\mathbf{a}_i x_i \ \geq \mathbb{1}^m, \\ & \sum_{i\in P}\mathbf{b}_i x_i - \mathbf{y} \ \leq \mathbf{d}. \end{aligned} \tag{37}$$

The connection weight is defined as

$$\mathbf{W}_{gh} := \sum_{i\ \in P_{gh}} \bar{x}_i \tag{38}$$

where $\bar{x}_i,\ i \in P$ is an optimal solution to (37).

Recalling the discussion in Section 5.1, particularly the definitions (26) and (28) and the equation (30), connection weights indicate the inclusion of certain connections in a solution. If $\mathbf{W}_{gh} > 0$ then $\textsc{conx}_{gh}$ is represented in the optimal solution to (37)—that is, there exists an $i \in P$ such that $\textsc{conx}_{gh} \in \textsc{pairing}_i$ and $x_i > 0$. Moreover, for a solution without overcovers, definitions (26), (28), and (38), imply that the pairing solution is integral if and only if $\mathbf{W}_{gh} \in \{0, 1\}$ for all $\textsc{conx}_{gh}$ and the pairing solution is not integral if and only if $\mathbf{W}_{gh} \in (0, 1)$ for some $\textsc{conx}_{gh}$. For proofs of these results, see [27].

Another consequence of the definitions (28), (26), and (38) is that for a pairing solution without overcovers, given a fixed flight constraint g, the following holds

$$\sum_{h \in H_g} \mathbf{W}_{gh} = 1. \tag{39}$$

The above results suggest utilizing connection weights to heuristically find good integral solutions as well as reduce the number of subproblems. Finding good integral solutions heuristically is accomplished with a selection rule, referred to as the 'high weight' rule, branches on a flight g for which $\mathbf{W}_{gh}$ is near 1 for some $h \in H_g$—creating subproblems by restricting P to P^+_{gh} such that $h \in H_g$ and associating each subproblem with the $\mathbf{W}_{gh}$—and an ordering rule that prioritizes branches associated with high connection weights. Mathematically, the 'high weight' rule branches on flight t such that

$$t \in \arg \max_{q \in \{r \ | \ |H_r| > 1\}} \left\{ \max_{s \in \{v \in H_q \middle| \mathbf{W}_{qv} < 1\}} \{\mathbf{W}_{qs}\} \right\}, \tag{40}$$

where $|H_r|$ denotes the cardinality of H_r, defined by (27). Note that while branching on P^+_{gh}, where $\mathbf{W}_{gh} = 1$, will reduce the space of pairings, it will not change the solution of the RMP since branching P^+_{gh} effectively forces $\mathbf{W}_{gh} = 1$ in that branch. Thus branching on such P^+_{gh} is avoided at this stage, since such a branching cannot perturb the solution towards integrality. The 'high weight' rule is supported by the work of Vance et al [27] and Barnhart et al [5].

Assuming that the best upper bound has been found, the reduction of the number of subproblems (that will be solved before B&P terminates) is independent of the ordering rule and depends solely on the selection rule. A first attempt at such a selection rule might be to branch on a flight g such that some branches corresponding to near-zero $\mathbf{W}_{gh}$ are created. While such near-zero branches are often pruned, the equation (39) implies that if all but one $\mathbf{W}_{gh}$ is near-zero, then one branch corresponding to $\textsc{conx}_{gh}$ such that $h \in H_g$ will have a near-one connection weight which probably will not be pruned. These unpruned branches problematically resemble their parents to such an extent that B&P has not made any significant progress. In fact, since some $\mathbf{W}_{gh}$ is near one, this first attempt has recreated the 'high weight' rule.

In practice, there usually exists other branching choices that, while resulting in fewer pruned children per branching, actually reduce the total number of nodes required. Given a flight g such that many $\mathbf{W}_{gh}$ are near zero, there often exists a branching that causes $\mathbf{W}_{gh} \in \{0, 1\}$ for $h \in H_g$ in their associated subproblems and thus removing the option of branching on g. An explanation for this phenomenon may be that branching choices for near-zero $\mathbf{W}_{gh}$ correspond to a relatively small portion of the feasible set and after pruning such branches, most of the solution space remains feasible in the unpruned branch. On the other hand, choosing to branch on a flight e that has more balanced values of $\mathbf{W}_{eh}$ for $h \in H_e$—meaning for all $h \in H_e$, $\mathbf{W}_{eh}$ are neither near zero nor one—may create larger, more balanced, partitions of the feasible set. Furthermore, the smaller portions of the feasible (created when branching on g) may lie in one of the larger partitions (created when branching on e), so the subproblem corresponding to one of these larger partitions is pruned, some of the $\mathbf{W}_{gh}$ may become integral or branching on $\textsc{conx}_{gh}$ may not longer be possible, i.e. $P^+_{ef} \cap P^+_{gh} = \emptyset$. In either of these two cases, some of the branching possibilities corresponding to $\textsc{conx}_{gh}$ such that $h \in H_g$ have been eliminated.

A selection rule for reducing the number of subproblems, in lieu of the above insights, is branching on a flight g such that

$$g \in \arg \min_{q \in \{r \ | \ |H_r|>1\}} \left\{ \sum_{s \ \in \ H_q} \frac{\big||H_q|\mathbf{W}_{qs} - 1\big|}{2(|H_q| - 1)} \right\}. \tag{41}$$

This selection rule, which is referred to as the 'most balanced' rule, attempts to find the start flight g such that all the $\textsc{conx}_{gh}$, $h \in H_g$, are equally used in the LP relaxation. The term $|H_g|$ scales the connections weights so that start flights with a different number of possible connections can be compared.

Additionally, we develop another basis for selection rules by considering the 'opposite' of the connection weight, referred to as the *connection break weight*,

$$\mathbf{B}_{gh} := \sum_{i \in B_{gh}} \bar{x}_i, \tag{42}$$

where $\bar{x}_i,\ i \in P$ is an optimal solution to (37). A connection break weight $\mathbf{B}_{gh}$, in terms of problem (37), is the sum of the optimal values of the solution pairings that would be excluded if $\textsc{conx}_{gh}$ were branched upon. Thus $\mathbf{B}_{gh}$ roughly measures how the LP relaxation will be perturbed by various branch selections. A selection rule, referred to as the 'most broken' rule, is based on this observation and consists of branching on the flight t where

$$t \in \arg \max_{q \in \{r \ | \ |H_r|>1\}} \left\{ \sum_{s \in H_q} \mathbf{B}_{qs} \right\}. \tag{43}$$

Branching on t hopefully perturbs the respective LP solutions such that many branches become pruned by bound since these subproblems have eliminated many of the pairings that appeared in the optimal LP solution of the parents of these subproblems.

The selection rules described in this section, ‘high weight’, ‘most balanced’, and ‘most broken’, play an important role in the B&P system presented in Section 5.4.

5.3 Connection Fixing heuristic

The ideas in Sections 5.1 and 5.2 can also be utilized in a heuristic procedure referred to as the *connection fixing heuristic*. The heuristic—instead of branching and solving a number of subproblems and repeating this process by further restricting the set of pairings considered—restricts pairings more drastically by fixing many connections before solving any subproblems. Thus the set of pairings P is restricted to some set $\bar{P}_k = P^+_{g_1h_1} \cap P^+_{g_2h_2} \cap \ldots \cap P^+_{g_kh_k}$ before any subproblems are solved. The restrictions $P^+_{g_1h_1}, \ldots, \cap P^+_{g_kh_k}$ can be chosen according to connection weights. A simple choice is selecting $P^+_{g_jh_j}$ for $j = 1, \ldots, k$ such that $0.9 < \mathbf{W}_{g_jh_j} < 1.0$. The objective of this operation is that such choices will guide the solution of the LP relaxed pairing problem (37) to a good integral solution. In practice, connection fixing adds and removes many of the sets P^+_{gh} in the intersection $P^+_{g_1h_1} \cap P^+_{g_2h_2} \cap \ldots \cap P^+_{g_kh_k}$ as it searches for near optimal integral solutions. Although the details of this process are not discussed here, it should be noted that the efficiency of connection fixing is the impetus for this thesis and the attempt to utilize connection branching in a B&P algorithm.

5.4 Implementation of the Branch and Price System Using Connection Branching at Jeppesen AB

This section describes the test B&P system using connection branching, hereafter simply referred to as the B&P system, which was implemented Jeppesen AB, Gothenburg, Sweden. The description outlines the high level program flow and the main components, but does not describe all of the minor implementation details. We begin this section by describing the software development environment and software development tools and then the implementation of the B&P system.

The B&P system comprises approximately 10,000 lines of C++ code that manage the pairings and create the search tree and subproblems. In order to solve the subproblems, the B&P system links in the linear program solver XPRESS® by Fair Issac Co. [32]. Furthermore, the IP heuristic PAQS [30] provided by Jeppesen AB is also linked to the B&P system in order the find integral solutions to the pairings in some subproblems—while these integral solutions are usually quite good, there is no proof that they are optimal and thus can only be used as upper bounds.

The B&P system has two phases: the initialization phase and the node processing phase. The initialization phase loads the pairing problem data from a file that describes the flights and their overcover penalties, the GLCs and all the possible pairings to form the MP (20). The implementation does not use a compact *pricing subproblem*, but rather has all the possible pairings explicitly enumerated—these pairings referred to as the *pricing pool*. Since the size of the pricing pools of the instances solved (see Section 6) range from several hundred to several million, solving the *pricing subproblem* in our implementation is often slower than solving the corresponding compact *pricing subproblem*. After the pairing problem data has been loaded, the program solves the RMP (21).

The RMP is initialized with artificial pairings that each covers one flight and costs more than the most expensive pairing in the pricing pool multiplied by the number of flights. These initial pairings cannot be considered actual pairings since they do not constitute a sequence of flights starting and ending at the same base. The cost of the initial pairings, guarantees that their associated $x_i = 0$ when column generation at the root node is completed. Since the RMP starts with these artificial pairings—that are very far from the optimal solution—column generation at the root node, often, adds hundreds of pairings to the RMP when only a few percent of these have nonzero x_i when it completes. Consequently, many of the columns generated at the root node are not very useful, since many are generated when the solution of the RMP is very far from optimality. Thus after computing the optimal solution at the root node, all pairings with $x_i = 0$ are discarded—that, is removed from the RMP. Then to accelerate future pricing, a number of columns with low reduced cost $\bar{c}_i < 0$ are added to RMP. The idea is that the first branch choices will shift the LP relaxation slightly causing pricing to add columns to the RMP that were just nearly excluded when pricing at the root node. This method of initializing the RMP grew out of conversations at Jeppesen AB [2].

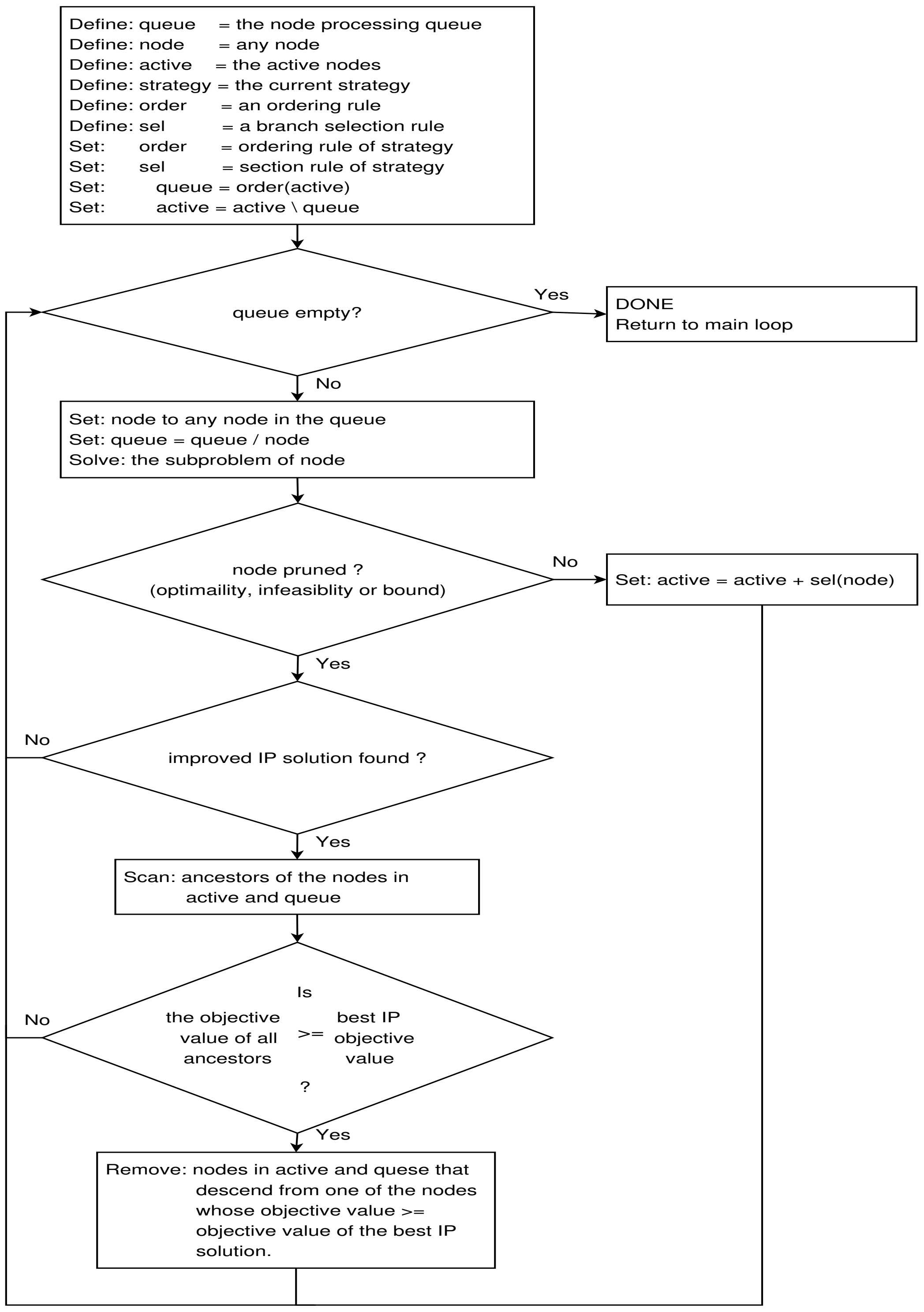
The columns in the RMP are then sent to the heuristic IP solver, PAQS [30], which attempts to quickly construct an integral solution. If an integral solution is constructed, it serves as an initial upper bound for the B&P and otherwise, the upper bound is set to infinity. Initialization concludes by scanning the RMP for connections in RMP and then determining an initial branching from these connections. The initial branching is selected according to the 'most balanced' rule to create a variety of different nodes for future branching. The nodes created from this initial branching are added to the active node pool (the set of nodes that have been created but not processed) without being solved. Now that the B&P system has some active nodes, it can begin the node processing phase.

The node processing phase processes the active nodes until no active nodes remain—effectively completing the B&B component of the B&P system. When node processing is completed, the best IP solution found is the best solution to the pairing problem. Active nodes are processed according to the current strategy. The main loop of node processing cycles between selecting a strategy and the actual node processing, i.e. applying the selected strategy to the active nodes. The main loop is depicted in Figure 8 and the node processing is depicted in Figure 7.

A strategy consists of an ordering rule and a selection rule. For node processing, the B&P system uses the three strategies: 'find IP', 'prune', and 'heuristic IP'. While these three strategies suffice and we limit our empirical experiments to these strategies, it should be noted that it is possible to construct other strategies by combining an ordering rules with a selection rule. Accordingly, if there were to exist 10 ordering rules and 10 selections rules, these could be combined to construct 100 different strategies. Both 'find IP' and 'prune' have an ordering rule that sorts the active nodes by their **W** and removes the highest 20% of these nodes from the active node pool and puts them in the processing queue. (There is no particular reason for removing the specific percent: 20%. However, removing 100%—that is, putting all the active nodes into the processing queue—would nullified the effect of having an ordering rule, since then there would be no reason for sorting them.) The subproblems of the nodes in the processing queue are solved. If a node is not pruned according to the B&B pruning criteria, then a selection rule is used to branch on this node.

The selection rule 'high weight' is used for 'find IP' and the rule 'most broken' is used for 'prune'. According to the discussion in Section 5.2, the 'find IP' strategy should lead to the discovery of good integral solutions and the 'prune' strategy should lead to a reduction in the total number of nodes provided that a good integral solution has been found. Although the results in Section 6 confirm this usage of these strategies in general, the results suggest that the best settings are somewhat problem specific. Once a selection rule has branched on a node—creating its children—its children are added to the active node pool.

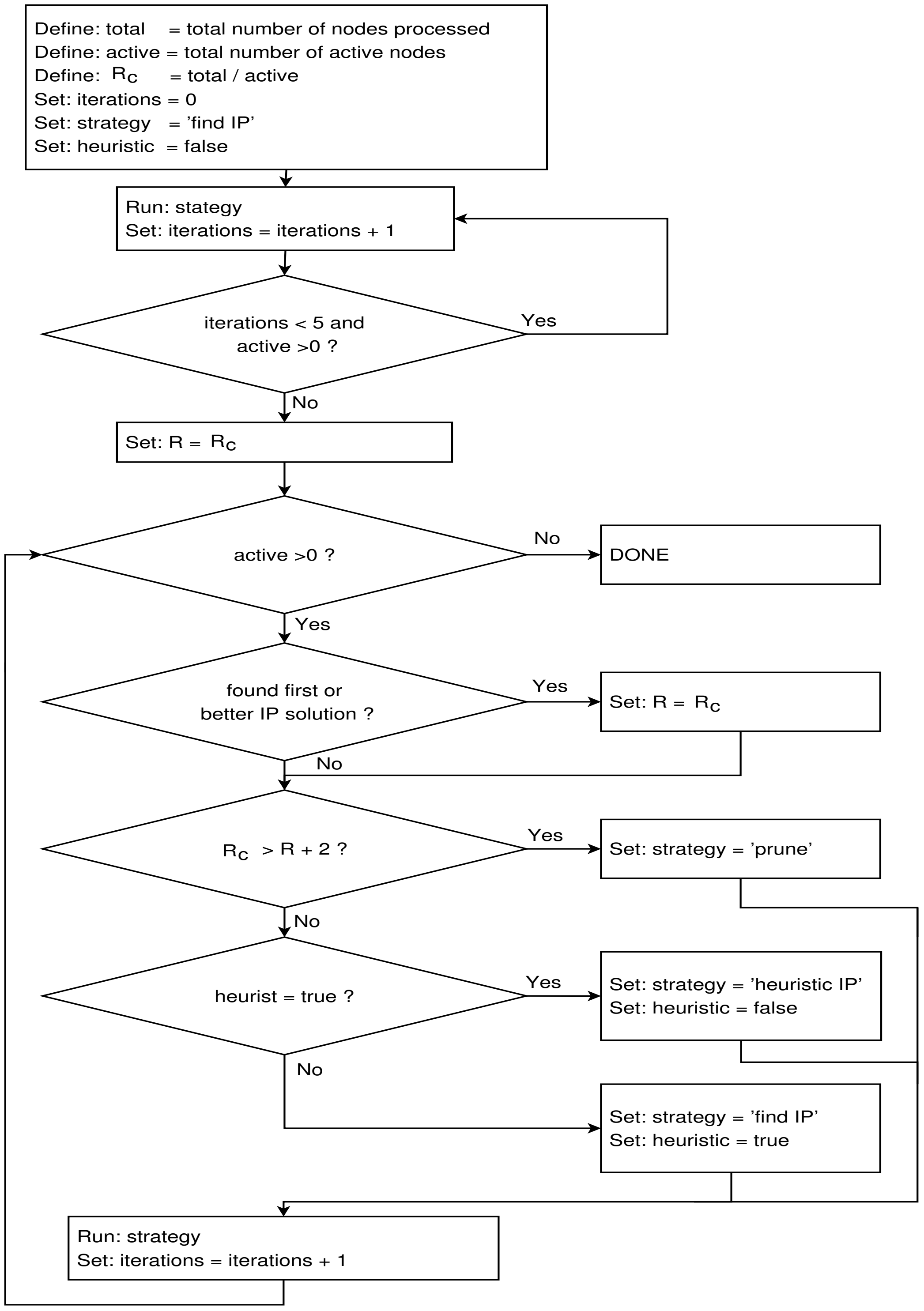
Figure 7: Node Processing or Running A Strategy


Define: queue = the node processing queue
Define: node = any node
Define: active = the active nodes
Define: strategy = the current strategy
Define: order = an ordering rule
Define: sel = a branch selection rule
Set: order = ordering rule of strategy
Set: sel = section rule of strategy
Set: queue = order(active)
Set: active = active \ queue
queue empty?
Yes
DONE
Return to main loop
No
Set: node to any node in the queue
Set: queue = queue / node
Solve: the subproblem of node
node pruned ?
(optimaility, infeasiblity or bound)
No
Set: active = active + sel(node)
Yes
improved IP solution found ?
No
Yes
Scan: ancestors of the nodes in active and queue
Is the objective value of all ancestors >= best IP objective value ?
No
Yes
Remove: nodes in active and quese that descend from one of the nodes whose objective value >= objective value of the best IP solution.

The strategy 'heuristic IP' does not follow the standard schematic discussed above and shown in Figure 7. Instead, 'heuristic IP' executes the heuristic PAQS on two of the active nodes on which PAQS has not previously been run and have the highest average cumulative connection weight. The average cumulative connection weight is the sum of the connection weight $\mathbf{W}_{gh}$ associated with a node and all its ancestors divided by the number of ancestors. The justification for using the average cumulative connection weight is that high connection weights are known to correspond to nodes whose LP solutions are integral or nearly integral—that the LP solution is nearly integral and the solution space is smaller than the solution space of the root node, makes it easier for the heuristic to find good IP solutions. Since 'heuristic IP' does not solve any subproblems, it does not necessarily remove or add any nodes to the active pool unlike the other strategies. Nodes are removed from the active pool only when 'heuristic IP' finds a better integral solution that can prune some nodes by the bound. Following the discussion in the preceding paragraph, we scan the ancestors of the active nodes to find nodes that can be removed.

If while processing nodes, a better IP solution (an IP solution with a lower objective value) is found, the ancestors of the active nodes and the nodes in the processing queue are scanned. If the objective value of any of these ancestors exceeds or equals the new IP objective value, the corresponding nodes are removed from the active pool and the processing queue.

Figure 8: The Main Loop of the B&P System


Define: total = total number of nodes processed
Define: active = total number of active nodes
Define: R_C = total / active
Set: iterations = 0
Set: strategy = 'find IP'
Set: heuristic = false
Run: stategy
Set: iterations = iterations + 1
iterations < 5 and active >0 ?
Yes
No
Set: R = R_C
active >0 ?
No
DONE
Yes
found first or better IP solution ?
Yes
Set: R = R_C
No
R_C > R + 2 ?
Yes
Set: strategy = 'prune'
No
heurist = true ?
Yes
Set: strategy = 'heuristic IP'
Set: heuristic = false
No
Set: strategy = 'find IP'
Set: heuristic = true
Run: strategy
Set: iterations = iterations + 1

The current strategy can be determined according to a number of factors related to all the existing nodes, the active nodes, the pairing problem, the progress of the B&P system, etc. We present a simple, yet effective, decision logic for determining which strategy to run based on the ratio between the total number of nodes processed and the number of active nodes, see Figure 8. This ratio varies as the B&P system processes nodes. We define R_c to be the current ratio. We define a 'decision' ratio, R, which is set to R_c once when node processing has completed four cycles using the 'find IP' strategy and thereafter only if a better integral solution has been found since the last time R was set. The reasoning behind R and R_c is that at some point after the optimal integral solution has been found, R_c must start increasing as more nodes are being processed than created (and added to the active pool). Thereafter R remains constant unless the first integral solution or a better integral solution is found which triggers that R is set to R_c. If $R_c > R + 2$ then 'prune' is run, otherwise, the B&P system alternates between the 'find IP' or 'heuristic IP' strategies.

Finally, building on the discussion at the end of Section 5.1, some of the children with the same parent are regrouped into a fewer number of nodes in order to reduce the total number of nodes processed. After some experimentation, the following logic for group was developed: children that have a connection weight $\mathbf{W}_{gh} \geq 0.80$ are never grouped with other children. In order to discuss the grouping of other children, let the pairings considered in the subproblem of the parent be denoted by $\bar{P}$, pairings of the children with $\mathbf{W}_{gh_i} < 0.80$ to be $\bar{P} \cap P^+_{gh_m}, \ldots, \bar{P} \cap P^+_{gh_n}$, and let the intersection of all these children be denoted by $Y = \left(\bar{P} \cap P^+_{gh_m}\right) \cup \ldots \cup \left(\bar{P} \cap P^+_{gh_n}\right)$. The set of these children can by partitioned in a number of grouping subsets S_i's, each containing a number of children that will be grouped into single nodes. The feasible pairing for each of these grouped nodes is

$$F_i := \bigcup_{\mathrm{CONX}_{gh} \in S_i} \left(\bar{P} \cap P^+_{gh}\right). \tag{44}$$

Since nodes that have a large number of feasible pairings are less likely to be pruned, it is important choose the S_i's such that $|F_i|$ are minimized. Of course, assigning one child per S_i would minimize $|F_i|$ but that would defeat, altogether, the idea of grouping children to minimize the total number of nodes evaluated. The solution is to group children that are 'most similar' into a few nodes—this is accomplished by counting the number of pairings shared by pairs of the children and then using a greedy algorithm.

6 Results

In this chapter, we present the numerical results for the B&P system. The B&P system is run on four different pairing test problems with several different settings. The test problems, which were provided by Jeppesen AB, are presented in Table 7. Table 8 displays the time the B&P system requires to completely solve a test problem and Table 9 displays the objective value of the best integral solution found.

Table 7: Pairing Test Problems

Test Problem	A	B	C	D
Number of Flights	51	51	385	385
Number of Pairings	425	166095	99998	999998
Number of GLCs	2	2	0	0
Sparsity	0.117	0.200	0.0227	0.023

Table 8: Processing Time(sec) for each Pairing Test Problem

Test Problem	A	B	C	D
Default Settings	2.38 (6)	6.41 (3)	30.26 (3)	10616.41 (1)
Without 'heuristic IP'	0.08 (3)	3.96 (1)	17.71 (1)	>74000.00 (2)
Only 'high weight'	0.31 (5)	6.43 (4)	1803.49 (5)	>74000.00 (2)
Only 'most broken'	0.07 (2)	4.75 (2)	13.43 (2)	>74000.00 (2)
No PAQS	0.06 (1)	20.48 (5)	92024.36 (6)	>74000.00 (2)
XPRESS	0.17 (4)	49.28 (6)	44.60 (4)	>74000.00 (2)

The numbers in parentheses correspond to the ranking of the results: (1) is the best and (6) is the worst.

Table 9: Best Objective Value found for each Pairing Test Problem

Test Problem	A	B	C	D
Default Settings	2867525	2860910	20233435 (2)	15283590 (1)
Without 'heuristic IP'	2867525	2860910	20293530 (5)	15516460 (3)
Only 'high weight'	2867525	2860910	20288120 (4)	15516460 (3)
Only 'most broken'	2867525	2860910	20293530 (5)	15516460 (3)
No PAQS	2867525	2860910	20249280 (3)	15516460 (3)
XPRESS	2867525	2860910	20233215 (1)	15285795 (2)

The numbers is parentheses correspond to the ranking of the results: (1) is the best and (6) is the worst.

The test problems, shown in Table 7, increase in size, from A to D. The pairings in these test problems are the pairings that are used as the pricing pool (the purpose of this pool is described in Section 5.4). Test problems B and D can be considered as realistic problems in the sense that the number of pairings greatly exceeds the number of flights. In fact, the test problem A was created by selecting a subset of the pairings from B and C was created, in the same fashion, from D. Since in practice, we are most interested in being able to solve larger instances quickly, the importance of a test problem increases with its size.

Before comparing the results of various tests and configurations of the B&P system, it can be useful to think about the solution process unfolds. Figure 9 visualizes the search that the B&P system performs on test problem A when run with the configuration 'Default Settings' (the various configurations are explained later in this section). The notation for the tag of each node is somewhat complex. The tag begins with the node index which is followed by ':'; this is followed by the connection that was branched upon in '(,)'; the tag ends with the connection weight. For nodes in which several connections with the same start flight and different followers were grouped, the connection information is replaced by a 'm' followed by the number of connections, or nodes, that are grouped. Any node that has the same parent will have the same start flight as this grouping node and thus in this way, the start flight for the grouping node can be ascertained. The connection weight for a grouped node is the sum of the connection weights of the nodes that are grouped. The root node is shown with connection $(-1, -1)$, which means that the root node has not branched on any connections. For example, the tag '1:(28,22) 0.5' indicates that the index of the node is 1, that the node branched on connection, $\textsc{conx}_{28,22}$, and that the connection weight $\mathbf{W}_{28,22} = 0.5$. Figure 10, visualizes the search that the B&P system performs on test problem C when run with the configuration "Without 'heuristic IP"'. This figure, Figure 10, is displayed without any node information due to the number of nodes; its primary purpose is to show how the search tree grows rapidly with the size of the problem.

Figure 9: The Node Search Performed by the B&P System on Test Problem A with the Configuration 'Default Settings'

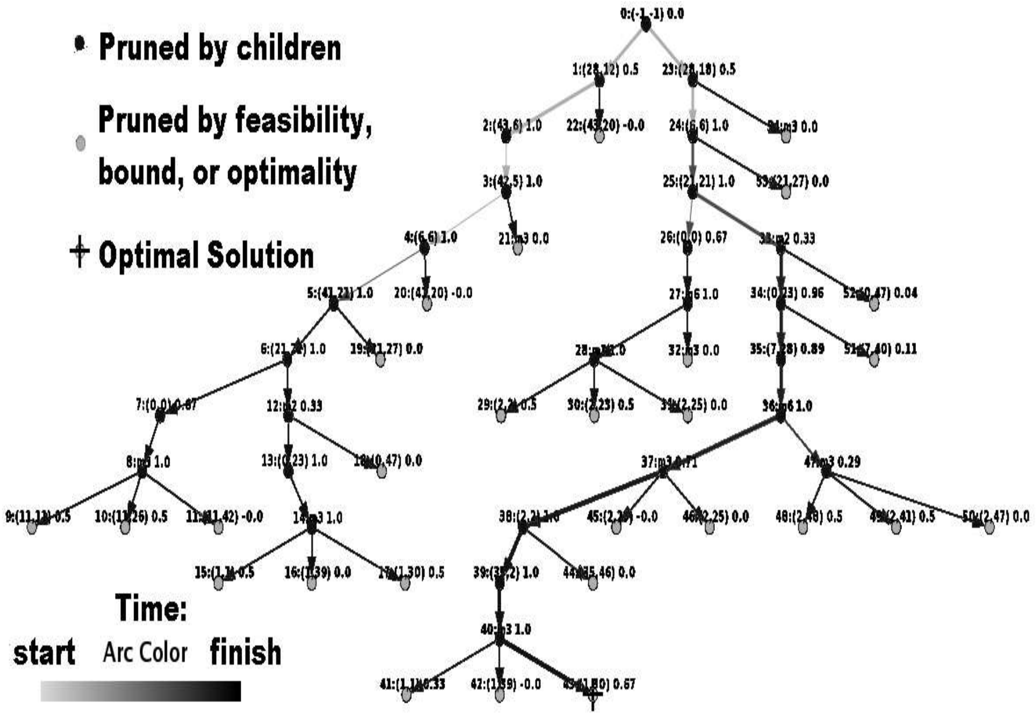
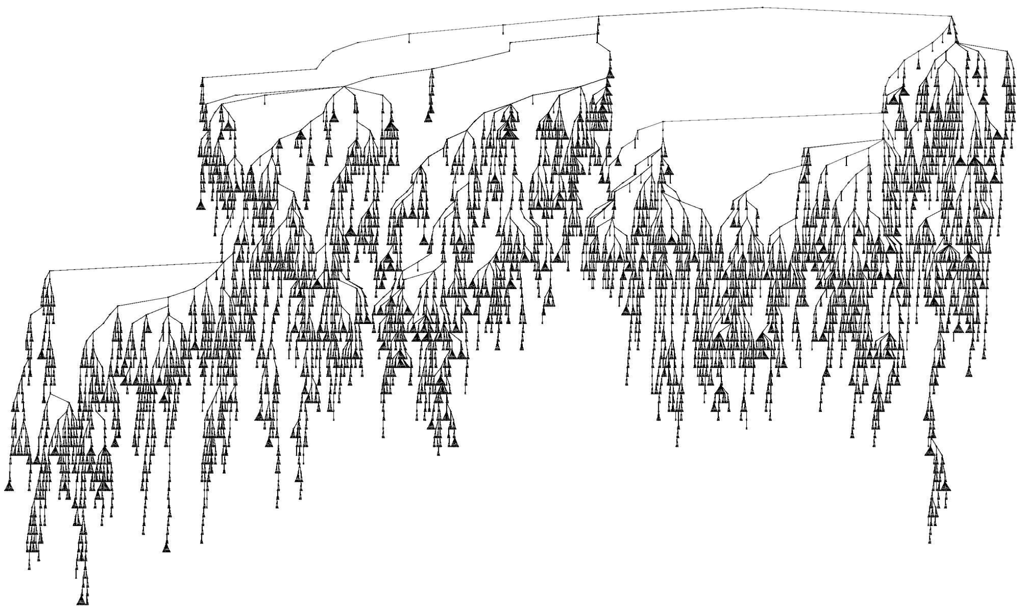


Figure 10: Example of the Node Search Performed by the B&P System



For all test problems and all the settings, the tolerance for pruning is set to 0.3%—meaning that if a node has an objective value of v and the current upper bound is b, then this node will be pruned by bound if $1.003v >= b$. Using a tolerance of 0.3% garuantees that the objective value of the best integral solution found by the B&P system will be within 0.3% from the objective value of the optimal solution. Thus for the test runs for which the processing time is explicitly stated in Table 8, the objective value found in Table 9 is within 0.3% of the optimal solution. The only test runs that did not run to completion were test runs for test problem D which were allowed the maximum time of $74,000$ seconds. We anticipated that test problem D would result in very long running times and thus we decided to stop all test runs for test problem D at this maximum run time.

The different configurations presented in Tables 8 and 9 require some explanation. The 'Default Settings' is the configuration described in Section 5.4. The next three configurations, "Without 'heuristic IP"', "Only 'high weight"', and "only 'most broken"' are identical to the 'Default Settings', except for the described modifications: "Without 'heuristic IP"' runs PAQS during the initialization phase only and then never runs PAQS again; "only 'high weight"' runs PAQS during the initialization phase only and then during the processing phase it uses only the strategy 'high weight'; and "only 'most broken' runs PAQS during the initialization phase only and then during the processing phase uses only the strategy 'most broken'. The 'No PAQS' configuration is identical to the 'Default Settings' except that PAQS is neither run during the initialization nor during node processing. The 'XPRESS' configuration is the result obtained from letting the XPRESS® Integer Programming Solver solve the test problem. Additionally, the start time for all the tests of the B&P system begins when the B&P is done with the initialization phase. For the 'XPRESS' configuration, the start time is when XPRESS has finished solving the LP relaxed problem. We choose this start time since we are interested in comparing and measuring the time used to obtain integral solutions and not the initial set up time. Note additionally that all times were measured as CPU or processor time for actually solving the problem.

While keeping in mind, that the B&P solver is a proof of concept and is not designed to compete head-to-head against XPRESS®, it does perform quite well in general. Before discussing the results in detail, we note that test problem A is almost too small to be considered significant for meaningful comparisons. The duration of the test runs for test problem A range from 0.17 to 2.38 seconds. Thus, test problem A is really too small to be interesting. However, the fact all the optimal objective values found for test problem A are identical indicates that the B&P system is working correctly.

For the remainder of the discussion of the results, we focus on test problems B, C, and D and only use A as a reference. All the objective values for each test problem are within 0.3% of each other as being within 0.3% of the optimal solution necessitates. Rerunning the 'Default Settings' test runs with a lower tolerance, shows that the results for the 'Default Settings' displayed in Table 9 are actually within 0.1% of the optimal integral solution. This experiment was not performed on the remaining problems due to time constraints of this thesis project. Besides noting that all the objective values for each test problem are within the given tolerance, the individual values do not have any meaning.

Comparing running times, we observe that the 'Default settings' are never the worst choice, in terms of time, and for the most significant test problem, D, they were far superior. The configuration "only 'most broken' performs unexpectedly well in the other test cases—especially, considering that it was not designed to find integral solutions needed for pruning nodes according to the reasoning in Section 5.2. In fact, the log file from "Only 'high weight"' for test problem D reveals that the only integral solution found is found at the root node by PAQS. Thus, the reason for "Only 'high weight"' not competing well on test problem D is that it fails to find the integral solutions necessary for pruning. Although this explanation seems likely, the same phenomena of not finding integral solutions occurs for "Only 'high weight"' for test problem D as well, meaning that for some problems running PAQS periodically is crucial. Furthermore, we note that for the B&P system the 'No PAQS' configuration had the worst average performance for the significant problems B, C, and D. The main reason for the success of 'No PAQS' for problem A is the initialization time for PAQS. For the two most significant and realistic problems, B and D, we note that the default setting are far superior to the 'XPRESS' configuration and that solutions are quite reasonable. On the downside, as expected, Table 8 indicates that the solution time does not scale well with the size of the problem. Note that even XPRESS® uses an IP heuristic when solving IP problems.

7 Conclusion

In this thesis, the airline crew pairing problem was explained and several suitable mathematical models for the crew pairing problem were formulated. These models were ILP, MILP, or IP problems that could be solved with the discussed optimization algorithms such as the B&P algorithm. The key components of the B&P algorithm were explained as well as the applicability of the algorithm to models developed. The B&P algorithm was adapted to the crew pairing problem by developing connection branching and developing the branching and ordering rules in Section 4.3. Finally, these ideas were tested by developing the B&P system and performing several experiments on a number of test problems.

The main conclusion is that utilizing connection branching to solve small and medium airline crew pairing problems is a promising approach. The results for the rather small test problems indicate that even medium size problems (problems with approximately a thousand flights and millions of feasible pairings) could be optimized using a connection branching B&P system without much additional work.

The strategies, discussed in Chapter 6, affect the solution process, but the limited number of tests makes difficult to draw any definite conclusions on how these strategies could be better utilized, improved or combined. Drawing such conclusions would require combining and changing the current strategies and then additional experimentation. The results indicate that the default combination of strategies is a safe choice, especially for large crew pairing problems. Moreover, the results indicate that the strategy based on the 'most broken' rule and the connection break weight shows promise and could be developed further (developed in Section 5.2). Furthermore, the solution process of the B&P system is sensitive to whether or not PAQS is run. Accordingly, future work could attempt to better define when PAQS should be run.

Outside the scope of developing and refining strategies, future work could explore employing early termination and column management [21]. Early termination is the idea that not all subproblems need to be solved to optimality, since their associated nodes will be pruned anyways if their children are pruned. This at first seems a little backwards, since it results in processing more nodes. However, early termination can prove advantageous if the number of times the *pricing subproblem* must be solved for certain nodes is excessive. Column management means that not all the columns, or in our context, pairings, that are added to the RMP are allowed to remain in the RMP, since the time required to solve RMP grow with the number of columns. Instead, columns are strategically removed to make solving RMP simpler and faster. Another area for refinement, which could potentially boost performance, is the grouping of nodes (see Section 5.4).

In summary, this thesis demonstrates that connection branching is a viable approach for optimally solving crew pairing problems. The many areas for future work imply that this approach is far from exhausted and is suitable for future research.

References

[1] Achterberg, T., K. Thorsten and A. Matrin, 2005. *Branching rules revisited*, Operations Research Letters Vol 33, No 1, pp. 42–54.

[2] Altenstedt, F., 2008. *personal communication*, Jeppesen AB, Gothenburg, Sweden.

[3] Andersson, E., E. Housos, N. Kohl, and D. Wedelin , 1998. *Crew Pairing Optimization*, in: Operations Research in the Airline Industry Kluwer Academic Publishers, Boston, MA.

[4] Andréasson N., A. Evgrafov, and M. Patriksson, 2005. *An Introduction to Continuous Optimization*, Studentlitteratur, Lund.

[5] Barnhart, C., E.L. Johnson, G.L. Nemhauser, M.W. Savelsbergh and P.H. Vance, 1998. *Branch-and-price: Column generation for solving huge integer programs*, Operations Research. Vol 46, No 3, pp. 316–329.

[6] Barnhart, C., E.L. Johnson, G.L. Nemhauser and P. Vance, 2003. *Crew Scheduling*, in: HandBook of Transportation Science, 2nd Ed. Editor: R.W. Hall. pp. 517–560. Kluwer Academic Publishers, Boston, MA.

[7] Barnhart, C., and A. Cohn, 2004. *Airline Schedule Planning: Accomplishments and Opportunities*, Manufacturing & Service Operations Management, Vol 6, No 1, pp 3–22.

[8] Bazaraa, M.S., H.D. Sherali and C.M. Shetty, 2006. *Nonlinear Programming: Theory and Algorithms*, 3rd Ed. Wiley-Interscience, New York, NY.

[9] Benichou, M., J.M. Gauthier, P. Girodet, G. Hetges, G. Ribiere and O. Vincent, 1971. *Experiments in Mixed–Integer Linear Programming*, Mathematical Programming, Vol 1, No 1, pp. 76–329.

[10] Chvátal, V., 1983. *Linear Programming*, W. H. Freeman.

[11] Delin, H., 2008. *personal communication*, Jeppesen AB, Gothenburg, Sweden.

[12] Desrosiers, J., F. Soumis and M. Desrochers, 1984.
Routing with Time Windows by Column Generation, Networks, Vol 14, Issue 4, pp. 545–565.

[13] Eclipse C/C++ Development Tooling - CDT 5.0, 2008.
http://www.eclipse.org/cdt/

[14] GCC, GNU Compiler Collection 3.4.6, 2006.
http://gcc.gnu.org/

[15] Graphviz - Graph Visualization Software 1.xx, 2004.
http://www.graphviz.org/

[16] The igraph library 0.5, 2008.
http://cneurocvs.rmki.kfki.hu/igraph/

[17] Jeroslow, R.G., 1974.
Trivial Integer Programs Unsolvable by Branch-and-Bound, Mathematical Programming, Vol 6, No 1, pp 105–109.

[18] Matoušek, J. and B. Gärtner, 2006.
Understanding and Using Linear Programming, Springer-Verlag, New York, NY.

[19] Kleinberg, Jon and Éva Tardos, 2005. *Algorithm Design*, Addison Wesley, New York, NY.

[20] Lasdon, Leon L., 2002.
Optimization Theory for Large Systems, Dover, Mineola, NY.

[21] Lübbecke, Marco E. and J. Desrosiers, 2005.
Selected Topics in Column Generation, Operations Research, Vol 53, No 6, pp. 1007–1023.

[22] matplotlib 0.87.3, 2006.
http://matplotlib.sourceforge.net/

[23] Mercurial Distributed SCM 1.0, 2008.
http://www.selenic.com/mercurial/wiki/

[24] Nemhauser, G.L. and L. A. Wolsey, 1988
Integer and Combinatorial optimization, John Wiley & Sons Inc., New York, NY.

[25] Python 2.4.3, 2006.
http://www.python.org

[26] Ryan, D. M. and B. A. Foster, 1981.
An Integer Programming Approach to Scheduling, in: Computer Scheduling of Public Transport. Editor A. Wren. North-Holland. 269–280.

[27] Vance, P.H., A. Atamtürk, C. Barnhart, E. Gelman, E.L. Johnson, A. Krishna, D. Mahidhara, G.L. Nemhauser and R. Rebello, 1997.
A Heuristic Branch-and-Price Approach for the Airline Crew Pairing Problem, http://citeseer.ist.psu.edu/vance97heuristic.html.

[28] Vanderbei, R.J., 1998.
Linear Programming: Foundations and Extensions, 2nd Ed. Springer-Verlag, New York, NY.

[29] Vazirani, V.V., 2001.
Approximation Algorithms, Springer-Verlag, New York, NY.

[30] Wedelin, D., 1995.
An algorithm for large scale 0-1 integer optimization, Annals of Operations Research, Vol 57, pp. 283–301.

[31] Wolsey, L.A., 1998.
Integer Programming, John Wiley & Sons Inc., New York, NY.

[32] Xpress-MP 18.10, 2008.
Fair Isaac Corporation.