Integrated Aircraft Routing,
Crew Pairing,
and Tail Assignment

Sebastian Ruther
(MSIE, Dipl. Wirt-Ing.)

The University of Newcastle

Thesis submitted for the degree of
Doctor of Philosophy in Mathematics

March 2013
Statement of Originality

The thesis contains no material which has been accepted for the award of any other degree or diploma in any university or other tertiary institution and, to the best of my knowledge and belief, contains no material previously published or written by another person, except where due reference has been made in the text. I give consent to the final version of my thesis being made available worldwide when deposited in the University’s Digital Repository**, subject to the provisions of the Copyright Act 1968. **Unless an Embargo has been approved for a determined period.

Sebastian Ruther
Acknowledgements

First and foremost, I would like to thank my wonderful supervisor, Natashia Boland. Without the stimulating discussions, guidance, and support, this work would not have been possible. I would also like to thank my co-supervisor Faramroze Engineer, especially for the invaluable advice regarding the implementation of the branch-and-price algorithm. It has been an honour to work with such highly professional researchers.

I would like to extend my gratitude to Constraint Technologies International for providing the real world data necessary for this project and also providing financial support. I would like to specially mention Ian Evans for directing this research and for his priceless insights into real world problems.

Olivia Smith provided a first implementation of the pricing problem solver, for which I am very grateful. She further provided a wealth of knowledge regarding labelling algorithms.

I am thankful for the help I received from all the people in the Department of Mathematics. Mohsen and Rachel, thanks for being there for me through the years.

Thanks to all my friends all over the world, particularly in Germany and Australia, for making my life as enjoyable as it is. Stefan, your support has been great. Jenna, thank you for being a great friend and also proofreading this thesis.


Sebastian
Preface

Ian Evans posed the initial idea of the new airline scheduling paradigm discussed in this thesis. A high level problem description was given in the Ph.D thesis of Olivia Smith. In her thesis, Olivia Smith also presented preliminary work on a multi-commodity flow formulation, however, the work presented in this thesis is the first time a solution algorithm is proposed for the new paradigm.

A first version of the implementation of the pricing problem solver was provided by Olivia Smith. She also posed the idea to model the resource consumption in consecutive duties based on the consumption in a single duty. She assisted with the development of some of the extensions presented in this thesis, however, the extensions were implemented and tested by myself.

Rachel Bunder provided professional editing of the figures presented in this thesis. All figures were, however, designed by myself.

The remainder of the work was completed by Sebastian Ruther, including much of the formulation of ideas, all of the code development and testing, bar the exceptions given above, and all of the written report.
# Contents

1 Introduction .......................................................... 1  
   1.1 An Overview of Airline Planning and Thesis Motivation .......... 1  
   1.2 Main Contributions ............................................. 5  
   1.3 Thesis Outline ................................................. 7  

2 Mathematical Background ........................................... 9  
   2.1 Decomposition Methods ......................................... 9  
      2.1.1 Dantzig-Wolfe Decomposition .......................... 10  
      2.1.2 Lagrangian Relaxation ................................ 13  
      2.1.3 DWD for Integer Programs with Block-Diagonal Structures . 14  
   2.2 Column Generation ............................................. 17  
   2.3 Recent Developments in Column Generation ...................... 21  
   2.4 Branch and Price ............................................. 24  
   2.5 Resource Constrained Shortest Path Problems .................. 26  
   2.6 RCSPP with Replenishments ................................... 27  

3 Airline Scheduling .................................................. 29  
   3.1 Terminology of Airline Scheduling Networks and Models .......... 32  
   3.2 Schedule Design .............................................. 39  
   3.3 Fleet Assignment ............................................. 40  
   3.4 Aircraft Routing ............................................. 42  
   3.5 Crew Pairing .................................................. 47  
   3.6 Crew Rostering .............................................. 54  
   3.7 Tail Number Assignment ....................................... 55  
   3.8 Integrated Airline Scheduling .................................. 58
3.8.1 Schedule Design and Fleet Assignment ............... 60
3.8.2 Schedule Design and Aircraft Routing ............... 60
3.8.3 Fleet Assignment and Aircraft Routing ............... 61
3.8.4 Fleet Assignment and Crew Pairing ................. 62
3.8.5 Aircraft Routing and Crew Pairing ................. 63
3.8.6 Crew Pairing and Crew Rostering ................. 65
3.8.7 Schedule Design, Fleet Assignment, and Aircraft Routing .......... 66
3.8.8 Schedule Design, Aircraft Routing, and Crew Pairing .......... 67
3.8.9 Fleet Assignment, Aircraft Routing, and Crew Pairing .......... 67
3.9 Past, present, and future developments in airline scheduling .......... 69
3.9.1 Overview of Airline Scheduling Optimisation Methodology .......... 71

4 Near Day-of-Operations Integrated Airline Scheduling .......... 73
4.1 Motivation ............................................. 74
4.2 Problem Description ..................................... 80
4.2.1 Aircraft Rules ........................................ 91
4.2.2 Crew Rules ........................................ 94

5 A CG Formulation for Integrated Airline Scheduling .......... 97
5.1 Master Problem ......................................... 98
5.2 Pricing Problems ....................................... 101
5.2.1 Aircraft Routing Pricing Problem ............... 104
5.2.2 Crew Pairing Pricing Problem ............... 113

6 A Branch-And-Price Algorithm .......... 121
6.1 Description of Test Instances .......................... 123
6.1.1 Maintenance ....................................... 127
6.1.2 Preprocessing of Maintenance Types and Indicators ........ 129
6.2 LP solving ............................................. 133
6.3 Initialisation of the RMP ................................ 140
6.4 Adding Multiple Columns per Pricing Problem ........... 144
6.5 Column Management .................................. 147
6.6 Branching ........................................... 151
6.6.1 Early Branching .................................. 155
## CONTENTS

9.5 Summary .................................................. 319

10 Comparison of Approaches and Conclusions 321
  10.1 Summary ............................................. 321
  10.2 Numerical Results for Large Instances .......... 324
  10.3 Future Work ......................................... 331

A Numerical Results for Pricing Problem Selection Strategies 335

B Numerical Results for Large Instances 341

Glossary 363

Mathematical Notation 369
List of Figures

2.1 Typical convergence of the LP solution value and the lower bound of a column generation procedure. ................................................. 19

3.1 A connection network for the example given in Table 3.1. ........... 38
3.2 A time-line network for the example given in Table 3.1. .............. 39
3.3 A pairing consisting of three duties with layovers in-between. ....... 48

4.1 Timing in the sequential approach to airline scheduling. ............ 75
4.2 Rolling planning horizon. ..................................................... 81
4.3 Monthly rosters of two crews. ............................................. 84
4.4 Actual vs. planned flying time in a roster. ............................... 85
4.5 Timing in the new approach to airline scheduling. .................... 86
4.6 Cost of a crew connection. .................................................. 88

6.1 Number of iterations when using different limits on the number of columns added per pricing problem. ........................................... 146
6.2 Run times (s) when using different limits on the number of columns added per pricing problem. ........................................... 147
6.3 LP solution times and number of columns added per iteration in the first 100 iterations after the root node is solved for instance 5-L-1. 154
6.4 LP solution times in each iteration when solving instance 5-L-1. .. 154
6.5 Cumulative number of columns generated and cumulative number of incompatible columns in each iteration when solving instance 5-L-1. 156
6.6 Number of fractional connections in fractional solutions. ............ 160
6.7 Number of fractional connections at each branching decisions for $\epsilon = 0.3$ and $\epsilon = 0$. ..................................................... 161
6.8 Relationship of average IP gaps and reduction in run time for different combinations of early branching values and tightening depths. . . . . . 163
6.9 Behaviour of the LP value and the local dual bound over all iterations after the root node for instance 3-S-A. . . . . . . . . . . . . . . 164
7.1 Connections along which a pairing can be extended due to Rule 3. . 181
7.2 Consecutive duty accumulation over three duties. . . . . . . . . . . . 183
7.3 Incorrect lower bound on cost resulting due to the credit accumu. . . 191
7.4 Incorrect lower bound on resource 3 when extending backward. . . 205
7.5 Correct lower bound on resource 3 when extending backward. . . . 207
7.6 Run times (s) for the small instances when using different limits on the number of preprocessing iterations. . . . . . . . . . . . . . 213
7.7 Average number of flights covered in a route or pairing generated during Algorithm 6.1. . . . . . . . . . . . . . . . . . . . . . . . . . . . 215
7.8 Reductions in initial objective values and run times. . . . . . . . . . 216
7.9 Convergence of the LP objective function value at the root node for instance 5-S-1. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 217
7.10 Frequency polygon of the number of non-zero entries of columns generated in our initialisation method. . . . . . . . . . . . . . . . . 219
7.11 Frequency polygon of the number of non-zero entries of columns generated in the first 20 iterations at the root node for instance 5-S-1. . 221
7.12 Average change in pricing time when performing various levels of preprocessing. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 226
7.13 LP value and lower bound at the final 80 iter. at the root node when using heuristic pricing. Instance 5-S-1. . . . . . . . . . . . . . . . 228
7.14 LP value and lower bound in the final 80 iter at the root node when using heuristic pricing. Instance 3-L-A. . . . . . . . . . . . . . . . . 228
8.1 Generation of columns per PP and iteration when solving all PPs per iteration. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 238
8.2 Convergence of LP value at the root node for instance 5-S-1 when using different pricing problem selection strategies. . . . . . . . . 250
8.3 Generation of columns per PP and iteration when solving a subset of PPs per iteration. ........................................... 252
8.4 Time spent pricing in each iteration when solving instance 5-S-1 while solving every pricing problem per iteration. .......... 255
8.5 Time spent pricing in each iteration when solving instance 5-S-1 while solving only a subset of pricing problems per iteration. 255
8.6 Average change in run time, pricing time, and LP time for different PP selection strategies. ............................................. 262
8.7 Average change in the number of iterations and number of columns for different selection strategies .......................... 263
9.1 Networks of three different aircraft routing pricing problems. ... 271
9.2 Aggregated network of aircraft AC1 and AC2. ....................... 272
9.3 Aggregated network of aircraft AC1, AC2, and AC3. ............... 272
9.4 Separate and aggregated networks of two crew blocks with different end locations. ..................................................... 281
9.5 Separate and aggregated networks of two crew blocks with varying availabilities. ......................................................... 284
9.6 Aggregated network of three crew blocks including resource 7. ... 287
9.7 Convergence of LP objective function value at the root node when solving instance 5-S-1 using setting AllSUPP+3 and AllPP+3. 297
9.8 Number of columns that were generated in each iteration at the root node when solving instance 5-S-1. ....................... 298
9.9 Number of crew columns that were present in the RMP per iteration. 299
9.10 Number of aircraft columns that were present in the RMP in each iteration at the root node when solving instance 5-S-1. .... 300
9.11 Number of aircraft columns that were generated in each iteration at the root node when solving instance 5-S-1. .................. 300
9.12 Cumulative difference between the average column size per iteration. 302
9.13 Illustration in which iteration the SUPPs generated columns for which OPP. .......................................................... 305
9.14 Illustration in which iteration a PP generated columns. ............ 306
9.15 Convergence of the LP objective function value and the lower bounds.
First 117 iterations at the root node. . . . . . . . . . . . . . . . . . . 307

9.16 Convergence of the LP objective function value and the lower bounds.
Last 118 iterations at the root node. . . . . . . . . . . . . . . . . . . 308

9.17 Time it took to solve in each iteration the five ASUPPS and the five
original ARPPs that took the longest to solve. . . . . . . . . . . . . 310

9.18 Time it took to solve in each iteration the three CSUPPs and the
three original CPPPs that took the longest to solve. . . . . . . . . . 311

9.19 Time it took to solve in each iteration the eight SUPPs compared to
the time it took to solve all OPPs. . . . . . . . . . . . . . . . . . . 311

9.20 Several performance measures for varying SUPP strategies without
preprocessing. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 314

9.21 Illustration of when SUPPs generated columns for OPPs when per-
forming preprocessing. . . . . . . . . . . . . . . . . . . . . . . . . . 316
List of Tables

3.1 Example timetable for SYD ............................................. 37
3.2 Hypothetical maintenance types ........................................ 44

5.1 Example maintenance types with sequencing restrictions. .......... 106
5.2 Permissible arcs for the example presented in Table 5.1. .......... 107
5.3 Resources for the different maintenance types in the example presented in Table 5.1. ............................................. 108
5.4 Replenishment arcs for the example presented in Table 5.1. ....... 109
5.5 Crew rules and respective resources considered in our problem. .... 116

6.1 Sizes of real-world and semi-artificial instances. .................... 125
6.2 Sizes of pricing problems in each instance. .......................... 128
6.3 Considered maintenance types for each fleet. ....................... 129
6.4 Number of aircraft in each instance with equal maintenance requirements ................................................................. 132
6.5 Overall run times (s) at root node when using different LP solution methods. ......................................................... 134
6.6 Number of CG iterations to solve the root node when using different LP solution methods. ............................................. 134
6.7 Total LP solutions times (s) and number of columns generated in the first 500 iterations when solving the crew and aircraft part separately. 138
6.8 Average number of flights covered per route or pairing for different instances. .......................................................... 139
6.9 Run time and number of iterations required to solve the root node when using various initialisation methods. ....................... 143
<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.10</td>
<td>Number of CG iterations required to solve each instance when using different limits on the number of columns added to the RMP per PP.</td>
<td>145</td>
</tr>
<tr>
<td>6.11</td>
<td>Run times (s) when using different limits on the number of columns added per pricing problem.</td>
<td>146</td>
</tr>
<tr>
<td>6.12</td>
<td>Root node solution times (s) when using different column age limits.</td>
<td>149</td>
</tr>
<tr>
<td>6.13</td>
<td>Run times (s) excluding time spent at the root node when using different column removal strategies.</td>
<td>151</td>
</tr>
<tr>
<td>6.14</td>
<td>Average change in run time (%), average integrality gaps, average number of iterations, and average depth of the tree for different early branching values.</td>
<td>158</td>
</tr>
<tr>
<td>6.15</td>
<td>Average integrality gaps when tightening early branching gaps.</td>
<td>162</td>
</tr>
<tr>
<td>6.16</td>
<td>Average reduction in run time compared to solving all nodes to optimality when tightening early branching gaps.</td>
<td>162</td>
</tr>
<tr>
<td>6.17</td>
<td>Various performance measures when branching on replenishments.</td>
<td>167</td>
</tr>
<tr>
<td>6.18</td>
<td>Change in run times in percent for different values of ( \omega ) compared to not suppressing the pricing step.</td>
<td>170</td>
</tr>
<tr>
<td>6.19</td>
<td>Reduction in run times in percent for different values of ( \omega ) compared to not suppressing the pricing step.</td>
<td>171</td>
</tr>
<tr>
<td>6.20</td>
<td>Average number of CG iterations that were allowed despite suppressing pricing for different values of ( \xi^A ) and ( \xi^B ).</td>
<td>171</td>
</tr>
<tr>
<td>6.21</td>
<td>IP gaps (%) for different values of ( \omega ) compared to not suppressing the pricing step.</td>
<td>172</td>
</tr>
<tr>
<td>7.1</td>
<td>Average fraction of arcs remaining in pricing problems after preprocessing when carrying out a different number of preprocessing iter.</td>
<td>211</td>
</tr>
<tr>
<td>7.2</td>
<td>Fraction of pricing problems solved to optimality in preprocessing when carrying out a different number of preprocessing iterations.</td>
<td>212</td>
</tr>
<tr>
<td>7.3</td>
<td>Average time (s) spent in pricing per CG iteration when performing different levels of preprocessing.</td>
<td>212</td>
</tr>
<tr>
<td>7.4</td>
<td>Run times (s) for the small instances when using different limits on the number of preprocessing iterations.</td>
<td>213</td>
</tr>
<tr>
<td>7.5</td>
<td>Number of column generation iterations at the root node when using different limits on the number of preprocessing iterations.</td>
<td>214</td>
</tr>
<tr>
<td>Table Number</td>
<td>Table Title</td>
<td>Page</td>
</tr>
<tr>
<td>-------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>7.6</td>
<td>Average change of run time (%) for different heuristic pricing strategies when performing a different number of preprocessing iterations.</td>
<td>224</td>
</tr>
<tr>
<td>7.7</td>
<td>Average change in number of CG iterations (%) for different heuristic pricing strategies and number of preprocessing iterations.</td>
<td>225</td>
</tr>
<tr>
<td>7.8</td>
<td>Percentage of pricing problems solves in which only the preprocessing procedure was executed.</td>
<td>226</td>
</tr>
<tr>
<td>7.9</td>
<td>Iteration at which certain LP gaps are reached at the root node when using $z_{LB}^{-}$ compared to using $z_{LB}^{+}$.</td>
<td>229</td>
</tr>
<tr>
<td>8.1</td>
<td>Number of resulting pricing problems when dividing by certain divisors for each instance.</td>
<td>246</td>
</tr>
<tr>
<td>8.2</td>
<td>Solution times (s) for different selection strategies when adding three columns per pricing problem solve.</td>
<td>247</td>
</tr>
<tr>
<td>8.3</td>
<td>Change in total time spent pricing (in %) for different selection strategies when adding three columns per pricing problem and iteration.</td>
<td>248</td>
</tr>
<tr>
<td>8.4</td>
<td>Total number of pricing problems solved for different selection strategies when adding three columns per pricing problem and iteration.</td>
<td>248</td>
</tr>
<tr>
<td>8.5</td>
<td>Total number of CG iterations for different selection strategies when adding three columns per pricing problem and iteration.</td>
<td>249</td>
</tr>
<tr>
<td>8.6</td>
<td>Average change in number of iterations (in %) for different selection strategies when adding three columns per PP and iteration.</td>
<td>251</td>
</tr>
<tr>
<td>8.7</td>
<td>Average change in run time, time spent in pricing, and time spent solving LPs for different selection strategies.</td>
<td>257</td>
</tr>
<tr>
<td>8.8</td>
<td>Keys to identify SetPP and SglPP strategies.</td>
<td>260</td>
</tr>
<tr>
<td>9.1</td>
<td>Change in pricing time when solving using strategy AllSUPP with several limits on the number of columns to add per OPP.</td>
<td>293</td>
</tr>
<tr>
<td>9.2</td>
<td>Run time when solving using strategy AllSUPP with several limits on the number of columns to add per OPP.</td>
<td>294</td>
</tr>
<tr>
<td>9.3</td>
<td>Varying performance measures for the AllSUPP strategies.</td>
<td>295</td>
</tr>
<tr>
<td>9.4</td>
<td>Change in number of iterations at the root node to reach the early branching criteria.</td>
<td>295</td>
</tr>
</tbody>
</table>
9.5 Iteration at which certain LP gaps are reached at the root node when solving all SUPPs compared to solving all OPPs. 309

9.6 Pricing time when solving using settings SwSUPP, AllPP+3, and SetPP/10/20+5. 318

9.7 Resulting run times when solving using settings SwSUPP, AllPP+3, and SetPP/10/20+5. 318

9.8 Various performance measures when solving using settings SwSUPP, AllPP+3, and SetPP/10/20+5. 319

10.1 Parameters used in each setting. 325

10.2 Run times (s) when solving the large instances using the best settings of each chapter. A * indicates that solving was aborted due to reaching the time limit of 12 hours. 326

10.3 Pricing time when solving the large instances using the best settings of each chapter. 327

10.4 Ratio of pricing time and LP time when solving the large instances using the best settings of each chapter. 328

10.5 Number of CG iterations when solving the large instances using the best settings of each chapter. 328

10.6 Number of columns generated when solving the large instances using the best settings of each chapter. 329

10.7 Number of PP solved when solving the large instances using the best settings of each chapter. 330

A.1 Various performance measures for strategy SetPP/5/10. 336

A.2 Various performance measures for strategy SetPP/7/15. 337

A.3 Various performance measures for strategy SetPP/10/20. 338

A.4 Various performance measures for strategy SglPP. 339

B.1 Time spent solving LPs when solving the large instances using the best settings of each chapter. 341

B.2 Number of branch-and-price node when solving the large instances using the best settings of each chapter. 341
B.3 IP gaps when solving the large instances using the best settings of each chapter. ......................... 342
Abstract

Airline scheduling is the process of scheduling available aircraft and crews such that all flights that an airline operates are serviced. The goal is to develop a minimum-cost plan in which all aircraft and crews have tasks assigned to them. Aircraft operate sequences of flights, so-called routes, that must not violate maintenance requirements. Crews have pairings assigned to them, which are sequences of flights that respect several rules regarding work-load and rest periods.

Due to its complexity, the airline scheduling is typically done sequentially, decomposing the entire problem into six more tractable sub-problems: the schedule design, fleet assignment, aircraft routing, crew pairing, crew rostering, and the tail assignment problem. The schedule design problem determines which flights should be offered, while the fleet assignment problem assigns the flights to aircraft types, matching aircraft capacity and expected demand as much as possible. The aircraft routing problem generates generic aircraft routes, whereas the crew pairing problem creates generic pairings. These pairings are then put together to form monthly rosters in the crew rostering problem. Finally, in the tail assignment problem, the generic routes are assigned to individual aircraft.

An issue of the sequential approach is that later stages depend on decisions made in earlier stages. For example, the initial location, time since last maintenance checks of each required type, and usage history of the individual aircraft is not known with certainty in the aircraft routing problem. Thus, in the tail assignment problem, an assignment of the generic routes to available aircraft may not be possible without making costly adjustments to the routes.

Another major disadvantage is caused by the long lead times and long planning horizons of the individual stages. Usually the aircraft routing and crew pairing problems are solved several weeks or even months before putting a plan into action,
the so-called day of operations. As a result, routes and pairings will be formed based on forecasts and historical flight data. Often the circumstances do not eventuate according to the predictions, requiring modifications to routes and pairings close to or on the day of operations.

While disruptive events on the day of operations will still occur, and have to be dealt with accordingly, many events leading up to that date may be considered if scheduling closer to the day of operations. For example, a change in workforce between solving the crew pairing problem and the day of operations may invalidate the crew pairing solution. Another example is that often minor equipment failures, such as a defective toilet, are tolerated if an aircraft’s scheduled maintenance is not too far in the future. However, these issues may accumulate, possibly resulting in additional unscheduled maintenance. When scheduling close to the day of operations, these minor failures are largely known and can be accounted for by scheduling maintenance earlier than originally anticipated.

Scheduling close to the day of operations also enables re-visiting fleeting decisions. The fleet assignment problem is solved months in advance, at which point demand is largely unknown. The literature indicates that the majority of the demand for each flight is realised during the last weeks prior to flight departure and that profit can be increased significantly if the fleeting decisions are revisited. However, since the aircraft routing problem is solved separately for each aircraft fleet, reassigning a flight to a different fleet due to updated demand information will invalidate the aircraft routes. Scheduling close to the day of operations overcomes this obstacle as routes are generated when most of the demand has been realised.

Naturally, information becomes significantly more accurate as the day of operations approaches. The paradigm this thesis is based on allows an integrated aircraft routing, crew pairing, and tail assignment problem to be solved only a few days before the day of operations. At that point in time, it should be possible to predict location and status of crews and aircraft more accurately. In this paradigm, crews are not told the exact pairings they will be operating after the crew rostering phase; they will only be notified of days on and off. We then form “crew blocks”, in which we aggregate all crews that have the same work period, i.e. consecutive working days, and are based at the same location. Then, for the purpose of our mathemat-
ical model, the crews represented by a crew block are considered identical. The solver will generate an appropriate number of pairings for each crew block. Since the work periods differ between crew blocks, we have to generate pairings for each individual crew block.

Solving an integrated model this close to the day of operations means we can generate routes specifically for each aircraft, thereby eliminating the need to later assign routes to aircraft. The routes consider the location and maintenance requirements of the individual aircraft. Unlike standard aircraft routing, we consider all maintenance requirements that the aircraft has during the planning horizon.

A downside of scheduling this close to the day of operations is that other processes and company divisions may be affected negatively. For example maintenance planning and technician rostering have to be carried out more timely, and, as already mentioned, crew members have to be more flexible as their final pairings are only revealed four days in advance. This may require higher financial compensation, however, we believe this to be offset by the benefits of using more accurate information when scheduling routes and pairings.

To solve the integrated aircraft routing, crew pairing, and tail assignment problem, we use mathematical programming, more precisely we develop a column generation formulation in which the master problem ensures that all flights are covered by one aircraft and one crew. Additionally, it contains constraints that model the interaction of aircraft and crew along short and restricted connections: on short connections a crew must stay with an aircraft because insufficient time is available to physically move to another gate. Restricted connections are likely to propagate flight delays; staying with an aircraft avoids spreading the delay, thus increasing schedule robustness.

All maintenance and crewing rules are relegated to the pricing problems, which are modeled as resource constrained shortest path problems with replenishments, where the replenishments represent maintenance checks for aircraft or rest periods for crews. The specific rules considered in our problem require several extensions to the standard resource constrained shortest path problems with replenishments. We present a labelling algorithm that is capable of handling these extensions. Furthermore, we develop a preprocessing procedure that significantly reduces the computa-
To obtain integer solutions to the integrated problem, we develop a branch-and-price algorithm based on follow-on branching. We evaluate the fitness of several column generation acceleration strategies to solve our problem by conducting numerical experiments on real world instances.

Since we generate routes specifically for each aircraft and pairings specifically for each crew block, we must solve a large number of pricing problems. We develop several strategies to address this challenge. In the most straightforward strategy, we solve all pricing problems in each column generation iteration. Another option is to solve only a subset of the pricing problems per iteration where the pricing problems are in a fixed order. In a third strategy, we chose a subset based on how well each pricing problem performed recently, and on the current state of the overall problem. The impact of these strategies on solution time, convergence of the column generation procedure, and the quality of the dual bound is investigated. We present detailed numerical results for real world and artificial instances of varying sizes.

As perhaps the strongest methodological innovation in the thesis, we develop aggregated forms of pricing problems, which we call “superimposed pricing problems”. These pricing problems can be solved instead of the many individual pricing problems as they fulfill the requirements of column generation, that is to generate the most negative reduced cost column among the represented pricing problems or prove that no such column exists. These superimposed pricing problems are slightly larger than the original problems but a much smaller number needs to be solved. We present numerical results, in which we compare their performance and impact on the overall branch-and-price algorithm to that of the regular pricing problem selection strategies. Lastly, we combine the ideas by solving only a subset of the superimposed pricing problems.
Chapter 1

Introduction

In this chapter, we give an overview of the airline planning process and motivate this thesis. We list the contributions in this work and give a thesis outline.

1.1 An Overview of Airline Planning and Thesis Motivation

Airlines face complex decision making processes during planning and execution of flight schedules. Major airlines operate several hundred aircraft and employ thousands of flight personnel. These resources have to be used efficiently and in conjunction with each other to operate at low cost. Due to the size of airline operations, not all planning decisions can be made simultaneously.

In practise a sequential approach is used in the planning stage. After a flight schedule is developed, aircraft types are assigned to each flight leg in the fleet assignment problem so that the capacity of the aircraft matches the estimated number of passengers closely. The subsequent aircraft routing problem generates generic aircraft routes by selecting a number of flights that have to be flown in sequence. From these routes, it then selects a set such that all flights are covered. In the crew pairing problem, generic pairings are generated while crew costs are minimised and all crew rules are respected. The pairings are put together to form monthly rosters in the crew rostering problem. These are assigned to actual crews based on personal preferences and requirements. A few days before the day of operations, the tail assignment problem is solved. Individual aircraft are assigned to routes generated in
the aircraft routing problem. The planner needs to ensure that routes are feasible with respect to maintenance, taking into account the actual location and resource consumption of an aircraft at the beginning of the planning horizon. Each of these stages involves a multitude of decisions which in turn influence other decisions. To tackle these complex planning and operational problems, airlines have been using operations research methods for decades.

Due to the sequential nature of this planning process, it is unlikely that the overall optimal solution can be obtained. Decisions in early planning stages, like flight schedule generation or fleet assignment, restrict the choices in later stages. One example is the fleet assignment problem which is solved absent detailed information regarding aircraft availability. Therefore, the fleet assignment problem usually does not take into consideration that aircraft are temporarily unavailable due to maintenance checks. As a consequence, the aircraft routing problem may be infeasible if an insufficient number of aircraft is available to cover all flights that were assigned to this aircraft type. Additionally, many interdependences between stages exist and will result in higher cost when not considered conjointly. Airlines are aware of these shortcomings and provide feedback loops that enable re-visiting decisions of earlier planning stages. To avoid these issues as much as possible, several publications develop integrated models for the airline scheduling problem (see Section 3.8).

Until recently, models for individual and integrated problems have focused mostly on minimising planned cost by achieving a high utilisation of airline resources. This usually results in highly synchronised schedules with very little buffer times between flights because aircraft and crews spend as little time as possible on the ground. When disruptions occur due to e.g. bad weather or unexpected aircraft maintenance, flights may be delayed or get cancelled. With insufficient buffer times, subsequent flights will be disrupted as well. The airline often has to recover operations at high cost using actions such as providing stand-by crews or compensating passengers. Airlines and researchers have recognised the need for more robust schedules and introduced measurements in the planning stages to increase stability and flexibility of schedules in operation.

However, even with these new robustness measurements, all airline scheduling models suffer from long lead times and long planning horizons. The exact timing
of fleeting, aircraft routing, crew pairing, and crew rostering decisions varies from airline to airline but the literature suggests that fleeting, aircraft routing and crew pairing are all carried out several weeks or months before the day of operations. Crew rostering, as the penultimate stage of the planning process, is done at least two weeks in advance of the day of operations and has a planning horizon of one month.

As a result, decisions will be based on inaccurate information from estimates or forecasts based on flight operations of previous months and years. For example, aircraft fleets are assigned to flights based on historic demand, it is assumed that a certain number of aircraft are available during a specific time period of the planning horizon, or that a certain number of crews are available at crew bases every day. Naturally, information becomes significantly more accurate as the day of operations approaches. The paradigm this thesis is based on allows considering an integrated aircraft routing, crew pairing, and tail assignment problem a few days before the day of operations. We propose to solve the resulting mathematical model about four days before the day of operations, however this decision is up to the airline. At that point in time, it should be possible to predict location and status of crews and aircraft to a high degree of accuracy while sufficient time will be available to evaluate the solution and possibly resolve problems. On the other hand, solving the problem much more in advance will result in diminished returns as a larger number of deviations from the schedule are to be expected.

Unlike other aircraft routing or crew pairing problems, we propose to solve our problem every day using a rolling horizon. Only the decisions made for the first day of the planning horizon will be kept fixed while decisions made for the second day and onwards can be re-evaluated when resolving the problem the next day. As a result, final decisions about any activity are made only four days in advance. The planning horizon is of short duration; we propose up to seven days. A longer planning horizon would require inaccurate data for the last days of the horizon, which is what the paradigm was designed to avoid.

Solving an integrated model this close to the day of operations facilitates considering individual aircraft, i.e. tail numbers. This addresses another major disadvantage of models that are solved a long time before the day of operations. The typical
a aircraft routing problem generates generic routes, that are feasible with respect to only the most frequent maintenance requirement. Because these routes are generic, they do not consider the current state of the aircraft on the day of operations. Often the schedule has to be changed because aircraft are unavailable, a number of aircraft different to what was assumed at the time of the planning process is available at locations, or aircraft require maintenance earlier than anticipated. Four days before the day of operations, information will be available that allows a more accurate prediction of state and location of aircraft. Therefore, instead of generic routes, we propose to generate routes for each individual tail number. These routes can represent specific maintenance requirements, eliminating the need to assign aircraft and possibly having to change the schedule because a feasible assignment is not possible. Unlike the standard approach to the aircraft routing problem, which only considers one type of maintenance, we include all maintenance types to guarantee feasibility. Further, to increase robustness, we require aircraft to remain at a maintenance station for several hours at least every three days. When executing the routes, minor defects that otherwise result in additional unscheduled maintenance can be rectified during these maintenance opportunities.

Implementing the new paradigm will require re-engineering of business processes in airlines. Aircraft maintenance planning, and hence technician rostering, will have to be carried out in a more timely fashion as the routes and thus maintenance demand only becomes known four days in advance. The undoubtedly most significant change will be to the crew rostering process. In current practice crews are notified when and on which flights they work at the time the rosters are released, i.e. weeks in advance. In the new paradigm, this is not possible as the final pairings are not known at that point in time. Instead, crews only get notified about which days they work and which they are off work. The actual routes are then only revealed four days in advance. Such a shift in rostering may not be implementable for all airlines, especially those operating in strongly unionised environments. Crew members are unlikely to forego knowing the exact flights more in advance without some form of compensation. On the other hand, given the increasing competition, airlines strive to gain any competitive advantage. Increased flexibility and more robust schedules due to delayed decision making provides a competitive advantage since operational
cost can be expected to be lower compared to plans that require many changes on
the day of operations.

The goal of this thesis is to show practicality of the new paradigm. We need to
develop a solution algorithm for the resulting mathematical model that is capable
of solving real-world instances in reasonable time.

1.2 Main Contributions

The main contributions of this thesis are as follows.

- We discuss the advantages of solving an integrated aircraft routing, crew pair-
ing, and tail assignment problem close to the day of operations, and discuss the
challenges of doing so. We identify affected business processes and highlight
organisational structures that have to be altered to accommodate this new
approach. We describe how delaying routing and pairing decisions leads to
more flexibility and ultimately lower operational cost. The practicality of this
new paradigm is demonstrated with the development of a solution algorithm,
that can obtain good quality solutions to problems of realistic size, based on
real airline data, in reasonable time.

- We extend the tail assignment literature by modelling a richer set of mainte-
nance requirements for individual aircraft.

- We propose a column generation based mathematical model and solution
method for the integrated aircraft routing, crew pairing, and tail assignment
problem, and investigate with the aid of extensive numerical experiments how
to engineer this algorithm to obtain good solutions in practice.

- We develop a label based solution algorithm for the pricing problems resulting
in our integrated problem. In this algorithm, we extend the preprocessing and
labelling algorithms for the resource constrained shortest path problem with
replenishments as proposed by Smith (2011) to accommodate the problem spe-
cific requirements. These extensions are handling multi-arcs, a non-additive
resource consumption, and cost that increases in non-linear fashion and de-
pends on a resource.
• We develop strategies to address the challenge of a large number of pricing problems, and determine effective criteria for selection of pricing problems to solve at each iteration of column generation. Furthermore, we present extensive numerical results that evaluate alternative parameter settings in these criteria. We show how the time spent solving pricing problems as well as the overall solution time can be reduced significantly.

• As perhaps the strongest methodological innovation in the thesis, we introduce a concept we call superimposed pricing problems, which represent aggregations of the original pricing problems. Solving a superimposed pricing problem implicitly solves all original pricing problems that are represented by the superimposed pricing problem. As a result, a much smaller number of pricing problem has to be solved. We examine and highlight the behaviour of the solution process compared to solving the original pricing problems. We provide numerical results that show how this new method, both independently and when combined with pricing problem selection strategies reduces the time spent in pricing as well as overall solution times.
1.3 Thesis Outline

In the next chapter, we review mathematical solution methods that we employ throughout this thesis. Chapter 3 describes the airline scheduling process and reviews the literature on the scheduling problems relevant to this thesis, namely the aircraft routing problem, the crew pairing problem, and the tail assignment problem. In Chapter 4, we give a detailed problem description. A column generation formulation is presented in Chapter 5. A branch-and-price algorithm is developed in Chapter 6, while the pricing problem solver is described in Chapter 7. We develop selection strategies for the pricing problems in Chapter 8. In Chapter 9, we develop the superimposed pricing problems that can be solved instead of the original pricing problems. Finally, in Chapter 10, we summarise our findings, compare the algorithmic advances made throughout the chapters, and outline future research directions.

A glossary of terms used in this thesis can be found on page 363. Nomenclature of all mathematical notation used in Chapters 5 through 9 can be found on page 369.
Chapter 2

Mathematical Background

In this thesis we develop a mathematical model and a solution algorithm for an integrated airline scheduling problem. In this chapter, we review the mathematical theory and solution algorithms that are frequently employed to solve such scheduling problems. We assume the reader is familiar with basic linear and integer programming theory, specifically the simplex method, duality theory, and the branch and bound method. For more information on these topics, we refer to any linear and integer programming textbook, e.g. (Bertsimas and Tsitsiklis, 1997) and (Bazaraa et al., 2008). We first briefly review Dantzig-Wolfe decomposition and Lagrangian relaxation, and note key features of these techniques that arise when they are applied to problems with block-diagonal structure. We then discuss column generation methods and branch-and-price for solving the resulting master problems, and briefly review models and solution methods for solving the pricing subproblems that commonly arise in airline planning applications.

2.1 Decomposition Methods for Large-Scale Optimisation Programs

The size and complexity of many real world optimisation problems prohibits solving them as a single mathematical optimisation program. The airline scheduling problem in its entirety (see Chapter 3) is an example of such a problem. These large-scale problems are usually decomposed into smaller problems based on their hierarchical nature. However, even the smaller (sub)problems may be too large to
solve using standard optimisation tools. Fortunately, many of these problems exhibit special structures that can be exploited using decomposition methods. The idea is to solve several smaller programs iteratively, passing back and forth information about feasibility and/or solution quality. Two of the more common decomposition methods, Dantzig-Wolfe decomposition and Lagrangian relaxation, are discussed in the following sections. We review their theoretical background and highlight their relationship and applicability.

2.1.1 Dantzig-Wolfe Decomposition

In Dantzig-Wolfe decomposition (Dantzig and Wolfe, 1960), an optimisation problem is decomposed into a master and a pricing problem, each containing a subset of the entire set of constraints. This is achieved by reformulating the original problem from a polyhedral point of view such that the pricing problem is “separated” automatically while also being defined unambiguously. Then, based on its constraints, the restricted master problem chooses a solution from the current set of variables, which in turn represent entire solutions to the pricing problem. Whenever the restricted master problem is solved, the pricing problem evaluates if it can generate a new variable that improves the solution to the current restricted master problem. The Dantzig-Wolfe reformulation is solved using column generation (see Section 2.2).

Consider the following linear program, in this context called original formulation or compact formulation:

\[
\begin{align*}
\text{min } & \quad c^T x \\
\text{s.t. } & \quad Ax \geq b, \\
& \quad Dx \geq d, \\
& \quad x \geq 0,
\end{align*}
\]

where \( c, x \in \mathbb{R}^n, A \in \mathbb{R}^{m \times n}, b \in \mathbb{R}^m, D \in \mathbb{R}^{k \times n}, \) and \( d \in \mathbb{R}^k \). Let constraints (2.2) be complicating or coupling constraints, while constraints (2.3) are such that, in the absence of \( Ax \geq b \), they define a well structured problem, e.g. a network problem.

We define a polyhedral set \( X = \{ x \in \mathbb{R}^n : Dx \geq d, \ x \geq 0 \} \), which we assume to be non-empty and, for convenience of notation, to be bounded\(^1\). Then, using

\(^1\)The unbounded case can be found e.g. in (Bertsimas and Tsitsiklis, 1997)
Minkowski’s Theorem, we can express each \( x \in X \) as a convex combination of a finite number of extreme points \( x_e, e \in E \), where \( E \) is the index set of extreme points in \( X \):

\[
x = \sum_{e \in E} x_e \lambda_e, \quad \sum_{e \in E} \lambda_e = 1, \quad \lambda_e \geq 0.
\]

(2.4)

Substituting (2.4) for \( x \) in program (2.1), we obtain the equivalent extended formulation:

\[
\min \sum_{e \in E} c_e \lambda_e \\
\text{s.t. } \sum_{e \in E} a_e \lambda_e \geq b, \\
\sum_{e \in E} \lambda_e = 1, \\
\lambda_e \geq 0, \quad \forall e \in E,
\]

(2.5)

(2.6)

(2.7)

in which \( c_e = c^T x_e \) and \( a_e = Ax_e \). Program (2.5) is called the Dantzig-Wolfe master problem (DW-MP) and is solved using column generation as it has a much larger number of variables compared to the original formulation. The Dantzig-Wolfe pricing problem (DW-PP) is a linear program of the form

\[
z_{\text{DW-PP}}(\pi, \sigma) := \min (c - A^T \pi)^T x - \sigma \\
\text{s.t. } Dx \geq d, \\
x \geq 0,
\]

(2.8)

where dual values \( \pi \in \mathbb{R}^m, \pi \geq 0 \) are associated with constraints (2.6) and \( \sigma \in \mathbb{R} \) is associated with the convexity constraint (2.7). Since constraints \( Dx \geq d \) define a well structured problem, DW-PP can usually be solved efficiently. DW-MP still contains the complicating constraints, however, in most applications \( m \) is small and thus the program can be solved efficiently as well.

Both upper and lower bounds on the objective function value of DW-MP, \( z_{\text{DW-MP}} \), are available. Let \( z_{\text{DW-RMP}} \) be the optimal objective function value of the restricted DW-MP. Then the following equation holds at any iteration during the solution
algorithm (in particular, during column generation, discussed in Section 2.2):

$$z_{DW-RMP} + z_{DW-PP}(\pi^*, \sigma^*) = \sum_{e \in E} a_e \lambda^*_e - \sigma^* + (c - A^T \pi)^T x^*$$

$$\leq z_{DW-MP} \leq z_{DW-RMP} \quad (2.9)$$

where $\lambda^*$ and $\pi^*$, $\sigma^*$ are the optimal primal and dual solutions to the RMP, respectively, and $x^*$ is the optimal solution to the PP.

The master and pricing problem derived here are for a linear program. When applying Dantzig-Wolfe decomposition to an integer program, the original integer variables $x \in \mathbb{Z}^n$ can be represented in different ways. Let us first define the polyhedron $X = \text{conv}\{x \in \mathbb{Z}^n : Dx \geq d, \; x \geq 0\}$, which we assume to be non-empty but not necessarily bounded. Then, the discretization approach by Vanderbeck and Savelsbergh (2006) uses the property that an integer polyhedron can be expressed using a finite set of integer points and a non-negative integer linear combination of integer extreme rays. On the other hand, in the more popular convexification approach, the solution space is expressed using a finite number of extreme points and extreme rays of the convex hull of $X$. In both cases, integrality is obtained through branching decisions (see Section 2.4). It should be noted that integrality does not have to hold for the $\lambda$ variables but for the original $x$ variables. The pricing problem, on the other hand, now is an integer program. It is thus even more imperative for the pricing problem to be well structured. Usually, constraints $Dx \geq d$ are selected from the original problem with specifically this goal in mind.

As was the case in branch-and-price, a lower bound on the optimal integer solution value of the Dantzig-Wolfe master problem is available (see Section 2.4). Again, the current objective function value of the restricted master problem is not an upper bound on the optimal integer solution value, unless the solution to the restricted master problem is integral.

Applying Dantzig-Wolfe decomposition to an integer program does not only result in solving two or several smaller programs instead of a large complicated one but also usually results in a better dual bound in a branch-and-bound method. Recall that solving an integer program with branch-and-bound involves solving a linear relaxation at every node and making branching decisions if the solution is fractional. The objective function value of the fractional solution then becomes the dual bound
2.1. DECOMPOSITION METHODS

of the child nodes. The quality of this dual bound is of importance as it is used to prune nodes. Reformulating an integer program using Dantzig-Wolfe decomposition results in a stronger formulation, i.e. better bound, compared to the linear relaxation of the original, compact integer program (see e.g. (Nemhauser and Wolsey, 1988)). However, this is not the case if polyhedron $X = \text{conv}\{x \in \mathbb{Z}^n : Dx \geq d\}$ possesses the integrality property, that is if all extreme points of $\{x \in \mathbb{R}^n : Dx \geq d, \ x \geq 0\}$ are integral. In this case, the lower bound from the decomposition principle is equal to the bound obtained from the linear relaxation of the compact formulation.

For more details on the integer and mixed integer case of Dantzig-Wolfe decomposition, the interested reader is referred to e.g. (Wolsey, 1998) or (Desrosiers and Lübbecke, 2005) and Section 2.1.3, in which we discuss the important case where Dantzig-Wolfe decomposition is applied to an IP that has a block-diagonal structure in the constraint matrix.

2.1.2 Lagrangian Relaxation

Lagrangian relaxation (Geoffrion, 1974) relaxes the complicating constraints $Ax = b$ by penalising their violation in the objective function. This leads to the Lagrangian relaxation of problem (2.1):

$$L(\pi) := \min c^T x + \pi^T (b - Ax)$$

s.t. $Dx \geq d,$

$x \geq 0,$

where $\pi \in \mathbb{R}^m$, $\pi \geq 0$ is a penalty vector, usually referred to as Lagrange multiplier. Observe that this problem is equivalent to DW-PP($\pi, \sigma$); the two problems have the same feasible set, and their objective functions differ only by a constant.

$L(\pi)$ is a lower bound on the objective function value of Problem (2.1), since for any feasible solution $\bar{x}$ to (2.1), we have

$$L(\pi) = \min_{x \in X} c^T x + \pi^T (b - Ax) \leq c^T \bar{x} + \pi^T (b - A\bar{x}) \leq c^T \bar{x}. \quad (2.11)$$

The bound clearly depends on the Lagrange multiplier. Thus, to find the best such lower bound, we have to maximise $L(\pi)$ over all $\pi \geq 0$, which can be achieved...
by solving the so-called *Lagrangian dual* (LD):

\[
\max L(\pi) \quad \text{(2.12)} \\
\text{s.t. } \pi \geq 0
\]

A detailed discussion on applying Lagrangian relaxation to integer programs can be found in (Wolsey, 1998).

Dantzig-Wolfe decomposition and Lagrangian relaxation seem to be quite different decomposition principles. However, at least in the linear (not integer) case, it can be shown (Nemhauser and Wolsey, 1988) that they are in fact equivalent to each other, as well as to Benders’ decomposition (Benders, 1962). The column generation method for solution of the Dantzig-Wolfe master problem (discussed in Section 2.2) is equivalent to Kelley’s Cutting Plane algorithm for solving the corresponding Lagrangian dual problem. Moreover, the resulting lower bounds are equivalent as well. We omit the proof at this point as we merely use its result, that is the lower bound given in (2.9) for Dantzig-Wolfe decomposition.

### 2.1.3 Dantzig-Wolfe Decomposition for Integer Programs with Block-Diagonal Structures

The full benefit of Dantzig-Wolfe decomposition only becomes apparent when dealing with programs that exhibit a block-diagonal structure in their constraint matrix. In this section, we discuss the important case when, additionally, the program is an integer program. Many airline scheduling problems, including the one presented in this thesis, exhibit such a structure.

Consider an IP for which the rows of the constraint matrix can be partitioned into \( J + 1 \) subsets, of which \( J \) subsets induce a partition of the columns with the property that no row has nonzero coefficients in more than one column subset, and
one subset contains the coupling constraints:

\[
\begin{align*}
\min & \quad (c^1)^T x^1 + ... + (c^J)^T x^J \\
\text{s.t.} & \quad A^1 x^1 + ... + A^J x^J \geq b, \\
                     & \quad D^1 x^1 \geq d^1, \\
                     & \quad \vdots \geq \vdots \\
                     & \quad D^J x^J \geq d^J, \\
                    & \quad x \in \mathbb{Z}^n.
\end{align*}
\]  

(2.13)

In this case the Dantzig-Wolfe master problem (DW-MP) becomes

\[
\begin{align*}
z_{\text{DW-MP}} &:= \min \sum_{j \in J} \sum_{e \in E_j} c_e \lambda^j_e \\
\text{s.t.} & \quad \sum_{j \in J} \sum_{e \in E_j} a^j_e \lambda^j_e \geq b, \\
& \quad \sum_{e \in E^j} \lambda^j_e = 1, \quad \forall j \in J,
\end{align*}
\]  

(2.14)

(2.15)

(2.16)

(2.17)

where \( E^j, j \in J \) is the index set of extreme points of polyhedron \( X^j = \text{conv}\{x^j \in \mathbb{Z}^{n_j} : D^j x^j \geq d^j, x^j \geq 0\} \), \( n_j \) is the dimension of vector \( x^j \) for all \( j \in J \) and again we assume \( X^j \) for \( j \in J \) to be bounded and non-empty.

Usually, 2.14 is solved using column generation (see Section 2.2), in this case branch-and-price since it is an integer program. For this, we use the Dantzig-Wolfe restricted master problem (DW-RMP) which is equivalent to 2.14, except that only subsets \( \bar{E}^j \subset E^j, j \in J \) exist for a given iteration of the algorithm and that the integrality constraints (2.17) are replaced by

\[
\lambda^j_e \geq 0, \quad \forall j \in J, e \in \bar{E}^j.
\]

Let \( \pi \) be the dual variables associated with Constraints (2.15) and \( \sigma^j \) be the dual variable associated with the \( j \)-th convexity constraints, i.e. Constraints (2.16). The resulting \( J \) pricing problems are

\[
\begin{align*}
z_{\text{DW-PP}}(\pi, \sigma^j) := \min_x (c^j - (A^j)^T \pi)^T x - \sigma^j \\
\text{s.t.} & \quad D^j x \geq d^j, \\
& \quad x \in \mathbb{Z}^{n_j}.
\end{align*}
\]
Then, a solution to the current restricted master problem is optimal only if none of the \( J \) pricing problems returns a negative reduced cost column. The multitude of pricing problems also has to be considered when calculating the lower bound (see e.g. Wolsey (1998)):

\[
\text{z}_{\text{DW-RMP}} + \sum_{j \in J} \text{z}_{\text{DW-PP}}^{j}(\pi, \sigma^{j}) \leq \text{z}_{\text{DW-MP}} \leq \text{z}_{\text{DW-RMP}},
\]

where \( \text{z}_{\text{DW-RMP}} \) is the objective function value of the current Dantzig-Wolfe restricted master problem.

If all block constraints \( D^{j}x^{j} \geq d^{j}, \ j \in J \) are identical, and so the sets \( X^{j} \) for \( j \in J \) are identical, then the original formulation suffers from symmetry, which can be harmful when enforcing branching decisions (Vanderbeck, 2005). These identical subsets arise in our problem when considering identical work-periods for different crews (see Section 4.2). The symmetry can be eliminated by aggregating variables \( \lambda_{e}^{j} \) as follows:

\[
\mu_{e} = \sum_{j \in J} \lambda_{e}^{j}.
\]

The Dantzig-Wolfe master problem then becomes

\[
\begin{align*}
\min & \sum_{e \in E} c_{e}\mu_{e} \\
\text{s.t.} & \sum_{e \in E} a_{e}\mu_{e} \geq b, \\
& \sum_{e \in E} \mu_{e} = J, \\
& \mu_{e} \in \{0, 1\}, \ \forall e \in E,
\end{align*}
\]

(2.18)

where \( E \) is the index set of all extreme points of \( X^{1} = \text{conv}\{x \in \mathbb{Z}^{n_{1}} : D^{1}x \geq d^{1}, x \geq 0\} \) (recall \( X^{1} = X^{j} \) for all \( j \in J \)). As a consequence, the lower and upper bound are as follows:

\[
\text{z}_{\text{DW-RMP}} + J(\text{z}_{\text{DW-PP}}(\pi, \sigma)) \leq \text{z}_{\text{DW-MP}} \leq \text{z}_{\text{DW-RMP}},
\]

where \( \sigma \) is the dual variable associated with the single convexity constraint (2.18).

The lower bounds for block diagonal structure with and without identical blocks are used in combination in our method, (see Section 6.6.1), since the integrated airline planning problem has a mixture of identical and non-identical blocks.
2.2 Column Generation

Column generation, as originally proposed by Ford and Fulkerson (1958) and Dantzig and Wolfe (1960), is a solution technique for linear programs that contain a very large number of columns. Column generation, when used in a branch-and-price framework (see next section), has been successfully applied to large scale integer and mixed integer programs (Lübbecke, 2010), especially to airline scheduling problems where the main challenge is to formulate tractable pricing problems, e.g. (Desaulniers et al., 1997b), (Vance et al., 1997a), (Barnhart et al., 1998a), (Barnhart et al., 1998c), (Gamache et al., 1999), (Klabjan et al., 2001b), (Klabjan et al., 2002), (Coln and Barnhart, 2003), (Lan et al., 2006), (Barnhart et al., 2009a), (Gabteni and Grönkvist, 2009), (Borndörfer et al., 2010), (Weide et al., 2010), (Ionescu and Kliewer, 2011), (Dück et al., 2012), (Saddoune et al., 2012).

In this section, we present the theoretical background of column generation, followed by a brief summary of strategies that have been developed to accelerate column generation. The method described here is known as delayed column generation. Most researchers use the expression column generation to refer to delayed column generation. We likewise follow this practice.

Let us consider the linear program

\begin{equation}
\begin{align*}
    z_{\text{MP}} := & \min \sum_{j \in N} c_j x_j \\
    \text{s.t.} & \sum_{j \in N} a_{ij} x_j = b_j, \quad \forall i \in M, \\
                 & x_j \geq 0, \quad \forall j \in N,
\end{align*}
\end{equation}

where $N$ is the index set of all variables (assumed to be finite), $c \in \mathbb{R}^{|N|}$, $M$ is the index set of constraints, $A \in \mathbb{R}^{|M| \times |N|}$ and $b \in \mathbb{R}^{|M|}$. For the sake of this discussion, we assume 2.19 to be bounded and feasible. In column generation, Problem 2.19 is called the master problem (MP). If $|N|$ is huge, solving it directly using standard LP solution methods is usually prohibitive. Instead, the entirety of set $N$ is considered only implicitly. To this end, we iteratively solve the so-called

\footnote{In classical column generation, all columns need to be known a priori and are considered explicitly in the pricing problem (see e.g. (Bertsimas and Tsitsiklis, 1997)), whereas in delayed CG, as is described in the following, the columns are generated during the solution process.}

\footnote{The unbounded case can be found in (Lübbecke, 2010).}
CHAPTER 2. MATHEMATICAL BACKGROUND

restricted master problem (RMP) and a pricing problem (PP). The restricted master problem is equivalent to the master problem, except that it contains only a subset of variables $N' \subseteq N$. Given an optimal dual solution $\pi^*$ to the RMP, the pricing problem identifies variables that, when added to the RMP, can improve its objective function value. From the simplex method, we know that any non-basic column $j \in N$ that has negative reduced cost, i.e. $c_j - \sum_{i \in M} a_{ij} \pi_i^* < 0$, can improve the objective function value (Bazaraa et al., 2008). If the pricing problem finds any such variable, the corresponding column is added to the RMP, which is then solved again. This constitutes one column generation iteration, or CG iteration for short. If, on the other hand, all variables $x_j \in N \setminus N'$ have reduced cost $c_j - \sum_{i \in M} a_{ij} \pi_i^* \geq 0$, then the current solution to RMP is also optimal for the original master problem, i.e Problem 2.19. If anti-cycling measures are provided, the method guarantees that the optimal solution to the master problem is obtained within a finite number of iterations (Lübbecke, 2010).

In delayed column generation, the pricing problem is not solved by enumerating all variables but by modelling it as an optimisation problem. It should be noted that the ability to solve the pricing problem efficiently is essential to the performance of a column generation algorithm.

In airline scheduling, column generation problems often arise in the form of the LP relaxation of a set partitioning problem, i.e. with $a_{ij}$ binary for all $i \in M$ and $j \in J$, and $b_i = 1$ for all $i \in M$. $M$ usually represents a set of flights to which either crew or aircraft need to be assigned. Each column typically indicates a set of flights that can appear in a feasible sequence of work for either crew or aircraft. Often the pricing problem is formulated as some variant of the shortest path problem (see Section 2.5) in a network. This network can take the form of a time-space, also known as time-expanded network Ford and Fulkerson (1962) but may also be a flight network, which is what we use in this thesis. In a flight network, nodes represent flights in the schedule, and arcs represent possible connections, i.e. pairs of flights that can be operated in sequence by an aircraft or crew.

Column generation suffers from the so-called tailing-off effect. After an initial phase, in which the objective function value of the current restricted master problem $z_{\text{RMP}}$ reduces rapidly (heading-in effect), column generation typically performs many
iterations during which \( z_{\text{RMP}} \) decreases only very little, if at all (in case of primal degeneracy). Figure 2.1 illustrates the convergence of the objective function value.

\[ z_{\text{RMP}} \]

---

Gilmore and Gomory (1963) suggest terminating the column generation algorithm when \( z_{\text{RMP}} \) does not improve by a certain percentage over a given number of iterations. This strategy has two issues. First, the algorithm may terminate even though the algorithm has only reached a stalling point (see Figure 2.1). Second, no measurement on the solution quality is available. As discussed in the previous section, using Lagrangian duality theory (Geoffrion, 1974), we can calculate a lower and upper bound on the objective value of the master problem. If we know that \( \sum_{j \in N} x_j \leq k \), we have for CG iteration \( \nu > 0 \) (Lübbecke, 2010):

\[
z_{\nu}^{\text{LB}} := z_{\nu}^{\text{RMP}} + k \left( z_{\nu}^{\text{PP}} \right) \leq z_{\text{MP}} \leq z_{\nu}^{\text{RMP}}, \quad (2.20)
\]

where \( z_{\nu}^{\text{RMP}} \) is the solution to the RMP in iteration \( \nu \) and \( z_{\nu}^{\text{PP}} \) is the solution to the pricing problem given the optimal dual solution to the current RMP. These bounds can be used to evaluate the quality of the current solution and terminate
the algorithm when an appropriate gap is reached. It should be noted that if any form of heuristic pricing is performed (see below), \( z_{\text{PP}}^v \) is not correct and thus does not give a valid lower bound, unless the pricing problem explicitly provides a valid lower bound on \( z_{\text{PP}}^v \), in which case that lower bound is used instead of \( z_{\text{PP}}^v \). We give a detailed discussion on lower bounds in the presence of multiple pricing problems and heuristic pricing in Section 2.1.3, 6.6.1 and 9.3.

As Figure 2.1 illustrates, the lower bound oscillates, which is due to the high variability of the dual values between CG iterations (see Section 2.3). Therefore, for bounding purposes, the maximum over all lower bounds is kept. We define the dual bound as

\[
  z_{\text{DB}}^v := \max\{z_{\text{RMP}}^v + z_{\text{PP}}^v, z_{\text{DB}}^{v-1}\},
\]

where \( z_{\text{DB}}^0 = -\infty \).

In the remainder of this section, we give a brief overview of commonly used acceleration strategies for column generation. This summary is based on the excellent two-paper series (Lübbecke, 2010) and (Desrosiers and Lübbecke, 2010), which gives a full exposition of column generation and highlights its inherent challenges.

A key observation is that for column generation to proceed, any negative reduced cost column is sufficient as it has the potential to enter the basis. Finding the most negative column is not mandatory at an intermediate iteration. Therefore the pricing problem is often not solved to optimality in every iteration.

Another acceleration strategy is to add multiple negative reduced cost columns per iteration. The advantage is that a larger number of columns is available to enter the basis of the RMP in the next iteration. The idea is to reduce the number of column generation iterations required to reach the optimal solution. The drawback is a possibly large set of columns which results in deteriorating LP solution times. To counter this, columns may be removed from the RMP if they have not been used, i.e. have been non-basic, for several iterations. This strategy is referred to as ageing and is discussed in more detail in Section 6.5.

The RMP can be solved using any LP solver; most often one of the simplex methods or the interior point method is chosen. Adding columns to the RMP does not make the primal solution of the previous iteration infeasible. Thus the warm-start capabilities of the primal simplex method can be exploited. Furthermore, primal
simplex may “sometimes work better on problems where the number of variables exceeds the number of constraints significantly” - which is clearly the case here - “or on problems that exhibit little variability in the cost coefficients” (IBM Corporation, 2011). The dual simplex method, on the other hand, is well suited to solving linear programs that are primal degenerate and have little variability in the right hand side coefficients (IBM Corporation, 2011).

A disadvantage of using either of the simplex methods is that $z_{RMP}$ often converges slowly, thus requiring many column generation iterations. The interior point method, on the other hand, may require fewer CG iterations as it uses more stable dual values (Vanderbeck, 2005) (also see Section 2.3). However, the benefit from requiring fewer CG iterations may be offset by longer LP solution times as the interior point method cannot be warm-started. Details on the choice of LP solver for our problem can be found Section 6.2.

Any acceleration strategy that results in the pricing problem not returning the most negative reduced cost column is called heuristic pricing. This may be due to - but is not limited to - supplying the PP with dual variables that are not optimal for RMP or because the PP does not return the most negative cost column (given the available dual values). While it is not necessary, and often slower, to perform full pricing at every iteration, a full pricing step, i.e. using optimal dual values and solving the PP to optimality, is necessary to prove that the optimal solution to the MP was found.

Many other acceleration strategies for column generation have been developed over the past decades. For a survey, the reader is referred to (Desaulniers et al., 2001), (Desaulniers et al., 2005), (Lübbecke and Desrosiers, 2005), the papers previously mentioned in this section, and Section 2.3. In the next section we discuss some of the more recent advances in column generation. Acceleration strategies that work well for our problem are discussed in detail in Chapter 6.

2.3 Recent Developments in Column Generation

Over the past decade, research on the algorithmic part of column generation has been focussed on two main areas, stabilization of dual values and dynamic constraint aggregation. In the present section, we will only briefly discuss these methods as we
do not employ them in our algorithm.

Apart from the already discussed heading-in and tailing-off effect, column generation often suffers from primal degeneracy and instability in the dual values, which results in slow convergence of the objective function value (Vanderbeck, 2005). Instability of the dual values means that between two column generation iterations, the dual solution may change significantly. Stabilization aims at reducing these effects; the goal is to use better, i.e less extreme, dual information. If successful, many fewer iterations are required. Several methods of stabilization have been proposed, some of which we discuss in what follows. A primal solution corresponds to an extreme point in the dual space. In the next CG iteration, this extreme point is cut-off. Using an interior point, one can cut-off the entire optimal face instead of just the extreme point (Rousseau et al., 2007). Another more simple way to avoid large jumps in the dual space is to consider previous dual solutions. Wentges (1997) describes the weighted Dantzig-Wolfe decomposition method that uses dual values in the pricing problem which are calculated by taking a convex combination of the current dual values and the dual values that gave the best lower bound on \( z_{LP} \).

The by far most popular approach to stabilization is to restrict the change in the dual variables, an idea introduced in the box-step method (Marsten et al., 1975). Here, fixed upper and lower bounds are imposed on the dual variables, which forms a “box” around the current dual solution, also called stability center. If the new dual solution lies on the boundary of this box, the box is moved in the corresponding direction. The idea has been developed further by introducing surplus and slack variables to the restricted master problem (du Merle et al., 1999). In the dual space, this results in restricting the original dual values in a soft interval. A dual variable is allowed to assume a value outside the interval, however, this incurs a penalty. The 3-piecewise linear penalty function of du Merle et al. (1999) is extended to a 5-piecewise linear penalty function by Ben Amor and Desrosiers (2006). The goal is to penalise larger deviations more severely. Oukil et al. (2007) successfully apply the 5-piecewise linear penalty function to a highly degenerate multiple-depot vehicle scheduling problem. These piecewise linear penalty functions approximate a quadratic penalty function (Euclidean distance), as they are used in bundle methods (Kiwiel and Lemaréchal, 2009). Amor et al. (2009) compare the performance when
using the box-step, 3-piecewise and 5-piecewise linear penalty function, and the bundle method on large scale vehicle and crew scheduling problems. They find that for larger and complicated problems, the 5-piecewise linear penalty function works best, while for easier problems the box-step method and 3-piecewise penalty function perform better. The authors also note that while significant gains are achievable, these methods require careful fine-tuning to be effective. Briant et al. (2008) provide a comparison between standard column generation and bundle stabilization, where they use five different combinatorial problems to evaluate the performances. They arrive at a similar conclusion, i.e. the fitness of the approaches is problem dependent.

Valério de Carvalho (2005) and Ben Amor et al. (2006) add valid inequalities to the dual problem, where in the latter they may even cut off optimal dual solutions, although at least one such solution remains. This corresponds to relaxing the primal problem. The authors show how feasibility and optimality of the restricted master problem can be regained. Lee and Park (2011) propose to stabilize column generation by using the Chebyshev center, which is the deepest point inside the polyhedron. In degenerate primal problems, multiple dual solutions exist. Leitner and Raidl (2010), Leitner et al. (2011), and Leitner et al. (2013) use these alternative dual solutions to generate more beneficial columns.

Many large-scale set partitioning problems contain a number of constraints that makes solving the RMP in a column generation context challenging. Elhallaoui et al. (2005) propose to aggregate subsets of set-partitioning constraints, thereby reducing the size of the RMP. The initial aggregation is usually done based on some problem-specific information: tasks that are expected to be covered by the same vehicle or personnel are aggregated. During the CG procedure, the current aggregation of constraints may then be reviewed based on the resulting dual solution. Aggregating constraints also results in aggregated dual values, which have to be disaggregated carefully to ensure proper convergence and optimality. Elhallaoui et al. (2010) extend this method to a multi-phase dynamic constraint aggregation, which uses a partial pricing strategy that favours variables that are compatible or slightly incompatible with the current constraint aggregation. As a result, the constraints are disaggregated at a lower rate. This method is particularly beneficial if a good initial aggregation is known. Saddoune et al. (2012) successfully apply the
methods developed by Elhallaoui et al. (2010) to the integrated airline crew pairing and crew assignment problem.

Elhallaoui et al. (2008) extend dynamic constraint aggregation to also reduce the size of the networks in the pricing problems. In this bi-dynamic constraint aggregation, arcs are removed based on the dual values of the aggregated master problem. In Saddoune et al. (2011), the method is adapted to remove arcs based on reduced costs and on neighbourhoods that may change between CG iterations.

Benchimol et al. (2012) combine dual stabilization and constraint aggregation to a stabilized dynamic constraint aggregation method, which they apply to a highly degenerate multi-depot vehicle scheduling problem. They report impressive reductions by a factor of seven when compared to pure dual stabilization.

2.4 Branch and Price

Pure column generation often results in a fractional solution. However, in many applications an integer solution to the problem is required. Thus, when solving an integer program, column generation is usually combined with the branch-and-bound method to form the so-called branch-and-price method (Barnhart et al., 1998c). In this method, column generation is applied at every node of the branch-and-bound tree. As in the standard branch-and-bound method, a fractional solution at a node is cut off by enforcing branching decisions that forbid the solution. Care must be taken when enforcing these branching decisions in the master and pricing problem. Column generation hinges on the ability to solve the pricing problem efficiently, thus enforcing branching decisions must not destroy the characteristics of a well structured pricing problem.

Branching on fractional columns in the RMP, i.e. on variables $x_j$ with fractional part $0 < x_j - \lfloor x_j \rfloor < 1$ for $x$ the solution to the current RMP, is a very natural branching choice. However, for example in a set partitioning type master problem, it produces a very unbalanced tree (Desrosiers and Lübbecke, 2010). In the 1-branch, a variable is fixed to one, which usually means that many tasks are assigned at once. The 0-branch has only little effect because it only forbids this particular variable. Otherwise, the problem is identical. Furthermore, enforcing the 0-branch in the master problem requires an additional constraint and hence dual variable.
2.4. BRANCH AND PRICE

Considering this dual value in the pricing problem is difficult as it often complicates the well structured pricing problem significantly (Vance, 1998).

A strategy that is widely used for set partitioning type problems is Ryan-Foster branching (Ryan and Foster, 1981). In this scheme, in one branch two rows must have a coefficient of 1 in the same column, while in the other, they must not. The benefit of this branching strategy is two-fold. First, it produces a more balanced branch-and-bound tree compared to variable branching and second, it can be easily imposed in the pricing problem. This branching scheme has been extended to follow-on branching (Vance et al., 1997a). In the context of airline planning, instead of branching on a pair of flights, this branching scheme focuses on a consecutive pair of flights between which a connection is permitted. Again, this strategy is easily enforced in the pricing problem by removing appropriate arcs in the corresponding flight network. All columns that are incompatible with this branching are removed from the master problem. No additional constraints are required in the master problem since, due to the modifications to the pricing problem, the incompatible variables cannot be generated again.

For integer programs, we can calculate a lower bound similar to Equation (2.20). However, in this case, \( z_{\text{RMP}} \) is not an upper bound on the optimal objective function value of the integer program, \( z_{\text{IP}} \), unless the solution to RMP is integral.

The previous section discussed terminating column generation prematurely to avoid the tailing-off effect. In a branch-and-price algorithm, this strategy is called early branching. When a stopping criterion is reached, e.g. lower bound gap or small improvement in objective function value, a branching decision is made based on the current fractional solution to the RMP. We discuss early branching in more detail in Section 6.6.1.

Recall that in branch-and-bound the optimal LP value of the parent becomes the dual bound of the child nodes. The same applies to branch-and-price. However, if early branching was performed at the parent node, \( z_{\text{RMP}} \) is not a valid dual bound for the child nodes. Instead, the dual bound of the parent becomes the dual bound of the child nodes.
CHAPTER 2. MATHEMATICAL BACKGROUND

2.5 Resource Constrained Shortest Path Problems

Many airline scheduling problems have an underlying network structure, e.g. a flight network. In this case, the pricing problem can be formulated as a (constrained) shortest path problem, which is to find the minimum cost path through a network from a given start node to an end node, usually subject to side constraints. Each feasible path represents for example an aircraft route or a crew pairing. Side constraints may for instance be: crews may only be allowed to operate a limited number of flights in a pairing or aircraft require maintenance after a certain amount of flying. Considering these additional constraints leads to formulating the pricing problem as a resource constrained shortest path problem (RCSPP). A detailed introduction to the RCSPP used in the context of column generation can be found in (Irnich and Desaulniers, 2005).

We now define the RCSPP for the purpose of this thesis. Let $G = (\hat{N}, \hat{E})$ be a directed network with a set of nodes $\hat{N}$ and a set of directed arcs $\hat{E}$. Arc $(i, j) \in \hat{E}$ has a cost $c_{ij} \in \mathbb{R}$ and non-negative integer weights $u^k_{ij}$, for each $k \in K$, associated with it, where $K$ is the set of resources. Every resource $k \in K$ has a limit $U^k$ associated with it. The accumulation of resource $k$ at a node is the sum of weights $u^k_{ij}$ of the arcs $(i, j)$ in the path to said node. This accumulation must not exceed the limit $U^k$ at any point in the path. Then, the goal of the RCSPP is to find a minimum cost path in the network from a source node $s \in \hat{N}$ to a sink node $t \in \hat{N}$ while not violating any resource limits.

Several solution methods have been developed, mostly employing some form of labelling algorithm (Desrochers and Soumis, 1988). Important contributions can be found in (Desrosiers et al., 1995), (Desaulniers et al., 1998), (Dumitrescu and Boland, 2003), and (Carlyle et al., 2008). For an extensive overview of solution methods and their properties, the reader is referred to (Irnich and Desaulniers, 2005) and (Irnich, 2008). More recent contributions can be found in (Reinhardt and Pisinger, 2011), (Zhu and Wilhelm, 2012), (Lozano and Medaglia, 2013), and (Zhu and Wilhelm, 2013).
Resource Constrained Shortest Path Problem with Replenishments

Many airline scheduling problems exhibit some form of replenishment; personnel need rest periods and aircraft have to be maintained. A way of considering this in a column generation formulation is to implicitly handle the replenishments in the master problem, while the pricing problem generates paths between two replenishments. Two examples in airline scheduling are string based models (Barnhart et al., 1998a) and duty networks (Vance et al., 1997a). String models use a column generation master problem in which a column represents a sequence of flights followed by a maintenance. This is called a string. The strings are generated by a pricing problem. Then, when concatenating two strings, the maintenance between the sequences of flights replenishes all resources. In duty networks, first duties are generated and then, in a second step, pairings are formed by sequencing several duties. Resources which solely restrict duty generation only need to be considered in the first stage, thus do not need to be reset, i.e. replenished, when a duty finishes.

Replenishments can be considered in the pricing problem instead of the master problem. The resource constrained shortest path problem with replenishments (RCSPP-R) extends the RCSPP by considering replenishment arcs on which resources are reset. Formally, the RCSPP-R is defined as finding a minimum cost path in a directed network $G = (\hat{N}, \hat{E})$ from a source node $s \in \hat{N}$ to a sink node $t \in \hat{N}$ while not violating any limits $U_k$ on the accumulation of resource $k \in K$ at any time in the network. Every arc $(i, j) \in \hat{E}$ has a cost $c_{ij}$ and non-negative integer weights $u_{ij}^k$, for each $k \in K$, associated with it. The network contains replenishment arcs $(i, j) \in \hat{E}^k \subseteq \hat{E}$, on which resource $k \in K$ is reset to zero. (Smith, 2011).

The RCSPP is NP-hard, which was proven by (Garey and Johnson, 1979). It follows that the RPCSPP-R is NP-hard as well since the former is a special case of the RPCSPP-R. Recently in her thesis, Smith (2011) extended the preprocessing ideas of Dumitrescu and Boland (2003) to accommodate the RCSPP-R. Several solution algorithms like Lagrangian bounds (Smith, 2011), meta-networks (Smith et al., 2012), and branch-and-bound methods (Smith and Boland, 2013) are developed. More details on the RCSPP-R and solution algorithms can also be found in Chapter
7, in which we develop a variant of the RCSPP-R that is suitable for a graph with multi-arcs, non-linear cost, cost that are functionally depended on resources, and non-additive resources.
Chapter 3

Airline Scheduling

One of the goals of this thesis is to integrate three airline scheduling stages, namely aircraft routing, crew scheduling, and tail assignment. This chapter places airline scheduling in context with the airline planning process and introduces terminology used in the airline industry. The individual stages of airline scheduling are discussed in detail, especially the stages relevant to this thesis (see Sections 3.4, 3.5, 3.7). In the beginning of each section, we describe the relevant processes, followed by key publications on the individual stage. Section 3.8 describes issues inherent to the sequential airline scheduling process and reviews papers that propose models integrating several stages. The chapter concludes by summarising the state of the art in airline scheduling and drawing conclusions about suitable algorithms for the integrated aircraft routing, crew pairing, and tail assignment problem.

The airline industry is very competitive. The emergence of low cost carriers like Southwest Airlines and RyanAir has significantly increased competition while variability of demand due to e.g. terrorism and the financial crises together with rising oil prices increase financial risks (Weide, 2009; Wu, 2010). To counter these challenges, many airlines have grown through network expansion or mergers. As a result, many major airlines operate several hundred aircraft and employ tens of thousands of employees, making planning and execution of airline operations a very complex task.

The quality of air transportation as perceived by the customer is determined by availability of airfares, delays, in-flights service, baggage handling, and ticket prices. These criteria are influenced by an airline’s operations such as demand forecast,
market selection, assignment of aircraft and crews to flights, ground operations, and disruption management. Airlines need to plan and execute these operations efficiently to be able to offer a high quality product at a low price (Wu, 2010).

Airlines have employed operations research methods to tackle their complex operations since the late 1960s. Simulation and optimisation models have been developed specifically for and successfully employed by the airline industry.

Airline operations can be classified into strategic planning, tactical planning, and operational management. Similar to other industries, the classification is mainly time based and with reference to the day of operations (DOO), i.e. the day a plan is put into action. Strategic planning includes acquisition of aircraft, long-term market forecast, route structure, determining location of crew bases, etc. These activities are usually carried out more than 12 months in advance and determine the airline’s long term success.

Tactical planning starts about 12 months before and lasts until the DOO. It includes pricing and revenue management problems, demand forecast, and airline scheduling. Due to its complexity, the latter is usually decomposed into six individual planning problems, which are solved sequentially.

First, the schedule design problem uses demand forecast to determine which markets are serviced with what frequency. Second, an aircraft type is assigned to every resulting flight in the fleet assignment problem so that estimated demand is matched closely. The subsequent aircraft routing problem selects a minimal cost set of routes such that all flight legs are covered. Generic aircraft routes are created by selecting a number of flights that have to be flown in sequence. While general maintenance requirements like days between maintenance checks have to be respected, the actual state of an aircraft, for example time spent flying so far, is ignored. In the crew pairing problem, generic pairings are generated such that the crew costs are minimised. A pairing is generated by selecting a number of flights that are serviced in succession by a crew while obeying several restrictions such as limits on time spend working or over-night rest periods. The pairings are put together to monthly rosters in the crew rostering problem. These are then assigned to actual crews based on personal preferences and requirements. In the tail assignment problem, the generic routes are assigned to individual aircraft (also called tails or tail numbers).
On the DOO, operations management has to implement planned schedules and make adjustments whenever necessary. Plans produced by airline scheduling problems are delicate schedules with a high degree of synchronisation among flights, ground operations, aircraft movements, crew movements, passenger itineraries, and general airport operations. Disruptions from mechanical problems, crews, late passengers, weather, etc. are commonplace and can disturb airline operations significantly. If disruptions occur, operations management, often in the form of the airline operations control centre, has to delay flights or reschedule aircraft, crews, and passengers. These decisions have to be made quickly and are often made manually or using heuristics that deliver good solutions within minutes. Surveys of disruption management and irregular airline operations can be found in (Butler and Keller, 2000; Ball et al., 2007; Wu, 2010; Clausen et al., 2010).

The goal of most mathematical models presented in airline scheduling literature is to generate minimal cost solutions by scheduling aircraft and crew such that they spend a minimal amount of time on the ground. This maximises aircraft and crew utilisation but results in very short connection times. Even minor disruptions may cause delays that propagate through the network as insufficient buffer times are available. While the planned cost is minimal, often costs associated with recovery operations are very high as more standby crews are needed, passengers may have to be compensated, etc. Moreover, not all costs are easy to quantify. On-time-performance is likely to deteriorate, resulting in a bad reputation and fewer customers.

An emerging trend in airline scheduling is to not only focus on minimal cost solutions but to simultaneously consider robustness. The goal is to generate schedules that are less sensitive to disruptions and/or are easier to recover from. Models focusing on robustness can be found in (Kang, 2004; Listes and Dekker, 2005; Pilla, 2006; Wu, 2006; AhmadBeygi et al., 2010). Additional models are discussed in the following sections.

Recent surveys on airline scheduling and airline operations in general can be found in (Gopalakrishnan and Johnson, 2005; Barnhart, 2009b; Grosche, 2009; Cohn and Lapp, 2010; Wu, 2010).

In the following section, common characteristics of underlying networks used in
CHAPTER 3. AIRLINE SCHEDULING

airline scheduling are described. The individual stages are discussed in more detail thereafter. We present key literature on all stages and give in depth discussions on models and algorithmic advances for the three stages relevant to this thesis. The individual stages are discussed first, followed by publications integrating two stages and then three stages.

3.1 Terminology of Airline Scheduling Networks and Models

Passengers usually wish to travel between two cities. In the airline industry this is called an origin and destination pair (O-D pair). The passengers generally do not care via which route they travel from origin to destination except that they prefer a quick and simple route, i.e. few aircraft changes. Naturally, airports in larger cities experience higher demand and therefore have a larger number of arriving and departing flights. Usually, these airports are connected to a larger number of other airports and there may be more frequent flights. On the other hand, most smaller cities experience low demands and it may therefore not be profitable for an airline to fly directly from such a location to several others (Wu, 2010).

This led to the development of hub-and-spoke networks in the USA after deregulation of its aviation market in the late 1970s. Low demand airports (spokes) are connected to larger airports (hubs) via direct flights but usually no direct flight between two spokes exists. Passengers wishing to travel between two spokes will have to travel via at least one hub. This increases travel time compared to direct flights but has the advantage that an airline can offer low demand O-D pairs.

To reduce travel time, activity at hubs comes in waves, also called banks. A number of incoming flights is followed by a number of outgoing flights. This allows for a large number of relatively short connections between flights.

An advantage of hub-and-spoke networks from an operational point of view is that because of the large number of incoming flights, and therefore aircraft, maintenance activities can be centralised at hubs. In fact most airlines operate only a small number of such maintenance stations. For similar reasons crew bases, i.e. the location crews have to return to after their work week, are located at major hubs.
Hub-and-spoke networks may mitigate the effect of the sequential approach. If flights arrive in banks, several opportunities to re-route aircraft or crews exist. By reassigning (swapping) an aircraft, a different aircraft type may be assigned to a flight, thus re-evaluating the decisions of the fleet assignment. Further, several re-routing opportunities exist so that an aircraft can get to a maintenance station earlier if this is necessary. Since hubs are often maintenance stations, the aircraft may simply get maintained early. Similarly, because crews are often based at hubs, they may finish pairings early in the event of disruptions. Further, because of the large number of outgoing flights, hubs are a good choice for positioning *standby crews*. A standby crew has no pairing assigned to it in the original schedule but may be assigned to fly parts of another crew’s pairing if the original crew is delayed or cannot continue the pairing.

However, any such change does not simply affect the aircraft or crew under consideration but also the aircraft or crew that the first one is swapped with. Generally such disruptions propagate through the network by triggering further reassignments and, if not done carefully, trigger additional delays. These may cause major disruptions later on in the schedule.

Beginning in the early 2000s, more and more airlines *de-bank* their hub operations (Jiang and Barnhart, 2009). Banked operations provide many passenger connections but require many aircraft to be on the ground at the same time and often for a longer duration thus decreasing aircraft utilisation. The larger amount of aircraft and passengers that need to be processed during banks create peak demands for manpower, equipment, and infrastructure facilities. However, because of the nature of banked schedules, these resources are idle during non-peak times, while still incurring cost. In de-banked operations flight arrivals and departures are not highly coordinated. While resulting in fewer passenger connections, productivity of aircraft, crews, and other resources is increased as demand is smoothed.

Conversely, **point-to-point networks** offer a larger number of direct flights between two cities given the demand is sufficient to sustain these flights. This type of network was pioneered by Southwest Airlines after the deregulation of the US market. Today a large number of *low cost carriers* (LCCs) use point-to-point networks, e.g. Ryanair in Europe and Virgin Australia in Australia. LCCs focus on markets with high
demands, thus realising high utilisation of their seat capacities.

The disadvantage of this type of flight network is the lack of a centralised airport serving as crew base and maintenance station. Further, fewer swapping opportunities for aircraft and crews exist. As a result, maintenance has to be planned more carefully and crews may have to deadhead in the event of disruptions if they otherwise exceed their allowed working hours.

Traditionally, airlines in the US market operate hub-and-spoke networks while, in the rest of the world, point-to-point networks are more common. However, today most airlines actually operate a mixed network. To be able to compete with LCCs, airlines with hub-and-spoke networks offer additional spoke-to-spoke flights for high demand O-D pairs. LCCs, on the other hand, want to reduce cost by centralising operations at a major airport, and as a result often offer more flights to this location. An example is AirAsia with its main hub in Kuala Lumpur, Malaysia.

When an aircraft arrives at an airport, it is parked at a gate, passengers disembark, luggage is unloaded, and the aircraft is prepared for its next flight. The crew connects to its next flight by entering the airport and walking to the gate from which its next scheduled flight departs. Even though the aircraft does not physically move it also “connects” to its next scheduled flight. Regardless of aircraft or crew, a feasible connection is a sequence of two flights where the first flight ends at the same airport the second departs from. Further, sufficient time must be allowed between the two flights.

The minimum time for an aircraft connection is known as minimum turn time (MTT), also known as turnaround time. Several activities such as disembarking, unloading luggage, cleaning, loading luggage, refuelling, and boarding have to be performed between two flights. The duration of these activities depend on the capacity of an aircraft. Therefore the minimum turn time depends on the aircraft type.

The minimum connection time for crew connections is referred to as minimum sit time (MST). An incoming crew has to enter the airport, travel through the airport to the departure gate, and perform several aircraft checks prior to departure. During the crew pairing stage, the gate assignment is not known and therefore the planner has to assume a worst case scenario and account for long travel times between gates.
The travel time and thus the minimum sit time depends on the size of the airport.

Generally, the MTT is shorter than the MST. However, when the crew does not have to travel through the airport because they work on the same aircraft as on the previous flight, the MTT also applies to the crew connection. This type of connection is known as a short connection. Using many short connections is beneficial with respect to utilisation as crews spend less time on the ground. Further, robustness may be increased: if a flight is delayed and the crew and aircraft are scheduled for a short connection, only one, the successor flight is delayed. In contrast, if the crew and aircraft connect to different flights, the two following flights may be delayed if only little connection times exist.

In the crew pairing stage of the sequential approach, the possible short connections are limited as they are determined by the aircraft connections in the solution of the aircraft routing problem. This highlights one of the main disadvantages of the sequential approach. The aircraft routing problem is solved without consideration of crew connections and thus crew cost or robustness.

In an integrated aircraft routing and crew pairing problem, all crew connections with a duration between the MTT and MST are included in the underlying network but such a connection can only be scheduled if an aircraft and a crew is scheduled for the same connection, i.e. the crew stays with the aircraft.

The concept of short connections was generalised by (Mercier et al., 2005). A connection with a duration between the MST and MST + 30 or 45 minutes, depending on publication, is considered a restricted connection. If a flight is delayed and the servicing crew is scheduled to have a restricted connection while the aircraft uses a different connection, i.e. an aircraft change occurs, the delay of the first flight is likely to be propagated by the crew connection. To increase robustness, a penalty is incurred whenever such a restricted connection is scheduled for a crew without scheduling the same connection for an aircraft.

Many airlines in North America offer the same flights on each day during the week and only a subset of these flights on weekends. European airlines typically run more irregular schedules (Gopalakrishnan and Johnson, 2005). Depending on the regularity of their flight network, airlines solve daily, weekly, or dated problems in the aircraft routing and crew pairing stages, while in fleet assignment, often a daily
repeating schedule is assumed (Hane et al., 1995; Clarke et al., 1996; Talluri, 1996; Desaulniers et al., 1997b; Gopalan and Talluri, 1998a; Barnhart et al., 2002; Sherali et al., 2006).

In daily problems, feasible sequences of flights are found such that all flights during a typical day are covered. These sequences are then concatenated to form legal pairings and routes. Airlines using this approach have to solve a weekly exception problem to account for the different set of flights operated on weekends. To create a monthly schedule, the solution of the daily problem in conjunction with the weekly exception problem is repeated for every week of the month. Further exceptions may be necessary if flight schedules differ for a specific week, resulting in even higher cost due to additional adjustments.

In weekly problems, a set of pairings and routes has to be found such that all flights during a week are covered. The advantage is that exceptions due to the weekend are automatically considered. Because this schedule is repeated weekly, a regular week has to be chosen when generating the pairings and routes. For irregular weeks such as weeks including a long weekend, the solution has to be adjusted. European airlines face more irregular schedules than airlines in North America. Therefore they usually solve weekly problems because modifying the schedule of a daily problem may incur unacceptably high cost (Gopalakrishnan and Johnson, 2005).

A dated approach, on the other hand, includes the actual flights of every day in the planning horizon. Therefore, all exceptions are accounted for and no adjustments have to be made other than those due to disruptions on the day of operations. The planning horizon is usually one month (Gopalakrishnan and Johnson, 2005). The obvious drawback is that due to the longer horizon the problem is much harder to solve. For large airlines, a monthly schedule contains many thousands of flights and therefore a dated approach may be prohibitive. Airlines may introduce new flights at the beginning of a month or when a new schedule is introduced. In this case even airlines that usually solve daily or weekly problems solve a dated transition problem (Gopalakrishnan and Johnson, 2005).

Lacasse-Guay et al. (2010) give an in depth discussion of dated, daily, and weekly approaches. Resulting string based, big-cycle, and one-day models for the aircraft
3.1. TERMINOLOGY OF AIRLINE SCHEDULING NETWORKS AND MODELS

Routing problem are compared with respect to complexity and robustness.

Most literature on aircraft routing and crew pairing considers daily problems (Desaulniers et al., 1997b; Vance et al., 1997b; Klabjan et al., 2002; Mercier et al., 2005; Sandhu and Klabjan; Mercier and Soumis, 2007; Gao et al., 2009; Papadakos, 2009). Weekly problems are discussed in Ioachim et al. (1999), Klabjan et al. (2001a), Sriman and Haghani (2003), Sherali et al. (2010), Weide et al. (2010), and Sherali et al. (2011). Only few models consider dated problems (e.g. Cordeau et al. (2001)). Tail assignment problems on the other hand are usually dated problems as they are solved much closer to the day of operations (Sarac et al., 2006; Grönkvist, 2005; Gabteni and Grönkvist, 2009; Saddoune et al., 2013; Froyland et al., 2012; Lapp and Wikenhauser, 2012).

Flight networks can be modelled as connection networks or time-line networks. In a connection network a node represents a flight leg, while an arc between two nodes indicates that a feasible connection between the corresponding flights exists.

A small fictional example of a connection network is shown in Table 3.1 and Figure 3.1. It shows activities at the Sydney airport (SYD), with flights departing for and arriving from Melbourne (MEL) and Brisbane (BNE).

<table>
<thead>
<tr>
<th>Flight No.</th>
<th>From</th>
<th>To</th>
<th>Dep. time</th>
<th>Arr. time</th>
</tr>
</thead>
<tbody>
<tr>
<td>EX01</td>
<td>MEL</td>
<td>SYD</td>
<td>18:30</td>
<td>20:05</td>
</tr>
<tr>
<td>EX02</td>
<td>BNE</td>
<td>SYD</td>
<td>19:00</td>
<td>20:30</td>
</tr>
<tr>
<td>EX03</td>
<td>SYD</td>
<td>BNE</td>
<td>20:15</td>
<td>21:45</td>
</tr>
<tr>
<td>EX04</td>
<td>SYD</td>
<td>MEL</td>
<td>21:00</td>
<td>22:35</td>
</tr>
<tr>
<td>EX05</td>
<td>BNE</td>
<td>SYD</td>
<td>21:55</td>
<td>23:25</td>
</tr>
<tr>
<td>EX06</td>
<td>SYD</td>
<td>MEL</td>
<td>22:00</td>
<td>23:35</td>
</tr>
</tbody>
</table>

Table 3.1: Example timetable for SYD

The connection network is very flexible as it allows to specify the meaning of an arc. For example two arcs between two nodes may exist, one representing that an aircraft is simply scheduled to service the two flights in sequence, while the other arc means that in addition to servicing the two flights, a maintenance check of some type is performed in-between. Such additional arcs can only exist if enough time is
available to carry out a maintenance check between the two flights.

Furthermore, it is possible to forbid certain feasible sequences of flights by simply removing the corresponding arcs from the network. For the previous example, let us assume a minimum turn time of 10 minutes, and that Brisbane has a curfew at 22:00. Without any delays, flights EX01, EX03, and EX05 can be operated by the same aircraft. However, if flight EX01 is delayed, EX03 and further EX05 will be delayed as well. If the delay is more than five minutes, the aircraft is then not allowed to operate flight EX05 since the departure of that flight would be pushed past the curfew time. The aircraft is thus not able to return to Sydney on the same day causing serious disruptions. If historical data indicates that EX01 is frequently delayed by more than five minutes, the connection between EX01 and EX03 should be forbidden to prevent the disruption. This can be achieved by removing the arc between EX01 and EX03.

On the other hand, in some cases certain connections must be made. This can be achieved by removing all arcs other than the one representing the connection that has to be made.

Connection networks are used in (Cordeau et al., 2001; Cohn and Barnhart, 2003; Mercier et al., 2005; Grönkvist, 2006; Gabteni and Grönkvist, 2009) to name just a few.

In a time-line network, also called time-space network, each airport is represented by a time line. Nodes along the time line correspond to arrival and departure times of flights. The arrival times are typically adjusted to include the minimum turn time and minimum sit time, respectively. Two types of arcs exist. Arcs between airports
3.2. SCHEDULE DESIGN

are flight arcs and arcs along the time-line of an airport are called ground arcs. The latter represents that all aircraft assigned to the arc remain at the airport between flight events.

In the time-line network it is not possible to distinguish between individual connections. It is therefore impossible to forbid or enforce connections. Also, cost cannot be assigned to an individual connection. On the other hand, the time-line network is much more compact than a connection network.

The above example is modelled using a time-line network in Figure 3.2. Ground arcs are depicted as dotted lines, while flights arcs are solid lines.

Figure 3.2: A time-line network for the example given in Table 3.1.

Models based on time-line networks can be found in (Hane et al., 1995; Clarke et al., 1997; Barnhart et al., 1998a; Rexing et al., 2000; Klabjan et al., 2002; Sandhu and Klabjan; Papadakos, 2009; Dück et al., 2011).

The following sections describe the individual stages of the sequential scheduling approach and review key publications for each of them.

3.2 Schedule Design

In schedule design, an airline uses market models to forecast demand for O-D pairs and subsequently determine which direct flights to offer between cities, their frequency, and the departure and arrival times of each flight. This is a crucial process as it indirectly determines the profitability of an airline. Unfortunately, designing an optimal schedule is impractical due to the numerous factors influencing this process. External factors such as flights offered by competitors, synchronisation opportunities with partners, and spill cost have to be considered. Internal factors include passen-
ger itinerary forecasts as well as all aircraft routing and crew scheduling decisions of
the airline. Because the resulting timetable is used as input for aircraft routing and
crew pairing, not only is the schedule design influenced by but also does it influence
these subsequent stages. Furthermore, the demand an airline experiences in turn
depends on the capacities offered (Akartunalı et al., 2013).

Due to its complexity, schedule design is typically carried out manually and in
an incremental fashion. New markets are added to or dropped from an existing
schedule by adding/removing flights to the servicing airport. Offered capacities for
existing markets can be varied by changing the frequency of flights. Often, only small
changes are made as the airline and customers prefer a high consistency between
one schedule and the next.

Generally, schedules are not changed often. Most airlines release between one
and four flight schedules per year, often a winter and a summer schedule. Schedule
design typically starts 12 months before and is completed three months before the
day of operations.

A detailed description of the schedule design process can be found in (Etschmaier
and Mathaisel, 1985) as well as (Akartunalı et al., 2013) and references therein. Be-
cause of its complexity and incremental nature, schedule design has not received
much attention in the literature except when departure times are allowed to vary
within time windows, an idea proposed by Levin (1971). In this approach, also
known as flight re-timing, the departure time can be restricted to only a few choices
for each flight (e.g. Mercier and Soumis, 2007) or the departure time may be chosen
from a continuous period (e.g. Desaulniers et al., 1997b). Time windows are used
especially in integrated models, see Section 3.8. An exception is the approach pre-
sented in Lohatepanont and Barnhart (2004), in which sets of flights can be added
or dropped from the current schedule.

3.3 Fleet Assignment

The goal of fleet assignment is to find an optimal assignment of fleet types to each
flight in the schedule. This process is crucial as it determines projected revenue and
affects operating cost associated with each flight. Operating cost depends on the fleet
type as fuel cost and landing fees differ between aircraft types. Fleet assignment
further affects crew cost as the number of cabin crew and ground crew members required to service an aircraft varies between fleets.

Another cost that is often considered are spill costs. An aircraft type has a certain number of seats and therefore limits the number of customers that can be transported on the flight the aircraft is assigned to. If the realised demand exceeds the capacity of the assigned aircraft, passengers are spilled. These customers are either lost to competitors or recaptured by the airline by transporting them via a different flight or route. On the other hand, assigning too large an aircraft may result in “wasted” seat capacity.

Because of its direct impact on profit, fleet assignment has received significant attention in the industry and literature. Abara (1989) and Hane et al. (1995) propose a basic fleet assignment model (FAM). The problem is formulated as a multi-commodity flow problem with side constraints. Hane et al. (1995) use a time-line network and introduce preprocessing methods such as node aggregation and identification of isolated islands at stations. The preprocessing techniques enable solution of large instances of up to 2500 flights and 11 fleet types.

A major disadvantage of the classical FAM is that it assumes spill cost to be flight based. In reality many passengers, especially in a hub-and-spoke network, have multi-leg itineraries. Therefore, revenue and thus spill cost are passenger itinerary based. For this reason, FAM has been enhanced by considering through-values, also called through-fares (Gopalan and Talluri, 1998a; Jarrah and Strehler, 2000; Ahuja et al., 2007). Passengers prefer to fly directly from origin to destination. If, however, this is not possible, they prefer to stay on the same aircraft, thus minimising the risk of missing a connection or losing luggage. A through fare captures the additional amount customers are willing to pay for this convenience.

Further development led to itinerary-based fleet assignment models (Barnhart et al., 2002, 2009a). If a passenger that has a multiple flight itinerary gets spilled because not enough capacity is offered on one of the flight legs, the whole itinerary is lost. It is unclear how to model this in a basic fleet assignment model as the spill cost have to be assigned to individual flight legs. Itinerary-based fleet assignment models include variables that represent the assignment of passengers to actual itineraries, eliminating the need to calculate spill cost based on flight legs. While the classic
CHAPTER 3. AIRLINE SCHEDULING

FAM is relatively easy to solve using today’s commercial solvers, itinerary-based fleet assignment models are much more complicated. The authors propose a heuristic column generation method (2002) and a branch-and-price algorithm (2009a).

Another area in fleet assignment that has received significant attention is re-fleeting. Fleet assignment is typically done months before the DOO because it is required as input for aircraft routing and crew pairing. At that point in time, demand for a flight is based on forecast and may change significantly as the DOO approaches. Dynamic re-fleeting models have been developed, facilitating reacting to such demand changes by reassigning a larger or smaller aircraft to the flight. One example of re-fleeting is demand-driven dispatch which was introduced by Berge and Hopperstad (1993). Given an initial fleet assignment, fleet types are re-assigned several times as the DOO approaches. The authors propose two heuristics instead of solving the problem to optimality. Talluri (1996) develops a more general aircraft swapping algorithm that is based on the work by Berge and Hopperstad (1993).

Solution robustness is improved in the work of Smith and Johnson (2006). The authors develop a model that limits the number of fleet types that can serve each airport, thereby increasing robustness. This strategy is called “station purity”. A column generation method is proposed that uses heuristic fixing to overcome highly fractional LP solutions.

Other recent publications include (Sherali et al., 2005; Sherali and Zhu, 2008). Detailed reviews of fleet assignment can be found in Gopalan and Talluri (1998a); Sherali et al. (2006); Shebalov (2009). Fleet assignment has significant impact on cost since it is used as input to aircraft routing and crew scheduling. The assignment of fleet to flights constrains choices in these problems. Therefore, significant effort has been made to integrate fleet assignment with aircraft routing and/or crew pairing. Several models have been developed in the literature and are reviewed in Section 3.8.

3.4 Aircraft Routing

This section describes the aircraft routing process and approaches to modelling the problem. Mostly early publications are reviewed in this section, while more recent papers, which integrate aircraft routing with other stages, are discussed in Section
3.4. AIRCRAFT ROUTING

3.8. A summary of the literature is provided in section 3.9.

The aircraft routing problem, or maintenance routing, determines routes so that all scheduled flights are covered by an aircraft. A route, sometimes also referred to as a routing, is a feasible sequence of flights that allows for at least the minimum turn time between two flights. The end location of a flight must be the same as the start location of its successor. In some publications a line-of-flights denotes a sequence of flights on a single day, while a series of several such line-of-flights constitutes a route.

In addition to sequencing requirements, all maintenance requirements that a later assigned aircraft has must be satisfied by the route. Maintenance requirements are dictated by manufacturers and aviation authorities. Often airlines specify even more restrictive maintenance requirements to decrease the likelihood of unscheduled maintenance. Maintenance requirements depend on the aircraft type but generally are categorised into four categories, A, B, C, and D. The categories roughly represent the frequency with which they have to occur. A maintenance of Type A has to occur every 60 flight hours, which is usually approximated by requiring a Type A check every three to four days. It includes inspection of all major systems and takes about four hours. Type C and D checks are performed every one to four years. The aircraft is taken out of service for several weeks as many components are taken apart or are replaced. Maintenance checks can only occur at specific maintenance stations that have appropriate facilities and skilled technicians.

A maintenance check has to be carried out before specific indicators are exceeded. These indicators are the actual time (AT) between two checks of the same maintenance type, the amount of flying time (FT), and the number of pressure cycles (PC), i.e. take-offs, since the last maintenance of the same type. Not every maintenance type is triggered by all three indicators. The hypothetical maintenance type C001 in Table 3.2 has to be carried out at least every 12 months, while C002 has to be carried out at least every 5000 hours of flying or before the aircraft accumulates 2000 pressure cycles, whichever happens first. In the following, we will refer to a maintenance type-indicator pair as a resource. Thus, C002-FT and C002-PC are two separate resources.

Modelling several types of maintenance increases complexity significantly. There-
CHAPTER 3. AIRLINE SCHEDULING

<table>
<thead>
<tr>
<th>Maintenance</th>
<th>Actual Time</th>
<th>Flying Time</th>
<th>Pressure Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>A001</td>
<td>60 hours</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>C001</td>
<td>12 months</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>C002</td>
<td>-</td>
<td>5000 hours</td>
<td>2000</td>
</tr>
<tr>
<td>C003</td>
<td>36 months</td>
<td>10000 hours</td>
<td>4000</td>
</tr>
</tbody>
</table>

Table 3.2: Hypothetical maintenance types

Therefore, almost all aircraft routing models proposed in the literature make simplifications. Some models only consider maintenance implicitly by requiring that every aircraft has an overnight stay at a maintenance station without considering the e.g. flying time or number of take-offs along the route. Almost all other publications only consider A-checks and simply require a route to visit a maintenance station before some maximum time between two A checks is exceeded. Modelling maintenance explicitly is one of our main contributions. Therefore, in the following we will discuss which assumptions with respect to maintenance have been made in published work whenever applicable.

A common issue with all aircraft routing problems is that the location of the aircraft on the DOO and its accumulation of resources (flying time, number of take-offs, etc.) since the last maintenance of each type is unknown. Therefore, only generic routes can be generated.

Often, the objective of the aircraft routing problem is to simply find a feasible set of routes such that all flights are covered. However, the problem can be extended to finding a solution that maximises through-values as was introduced for the fleet assignment problem (Section 3.3) (Gopalan and Talluri, 1998a). The objective function can also be used to increase robustness by assigning a penalty to unfavourable connections.

An aircraft routing problem is solved for each fleet type separately. Only flights that were assigned to this fleet type in the preceding fleet assignment are considered. Models that integrate aircraft routing and fleet assignment have been developed and are discussed in Section 3.8.

Several airlines, especially in North America, require that each aircraft in a given
fleet is assigned to every flight in the schedule. The advantage of this is equal wear and
and tear on every aircraft. This *big-cycle* requirement is honoured by concatenating
all created lines-of-flight to one cycle. Each aircraft then flies this cycle but starts
the cycle with a different line-of-flight. However, as these cycles are often very long
(weeks), deviations due to disruptions are to be expected. Mathematically the big-
cycle requirement is equivalent to finding a Hamiltonian path in the network (Clarke
et al., 1997; Barnhart et al., 1998a).

Detailed descriptions of the aircraft routing problem can be found in (Clarke
et al., 1997; Gopalan and Talluri, 1998b; Grönkvist, 2005).

In early contributions, the aircraft routing problem is formulated as a multi-
commodity flow problem (e.g. Feo and Bard, 1989) or travelling salesman problem
(e.g. Clarke et al., 1997). The majority of recent models are set partitioning/set
covering formulations, in which variables represent routes. The advantage of these
models is that most maintenance requirements can be considered implicitly by only
including maintenance feasible routes in the set partitioning/covering problem.

Feo and Bard (1989) solve a combined aircraft routing and maintenance base
location problem. The goal is to find the minimal number of maintenance bases while
respecting four day maintenance requirements. Two different models are proposed.
An arc based model is found to be too complicated while a path based model is
solved using a two stage heuristic method. The first stage generates sets of good
routes for each aircraft. The second stage solves a set covering formulation using a
greedy heuristic.

Another early contribution is the work of Daskin and Panayotopoulos (1989).
They solve an aircraft routing problem for a hub-and-spoke network with only one
hub while ignoring maintenance. The problem is actually an aircraft routing model
with fleet assignment considerations as the profit from assigning an aircraft to a flight
depends on the individual aircraft. The resulting set packing problem is solved using
a method that obtains an upper bound using Lagrangian relaxation and then finds
feasible solutions via a heuristic. They show that the problem is more difficult to
solve in cases where some flights can not be assigned to aircraft, i.e. the problem is
infeasible.

Clarke et al. (1997) solves an aircraft routing problem that considers through-
values and two different types of maintenance checks. One type has to be carried out every three days, the other every four days. The problem is formulated as a travelling salesman problem with side constraints. The side constraints ensure maintenance feasibility while the sub-tour elimination constraints of the travelling salesman problem function as the big-cycle constraint. The authors solve the problem by Lagrangian relaxing the sub-tour elimination and maintenance feasibility constraints, which are added dynamically when violated.

Gopalan and Talluri (1998b) consider a problem in which no costs are considered and all maintenance occurs overnight when all aircraft are grounded. In the first phase of their algorithm they construct feasible routes for each day. With a fixed set of routes an Eulerian tour problem is solved in the second phase. Constraints require that every aircraft has to return to a maintenance station every three days. The problem can be solved in polynomial time. However, Talluri (1996) shows that the problem is NP-hard when maintenance stations have to be visited every four days.

The string based model proposed by Barnhart et al. (1998a) has received significant attention. A string is a sequence of flights that ends with a maintenance check. It respects the maximum time between two maintenance checks, where only one type of maintenance is considered. Routes are generated by concatenating several strings for each aircraft. The authors propose two models, one a pure aircraft routing problem, the other a combined fleet assignment and aircraft routing problem. The first model is a set partitioning problem with side constraints and is solved using a branch-price-and-cut algorithm. Equal aircraft utilisation is achieved by including a big cycle constraint. Strings are generated using a constrained shortest path algorithm. The proposed algorithm solves instances of up to 190 flights with 6244 connections. The integrated model is discussed in Section 3.8.3.

Ageeva (2000) maximises the number of times aircraft routes meet, i.e. are at the same airport at the same time. This provides opportunities to swap parts of the routes, which may be beneficial if one aircraft is late and another aircraft’s connection has sufficient slack to absorb the delay.

A robust aircraft routing model that minimises the expected propagated delay along aircraft routes is developed by Lan et al. (2006). The mixed integer problem
3.5 Crew Pairing

In this section, crew pairing generation is described in detail. Early publications and publications introducing robustness to the crew pairing problem are reviewed. Literature integrating crew pairing with other stages is reviewed in Section 3.8. A summary of the literature is provided in Section 3.9.

Most major airlines operate more than 1000 flights per day. This involves scheduling thousands of cockpit crews and flight attendants on a daily basis. Several rules restricting crew scheduling are imposed by national aviation authorities, unions and the airline itself. Therefore, crew scheduling is a complex task that has to be carried out carefully especially since crew cost represent the second largest expense of an airline after fuel cost (Barnhart and Cohn, 2004). Operations research has been employed for crew scheduling since the 1960s (Arabeyre et al., 1969). A large number of publications on crew scheduling exists. We review key publications and
recent integrated models.

The objective of the airline crew pairing problem is to find a set of minimal cost pairings such that all flights are covered. The pairings must be feasible with respect to all crewing rules. Pairings are not yet assigned to individual crews, which is done in the subsequent crew rostering phase (see Section 3.6). Since crews are trained for specific aircraft types, the crew pairing problem is usually solved for each fleet type separately. Traditionally, it is solved after the aircraft routing problem as aircraft routes determine the possible set of short connections.

A *duty* is a sequence of flights that are flown in sequence with only short rest periods in-between. Every duty has a brief and debrief time at the beginning and end of the duty, respectively. The duration of a duty is called *duty time* and can be up to 8-11 hours. Several duties with overnight rest periods between them form crew *pairings* (Figure 3.3). A pairing usually has a duration of two to five days and starts and ends at the crew base. The overnight rest periods are called *layovers* and must be of a minimum duration.

![Duty 1 Duty 2 Duty 3](image)

Figure 3.3: A pairing consisting of three duties with layovers in-between.

Duties and pairings have to respect several rules and regulations. Aviation authorities impose regulations to increase safety, while labour negotiations with unions have led to rules that improve working conditions. An airline may further specify or tighten rules to increase robustness. For example, an airline may require rest periods with durations slightly more than legally required, which allows delayed crews to still have rest periods that are within their legal requirements.

Examples of rules governing crew pairings are the maximum number of duties in a pairing, minimum and maximum amount of rest between duties, the maximum time the crew is away from its home base, and the so called 8-in-24 rule. The latter requires that if a crew is flying more than eight hours in any 24 hour interval, it must have a compensatory extended rest period afterward. The 24 hour interval does not necessarily coincide with one day but may cover parts of two days. Examples of rules that have to be respected by individual duties are limits on the total flying
Cost associated with crew pairings depends on several factors such as flying time in a duty, duration of a duty, a minimum guaranteed pay per duty, the flying time in the pairing, total duration of all duties in the pairing, and the time away from base. Additional cost such as transport, meal and hotel cost are incurred if a layover takes place at a city that is not the crew base of the assigned crew.

Cockpit crews are trained for specific aircraft types while cabin crews are more flexible as they are often cross-trained. The number of cockpit crew members is fixed depending on the aircraft type, while the size of cabin crews depends on the expected number of passengers on a flight. Airlines therefore may split-up cabin crews for subsequent flights and join them with cabin crews from other flights. Additionally, rules regarding rest periods and working hours are different for the two types. It is therefore customary to solve the crew pairing problem separately for cabin and cockpit crews. The cockpit crew problem is then decomposed by fleet type, while the cabin crew problem is usually not due to the aforementioned cross-training. Cost associated with cockpit crews are significantly higher than cost for cabin crews. Therefore, publications almost exclusively focus on the cockpit crew problem. An exception can be found in (Wallace, 2001).

Crew scheduling is a challenging task due to the many complex rules and non-linear cost. The problem is aggravated in domestic operations, especially with hub-and-spoke structures, where a duty can be made up of several short flights, resulting in billions of combinations even with as little as 200 daily flights (Klabjan, 2005). Therefore, the problem is often solved as a daily problem, followed by a weekly exception problem (see Section 3.1).

The airline may choose to transport crews as passengers to reposition them. The cost of doing so is high as this counts towards time away from base and often duty time. Furthermore, if the crew is transported on a flight operated by the airline, the seat capacity is effectively reduced, potentially spilling passengers. If operated by a competitor, flight tickets have to be purchased. However, especially in international operations, deadheading may be beneficial if only few flights are scheduled in and out of a location (Barnhart and Shenoi, 1998d).

The crew pairing problem is most often modelled as a set partitioning or set
covering problem. In early publications, the problem is solved using heuristics (Gershkoff, 1989; Anbil et al., 1991; Hoffman and Padberg, 1993; Graves et al., 1993) while more recent publications use a column generation or branch-and-price algorithm (Barnhart et al., 1998c; Klabjan et al., 2001b; Cohn and Barnhart, 2003; Shebalov and Klabjan, 2006; Weide et al., 2010; Saddoune et al., 2013).

The pairings that may be chosen in the set partitioning/set covering problem are generated in pricing problems. Two main types of networks are frequently used in the literature, flight networks and duty period networks (Vance et al., 1997a). In a flight network, each flight is represented by a node, while every legal connection between two flights is represented by an arc. The duty period network consists of a node for each legal duty in the schedule. The duties are usually enumerated beforehand. Arcs connecting nodes represent layovers between duties.

The advantage of the duty period network is that all duty rules can be embedded in the network. However, a large number of nodes and arcs may result. Therefore, a duty period network is usually only used for international schedules where duties rarely consist of more than one flight. In that case, the number of duties does not exceed the number of flights by much. In a flight based network, an algorithm has to check duty and pairing rules simultaneously, which is computationally more expensive.

Enumerating all pairings for a medium sized or major airline is impractical. Instead, only a subset of all pairings is considered at any given time. Early publications employ a so called local search row approach. A subset of flights (i.e. rows) is chosen and pairings are generated for these flights. Then, a new set of flights is chosen and the process repeats until no further improvements can be made or a time limit is exceeded (Gershkoff, 1989; Anbil et al., 1991; Graves et al., 1993).

The local search row approach can get stuck in a local optimum. Column generation, on the other hand, finds the global optimum by implicitly considering all pairings. The subproblem finds all pairings that potentially improve the objective function of the set partitioning/set covering master problem (see Section 2.2).

The most common method to solve the subproblem is a resource constrained shortest path problem (see Section 2.5). Publications using a resource constrained shortest path problem can be found in (Desaulniers et al., 1997a; Vance et al., 1997a;
3.5. CREW PAIRING

Cordeau et al., 2001; Gabteni and Grönkvist, 2009; Weide, 2009; Saddoune et al., 2013; Dück et al., 2011). Other subproblem solvers can be found in (Klabjan et al., 2001b; Galia and Hjorrig, 2003).

Detailed descriptions of the crew pairing problem can be found in (Barnhart et al., 2003; Gopalakrishnan and Johnson, 2005). The remainder of this section reviews key publications for the crew pairing problem.

An approach not using column generation is presented in (Hoffman and Padberg, 1993). A limited number of pairings are generated a priori. A set partitioning problem with side constraints is solved using a branch-and-cut method.

Vance et al. (1997b) decompose the daily crew pairing problem into two steps. In the first step, beneficial duty periods are selected to cover all flights in the schedule. In a second step, pairings are generated from these duty periods. Applying Dantzig-Wolfe decomposition results in a column generation formulation with two pricing problems. One pricing problem generates pairings, the other generates duty sets while ensuring that every flight is covered. The formulation yields a tighter LP-bound than the traditional set partitioning formulation but is harder to solve. The authors solve instances of up to 174 flights.

A multi-commodity network flow model for the crew pairing problem with non-linear cost and nonlinear constraints is presented in (Desaulniers et al., 1997a). Applying Dantzig-Wolfe decomposition results in a linear set partitioning master problem and several pricing problems that are solved with a resource constrained shortest path algorithm. Instances of up to 1157 flights and 63 airports are solved to within an acceptable gap in less than four hours.

Andersson et al. (1998) describe the pairing generator at Carmen Systems. The set covering master problem is solved using the Lagrangian based algorithm presented in (Wedelin, 1995).

Most crew pairing problems use follow-on branching to obtain integer solutions. An alternative branching rule is presented in (Klabjan et al., 2001b). The so called time-line branching is based on the connection time between flights. In one branch, crews are not permitted to connect from a specific flight $i$ to flights $j$ if the connection time exceeds a time $t$. In the other branch, connections from $i$ to $j$ are forbidden if the connection time is below $t$. The authors show that this is a valid
branching rule if all flights have different departure times, which can be achieved by slightly perturbing the departure times. They further combine time-line branching and follow-on branching with strong branching. Klabjan et al. generate pairings by randomly choosing connections where the probability of choosing a connection increases with shorter connection times.

The same authors present a model for the crew pairing problem that considers schedule regularity (Klabjan et al., 2001a). Schedule regularity is beneficial as it facilitates smoother operations management and often crews prefer flying pairings repetitively. Despite using a heuristic solution method, solution times for a weekly problem were up to 47 hours.

A model that generates robust solutions is presented in (Ehrgott and Ryan, 2002). The authors develop a bi-criteria approach that considers crew cost and penalises connections with buffer times less than the expected delays. Connections on which crew stay with an aircraft are not penalised.

Schaefer et al. (2005) increase robustness by solving a deterministic model that uses expected operational cost instead of planned cost. A simulation tool known as SimAir is used to compute expected cost based on estimated delay propagation. Delay propagation can only occur within the pairing; no interdependencies between pairings are considered.

An idea similar to aircraft swapping in (Ageeva, 2000) is proposed for crew pairing in (Shebalov and Klabjan, 2006). They maximise the number of swapping opportunities, called move-up crews. However, the authors show that increased planned cost when having many move-up crews quickly outweighs operational gains. A combination of Lagrangian relaxation and column generation is used to solve their model.

Yen and Birge (2006) use a two-stage stochastic program to generate robust crew schedules. The first stage corresponds to the basic deterministic crew pairing problem, while the second calculates the propagation of delays due to interdependencies of crew pairings and aircraft routes. The solution approach is computationally expensive. Only small instances with 79 flights are solved.

The idea of schedule regularity is further investigated in (Saddoune et al., 2013). The authors show that using a rolling time horizon approach is superior to sequen-
3.5. CREW PAIRING

tially solving a daily, weekly-exception, and finally monthly problem as is customary in many airlines. They further show that more regular solutions can be obtained by allowing a pairing to contain the same flight more than once, i.e. repeat the same flight on a different day. A column generation algorithm is used to solve the model.

Dück et al. (2011) develop a heuristic branch-price-and-cut algorithm for the standard crew pairing problem. The authors use subset-row inequalities (Jepsen et al., 2008) to strengthen the master problem and introduce label categorising to generate more diverse columns. Instead of storing one list of non-dominated labels at each node, their label categorising algorithm stores and extends the least cost label for each incoming arc at any node. The authors use several column generation acceleration strategies and, in a last step, use a standard mixed integer programming solver after column generation termination criteria were met. Twenty instances of up to 539 flights are solved with average solution times of less than 2 hours.

Ionescu and Kliewer (2011) propose a stochastic optimisation model that creates swapping opportunities for crews. Unlike the work of Shebalov and Klabjan (2006), which simply maximises the number of swapping opportunities, their approach identifies connections that are likely to propagate delays and places swapping opportunities at those connections. The solution algorithm used is the same as in (Dück et al., 2011).

Muter et al. (2013) solve a robust crew pairing problem in which robustness is measured by the solutions ability to handle extra flights that must be added to the flight network after the schedule is finalised. The solution must provide a certain number of recovery options, namely pairings that have enough slack to accommodate the additional flights and swapping opportunities between crews. The authors use column generation to solve the problem. Due to the swapping opportunities of pairings, a row is added to the master problem for each combination of pairings whenever a new pairing is generated. Since not all constraints are known, the dual information from the restricted master problem is incomplete and thus optimality cannot be guaranteed.

Other recent literature on pure crew pairing problems include (Subramanian and Sherali, 2008; Chen et al., 2011; Deng and Lin, 2011). Subramanian and Sherali (2008) use a deflected subgradient scheme to solve the problem, Chen et al. (2011)
consider fairness among crews, and Deng and Lin (2011) propose solving the crew pairing problem with an ant colony optimisation based algorithm.

### 3.6 Crew Rostering

After a set of pairings that covers all flights has been chosen in the crew pairing stage, these pairings need to be assigned to individual crew members. In doing so, additional activities such as stand-by duties, pre-scheduled training, or days off must be considered. The objective is often to find a fair distribution of workload and unfavourable flights and pairings. The crew rostering problem usually generates rosters with a duration of one month.

As is the case for the crew pairing problem, aviation authorities, unions and the airlines themselves impose several rules that must be respected when concatenating pairings to form monthly rosters. For example the maximum monthly flying time, maximum yearly flying time, maximum duty time in a month, rest time between activities, and the minimum total number of days off must be considered. Many more rules exist and are described in (Kohl and Karisch, 2004).

Airlines in the United States use a two-phase approach, called *bidline approach*. In the first phase generic feasible monthly rosters, called bidlines, are created. These rosters are released and individual crews bid on these rosters based on their personal preferences. The airline then assigns rosters to crews based on priority which usually depends on seniority. Rostering with bidline approaches can be found in (Campbell et al., 1997; Jarrah and Diamond, 1997; Christou et al., 1999; Boubaker et al., 2010; Saddoune et al., 2012).

An advantage of this approach is that crew members know the exact monthly roster when bidding for the bidline. A drawback is that rosters may be in conflict with pre-scheduled vacation or long term training periods. In this case only partial rosters can be assigned, requiring other crews to work the remainder of the roster.

An approach widely used outside of the United States is *personalised rostering* or *preferential bidding*. Here rosters are generated directly for and assigned to individual crews members. Individuals can express personal preferences for certain attributes of a roster. For example an individual may wish to have certain days of the week off. These personal preferences are considered during roster generation.
3.7. \textit{Tail Number Assignment}

An airline usually either tries to maximise the number of met preferences of their most senior crew members or seeks a fair share of met criteria. A disadvantage of this approach is the increased complexity of the optimisation problem. Models for personalised rostering can be found in (Ryan, 1992; Gamache et al., 1998; Dawid et al., 2001; Sellmann et al., 2002; Kohl and Karisch, 2004).

An in-depth discussion of the crew rostering problem as well as models and algorithms for solving it can be found in (Kohl and Karisch, 2004). The authors also give insight into the rostering procedure at Carmen Systems. Additional publications on crew rostering can be found in (Day and Ryan, 1997; Gamache et al., 1999; Cappanera and Gallo, 2004).

3.7 \textit{Tail Number Assignment}

As previously described, the aircraft routing stage selects a set of generic aircraft routes such that all scheduled flights are covered. Depending on the planning process used by the airline, the aircraft routes may only be feasible with respect to A-checks while other maintenance considerations are neglected.

The goal of the tail assignment problem then is to assign an individual aircraft (tail number) to each route. Often such an assignment is not feasible as the location and history of the tail numbers were not considered in the aircraft routing problem. Therefore, the routes generated during the tail assignment problem follow those of the aircraft routing as much as possible but explicitly consider preassigned maintenance, the anticipated location of the aircraft on the day of operations, and the history with respect to previous maintenance checks.

The tail assignment problem is solved only a few days before the day of operations. At that point in time, location and resource consumption of each aircraft can be predicted with a high degree of certainty. The process of assigning tails varies by airline but usually a problem with a rolling horizon of several days is solved every day. The tail assignment problem is described in detail in the thesis of Grönkvist (2005). Only scant literature on the tail assignment problem exists. Furthermore, the literature is fairly recent, which suggests that the actual tail assignment process in airlines is not optimised. Tools like TPAC\textsuperscript{TM} Operation suite by CTI exist that automate the tail assignment process. These tools consider several rules such
as conserving crew connections and through-values but do not use optimisation to achieve an optimal solution. Instead, heuristic rules such as first-in-first-out are used (Evans, 2013).

Sarac et al. (2006) solves a tail assignment problem on the day of operations. The goal is to route aircraft such that every aircraft that needs maintenance at the end of the day ends its route at a maintenance station. The resulting aircraft schedule may be completely different from the planned schedule. The problem is solved daily and has a one day horizon. The authors constrain the number of maintenance checks that can be carried out at a maintenance station by the availability of technicians and hangar space at the station. This is respected by including resource availability constraints in the set-partitioning master problem. These constraints require tail-dependent follow-on branching once standard follow-on branching cannot resolve any remaining fractionality. While several maintenance types are considered in the master problem, only flying time until the earliest necessary maintenance for each aircraft is considered in the pricing problems. It is assumed that during this maintenance check all other maintenance types are carried out as well. Each aircraft is represented by one pricing problem, and all pricing problems are solved in every iteration of the column generation algorithm. The model is tested on ten randomly generated instances with 175 flights and 32 aircraft.

In a series of publications (Grönkvist, 2006; Gabteni and Grönkvist, 2009) combine column generation and constraint programming to solve the tail assignment problem. A constraint programming preprocessing technique is shown to effectively remove connections which cannot be part of an optimal solution. Furthermore, the algorithm uses a constraint programming model to evaluate heuristic variable and connection fixing decisions before applying them to the set partitioning master problem. After an initial feasible solution is found, an improvement heuristic selects a small time period and re-optimises over the set of flights during that period. Routes are generated for each individual tail via resource constrained shortest path problems that are solved with a label setting algorithm. The problem is dated as tail specific preassigned activities are considered. The authors show the flexibility of their approach by solving three problems with different objectives and constraints. The instances have a duration of one month and include up to 5000 flights and 33
3.7. TAIL NUMBER ASSIGNMENT

Borndörfer et al. (2010) propose a stochastic optimisation model that generates individual aircraft routes while minimising the total probability of delay propagation along the routes. They explicitly model the probability with respect to the length of a delay during both ground time and flying time. The model is solved using column generation at the root node followed by a rounding heuristic to obtain an integer solution. The authors show that their model produces more robust solutions than models with more traditional objectives such as maximising swapping opportunities, maximising opportunities to cancel short cycles, distributing ground time evenly, etc.

Froyland et al. (2012) apply the recoverable robustness technique (Liebchen et al., 2009) to the tail assignment problem. Their stochastic model simultaneously solves the planning phase and the recovery operations of tail assignment. The goal is to develop a daily tail assignment that simultaneously minimises the cost of recovery and the deviation from the planned solution of the aircraft routing. Maintenance requirements are not considered. Benders decomposition is applied such that the master problem handles the planning process while the subproblems model all recovery operations for several predefined disruption scenarios. The subproblems can delay or cancel flights or perform aircraft swaps to recover the master problem solution from the subproblem’s disruptions scenario. Information about the resulting recovery cost are relegated to the master problem to create a better solution. Both the Benders master and subproblem are solved with column generation. Only small instances with 53 flights and ten aircraft are solved.

A completely different approach from a real world point of view is presented in Lapp and Wikenhauser (2012). The authors consider that individual aircraft have different fuel consumption even though they are of the same fleet type. They develop two models that assign routes to tails while minimising overall fuel consumption. The first model creates routes from scratch but includes a constraint that controls the number of deviations from the original solution to the aircraft routing problem. The second model does not create new routes instead simply assigns complete routes to individual aircraft based on their fuel consumption. In both models maintenance is not considered. The authors show final results but do not provide detailed computational experiments nor instance sizes.
3.8 Integrated Airline Scheduling

Airline scheduling is traditionally carried out in a sequential way. The scheduling problems described in the previous sections are solved independently, and usually with few feedback loops. Because they are solved independently, the solution of an early stage may restrict the solution space of a subsequent stage such that the optimal solution for that stage cannot be obtained. In a worst case scenario, the subsequent stage may even be infeasible due to choices in early stages.

Apart from restricting choices in subsequent stages, interdependencies exist that can only be exploited when considering decisions for the stages simultaneously. In Section 3.1, we already discussed the issues of short connections. This section discusses further issues with the sequential scheduling approach while following subsections review publications that partially or fully integrate several stages to alleviate these issues. Publications integrating two stages are discussed in Sections 3.8.1 through 3.8.6, while publications integrating three stages are reviewed in Sections 3.8.7 through 3.8.9.

Schedule design determines the frequency of flights while fleet assignment determines the aircraft type and thus seat capacity for each flight. When these two decisions are not considered simultaneously, a suboptimal daily capacity may be offered by the airline. As an illustration consider the following small example. The daily demand between two cities is 600 passengers and schedule design determined to offer two flights per day. Two aircraft types are available, one with a capacity of 200 seats, the other with a capacity of 500. The fleet assignment then may choose to assign two smaller or one small and one large aircraft to the two flights. These assignments result in daily capacities of 400 and 700 passengers, respectively. Therefore either 200 passengers get spilled or 100 seats are wasted. Instead, in a simultaneous decision process, the airline would offer three flights per day with a small aircraft assigned to each of them, realising the full 600 passengers while not wasting any seat capacities. Therefore, integrating schedule design and fleet assignment has significant impact on an airline’s revenue.

Fleet assignment itself has a major impact on aircraft routing and crew pairing (Barnhart et al., 1998a). When maintenance requirements and rest periods for crews are not considered in fleet assignment, the two problems may be infeasible.
For example, if several aircraft have to undergo maintenance at the same time and fleet assignment did not consider this, not enough aircraft may be available at that point in time to service all flights.

Lower cost solutions to both the aircraft routing and the crew pairing problem may be available if flight departure times can be altered slightly. Integrated models have been proposed that allow flight re-timing within a \textit{time-window}. Time windows may be discretised to reduce the number of variables. Of course, it has to be ensured that if the departure time of a flight is changed in one problem, it also is changed to the same time in the other.

Crew cost are the second highest variable expense of an airline after fuel cost. Unfortunately, decisions in the crew pairing problem are fairly restricted as it is solved after the schedule design, fleet assignment, and aircraft routing problem and thus inherits decisions and constraints from all these stages. It is therefore beneficial if crew considerations are included in earlier stages.

Since aircraft routing is solved weeks or months in advance of the DOO, neither the location nor the resource depletion of individual aircraft is considered. Therefore, only generic routes are generated. In the tail assignment problem, an assignment of these routes to individual aircraft may not be feasible. This can occur if for example fewer than expected aircraft are at a location or because no aircraft can be assigned to a certain route because all individual aircraft require maintenance earlier than scheduled in the route. As a result, routes may have to be adjusted which can be very costly.

In the 1990s, considerable progress was made for each individual stage. While further achievements are possible, especially for major airlines and their large-scale problems, most researchers post 2000 turned their attention to integrating several stages. The goal is to counter the effects of solving the stages sequentially and to increase robustness. In the early 2000s, most models integrate key elements of one stage with the full core problem of another. More recently, two and even three stages have been fully integrated successfully. These models are discussed in the following sections.
3.8.1 Schedule Design and Fleet Assignment

Early models integrating schedule design and fleet assignment solve the two problems iteratively (Etschmaier and Mathaisel, 1985). First, the demand for a given schedule is determined. Then, after the fleet assignment model is solved, flights for deletion or addition are identified. The procedure repeats by evaluating the resulting demand of the new schedule.

Fully integrated models that freely choose frequency and timing are still not solvable today. Instead, recent models integrate the two problems by allowing the flight departure times to vary within a relatively narrow time window. In the work of Rexing et al. (2000), departure times are allowed to vary within a 20-minute time window. The resulting network is significantly larger than a normal fleet assignment model. The authors use preprocessing such as consolidating nodes, deleting redundant arcs, and identifying “islands” to reduce the network size and develop a direct and an iterative solution algorithm. They show that by re-timing eight percent of all flights and re-fleeting 10-20\% of them, savings of $65,000 per day can be achieved.

Lohatepanont and Barnhart (2004) extend the itinerary-based fleet assignment model of (Barnhart et al., 2002). Their integrated model simultaneously determines frequency, departure times, and fleet assignment of a set of flights. A list of mandatory flights have to be offered, while flights from a set of optional flights may be included to vary service frequency. This set also includes several copies of each specific flight where the copies differ by departure times. By choosing one such flight, the departure time is determined. The authors consider that demand changes when frequency is altered by evaluating the demand of each flight in a solution. Given the new demands, the model is resolved iteratively several times. For the full scale problem of a major airline, they achieve savings of close to one million dollar per day.

Other recent publications are Yan and Tseng (2002); Bélanger et al. (2006); Jiang and Barnhart (2009); Sherali et al. (2010, 2011).

3.8.2 Schedule Design and Aircraft Routing

Burke et al. (2010) apply a multi-meme memetic algorithm to an aircraft routing
problem that allows departure time to vary within a 20-minute time window. The algorithm solves the problem by making incremental changes to departure times and aircraft routes. It generates Pareto optimal solutions with respect to schedule flexibility (probability of delay propagation between flights) and schedule reliability (aircraft swapping opportunities). The model only considers A-checks (i.e. one maintenance check every 60 flight hours). Instances of up to 504 flights per week are solved.

3.8.3 Fleet Assignment and Aircraft Routing

In (Barnhart et al., 1998a), the authors present a model which integrates fleet assignment and aircraft routing. As is the case in their pure aircraft routing model, only one maintenance type is considered and a resource constrained shortest path algorithm generates strings that end at a maintenance station. The combined model solves an aircraft routing problem for each fleet and additionally ensures that each flight is assigned to only one fleet. Through-values and operational cost are considered. Their branch-and-price algorithm solves a seven day problem including 1124 flights, nine fleet types, and a total of 89 aircraft in five hours.

Sriram and Haghani (2003) propose a multi-commodity flow problem with side constraints and solve the model using a heuristic solution method that combines random and depth-first search. The authors allow re-fleeting and thus solve a combined fleet assignment and aircraft routing problem. The model requires fixed daily lines-of-flights as input and combines these to form weekly schedules. Two types of maintenance, A and B-checks, are considered. It is assumed that all maintenance is carried out overnight.

An integrated fleet assignment and aircraft routing model that considers robustness is presented in (Rosenberger et al., 2004). Their model generates solutions that contain many short cycles and low hub connectivity. Short cycles are sequences of a few flights that start and end at the same station. Robustness is increased because cancelling short cycles affects only these few flights. Hub connectivity measures how many hubs an aircraft visits during a route. Low hub connectivity helps to isolate delays to a limited number of hubs.

A multi-commodity network flow formulation that considers fleet dependent
operating cost, repositioning of aircraft, and passenger spill-cost is presented in (Haouari et al., 2009). The authors propose a two-phase heuristic. In a first step, an initial solution is obtained by assigning flights to aircraft in a greedy fashion, while in the second step, an improved solutions is found by reassigning flights. Instead of directly swapping two flights, subpaths are swapped by solving several minimum-cost flow problems. The method generates near optimal solutions with solution times of only few minutes. Instances ranging from one to three weeks with up to 1745 flights and 63 aircraft are solved.

Exact methods for integrated fleet assignment and aircraft routing at TunisAir are presented in (Haouari et al., 2011). Spill cost and planned cost are minimised, but maintenance requirements are not considered as maintenance checks can be carried out at any airport overnight. A Benders decomposition method using strong Benders cuts (Magnanti and Wong, 1981), combinatorial cuts, and maximal clique constraints is used to solve instances with 1050 flights, 34 aircraft, and eight different fleets. The authors also present a branch-and-price algorithm that uses variable branching to obtain integer solutions. Benders decomposition generates feasible solution very quickly but is outperformed by branch-and-price when proving optimality.

3.8.4 Fleet Assignment and Crew Pairing

A partial integration of fleet assignment and crew pairing has been shown to reduce crew costs of a long-haul airline by 7% while fleeting cost are increased by only 1% compared to the sequential approach (Barnhart et al., 1998b). A two stage method is proposed. In the first stage, a fleet assignment model is solved that includes decision variables representing partial pairings. In the subsequent stage, a crew pairing problem is solved for each fleet.

The station purity concept of Smith and Johnson (2006), which originally only refers to fleet purity, is extended to crew base purity by (Gao et al., 2009). The authors simultaneously restrict the number of fleet and crew bases that service an airport, thereby increasing robustness. Crew base purity facilitates easier recovery operations as more crew swapping opportunities are available. Their model only partially integrates fleet assignment and crew pairing by including crew connection constraints in the fleet assignment model. The problem is formulated as a mixed
3.8. INTEGRATED AIRLINE SCHEDULING

integer program and solved using branch-and-bound with an additional variable fixing heuristic. The authors note that the subsequent crew pairing problem can be solved much faster because of the smaller size of the problem due to the limited number of airports that each crew base can service.

3.8.5 Aircraft Routing and Crew Pairing

A model that fully integrates aircraft routing and crew pairing is developed by Cordeau et al. (2001). The aforementioned issue of short connections is considered for the first time in this work. Due to the resulting large number of constraints, the authors use Benders decomposition. Aircraft routing is handled in the master problem and crew pairing in the Benders subproblem. In every Benders iteration, short connections are temporarily fixed by the master problem. The subproblem then generates crew pairings using only the currently allowed set of short connections. Both the Benders master and subproblem are solved by column generation. No costs are associated with aircraft routes, reducing the aircraft routing problem to a feasibility problem. Only one type of maintenance is considered that has to occur at most every four days. Instances covering up to three days with 525 flights, 35 aircraft, and 67 crews were solved in less than five hours.

Cohn and Barnhart (2003) include maintenance routing decisions in their extended crew pairing model. The authors realise that if aircraft routing is a feasibility problem, the only impact of aircraft routing on the integrated problem is the choice of short connections. Under this assumption, variables representing entire aircraft routing solutions can be added to the basic crew pairing model. The authors show that only a fraction of all maintenance solutions need to be considered, namely those representing unique and maximal short connection sets. A branch-and-price algorithm is presented that has two pricing problems, one for generating pairings, another for generating entire aircraft routing solutions.

The model of Cordeau et al. (2001) was developed further by Mercier et al. (2005). The authors introduce the concept of restricted connections (see Section 3.1). They also show that if the aircraft routing is a feasibility problem, solving the crew pairing problem as the Benders master problem and the aircraft routing problem as the subproblem is superior to the reversed order, originally proposed by
CHAPTER 3. AIRLINE SCHEDULING

(Cordeau et al., 2001). Fewer Benders cuts are necessary as only feasibility and no optimality information has to be transferred back to the master problem. Instances of up to 707 flights and 143 aircraft are solved in less than five hours.

Weide et al. (2010) use a procedure in which the aircraft routing and the crew pairing problem are solved separately and iteratively. The algorithm starts with an unconstrained crew pairing problem to calculate a lower bound. That is, no aircraft solution is used as input to the crew pairing problem. Then, in the aircraft routing problem, the algorithm maximises the number of restricted connections that are used in the current crewing solution. Next, the crew pairing problem minimises crew cost and penalties assigned to changing aircraft on a restricted connection. These penalties depend on the duration of the connection and are incremented from iteration to iteration until the robustness measures cannot be improved any further. The method does not aim to solve the problem to optimality but instead create different solutions with different degrees of robustness. A weekly flight schedule is generated which assumes that maintenance is carried out overnight when aircraft are grounded. Several instances of approximately 750 flights and 14 aircraft are solved. Weide (2009) also proposes to model the problem using Dantzig-Wolfe decomposition and Benders decomposition. The latter model is solved using the solution algorithm of (Mercier et al., 2005) while the former is solved with branch-and-price. In terms of solution time, the two algorithms are outperformed by the iterative approach.

An integrated stochastic aircraft routing and crew pairing problem is formulated in (Dück et al., 2012). Since the problem is nonlinear, the authors decompose the model into two separate stochastic linear problems that are connected via the objective function. The goal is to simultaneously minimise crewing cost and the sum of secondary delays. Similar to (Weide et al., 2010), the aircraft routing and the crew pairing problems are solved iteratively. Each problem is solved using the solution algorithm proposed in (Dück et al., 2011). The authors compare their indicators of robustness, i.e. sum of secondary delays, with the deterministic indicators (aircraft swaps) used in (Weide, 2009) by generating flight schedules for both approaches and evaluating them using a Monte-Carlo simulation. The authors show that the two approaches are comparable in terms of solution quality, while their algorithm is only slightly more time consuming. The advantage of their approach is that
it automatically calibrates itself due to the delay scenarios used in the stochastic models while the algorithm of Weide has to be calibrated first.

Dunbar et al. (2012) also propose to solve the aircraft routing and crew pairing problems iteratively. Both problems minimise the cost associated with expected propagated delays along routes and pairings, respectively. The interaction of aircraft and crew is considered when calculating delays: for example the delays along a route considers the delay due to the aircraft route itself and pre-calculated propagated delays due to crew connections. The aircraft and crew pairing problems are solved with column generation. The algorithm finds integer solutions at the root node.

3.8.6 Crew Pairing and Crew Rostering

Solving the crew scheduling problem with the two phase approach by first solving the crew pairing problem followed by the crew rostering problem may lead to infeasibility. It may not be possible to assign the minimum cost set of pairings to crews because not enough crew members are available at certain days due to pre-assigned training or vacation. As a result pairings may have to be changed which incurs additional cost. It is therefore a natural approach to simultaneously solve the crew pairing and crew rostering problem. The complexity increases significantly as now the rostering rules have to be considered together with duty and pairing rules when sequencing flights.

Zeghal and Minoux (2006) partially integrate the problems by directly assigning duty periods to individual crew members while considering roster building rules, crew member availability, and compatibilities. It is assumed that all duties are generated a priori. The authors propose a heuristic branch-and-bound algorithm for the resulting mixed integer program. Small instances of up to 40 crew members are solved.

The first fully integrated problem is presented in (Souai and Teghem, 2009). The authors use a hybrid genetic algorithm to solve the problem. Saddoune et al. (2012) propose a model that fully integrates crew pairing generation and crew rostering. The model generates duties, pairings and resulting bidlines while considering the number of available crews at each base. The model is solved using column generation with dynamic constraint aggregation. Duty, pairing, and schedule rules are
considered explicitly in the pricing problem. Compared to the sequential two-stage approach, cost savings of 3.37% can be achieved for small instances. However, computational times increase by a factor of 6.8.

In subsequent work, the authors apply several bi-dynamic constraint aggregation (see Section 2.3) techniques to the same problem (Saddoune et al., 2011). Two methods perform well and reduce computational times to 3.0 and 3.8 times that of the sequential approach. Additionally, costs are reduced further to saving of 4.02% and 4.76%, respectively, compared to the sequential approach.

3.8.7 Schedule Design, Fleet Assignment, and Aircraft Routing

Desaulniers et al. (1997b) extend the model of Abara (1989) by introducing time windows. They present two models for the integrated fleet assignment and aircraft routing with time windows. One is a multi-commodity flow model, the other a set partitioning problem. The models are solved with column generation and branch-and-bound. One-day problems are solved with up to 383 flights and 91 aircraft.

Another model that integrates these three stages by using time windows is proposed by Ioachim et al. (1999). Additionally, they require that flights with the same flight number must depart at the same time throughout the week. Instead of solving a daily problem and repeating the solution on every day, they solve a one week problem that uses a multi-commodity flow formulation with side constraints to ensure departure time synchronisation. The model is solved with Dantzig-Wolfe decomposition. No maintenance requirements are considered.

Zeghal et al. (2011) propose a heuristic to solve real world instances for TunisAir. Departure times are allowed to vary within a 30 minute time windows. The model considers renting additional aircraft. Maintenance is not considered because maintenance checks can be carried out at night at any station. The authors propose a two-phase solution method that uses a similar first stage as the algorithm in (Haouari et al., 2009). The solution is then improved by either fixing the fleet assignment and re-optimising departure times and aircraft routes or by fixing departure times and re-optimising fleeting and aircraft routing. Up to nine fleet types with a total of 30 aircraft are considered.
3.8. INTEGRATED AIRLINE SCHEDULING

3.8.8 Schedule Design, Aircraft Routing, and Crew Pairing

Klabjan et al. (2002) semi-integrate aircraft routing and crew pairing. They reverse the traditional order by first solving the crew pairing problem instead of the aircraft routing problem. Plane-count constraints are introduced to the crew pairing problem, ensuring feasibility of the aircraft routing problem under the assumption that maintenance is performed at night when all aircraft are on the ground. To decrease crew cost, flight departure times are allowed to vary within time windows. The problem is solved using column generation at the root node followed by branch-and-bound to obtain an integer solution. Four instances of up to 450 flights are solved in 15 hours.

The model presented in (Mercier et al., 2005) is further enhanced in (Mercier and Soumis, 2007) by using flight re-timing, where flights are allowed to depart five minutes earlier or later. Binary variables are added that represent departure times, while additional constraints ensure that the same departure times are chosen for the assigned crew and aircraft. The model is solved with a Benders decomposition approach. For instances of 500 flights, solution times were up to 22 hours when using flight re-timing and less than one hour for fixed departure times.

3.8.9 Fleet Assignment, Aircraft Routing, and Crew Pairing

Clarke et al. (1996) propose a fleet assignment problem that includes some maintenance and crew constraints. Flying time restrictions are considered and aggregate maintenance constraints require a minimum number of maintenance opportunities. However, despite the maintenance considerations included, a feasible solution to a subsequent aircraft routing problem cannot be guaranteed.

Another partially integrated fleet assignment, aircraft routing, and crew pairing problem is presented in (Rushmeier and Kontogiorgis, 1997). The authors propose a multi-commodity flow network model with “soft” resource constraints. A penalty in the objective function is incurred if target resource consumption is not met. Only aggregate resource consumption such as “number of aircraft of each type overnight at a specific group of maintenance stations” and “maximum number of crew groups arriving and departing at certain stations” are considered. After an LP-relaxation is solved, a fixing heuristic and branch-and-bound are used to obtain integer solutions.
Sandhu and Klabjan integrate fleet assignment and crew pairing while considering some aircraft routing aspects. Plane-count constraints ensure that at most the number of available aircraft is used, while maintenance requirements are ignored. The model respects that crews are trained for specific aircraft types by requiring that if a pairing is assigned to a crew, all flights in the pairing have to be assigned to the correct aircraft type. Two solution methods are proposed, one based on Benders decomposition, the other on a combination of Lagrangian relaxation and branch-and-price. Overall, the first method finds good solutions quickly but is outperformed by the latter if more solution time is available. The largest instance of 942 flights and four fleets was solved in 29 hours using Benders decomposition and 34 hours using Lagrangian relaxation and branch-and-price.

Papadakos (2009) presents a model that fully integrates fleet assignment, aircraft routing, and crew pairing. A Benders decomposition approach similar to that of (Mercier et al., 2005) is used, however, the aircraft routing is solved in the master problem, while crew pairing is handled in the Benders subproblem. Crew are fleet dependent thus requiring a subproblem for each crew. The master problem relates information to the subproblems about which flights are currently assigned to which fleet, and which short and restricted connections are used in the current solution. Both the master and the subproblems are solved with column generation. Aircraft routes only consider A-checks, while several rules are considered in the crew pairing generation. The solution algorithm uses Pareto-optimal Benders cuts (Magnanti and Wong, 1981) and heuristic depth-first branching. After branching on fleet variables, the problem decomposes by fleet type. Follow-on branching is used in the decomposed problems. Despite the heuristic solution method the problems are very hard to solve. For the largest instance, which contains 700 flights, no solution can be found in 36 hours. However, the author estimates that the problem can be solved in 16.5 hours when using parallel computing.
3.9 Past, present, and future developments in airline scheduling

This section summarises the literature on airline scheduling and gives an outlook on future research in the area. The section concludes with a summary of solution methodologies in airline scheduling.

Airlines achieve annual cost savings of millions of dollars by using multi-commodity network-flow based fleet assignment models (Barnhart et al., 2003). These pure fleet assignment models can be solved even for major airlines. In the past decades, research has focused on introducing other aspects of the airline scheduling process to fleet assignment models. Resulting models are more complex and often column generation is used as a solution method.

In the past, small and medium sized aircraft routing problems have been solved as multi-commodity network flow problems or travelling salesman problems. Starting in the late 1990s column generation has been successfully applied to large scale aircraft routing problems that contain hundreds of aircraft and complex maintenance constraints. Column generation has become the standard solution methodology for aircraft routing problems.

The crew pairing problem has received significant attention from researchers for two reasons. First, crewing costs represent the second highest expense of an airline so even slight improvements result in significant savings. Second, numerous rules regulating pairing generation exist, complicating crew schedule generation. As a result, optimisation tools have been applied to crew pairing generation for several decades. Early on, heuristics were developed that generate only a small subset of all pairings. Over the past 20 years, column generation has been applied to crew pairing generation. Current solvers can generate provably optimal or near-optimal solutions even for large problems. Today, most major airlines use optimisation tools for their crew scheduling process (Barnhart and Cohn, 2004).

Researchers have mainly focused on scheduling cockpit crews, while scheduling cabin crews has attracted only little attention. While the problems are very similar some differences exist: cockpit crews stay together during a pairing while cabin crews may be split up and recombined on later flights. Also, cockpit crews are
usually trained for one or only a few fleet types while cabin crews may work different fleet types. Therefore, the cabin crew pairing problem is larger as it cannot be decomposed by fleet type. Additional research is needed to develop algorithms that can solve such a large scale problem.

Tail assignment is a largely manual process in airlines. Only few publications are available, all of which have been published in the last 10 years. Common among them is the use of column generation as a solution methodology, which is not surprising considering the close ties to the aircraft routing problem.

A currently heavily researched area is the integration of two or more stages of the airline scheduling process. Several models have been published over the past 15 years. Initially, only some aspect of another stage were considered in the other core problem, while recently fully integrated models have been developed. Integrated models overcome some of the shortcomings of the traditional sequential airline scheduling process, namely interdependencies of stages and restrictions of the solution space of subsequent stages. Researchers have reported savings in the millions due to such integration.

Ideally, all airline scheduling decisions are made in one single model. However, such a model is computationally intractable as each individual stage is already hard to solve. Moreover, making all decisions at the same time may not be sensible. For example, a tail and crew assignment to a flight that was made at the time when schedule design decisions are made, i.e. months in advance, will likely have to be reviewed as the day of operations approaches.

Another major trend in airline research is robust scheduling. Traditional models generate (near) optimal solutions with respect to planned cost by removing slack in the schedule. However, the schedules are rarely executed without changes as disruptions due to weather, personal delays, unforeseen maintenance, etc. are very common. Often, these changes require additional changes, which results in high operational cost. Airlines and researchers have recognised the need for schedules to perform better under irregular conditions. Over the past ten years, researchers have developed models that generate more stable and recoverable schedules.

An issue with these models is that measuring robustness is not straightforward. Several indicators have been proposed, for example swapping opportunities for crew
3.9. PAST, PRESENT, AND FUTURE DEVELOPMENTS IN AIRLINE SCHEDULING

and aircraft, station purity, or delay propagation along routes and pairings. However, none of them actually measures operational cost. Instead, they provide the ability to absorb or mitigate the results of disruptions. A solution is then said to be good if it provides many such opportunities. Stochastic programming has been introduced to airline scheduling to ensure that schedules are robust under different scenarios.

Undoubtedly, the trends of developing integrated models and robust schedules will continue. Integrated models will focus on including even more stages and on developing faster solution methodologies since as of today only medium sized problems can be solved. In robust scheduling, better and more complex indicators have to be developed. Additionally, better solution algorithms are required.

A natural development is to combine these two trends into robust integrated scheduling. The goal is to generate integrated solutions at lower cost that are even more robust. In recent years, the first such models have been developed, e.g. (Weide et al., 2010), (Dück et al., 2012), and (Dunbar et al., 2012).

3.9.1 Overview of Airline Scheduling Optimisation Methodology

Airline scheduling is a very complex task. Each single stage of airline scheduling in itself is a challenging problem. Nevertheless, increasingly complex models for each individual stage and integrated models are being developed, requiring sophisticated and often specifically tailored algorithms.

Several mathematical optimisation tools (see Chapter 2) have been used in these algorithms over the past 50 years. Column generation, often in combination with other tools, has become the standard approach for airline scheduling problems. It is well suited to handle the resulting combinatorial complexity of scheduling several tasks in sequence while respecting all rules governing the sequencing. A set partitioning master problem selects a set of columns such that all tasks are assigned. Columns may represent aircraft routes, crew pairings, crew rosters or even complete solutions to another stage as for example in (Cohn and Barnhart, 2003). The columns are generated in pricing problems that are almost always formulated as resource constrained shortest path problems. The RCSPP is typically solved using a label setting algorithm (see Section 2.5).

Pure column generation algorithms have been developed in (Hoffman and Pad-
berg, 1993; Desaulniers et al., 1997b; Vance et al., 1997a; Klabjan et al., 2002; Gabteni and Grönkvist, 2009; Borndörfer et al., 2010; Saddoune et al., 2013, 2011; Dunbar et al., 2012; Saddoune et al., 2012) to name just a few. Column generation only solves linear programs, which usually do not result in integer solutions. Therefore most of these models use branch-and-bound after the root node or a heuristic fixing procedure to deal with fractional solutions.

In branch-and-price, column generation is used to solve an LP-relaxation at every node. Follow-on-branching as introduced by (Ryan and Foster, 1981; Vance et al., 1997a) or variants of this branching rule have become the standard approach to eliminate fractional solutions in branch-and-price. The algorithms presented in (Barnhart et al., 1998a,c; Klabjan et al., 2001b; Cohn and Barnhart, 2003) are examples of using branch-and-price algorithms.

Algorithms using Branch-and-price and especially the heuristic fixing procedures usually employ a depth-first search strategy that selects the “follow-on-branch” of the branch-and-bound tree to obtain integer solutions quickly (Section 6.6). In many publications the algorithm terminates when the “follow-on-branch” has been evaluated, i.e. no backtracking is allowed.

Lagrangian relaxation has received only little attention as a sole solution method (Daskin and Panayotopoulos, 1989; Andersson et al., 1998). It also has been combined with other solution techniques (Shebalov and Klabjan, 2006; Sandhu and Klabjan).

A series of papers (Cordeau et al., 2001; Mercier et al., 2005; Sandhu and Klabjan; Mercier and Soumis, 2007; Papadakos, 2009; Sherali et al., 2010; Froyland et al., 2012) have successfully developed multi-phase algorithms based on Benders decomposition. This approach works well for integrated models as the entire problem decomposes into the individual models the integrated problem is made up of. These individual models then can be solved as before. This approach is particularly effective when many “linking” constraints exists, e.g. short and restricted connections exist when integrating aircraft routing and crew pairing.

In almost all publications, solutions are not provably optimal. Instead, problems are either solved to a high degree of accuracy or heuristics are used to obtain an integer solution. Often, such a solution is close to an optimal solution.
Chapter 4

Near Day-of-Operations
Integration of Aircraft Routing, Crew Pairing, and Tail Assignment

Aircraft routing and crew pairing decisions are made weeks or even months before the day of operations. Many developments leading up to the start of the planning horizon may render the original solutions infeasible. Furthermore, recovery actions that are necessary on the day of operations have an impact on the availability of aircraft and crews in the remaining planning period. These changes may require significant changes to original plans for subsequent days of the planning period.

We propose to solve an integrated aircraft routing, crew pairing, and tail assignment problem about four days before the day of operations to account for these new developments. We suggest the problem be solved daily with a rolling planning horizon of one week. Routes and pairings will be generated for individual aircraft and crews while considering their expected location and resource consumption.

The following section motivates our approach, while the section thereafter provides a general problem description, including all aircraft and crewing rules that are considered. It furthermore outlines affected business processes and describes how these may be modified to accommodate our approach. A mathematical model and
appropriate solution methods will be discussed in subsequent chapters.

4.1 Motivation

The individual stages of the sequential approach and also integrated models are traditionally solved well in advance of the day of operations (DOO). The exact timing of these problems varies between airlines and different times have been reported by several authors.

Cohn and Lapp (2010) report that the fleet assignment problem as well as the aircraft routing problem is solved months in advance. Wu (2010) notes that aircraft routing starts two months before the DOO and finishes one month later. (Klabjan, 2005) documents that the problem is solved “several weeks or months in advance”, while Dück (2010) reports that it begins up to six months in advance and finishes two weeks before the DOO.

The crew pairing problem is solved months in advance (Cohn and Lapp, 2010), four months to three weeks (Dück, 2010), three to one months (Klabjan, 2005), 12 to eight weeks (Sherali et al., 2005), or two to one month (Wu, 2010) prior to the DOO. Butchers et al. (2001) report that the crew pairing process finishes four weeks in advance.

Klabjan (2005) and Wu (2010) state that crew rostering is done one month before departure of flights, while Dück (2010) reports that it is carried out partially in parallel to crew pairing and starts eight weeks in advance and finishes two weeks prior to the DOO. Kohl and Karisch (2004) report six to two weeks before flight departures. Butchers et al. (2001) state one to two weeks, while Sandhu and Klabjan note “a few weeks before the day of operations”.

For tail assignment, Sandhu and Klabjan state the same time as crew rostering, i.e. a few weeks before the DOO, whereas Klabjan (2005) reports “a few weeks or even days before the day of operations”.

Our industry partner reports times that are much closer to the day of operations. Aircraft routes and crew pairings are generated one to two weeks before the day of operations, immediately followed by the crew rostering problem. The tail assignment occurs only a few days before the day of operations (Evans, 2013). Figure 4.1 shows the timing and sequence of the individual stages according to our industry partner.
4.1. MOTIVATION

Naturally, the crew rostering problem will make decisions about the last day of the roster. Even when assuming the best case scenario where crew rostering is carried out one week prior to the start of the roster, decisions for the last day are made five weeks in advance. The same holds for the aircraft routing and crew pairing problems which have to be solved prior to the crew rostering problem. Even though the problems themselves are daily or weekly problems (see Section 3.1) they are used to create schedules with a duration of at least one month. When the schedule for the last day of the month is operated the assumptions made in the daily/weekly problems may not be valid any longer.

As a result of the long lead times and long planning horizons, decisions in fleet assignment, aircraft routing, crew pairing, and crew rostering have to be made based on inaccurate or unknown data. This data is often based on forecasts or simply estimates or assumptions. For example, demand forecasts directly influence which capacity and thus aircraft type is assigned to each flight. Fleet assignment assumes the availability of a certain number of aircraft but often neglects the fact that aircraft are unavailable for several days due to maintenance checks that take a long time to complete, e.g. C- and D-checks. Aircraft routing selects a set of
routes for each day such that all flights are covered. In doing so, it specifies how many routes and therefore aircraft start the day at each airport. On the day of operations, a different number of aircraft may be available at an airport. Similarly, the crew pairing problem assumes a certain number of crew members to be available at each crew base or airport every day.

Flight schedules can be disrupted due to numerous events. With respect to aircraft routes, four categories can be identified. Flight cancellations or delays when operating a route, aircraft breakdown which requires immediate additional maintenance, events that result in additional maintenance but that do not have to be carried out immediately, and actions and events that result in earlier than anticipated maintenance. The first two types occur on the day of operations and can only be prevented or alleviated by providing sufficient slack and more frequent maintenance checks, respectively. Otherwise, they have to be dealt with immediately as they arise on the day of operations. The third category includes minor events such as a defective toilet or failed entertainment system of an individual seat. These events by themselves do not require immediate service but instead, are tolerated until an opportunity arises to fix all of these issues at once without disrupting the route too much. However, especially in highly efficient schedules, these opportunities may be rare, resulting in additional maintenance. If routes are generated for individual aircraft and close to the day of operations, the current state of the aircraft can be taken into consideration. For an aircraft that recently had several of these minor events, it may be beneficial to schedule an additional maintenance in the near future to fix all of these issues without disrupting other routes and crew pairings.

The category that requires earlier maintenance summarises events and actions that result in changed resource consumption. For instance airport closure or major airport congestion may result in severely changed routes. Another example is using a stand-by aircraft. An aircraft is said to be on stand-by when it is sitting at an airport and does not have any flights scheduled for several hours. The aircraft can then be used to fly a short cycle that returns to the current airport early enough for the aircraft to resume its original route. Both of these examples result in a different number of take-offs and accumulation of flying time. If such events happened in recent history and the aircraft is scheduled to have a say B-check in the near future,
it may be infeasible for the aircraft to operate all scheduled flights until that maintenance check. Planning routes close to the day of operations will then either assign fewer flights so that the scheduled maintenance check can be realised or move the check to an earlier date, if this is permitted.

On the day of operations, crew schedules can be disrupted due to a variety of reasons. Flights can be delayed, crews may not have reached the final destination on the previous day, and crews may call in sick or simply not show up for work. These events may have an influence on following duties and pairings throughout the remaining roster. For instance consider a crew member for which the limit on monthly flying time is binding. Further assume that the last pairing has a duration of five days but contains only little flying time because pairings earlier in the roster use up most of the monthly flying time. If the crew member misses the first pairing due to sickness, it is possible for the crew member to have a five-day pairing with more flying time as its last pairing since the monthly limit is not tight any more.

Additionally, events that develop between the time when the crew rostering was solved and the day of operations can be considered if the crew pairing problem is re-solved close to the day of operations. Such events include a change of work force, a crew member requesting additional days off to care for family members or run errands, or leave of absence of a crew member for several days or even weeks due to serious illness. If pairings are not rescheduled in these cases, the original crew pairing solution may be invalid, requiring significant recovery actions on the day of operations.

Since the traditional aircraft routing problem is solved weeks or months in advance and it is not known which individual aircraft a route is assigned to until later on, maintenance scheduling has to be done conservatively. The resource limits for a maintenance check cannot be exhausted completely because the later assigned aircraft may have accumulated some resources already. For example if aviation authorities mandate an A check every 65 hours, scheduling the first A check along a route after 62 hours severely limits the number of aircraft this route can be assigned to as it cannot be assigned to any aircraft that has already accumulated more than 3 hours since the last A check. If the aircraft routing solution consists of several such routes but only few aircraft with little accumulation are available, a feasible
tail assignment is not possible. In practice, airlines stipulate more stringent limits to facilitate easier tail assignment. Generating routes close to the DOO and for each aircraft individually allows these limits to be utilised more aggressively.

About two weeks before the day of operations, airlines transfer responsibility of aircraft and crew schedules to the operations control centre (OCC), also known as the operations management department. The OCC executes the planned schedules on the DOO but is also entitled to make changes by re-routing aircraft and re-scheduling crews during the days leading up to the DOO (Burke et al., 2010; Dück, 2010; Gabteni and Grönkvist, 2009). However, these changes are to be “minor”, i.e. only few other resources are to be affected. It can be expected that these adjustments lead to sub-optimal solutions and thus, together with disruptions on the day of operations, can explain the significant discrepancy between planned and operational cost (Wu, 2010).

The need to revisit fleeting decisions has been recognised as evident by the increasing number of publications on re-fleeting Berge and Hopperstad (1993); Talluri (1996); Rexing et al. (2000); Sriram and Haghani (2003); Jiang and Barnhart (2009). The demand that fleet assignment is based on was forecast three to six months before the day of operations and is often very inaccurate. As an illustration, Jiang and Barnhart (2009) report that only 50% of the demand for each flight is realised three weeks prior to flight departure. The goal of re-fleeting then is to match the offered seat capacity with the increasingly accurate demand forecasts for each day in the planning horizon. As a result, a flight number may now be assigned to different fleets on different days. However, because the fleet assignment is a daily problem the aircraft routing problem assumed that the same flight number has to be covered by the same fleet whenever the flight is offered. Therefore routes generated in the aircraft routing cannot be repeated on days where the re-fleeting model assigns a different fleet type.

More recent re-fleeting models allow departure times to vary within time windows (Rexing et al., 2000; Jiang and Barnhart, 2009). Changing departure and thus arrival times influences crew connection times, possibly invalidating the crewing solution. In a study at SAS, Warburg et al. (2008) found that when re-optimising 15 days before the start of a day, 21% of all crew pairings on that day were infeasible,
4.1. **MOTIVATION**

almost all due to insufficient connection times. However, this is not limited to cases which allow re-timing. Any aircraft swap may cause crew connections to be broken, requiring re-scheduling of at least two crews, the original crew and the crew that is now assigned to the changed flight.

Sherali et al. (2005) and Jiang and Barnhart (2009) note that the time when re-fleeting is carried out has significant impact on aircraft and crew schedules. If re-fleeting occurs after aircraft routes and crew pairings are generated, more accurate demand forecast is available and revenue may be increased significantly. On the other hand, more disturbances to aircraft routes and crew pairings have to be expected, increasing operational cost. Jiang and Barnhart (2009) solve the re-fleeting three weeks prior to flight departures. Bish et al. (2004) are more conservative and place the re-optimisation point six weeks prior to the day of operations. Re-evaluating fleeting decisions multiple times may be beneficial. However, if routes and pairings are adjusted every time re-fleeting decisions are made, cost associated with these often incremental changes will accumulate, possibly increasing cost past the gains due to the higher revenue. Warburg et al. (2008) analyse the gross profit that can be gained from re-timing and re-fleeting once during the time leading up to the day of operation. The authors find that profit is increased by 3.47% when re-fleeting 24 days prior to the DOO but can be improved to 7.46% if done only seven days in advance. Solving an aircraft routing and crew pairing problem close to the day of operations facilitates more effective re-fleeting. The time point of re-fleeting can be chosen very close to the day of operations as long as it is prior to the aircraft and crew schedule generation. Moreover, solving re-fleeting problems multiple times is possible since routes and pairings are generated last. As a result continuous re-fleeting becomes possible.

In summary, many events leading up to the day of operations and events that occur while executing flight schedules on a given day impact routings and pairings in the near future. Re-solving aircraft routing and crew pairing problems close to the day of operations allows these developments to be considered. It is, however, noteworthy that disrupting events on the day of operations will still occur and need to be dealt with accordingly by for example proving stand-by crews or swapping aircraft.
4.2 Problem Description

Given the issues raised in the previous section, we propose to solve an integrated aircraft routing, crew pairing, and tail assignment problem about four days before the day of operations. The model generates routes for each individual tail number, considering its (estimated) location and current state with respect to previous maintenance. Unlike regular aircraft routing, all maintenance requirements must be respected in the model as only little time is available until execution of plans, which may not be enough to make additional adjustments due to maintenance that was previously ignored (Section 4.2.1). For the same reason, all crewing rules have to be considered so that only feasible pairings are generated (Section 4.2.2).

We solve an integrated model to overcome some of the inherent issues of the sequential approach. Short and restricted connections are considered, thus capturing the interdependency between aircraft routing and crew pairing. Furthermore, generating routes for individual aircraft eliminates the need to assign generic routes to individual tails, which may be infeasible.

At about four days before the day of operations it should be possible to project location and status/resource consumption of crews and aircraft to a high degree of accuracy. On the other hand, sufficient time will be available to evaluate the solution and possibly resolve the problem. Solving the model much more in advance will result in diminished returns as a larger number of deviations from the schedule have to be expected. Of course, the decision when to optimise is ultimately up to the airline’s discretion. Before implementing this tool airline-wide, testing using simulations and running the tool for small fleets may be necessary.

As discussed in the previous section, the motivation of this approach is to use more accurate information in the optimisation problem. It is therefore not advisable to solve a problem with a long planning horizon as the data becomes increasingly uncertain for later days in the horizon. We therefore propose to solve about a seven day problem, which contains a first period in which it is required that one aircraft and one crew must be assigned to all flights. A second period is included to guarantee that crews are able to return to their respective bases. Aircraft, on the other hand, do not need to end routes at specific locations and therefore routes do not need cover flights in the second period. It should be noted that flights in this period do not
need either or both a crew or aircraft assigned. In a seven day problem we chose the first period to cover four days, and the second the remaining three days.

The problem is solved every day with a rolling horizon to account for all new developments and disruptions on the current day that influence routes and pairings four or more days later. Figure 4.2 shows how the horizon is rolled over from one day to another. Solving the problem on day 1 generates a schedule for days 4 through 10. Then, solving on day 2 creates the schedule for days 5 through 11. Of the routes and pairings generated on day 1 only decisions for day 4 are fixed, decisions for all following days can be revisited to account for new information. As a result, any final aircraft and crew scheduling decision is made only four days in advance, compared to months when using the sequential approach. It should be noted that because of the rolling horizons, flights that are in the second period of the planning horizon do eventually get covered by routes as time progresses.

![Figure 4.2: Timing for the integrated problem. The figure shows the point of optimisation, the duration of the planning horizon, and the duration of the respective periods in the planning horizon.](image)

Since the problem is to be solved daily, only limited computation time is available. Fast solution algorithms have to be developed that produce good solutions quickly. The airline has to decide what it considers an acceptable solution time. It may want to resolve the model on the same day in case it deems a solution unacceptable. We believe a solution time of up to four hours to be reasonable. Of course, changes in computer technologies/variation in platform performance makes this a moving target.

The decision to solve a daily, weekly, or dated problem depends on the regularity of the schedule. The instances provided by our industry partner are a small part of the domestic operations of a major Australian airline. In Australia, flight schedules
are fairly irregular as they are changed for major sporting events, holidays, and even music festivals. Solving daily or weekly problems with the many necessary exception problems would incur very high cost. Moreover, it is a natural to solve a dated problem when solving an optimisation problem each day and close to the day of operations while considering the individual state of each aircraft and crew. The routes and pairings need to cover the actual flights that must be operated on each day of the planning period.

To accommodate our paradigm, some adjustment to the business processes of an airline will be necessary. We do not propose to replace existing structures. In fact, as will be shown below, we need the original aircraft routing, crew pairing, and crew rostering problems to be solved as before since they are used as input to our model. Our model re-schedules aircraft and crews simultaneously so the business group in charge of the model has to have permission to make these combined decisions. In traditional airline scheduling, generating aircraft routes and crew pairings is often handled by different business units. Communication between involved parties is not always easy as the groups may have different objectives and deadlines. On the other hand, the operations control centre is in charge of both aircraft routes and crew pairings from two weeks before the day of operations and is therefore a natural choice as the process owner of our model. However, the OCC has to be empowered to make more than just “minor” changes as is current practice.

A major disadvantage of our approach stems from the fact that traditionally the monthly roster for each crew member is released after crew rostering. In our model, pairings get rescheduled close to the day of operations and so the exact rosters are not known after the rostering period. Therefore, we propose the following approach. The monthly rosters are generated based on “temporary” pairings generated in the crew pairing stage. We then keep the days-off for each crew member as they are in the roster but allow rescheduling within the originally proposed working days. In other words, after the rosters are generated, crews get notified when they are off work so they can manage their private lives but do not yet get told on which flights they are going to work. This information is revealed only four days before the day of operations, after the integrated model is solved. This is in contrast to current airline practice and may be hard if not impossible to implement for some
4.2. PROBLEM DESCRIPTION

airlines, especially those operating in a strongly unionised environment. On the other hand, newer airlines, which are often low-cost-carriers, may be able to start out with different business processes or are more flexible and open to re-engineering their operations. We expect that crews will require higher financial compensation for their increased flexibility. However, we believe that the increased planning cost will be offset by lower operational cost and higher revenue resulting from more accurate demand used in more timely re-fleeting.

As stated, the integrated model can generate new/different duties and pairings for each crew for the time the crew is scheduled to work according to the original roster. We define the time during which a crew is originally scheduled to be on a pairing as a work-period. The model can generate a new pairing with a duration of up to the number of days in the work-period. By doing so, the rostered days off are automatically respected. As an illustration, consider Figure 4.3, which shows partial original rosters for two crews. Crew A has a pairing of five days starting on day 10, followed by two days off work. The crew then has a five-day pairing with three days off after that. Crew B starts a four-day pairing on day 10, has three days off, and then another five-day pairing. Because crew B has a four-day pairing starting on day 10, the model can only schedule a pairing of up to four days for this work-period. It should be noted that it is allowed to schedule a say three-day pairing that starts on day 11 and finishes on day 14. In this case, when the new pairing is shorter than the original one, the airline - depending on negotiations with crew unions - may have to compensate the crew for the original four days.

Generally, the model does not have any information about which work-period belongs to which crew. It only knows about the rules applying to each work-period. As a consequence any work-periods to which the same rules apply can be considered identical. In Figure 4.3 crew A and B are based in Melbourne, thus all their pairings must start and end in Melbourne. Therefore the second pairing for both crews starts and ends at the same location and has the same duration. If all rules governing duty and pairing generation are identical as well, the solver is free to generate two new pairings that can be assigned to either of the two crews, thus the work-periods are considered identical.

Generalising this leads to the definition of crew blocks. A crew block represents
Figure 4.3: The figure shows the monthly rosters of two crews A and B, which are based in Melbourne. The crews have a common work-period and days-off.

all crews for which the same rules apply when generating new pairings for their work-periods. These rules are:

- The pairing has to start at the current location, where current location can be the crew base or the projected location if the crew is on a pairing at the beginning of the planning horizon.
- The pairing has to end at the crew base.
- The new pairing has to have a duration smaller or equal to the original pairing.
- All duty and pairing rules are identical.
- All additional constraints due to rostering rules are identical.

The last point is intended to capture the fact that in our paradigm, some crew rostering rules may influence pairing generation. This is best illustrated by an example in which we consider the rostering rule that limits the crew’s flying time per month. Figure 4.4 shows the original monthly roster of a crew that is scheduled to have the same work pattern every week: a pairing with a duration of five days and a flying time of 25 hours is followed by two days off. The total flying time in the month is 100 hours, which for this example we assume to be the monthly limit. Furthermore, let there be a limit of 30 hours of flying time per pairing. As time progresses, our model may have scheduled three five-day pairings with more than 25 hours of flying time each for the first three work-periods. In the example a total of 80 hours was scheduled. The fourth pairing now can still cover five days but is
limited to just 20 hours of flying time. The airline can choose to simply restrict
the flying time in the last pairing or instead enforce a 25 hour limit on all four
pairings so that the last pairing is not too restricted. By enforcing a tighter limit
(20 or 25 instead of 30 hours) the affected work-periods have to be singled out from
otherwise identical work-periods that belong to other crews. As a result additional
crew blocks are necessary, representing only one work-period each (unless the exact
same rules/limits are changed for other crews as well).

Figure 4.4: The figure shows the original monthly roster of a crew, which alternates
a five day pairing with two days off. During the pairings, 25 hours of flying time are
scheduled. The limit of flying time in a pairing is 30. In the new paradigm, as time
progresses, we may have scheduled a flying time different than 25 hours. As a result,
the final pairing cannot exceed 20 hours of flying time as otherwise, a monthly limit
of 100 hours is exceeded.

Usually the rostering rules simply change the limits of existing duty or pairing
rules and thus make them more restrictive (as was shown in the example). However,
many rostering rules can be quite complex, thus may require additional rules that
must be respected when generating pairings. However, our industry partner has not
encountered such a rule. Furthermore, since the goal of this work is primarily to
validate the new paradigm, we do not model additional constraints resulting from
rostering rules. Therefore, in addition to the original crew pairing rules, in our
experiments, we only respect the scheduled days off work and thus durations of
pairings.

If all duty and pairing rules as well as the additional constraints from rostering
rules are respected by the new pairings, a resulting roster will always be feasible. The new pairings for each crew block simply need to be assigned to the crews that are respected by the crew block.

Figure 4.5 shows how our approach fits into the sequential airline scheduling process, the flow of information, and all required input. Schedule design, fleet assignment, aircraft routing, crew pairing, and crew rostering are executed as before. From the crew rosters we use information about when crews are available and when they are off work as input to our model. Re-fleeting, which now can be carried out continuously after the original fleet assignment has terminated, determines which flights are to be covered by a specific fleet. The integrated model is then solved for each fleet separately. The airline may choose to consider the original aircraft routes and crew pairings in the new model by requiring that new routes and pairings follow old ones as much as possible. This is discussed in more detail at the end of this section.

We choose to formulate the integrated model based on a connection network. The main reasons are that it facilitates modelling maintenance checks explicitly
4.2. PROBLEM DESCRIPTION

(see Section 4.2.1) and individual crew rest periods (see Section 4.2.2). As a consequence, we are able to assign penalties or benefits to connections, which have to be in monetary values as the objective is to minimise maintenance and crew cost. Connection networks and assigning cost to connections facilitates modelling several interesting aspects of airline scheduling. We can model through fares (see Section 3.3) by assigning a benefit for the appropriate connection, can forbid connections that are known to cause or propagate delays, force an important sequence of flights by removing all other connections for these flights, and model short and restricted connections. Further, we can penalise aircraft connections of medium length (two to three hours). Airlines avoid such connections as the aircraft can neither be maintained nor be on stand-by, as both of these actions usually require more time. The data provided by our industry partner contained no information about through fares, unfavourable connections, or sequences of flights. We therefore do not include them in our experiments.

We do, however, model short and restricted connections (see Section 3.1). In reality the minimum sit time (MST) depends on the size of the airport as crews have to physically move from one gate to another. However, in our experiments we assume a constant MST. According to the definition of short connections, all connections with a duration smaller than the MST but larger than the minimum turn time (MTT) are considered short connections. Connections with a duration of more than the MST but smaller than a certain time \( T_{RC} \) are labelled restricted. A typical value for \( T_{RC} \) is between 60 and 90 minutes, and we have MTT < MST < \( T_{RC} \). Scheduling a restricted connection for a crew but not for an aircraft, i.e. the crew changes aircraft, incurs a linear penalty that decreases with the duration of the connection. Figure 4.6 shows the cost of a connection as a function of its duration. A crew connection with a duration of up to the MST, i.e. short connection, does not incur any cost but can only be scheduled if the corresponding aircraft connection is scheduled. An aircraft swap on a restricted connection with a duration equal to the MST incurs a penalty of \( \rho \). The penalty reduces linearly to 0 for restricted connections that have a duration of \( T_{RC} \).

Assigning cost to connections is even more valuable in our approach. By setting high benefits to correlating connections, the airline can control how much the new
Figure 4.6: The cost of a crew connection depends on its duration. Connections with a duration between MTT and MST do not incur cost but can only be scheduled for a crew if the same connection is scheduled for an aircraft. Connections with a duration between MST and $T_{RC}$ incur a penalty if they are scheduled for a crew but not for an aircraft. The penalty decreases linearly from $\rho$ to 0.

As was shown in Figure 4.2, the decision for day 4 in the solution of day 1 will be implemented while decisions for day 5 through day 10 can be revised the next day. Without any incentive, the model will schedule new routes and pairings with no regard to schedules generated on the previous day (Brown et al., 1997). Completely overhauling routes and pairings every day may result in decreased confidence in our paradigm. It may therefore be advisable to assign a benefit to connections that were chosen in the solution of the previous day. These benefits should be higher for earlier days as more certain information was available for these day. For instance when resolving on day 2 the solution of day 1 is considered. Of that solution the connections chosen for day 5 should receive a higher reward than connections on day 7 or 8. Day 5 is less further in the future, which means other business units or
individuals can plan with a higher degree of certainty.

Another business process that will have to be adjusted is the crew rostering process. In a bidline approach, crew members usually bid for entire rosters. The major advantage is that the rosters are known a priori, which of course is not the case in our approach. Instead, the crew member would bid on fixed work patterns, i.e. days on and off. As outlined above, the model generates one pairing for each crew represented by a crew block. In the large instances solved in this thesis, some crew blocks represent up to 18 crews. The pairings are not assigned to the individual crews by the model but instead by the operator. It is possible to carry out an additional bidline process at this point. Here, each crew member that is represented by the crew block can bid for a pairing generated for the crew block. However, it should be noted that the pairing may change again, which means that crews probably should only bid on the duties that are scheduled for four days later as these are not going to be re-scheduled.

In preferential rostering, personal preferences are considered when generating rosters. These include days off and preferences for specific flights but also more general requests such as not starting work days before a specific time of the day. Days off are automatically considered since they are kept fixed. Some other preferences can be considered in our model, however these preferences are on the crew level not on the individual crew member level as we generate pairings for crews not for crew members. For example if the crew prefers a specific flight, the flight can be removed from the set of flights for all other crew blocks. Then, exactly one pairing containing this flight will be generated for the crew block representing this crew. Naturally this pairing gets assigned to the crew. However, this does not work if two non-consecutive flights are desired. The solver may generate two pairings containing one flight each. This can be modelled correctly if the crew is singled out and represented by its own crew block, which then is the only crew block containing the two flights. Theoretically many preferences that are requested by the entire crew can be considered. However, this often changes the set of connections and flights that are considered for this crew, resulting in a separate crew block. Considering too many preferences could result in an explosion of the number of crew blocks.

Airlines which use a rostering approach that maximises the number of matched
preferences of its most senior crews, may wish to slightly modify our approach. In these airlines more junior crews can usually only choose from “bad” pairings, i.e. pairings with unfavourable work conditions. Therefore these crews may be more open to changing business processes. The airline can choose to keep the pairings and rosters of its most senior members fixed - given they are feasible - and only re-optimise over the resulting set of flights and crews.

As was pointed out in the discussion in Section 3.1, hub-and-spoke networks alleviate the effects of the sequential approach by providing many crew and aircraft swapping opportunities at hubs. In point-to-point networks, fewer such opportunities exist. Consequently, airlines with a point-to-point network benefit most from more accurate aircraft routing and crew pairing close to the day of operations as the likelihood of having to use such swapping opportunities is reduced. We therefore believe our approach is very beneficial especially for low-cost-carriers, who often operate a point-to-point network and are more open to re-engineering their business processes. Of course, the airlines also would benefit from more accurate scheduling and more timely re-fleeting.

Several airline scheduling models that increase robustness have been proposed. The goal of increasing robustness is to generate schedules that require fewer and/or less costly recovery actions in case of irregularities. By considering robustness measures, a schedule is generated that has higher planned cost but is generally considered to be operable at lower operational cost compared to a non-robust schedule. In that regard, our paradigm can be expected to generate a schedule with low planned but also low operational cost. Some of the irregularities are already considered when re-generating routes and pairings, thus fewer recovery actions are necessary. As pointed out in the previous section, these irregularities are events leading up to the day of operations and thus have an impact on the near future as for example additional resource consumption. Disruptions on the day of operations will still occur. We account for this by providing some robustness increasing measures, namely modelling restricted connections and providing maintenance opportunities throughout every route (see Section 4.2.1).

The following sections describe maintenance requirements and crew rules that have to be considered in near day of operations optimisation. Mathematical mod-
4.2. PROBLEM DESCRIPTION

elling of these rules is discussed in the following two chapters.

4.2.1 Aircraft Rules

The use of the traditional maintenance classification of A, B, C, and D checks has over recent years declined. Generally speaking, to guarantee airworthiness of an aircraft, many maintenance tasks have to be carried out within regular intervals. The majority of these tasks have to be carried out with a low frequency. Traditionally, aircraft manufacturers bundled these low frequency tasks as C and D checks. The large number of tasks explains the long duration of such a check. More frequent tasks were bundled as A and B checks. The specifications (tasks and intervals) for A, B, C, and D checks were then released in what is known as Maintenance Planning Data.

Advancements in aircraft maintenance analysis have resulted in more task-driven maintenance specifications, especially for new generation aircraft. Here aircraft manufacturers abstain from bundling tasks to C and D checks and instead, let the airline decide which tasks to package together (Muchiri, 2002). Bundling tasks such that the resulting check can be performed overnight when the aircraft is not in use, is beneficial because the number of calendar days an aircraft is not available is reduced. This maintenance approach is known as progressive maintenance. In addition to these short and frequent progressive checks, additional maintenance checks exist that have a duration of only a few hours and must be carried out about every seven to 14 weeks. Maintenance types of short duration and high or medium frequency are generally known as line maintenance (Joint Aviation Authorities, 2001). They usually do not require hangar facilities. The definition of line maintenance is not exact. Sometimes additional unscheduled maintenance is included in this category. However, we distinguish this maintenance. Therefore, we refer to all maintenance that can be scheduled beforehand, has a short duration, and a high or medium frequency as scheduled line maintenance (SLIM).

Only scheduling A checks, as is done in traditional aircraft routing, means that routes have to be changed often to allow for progressive checks. Our integrated model is solved close to the DOO and we consider each tail individually, including its resource consumption. We are therefore able to consider and schedule all
required SLIM types during the planning horizon. The solver will schedule all of these maintenance checks whenever necessary, i.e. without violating any limits. The checks can only be carried out at specific maintenance stations that have appropriate equipment and technicians. We do not restrict maintenance checks to be scheduled only overnight. However, we expect the solver to schedule few maintenance checks during the day as aircraft will be required to be operated during the day.

Not all C and D checks can be carried out progressively. For some tasks an aircraft has to be taken apart or undergo a major overhaul. Due to the extent of the work involved, these checks are categorised as heavy maintenance (HEAM). A heavy maintenance requires a hangar and spare parts. Hangar availability is limited and so these maintenance checks are usually scheduled well in advance. We consider HEAM checks as input and model them by ensuring that the correct tail arrives at the correct maintenance station before the start of its scheduled check. Additionally, we ensure that the route prior to the check does not violate any limits of this check. Since these maintenance types take several days or weeks to complete, we assume that an aircraft cannot be used again in the planning horizon once a heavy maintenance check commences.

According to the data provided by our industry partner, aircraft require additional unscheduled maintenance frequently. The duration of this maintenance depends on the particular fault but is normally much less than the standard overnight non-flying period. While some faults need to be repaired immediately, other faults can be rectified at a later date. To increase robustness, we schedule maintenance opportunities (MOPP) along a route to perform this additional maintenance if need be. We require that on average every $\phi$ days, each aircraft remains at a maintenance station for $\tau$ hours. It should be noted that the aircraft does not have to be maintained during this time if it is not necessary. In other words, we introduce slack to the schedule at appropriate times and locations.

In traditional aircraft routing, the cost of maintenance is usually not considered as it is assumed to be a fixed cost. In practice, airlines have to balance between carrying out maintenance as infrequently as possible and remaining feasible, i.e. avoiding unscheduled maintenance checks and changing routes. As discussed above, airlines usually set more stringent limits and then carry out maintenance checks as
late as possible while not violating these more stringent limits. Not associating cost with a maintenance check in a rolling horizon may lead to more frequent maintenance than necessary. Consider the following example: an aircraft requires a maintenance check every three days. Let the planning horizon be three days as well. If there is no cost associated with scheduling a check, the solver may schedule it to be on the first day (day 1). When rolling forward to the next day, the same aircraft will require another check within the planning horizon of three days. The solver again may schedule the check on the first day of the planning horizon (which is day 2). This may continue, resulting in a scheduled maintenance check on every day while the check is actually required only every three days. We therefore associate higher cost with scheduling a specific maintenance check early in the planning horizon compared to later on (see Section 5.2.1). Depending on the skewness, the solver will schedule maintenance checks as late as possible.

We do not require routes to end at a maintenance station. One of the reasons this is done in traditional aircraft routing is that by concatenating such routes a daily or weekly solution can be rolled out to cover an entire month. Since we use a rolling horizon and schedule maintenance checks whenever necessary, this condition is not necessary.

Some maintenance types can be carried out simultaneously, e.g. one involving work on the outside of an aircraft, while the other is entirely on the interior. Furthermore, some maintenance types may be carried out in sequence if sufficient time is available between two flights. This is modeled explicitly by including appropriate activity arcs in the flight network (see Section 5.2).

In summary we consider three maintenance categories:

- **Heavy maintenance (HEAM):** requires a hangar and is fixed in time and location. The correct tail has to be at the correct maintenance station at the right time. No cost is associated with this maintenance category as maintenance checks have been scheduled previously.

- **Scheduled line maintenance (SLIM):** all maintenance with high or medium frequency and a duration of several hours. The solver schedules these maintenance checks whenever necessary. Skewed costs are used to avoid scheduling maintenance checks too frequently.
• Maintenance opportunities (MOPP): on average every $\phi$ days, each aircraft has to remain at a maintenance station for $\tau$ hours. Scheduling this maintenance check does not incur any cost.

Each maintenance type of the first two categories may have up to three indicators, namely time between checks, flying time, and pressure cycles (see Section 3.4 and Section 6.1). Only one maintenance type exists in the MOPP category. It is triggered before a certain time between two MOPPs elapses. The limit and the duration of the MOPP depends on historical data collected for the aircraft type (see Section 6.1).

4.2.2 Crew Rules

Industry regulations regarding crew scheduling are very strict in the planning stage. We consider several crew rules in the integrated model. Some of these rules are mandated by aviation authorities, in Australia the Civil Aviation Safety Authority (CASA, 2012), unions, and our industry partner. As mentioned before, additional rules may result from the new paradigm itself. It should be noted that we do not consider all crew rules. To show proof of concept, we focus on crew rules that are standard in the literature and that arise from the new paradigm. Additionally we model a more complicated set of rules that relates two consecutive duties.

Each crew is based at an airport, called crew base, and has to start and end its pairing here. The pairing can only start after the start time of the crew block the crew is represented by. Similarly, the pairing has to end before the end time of the crew block. This rule guarantees that all previously released days-off are respected. Due to the rolling horizon, some crews may have completed a partial pairing at the beginning of the planning horizon. In this case, they may be at an airport other than their base. The crew then has to simply return to the base before the crew block end time.

Pairings are made up of one or several duties. A duty is restricted by several industry rules that mainly relate to two quantities, duty hours and block hours. Duty hours is the duration of the duty including briefing and de-briefing periods, which every duty starts and ends with, respectively. Let Brief be the duration of the briefing period and DeBrief be the duration of the debriefing period. We limit
4.2. **PROBLEM DESCRIPTION**

the duration of each duty, i.e. the duty hours to MD1 hours.

*Block hours* measures the time spend flying plus the taxi time, which is the time spent at an airport once all aircraft doors are closed. We limit the number of block hours that can be accumulated in a duty to MB1 hours.

When a duty ends, either the pairing has to conclude or a layover is required if the crew is to work a subsequent duty. A layover must have a minimum duration and accommodation as well as food must be provided. Additionally, since layovers are necessary between two duties, they contain a de-briefing period to end the first duty and a briefing period to start the second duty. Some airlines do not allow layovers at a crew’s home base. We chose to allow this. However, it can easily be forbidden by removing layover connections at the crew base.

Rules exist that mandate an extended layover if a crew experiences a heavier workload in two consecutive duties. They stipulate that if the total block hours in two consecutive duties exceed a certain value, MB2, and no long layover exists between the two duties, then the pairing has to end, or a long layover is required, immediately after the two duties. Similarly, if the total duty hours in two consecutive duties exceed a value, MD2, with no long layover between the duties, the pairing has to terminate, or a long layover is necessary, immediately following the two duties. Except for the work of (Smith, 2011), we are not aware of any publication considering these type of rules.

Additional rules limit the number of block hours in any seven day period to 30. We approximate this rule by limiting the number of block hours in a pairing to 30. It should be noted that this does not guarantee that the rule holds. For example, a crew may have a three day pairing which accumulated 20 hours, followed by two days off, and a second pairing with a duration of two days and a total of 12 block hours. This seven day span violates the 30 in seven rule. For the purpose of modeling, we denote the limit of this rule by MTB. Theoretically, the rule can be modeled accurately but this will reduce the number of crews that can be aggregated in a crew block dramatically. We therefore abstain from doing so.

For the sake of completeness, we again state the rule that crews cannot be scheduled on a connection that is shorter than the minimum sit time (MST) unless an aircraft is scheduled for the same connection, i.e. no aircraft change occurs. In
that case, crews can be scheduled on connections with a duration of at least the MTT.

We do not consider crew deadheading in our model but this can be easily incorporated as described in Section 10.3.

The cost of a pairing depends on several factors (see Section 3.5). We model cost using applicable credit, a concept used to describe the amount of pay that a crew member receives per duty. Each flight has an amount of credit associated with it, which is accumulated along a duty and thus along a pairing. The operating crew is credited with this value.

Complicating the credit calculation is that airlines avoid scheduling too many short duties as this may result in an infeasible overall schedule. Therefore, each duty will either incur a cost equal to the accumulated credit value or a minimum value called $\zeta$. As a result, the solver should not schedule many short duties because crew utilisation will be low.

In addition to applicable credit, the cost of a pairing depends on the layover cost $LC$, which accounts for expenses such as accommodation, meals, and transport to and from the airport.

Recall that the planning horizon consists of two periods. In the first period each flight must have an aircraft and a crew assigned to it, while in the second up to one crew may be assigned to a flight. This second period is included to ensure that it is feasible for crews to return to their home base. Since we are not concerned with a detailed scheduling for the second period, we do not assign any cost to a flight or connection in the second pairing.
Chapter 5

A Column Generation Formulation for the Integrated Aircraft Routing, Crew Pairing, and Tail Assignment Problem

The focus of our research is to develop a mathematical model for the integrated aircraft routing, crew pairing, and tail assignment problem that is solvable in reasonable time, thereby showing that the paradigm described in Chapter 4 is viable. In this chapter, we present a mathematical model for the integrated aircraft routing, crew pairing, and tail assignment problem. The model is a set partitioning formulation with side constraints. The solution method is based on column generation, which is well suited to solve this type of mathematical model. The chapter begins with a description of the column generation master problem. A detailed description of aircraft routing and crew pairing pricing problems follows. Solution methods for the master problem and pricing problems are discussed in Chapters 6 and 7, respectively. Nomenclature of all mathematical notation used in this and subsequent chapters can be found on page 369.
5.1 Master Problem

The integrated aircraft routing, crew pairing, and tail assignment problem is formulated as a set partitioning problem with side constraints. The master problem ensures that all necessary flights are covered by exactly one aircraft and one crew. Additional constraints deal with all aspects of integration, namely short and restricted connections. Maintenance regulations and crew rules are handled in the column generation pricing problems associated with the generation of routes and pairings, respectively (see Section 5.2).

As discussed in Section 4.2, the planning period consists of two periods. Flights in the first period need to be covered by exactly one aircraft and one crew. Flights in the second period are included to ensure that crews are able to return to their home base before the end of the planning horizon. In the following, we will refer to the first period as Period 1, and the second as Period 2. Up to one crew may work on a flight in the second period but it is not necessary to assign a crew to a flight. No aircraft is assigned to such a flight. Let the set of flights in Period 1 be $N^1$, while the set of flights in Period 2 is $N^0$. The set of all flights in the planning horizon is $N$.

Let $a \in A$ be the set of all aircraft (tails). A route $r \in R_a$ is a sequence of flights for aircraft $a$ that is feasible with respect to all maintenance requirements of aircraft $a$. Let parameter

$$v_{i}^{ra} = \begin{cases} 1 & \text{if route } r \in R_a, \ a \in A \text{ covers flight } i \in N^1, \\ 0 & \text{otherwise.} \end{cases}$$

We define binary variables

$$x^{ra} = \begin{cases} 1 & \text{if route } r \in R_a \text{ is chosen for aircraft } a \in A, \\ 0 & \text{otherwise.} \end{cases}$$

The integrated problem is solved for each fleet type separately. We assume that all aircraft of the same type have equal fuel consumption. Therefore fuel costs
5.1. MASTER PROBLEM

associated with covering all flights in the planning horizon are constant. The cost of a route \( c^{ra} \) thus solely depends on maintenance requirements (see Sections 4.2.1 and 5.2.1).

The set of crew blocks\(^1\) that are generated from a crew rostering solution is denoted by \( B \), where \( n_b \) is the number of crews in crew block \( b \in B \). Let \( P_b \) be the set of feasible pairings for crew block \( b \). A feasible pairing respects all necessary rest periods between flights, starts and ends at the crew base, and starts after the crew block start time and ends before the crew block end time. We define parameter

\[
v_{i}^{pb} = \begin{cases} 
1 & \text{if flight } i \in N \text{ is covered by pairing } p \in P_b, \ b \in B, \\
0 & \text{otherwise.}
\end{cases}
\]

Additionally we have binary variables

\[
y_{pb}^{pb} = \begin{cases} 
1 & \text{if pairing } p \in P_b \text{ is chosen for crew block } b \in B, \\
0 & \text{otherwise.}
\end{cases}
\]

The cost of a pairing is denoted by \( c^{pb} \) and is discussed in more detail in Sections 4.2.2 and 5.2.2.

As previously defined, a feasible connection \((i,j)\) with a duration \( d_{ij} \) such that \( \text{MTT} \leq d_{ij} < \text{MST} \) is called a short connection. The set of all short connections is denoted by \( C_{Sh} \). Connections with a duration \( \text{MST} \leq d_{ij} < T_{RC} \) are called restricted connections and are represented by \( C_{Re} \). Let parameter

\[
h_{ij}^{ra} = \begin{cases} 
1 & \text{if connection } (i, j) \in C_{Sh} \cup C_{Re} \text{ is part of route } r \in R_a, \ a \in A, \\
0 & \text{otherwise.}
\end{cases}
\]

Similarly, let

\[
h_{ij}^{pb} = \begin{cases} 
1 & \text{if connection } (i, j) \in C_{Sh} \cup C_{Re} \text{ is part of pairing } p \in P_b, \ b \in B, \\
0 & \text{otherwise.}
\end{cases}
\]

\(^1\) A detailed description of crew blocks can be found in Section 4.2
For restricted connections we define binary variable

\[ z_{ij} = \begin{cases} 
1 & \text{if connection } (i, j) \in C_{Re} \text{ is used by a crew block but not by an aircraft,} \\
0 & \text{otherwise.} 
\end{cases} \]

A penalty \( \rho_{ij} \), \( (i, j) \in C_{Re} \) is incurred if an aircraft change occurs on a restricted connection. The penalty depends on the duration of the connection (see Section 4.6). The integrality restrictions of \( z_{ij} \) can be relaxed because the objective function together with constraint (5.7) ensure that the variables assume values of either 1 or 0.

We formulate the integrated aircraft routing, crew pairing, and tail assignment problem (INT) as

\[
\min \sum_{b \in B} \sum_{p \in P_b} c_{pb} y_{pb} + \sum_{a \in A} \sum_{r \in R_a} c^{ra} x^{ra} + \sum_{(i,j) \in C_{Re}} \rho_{ij} z_{ij}
\]

s. t. 

\[
\sum_{a \in A} \sum_{r \in R_a} v_{i}^{ra} x^{ra} = 1, \quad \forall i \in N^1 
\]

\[
\sum_{b \in B} \sum_{p \in P_b} v_{i}^{pb} y_{pb} = 1, \quad \forall i \in N^1
\]

\[
\sum_{b \in B} \sum_{p \in P_b} v_{i}^{pb} y_{pb} \leq 1, \quad \forall i \in N^0
\]

\[
\sum_{r \in R_a} x^{ra} \leq 1, \quad \forall a \in A\quad (5.4)
\]

\[
\sum_{p \in P_b} y_{pb} \leq n_b, \quad \forall b \in B \quad (5.5)
\]

\[
\sum_{b \in B} \sum_{p \in P_b} h_{ij}^{pb} y_{pb} - \sum_{a \in A} \sum_{r \in R_a} h_{ij}^{ra} x^{ra} \leq 0, \quad \forall (i, j) \in C_{Sh} \quad (5.6)
\]

\[
\sum_{b \in B} \sum_{p \in P_b} h_{ij}^{pb} y_{pb} - \sum_{a \in A} \sum_{r \in R_a} h_{ij}^{ra} x^{ra} - z_{ij} \leq 0, \quad \forall (i, j) \in C_{Re} \quad (5.7)
\]

\[
x^{ra} \in \{0, 1\}, \quad \forall r \in R_a, a \in A
\]

\[
y_{pb} \in \{0, 1\}, \quad \forall p \in P_b, b \in B
\]

\[
z_{ij} \geq 0, \quad \forall (i, j) \in C_{Re}
\]

The objective function minimises the cost of all routes and all pairings and incurs a penalty if an aircraft change occurs on a restricted connection. Constraints (5.1)
5.2. Pricing Problems

The purpose of column generation is to not work with the entire set of columns. As a result, at any given time, the restricted master problem may not contain all columns that are required in the optimal solution. It is therefore necessary to evaluate whether additional columns are required or if the current solution to the restricted master problem is optimal. This is achieved in pricing problems which use dual values from the restricted master problem to evaluate the current LP solution. In standard column generation, only one pricing problem exists. In our integrated problem, however, aircraft and crews are different when considering resource consumption, location, and availability. Thus, we have to generate routes and pairings specifically for each aircraft and crew block. We therefore have to solve one pricing problem for each tail \( a \in A \) and crew block \( b \in B \).

Despite these differences, similarities exist between aircraft and crews as they use a similar flight network. As discussed in Section 3.1, flight networks can be modelled as time-line and connection networks. We choose to model pricing problems based on connection networks since we are interested in scheduling specific activities.
between flights such as maintenance checks and rest periods. Due to the common underlying network, we can model all pricing problems in a similar fashion. Crew layovers and aircraft maintenance can be interpreted as replenishments of resources. Thus, we model pricing problems as a variant of the resource constrained shortest path problem with replenishments (RCSPP-R) (see Section 2.6). The crew pairing pricing problem requires several extensions to the RCSPP-R. Non-linear costs are introduced, cost may be a function of resources, and the accumulation of a resource may depend on the accumulation of another resource. For the aircraft routing pricing problem (ARPP), the RCSPP-R must be extended in order to handle multi-arcs. Crew pairing pricing problems (CPPP), as described in this chapter, can be modeled without multi-arcs. However, the superimposed pricing problems for crews that will be introduced in Chapter 9, do require them. Thus, we define notation capable of describing multi-arc networks for both the aircraft routing and crew pairing pricing problems. The following two sections describe how aircraft routing and crew pairing can be modeled as a variant of the RCSPP-R. We discuss how the arc sets are constructed and parameters are calculated. The extensions to the RCSPP-R are discussed in Section 7.1.

The RCSPP-R with multi-arc capabilities requires a directed multi-arc network consisting of a set of nodes, a set of multi-arcs, a set of replenishment arcs, a source node, a sink node, a cost on each arc, weights on each arc, and limits on weight accumulation before use of a replenishment arc. These sets and parameters are defined for each aircraft \( a \in A \) and crew block \( b \). Therefore all sets and parameters carry an index \( a \) or \( b \), however, for ease in exposition, we choose to neglect the index unless necessary.

In a pricing problem every flight \( i \in N \) is represented by exactly one node. Thus, let \( N \) also denote the set of all nodes that represent flights the aircraft or crew is able to operate. The set is ordered topologically with respect to flight departure time. Node \( i < j \), \( i \in N, j \in N \) if \( \tau_i \leq \tau_j \), where \( \tau_i \) is the departure time of flight \( i \). Let \( \hat{N} = N \cup s \cup t \) be the set of all nodes in the pricing problem, where \( s \) is the source node and \( t \) is the sink node of the network.

Several parallel arcs may exist between a pair of nodes \((i, j)\), \( i \in \hat{N}, j \in \hat{N} \). We identify an arc as a triplet \((i, j, g)\), \( i \in \hat{N}, j \in \hat{N}, g \in G_{ij} \), where \( G_{ij} \) is the index
5.2. PRICING PROBLEMS

set of all arcs for node pair \((i, j)\). We define a set of source arcs, \(E^+\), to contain all arcs from \(s\) to \(j \in N\), i.e. \(E^+ = \{(s, j, g)\}, j \in N, g \in G_{sj}\). Similarly, we define the set of sink arcs, \(E^-\), to be all arcs from node \(i \in N\) to \(t\). Additionally, we define the set of arcs between two flight nodes as \(E = \{(i, j, g)\}, i, j \in N, g \in G_{ij}\). Then, the set of all arcs is \(\hat{E} = E^+ \cup E \cup E^-\). Some arcs replenish a resource \(k \in K\), where \(K\) is the set of resources in the pricing problem. Let \(\hat{E}^k \subseteq \hat{E}\) be the set of all replenishment arcs for resource \(k \in K\).

Weights along arcs model the resource consumption associated with connections and flights. Let the weight, or usage, of resource \(k\) along an arc be \(u^k_{ijg}, \forall k \in K, (i, j, g) \in \hat{E}\). Weights are accumulated along a path through the network. The accumulation of resource \(k \in K\) between replenishment arcs must not exceed a limit \(U^k\).

Traversing an arc incurs a cost. Let \(c_{ijg}, \forall (i, j, g) \in \hat{E}\) be the cost of edge \((i, j, g)\). The algorithm presented in Section 7.1 requires that network \(G\) contains no negative cost cycles. While \(c_{ijg}\) can be negative, we do have that the network in our problem is acyclic since the set of nodes is ordered topologically and all arcs \((i, j, g) \in \hat{E}\) satisfy \(i < j\).

The goal of the RCSPP-R is to find a path \(q\) from source node \(s\) to sink node \(t\) through network \(G = (\hat{N}, \hat{E})\) that minimises the total cost while not violating any resource limits at any given time. Resources may be reset (i.e. the accumulated weight is set to zero) by using appropriate replenishment arcs. A path consists of a sequence of connected arcs \((i, j, g) \in \hat{E}\). We say \((i, j, g) \in q\) if path \(q\) contains arc \((i, j, g)\).

To ease exposition, we define additional common notation. Let \(d_i, i \in N\) be the duration of flight \(i\). Let \(\tau_s\) be the time when the aircraft or crew block for this pricing problem becomes available, and \(\tau_t\) be the time when the aircraft or crew block becomes unavailable (recall, we are omitting aircraft/crew block indices in this section). The duration an aircraft or crew block is available is denoted by \(d_{st} = \tau_t - \tau_s\). It should be noted that the set of flight nodes \(N^a\) for aircraft \(a\) only contains flights that depart after \(\tau^a_s\) and arrive before \(\tau^a_t\). Similarly for crew block \(b\).

In the remainder of this work, we use the term connection whenever we refer to
the (real world) sequencing of two flights while the term arc is used when modeling an activity along a connection and/or discussing solution algorithms. To be consistent, we also refer to the period before the first flight and the period after the last flight as a connection. This can be seen as connecting the beginning of the period to the first flight and connecting the last flight to the end of the planning period. An arc \((i, j, g)\) can only exist if a connection \((i, j)\) exists. A connection \((i, j)\) may be represented by several arcs \((i, j, g)\), \(g \in G_{ij}\), while every arc corresponds to exactly one connection.

Let \(\hat{C} = C \cup C^+ \cup C^-\) be the set of all connections that the aircraft or crew can be scheduled on. \(C\) is the set of all legal flight connections, i.e. connections between two flights \(i \in N\) and \(j \in N\). \(C^+\) is the set of connections between the source node \(s\) and flight nodes \(j \in N\), and \(C^-\) is the set of connections between flight nodes \(i \in N\) and sink node \(t\). In analogy to source and sink arcs, we refer to connections in \(C^+\) and \(C^-\) as source connections and sink connections, respectively. Additionally, we define the duration of connection \((i, j)\) to be \(d_{ij}\), \((i, j) \in \hat{C}\).

Operating a flight and using a connection consumes resources. Let the resource consumption, or equivalently usage, of resource \(k \in K\) on flight \(j \in N\) be denoted by \(u^k_j\). Similarly, we define the consumption of resource \(k \in K\) on connection \((i, j) \in \hat{C}\) as \(u^k_{ij}\). At the beginning of the planning horizon, an aircraft or crew may have accumulated resources because the last replenishment of resource \(k\) has been some time before the beginning of the horizon. This accumulation has to be considered when scheduling maintenance checks and rest periods in the current planning horizon. We define the resource consumption of the aircraft or crew at the beginning of the horizon (resource start values) as \(u^k_s\), \(\forall k \in K\). As described in the following two sections, the usage \(u^k_{ijg}\) on an arc depends on the type of arc and is thus a function of \(u^k_j\), \(u^k_{ij}\), and \(u^k_s\).

### 5.2.1 Aircraft Routing Pricing Problem

As was previously described, we need to generate feasible routes for each aircraft that consider the location and resource consumption of the aircraft at the beginning of the planning horizon. Additionally, to prove LP-optimality of the master problem, we need to show that no further routes need to be added for any aircraft. We therefore model each aircraft \(a \in A\) as a pricing problem that generates routes specifically
for this aircraft. The mathematical representation of the aircraft pricing problem is given in this section.

As was described in Section 4.2.1, aircraft only need to cover flights in the first period of the planning horizon. An aircraft is generally assumed to be available throughout the first period unless it undergoes a heavy maintenance check at the beginning of the planning horizon or has to start such a check during Period 1. Therefore, \( \tau_s \) is either equal to the start time of the planning horizon or the end time of the HEAM that is being performed. \( \tau_t \) is equal to the end time of Period 1 or equal to the start time of the prescheduled HEAM.

As described in Section 4.2.1, safety regulations require that an aircraft has to undergo periodic maintenance checks. Let \( M \) denote the set of all maintenance types that apply to the aircraft. Let \( M^H \) be the set of heavy maintenance types, \( M^S \) be the set of SLIM types, and \( M^O \) be the set of maintenance opportunities, although the latter only contains one element. Furthermore we have \( M^H \cap M^S \cap M^O = \emptyset \) and \( M = M^H \cup M^S \cup M^O \).

A check for maintenance type \( m \in M \) has to be carried out before limits on indicators for this maintenance type are exceeded. Possible indicators are actual time (AT), flying time (FT), and pressure cycles (PC). Every maintenance type \( m \) may have between one and three indicators. In Section 3.4 we defined every maintenance type-indicator pair as a resource \( k \in K \).

The set of flight connections \( C \) contains all legal connections \( (i, j), \ i \in N, \ j \in N \) between two flights. A connection is legal if its duration is equal to or larger than the minimum turn time and flight \( i \) arrives at the same airport flight \( j \) departs from.

The aircraft is at a specific location at the beginning of the planning horizon. It can therefore only start a route with a flight that departs at the initial location. Additionally, an aircraft cannot operate a flight that departs before the aircraft becomes available. Therefore, \( C^+ \) contains all connections that connect source node \( s \) to flight node \( j \in N \) where flight \( j \) departs from the aircraft’s location and departs after the aircraft becomes available, i.e. after \( \tau_s \).

An aircraft does not have to return to a specific station at the end of the planning horizon. Therefore every flight node \( i \in N \) is connected to sink node \( t \). However, if the aircraft is scheduled to have a heavy maintenance, it must arrive at the main-
tenance station before the maintenance check is due to commence. This can be guaranteed by only including connections from flights that terminate at the correct maintenance station and that terminate before the check begins, i.e. before \( \tau_t \). Since every path in the network has to end at the sink node, no flight that is in contradiction with the HEAM will be included in a feasible route.

In aircraft pricing problems, a connection may be represented by several arcs, depending on what activity can be carried out during the connection. Not scheduling any activity apart from simply connecting from \( i \in \hat{N} \) to \( j \in \hat{N} \) is represented by a no-maintenance arc. One such arc exists for every connection. Additional arcs exist if some maintenance types can be carried out on the connection. A maintenance check is permissible if the duration of the check does not exceed the duration of the connection and the connection occurs at a maintenance station where this maintenance type can be performed. If several maintenance types are permissible on a connection, they may be carried out simultaneously or in sequence. Sufficient time has to be available and all sequencing constraints between the maintenance checks have to be fulfilled. Each individual permissible maintenance check as well as all allowed combinations of maintenance checks are represented by separate arcs. Let \( M_{ijg} \) be the set of maintenance checks that are carried out on arc \((i,j,g) \in \hat{E}\). The hypothetical example in Table 5.1 illustrates the set of arcs for a given connection.

<table>
<thead>
<tr>
<th>MTN Type</th>
<th>Duration</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>F001</td>
<td>4</td>
<td>not with or after F003</td>
</tr>
<tr>
<td>F002</td>
<td>3</td>
<td>not with or after F003</td>
</tr>
<tr>
<td>F003</td>
<td>5</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 5.1: Example maintenance types with sequencing restrictions.

In this example three maintenance types exist. F001 has a duration of four hours, F002 has a duration of three hours, and F003 has a duration of five hours. If to be carried out on the same connection, F001 and F002 have to be completed before F003. Given a connection time of eight hours, a total of eight possible arcs results. Table 5.2 lists all arcs and their respective meaning. A “||” in the second column means the two checks are done simultaneously, while a comma signifies they
are carried out sequentially.

<table>
<thead>
<tr>
<th>Arc #</th>
<th>Represents</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>No maintenance</td>
</tr>
<tr>
<td>2</td>
<td>F001</td>
</tr>
<tr>
<td>3</td>
<td>F002</td>
</tr>
<tr>
<td>4</td>
<td>F003</td>
</tr>
<tr>
<td>5</td>
<td>F001</td>
</tr>
<tr>
<td>6</td>
<td>F001, F002</td>
</tr>
<tr>
<td>7</td>
<td>F002, F001</td>
</tr>
<tr>
<td>8</td>
<td>F002, F003</td>
</tr>
</tbody>
</table>

Table 5.2: Permissible arcs for the example presented in Table 5.1.

Some combinations are not feasible because they exceed the time available on the connection, e.g. F001 followed by F003. Others are not permissible due to sequencing restrictions. For example F001 cannot be done in parallel with F003. Without this condition, F001 and F003 could be carried out on this connection because for parallel checks, the total duration is equal to the duration of the longer of the checks, here five hours.

It should be noted that arcs 5 through 7 differ only by whether F001 and F002 are carried out in sequence, in parallel, or which one is performed first. Our model does not consider detailed scheduling of operations at maintenance stations. The exact sequence is therefore irrelevant as long as both checks can be completed. However, in our model we have to ensure that we consider the worst case scenario. As an example, consider the case when in our solution F002 is carried out after F001, and the next check of F002 is exactly \( U_k \) hours later than the first F002, where \( U_k \) is the maximum time two checks for F002 can be apart. Then, on the day of operations, F002 and F001 cannot be carried out in parallel. However, detailed scheduling at the maintenance station may require this. To retain flexibility at that point in time, we therefore must assume the worst case in our model, which is that both checks are carried out at the beginning of the connection and are done in parallel. Of course if, as is the case in the example, sequencing rules require that F003 must be carried
out after F002, we then assume that F002 starts at the beginning of the connection, and that F003 starts as soon as F002 is completed.

We therefore only include the arcs that represent the worst case scenario. In the above example we include arc 5 and discard arc 6 and 7. Thus for this connection we have $G_{ij} = \{1, 2, 3, 4, 5, 8\}$. It should be noted that apart from the exact sequencing at a maintenance station, we also do not consider the capacity of the maintenance station.

Recall that a resource $k \in K$ is a combination of a maintenance type and an indicator. In this example let F002-PC and F002-FT be the two resources that relate to maintenance type F002. A check of F002 is necessary if otherwise the limit of either F002-PC or F002-FT is exceeded. Carrying out a check of type F002 will reset both resources. Naturally, any resource not related to F002, for example F003-PC, is not reset by a check of type F002. Table 5.3 continues the example and shows the three maintenance types and their related resources.

<table>
<thead>
<tr>
<th>Resource #</th>
<th>Resource</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>F001-AT</td>
</tr>
<tr>
<td>2</td>
<td>F002-PC</td>
</tr>
<tr>
<td>3</td>
<td>F002-FT</td>
</tr>
<tr>
<td>4</td>
<td>F003-PC</td>
</tr>
</tbody>
</table>

Table 5.3: Resources for the different maintenance types in the example presented in Table 5.1.

For the arc set described above, we get resource replenishments along arcs as displayed in Table 5.4. On arc 1, no maintenance is carried out so no resource is reset. Arc 3 represents a maintenance check for F002 and thus resets resource 2 and 3, while resources 1 and 4 are simply incremented. Arc 5 signifies a maintenance check for type F001 and F003, thus resources 1, 2, and 3 are reset while resource 4 is increased.

We defined $\hat{E}_k$ to be the set of all arcs that replenish resource $k$. In the example above, we have for $k = 1$, that $\hat{E}_k$ contains arcs $(i, j, 2)$ and $(i, j, 5)$.

Maintenance may be carried out before the first flight of a route if the aircraft is
5.2. **PRICING PROBLEMS**

<table>
<thead>
<tr>
<th>Arc #</th>
<th>Maintenance Type</th>
<th>Increases Res. #</th>
<th>Resets Res. #</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>no maintenance</td>
<td>1, 2, 3, 4</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>F001</td>
<td>2, 3, 4</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>F002</td>
<td>1, 4</td>
<td>2, 3</td>
</tr>
<tr>
<td>4</td>
<td>F003</td>
<td>1, 2, 3</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>F001 ∥ F002</td>
<td>4</td>
<td>1, 2, 3</td>
</tr>
<tr>
<td>8</td>
<td>F002, F003</td>
<td>1</td>
<td>2, 3, 4</td>
</tr>
</tbody>
</table>

Table 5.4: Replenishment arcs for the example presented in Table 5.1.

at an appropriate maintenance station at the beginning of the planning horizon and enough time is available before the first flight. Therefore, $E^+$ may contain several arcs for each $(s, j) \in C^+$, one representing the no-maintenance option, the others each representing a permissible maintenance check or a combination of permissible maintenance checks. Similarly, $E^-$ may contain several arcs (no-maintenance, single maintenance, combinations of maintenance) for each $(i, t) \in C^-$ if sufficient time is available after the last flight and before $\tau_t$ and the last flight terminates at an appropriate maintenance station.

Recall that for some aircraft the end node $t$ represents the start of a heavy maintenance. In this case, $E^-$ still contains arcs representing a single or combinations of maintenance types. It may seem unnecessary to schedule a SLIM check immediately prior to a HEAM since an argument can be made that the shorter SLIM can be taken care of during the HEAM. The only time the solver would schedule this is if otherwise a limit of the SLIM is exceeded. Let us consider what this actually means. Because the SLIM is *immediately* before the HEAM, the aircraft is already at the correct maintenance station for the HEAM. The fact that a limit of the SLIM would be exceeded simply tells us that the aircraft cannot fly anywhere before a check for this SLIM is performed. But since the aircraft does not have to fly anywhere to carry out either the SLIM or HEAM, it doesn’t matter whether the SLIM is performed before, during, or after the HEAM.

Operating a flight and using a connection consumes resources. The consumption of resource $k \in K$ on flight $j \in N$ depends on the indicator of the resource pair (e.g.
“F001-FT”). We have

\[
  u^k_j = \begin{cases} 
    1 & \text{if indicator is pressure cycles,} \\
    d_j & \text{if indicator is flying time,} \\
    d_j & \text{if indicator is actual time.}
  \end{cases}
\]

The consumption of resource \( k \in K \) on connection \((i,j) \in C\) depends on the indicator and on whether and when a maintenance is carried out on the connection (recall that several maintenance checks may be carried out sequentially or in parallel on appropriate connections). No resources are consumed if the indicator is pressure cycles or flying time as no flying occurs on a connection. For indicator actual time, the consumption equals the duration of the connection if no maintenance is carried out that relates to \( k \). If maintenance that relates to \( k \) is carried out, the consumption is equal to the time that passes between the end of the check and the departure of flight \( j \). Let this time be denoted by \( d_{ij}^k \).

For arcs between two flight nodes, we define the weights to represent the resource consumption of the connection as well as the resource consumption of the flight following the connection. Therefore the usage of resource \( k \in K \) along arc \((i,j,g) \in E\) is

\[
  u^k_{ijg} = \begin{cases} 
    u^k_j & \text{if indicator is pressure cycles,} \\
    u^k_j & \text{if indicator is flying time,} \\
    d_{ij} + u^k_j & \text{if indicator is actual time and } (i,j,g) \notin \hat{E}^k \\
    d_{ij}^k + u^k_j & \text{if indicator is actual time and } (i,j,g) \in \hat{E}^k.
  \end{cases}
\]

The weights on arcs in \( E^+ \) are used to model the resource consumption that the aircraft experienced before the beginning of the planning horizon. The weight of such a source arc is equal to the resource consumption at the beginning of the planning horizon plus the resource consumption between the start of the planning horizon and the departure of the first flight plus the resource consumption of the first flight of a route. However, on some source arcs, maintenance checks may be carried out. A check resets any previous consumption of resources that apply to this maintenance type. Therefore, if arc \((s,j,g) \in E^+\) is a replenishment arc for weight \( k \), the resource consumption of the aircraft before the beginning of the planning horizon can be disregarded.
5.2. PRICING PROBLEMS

The consumption of resource \( k \) between the start of the planning horizon and the departure of the first flight depends on whether and when a maintenance that relates to \( k \) is carried out on the source connection. If no such maintenance is carried out, the consumption of resource \( k \) is

\[
\begin{align*}
    u^k_{sj} = \begin{cases} 
    0 & \text{if indicator is pressure cycles}, \\
    0 & \text{if indicator is flying time}, \\
    d_{sj} & \text{if indicator is actual time},
    \end{cases}
\end{align*}
\]

for \( k \in K, (s, j) \in C^+ \), where \( d_{sj} \) is the time between \( \tau_s \) and the departure time of flight \( j \). If a maintenance that relates to \( k \) is carried out we have

\[
\begin{align*}
    u^k_{sj} = \begin{cases} 
    0 & \text{if indicator is pressure cycles}, \\
    0 & \text{if indicator is flying time}, \\
    d^k_{sj} & \text{if indicator is actual time},
    \end{cases}
\end{align*}
\]

for \( k \in K, (s, j) \in C^+ \), where \( d^k_{sj} \) is the time between the end of the maintenance and the departure of flight \( j \).

We then have the consumption of resource \( k \in K \) along a source arc \((i, j, g) \in E^+\) to be

\[
\begin{align*}
    u^k_{sji} = \begin{cases} 
    u^k_{sj} + u^k_{ij} & \text{if } (s, j, g) \in \hat{E}^k, \\
    u^k_s + u^k_{sj} + u^k_{ij} & \text{otherwise},
    \end{cases}
\end{align*}
\]

(5.8)

For arcs in set \( E^- \) the terminating node is the sink node, which does not consume resources. Thus we have the consumption of resource \( k \in K \) on sink arc \((i, t, g) \in E^-\)

\[
\begin{align*}
    u^k_{itg} = \begin{cases} 
    0 & \text{if indicator is pressure cycles}, \\
    0 & \text{if indicator is flying time}, \\
    d_{it} & \text{if indicator is actual time and } (i, j, g) \notin \hat{E}^k \\
    d^k_{it} & \text{if indicator is actual time and } (i, j, g) \in \hat{E}^k
    \end{cases}
\end{align*}
\]

where \( d_{it} \) is the time between the arrival of flight \( i \) and \( \tau_t \) and \( d^k_{it} \) is the time between the end of a maintenance that relates to \( k \) and time \( \tau_t \). \( d_{it} \) represents the time the aircraft is idle after the last flight of its route until the end of the planning horizon.
As discussed in Section 4.2.1, we do not model actual maintenance cost but instead, use a skewed cost function to schedule maintenance as late as possible in the planning horizon. A SLIM \( m \in M^S \) that is scheduled at the beginning of the planning horizon, incurs a cost of \( c^m \). The cost decreases linearly to zero if the check is scheduled at the end of the planning horizon. We thus define the cost of maintenance \( m \in M^S \) along a connection \((i,j) \in \hat{C}\) on which \( m \) is performed as

\[ c_{ij}^m = c^m(1 - \frac{\tau_{ij} - \tau_s}{\tau_t - \tau_s}), \quad (5.9) \]

where \( \tau_{ij} \) is the start time of the connection.

A heavy maintenance type is never scheduled by the solver. This is achieved by not including any arcs that represent checks for this maintenance type. We therefore arbitrarily define the cost of \( m \in M^H_a \) along \((i,j) \in \hat{C}\) as \( c_{ij}^m = 0 \). It should be noted however, that if a HEAM was pre-scheduled for the aircraft, we need to keep track of the corresponding resources for this \( m \in M^H \) since we must ensure that none of them is exceeded before the HEAM commences.

The single maintenance type in the MOPP class does not incur any actual cost as scheduling such a “check” simply signifies an opportunity for unplanned maintenance. The goal is to provide such an opportunity on average every \( \mu^k \) days. In other words, this is not a hard limit. We chose to model this by setting the limit for check \( m \in M^O_a \) to be \( 2\mu^k - 1 \). Usually this value is less than the duration of the planning horizon. Furthermore, we incur no cost for such a check, thus we define \( c_{ij}^m = 0, \ m \in M^O, \ (i,j) \in \hat{C} \). Generally, the solver will schedule most maintenance checks overnight when aircraft are not being used. Because of this and the combination of no cost and such a limit will result in scheduling \( m \) on average slightly less than \( \mu^k \) days.

It is possible to model MOPP using a limit that is equal to or slightly higher than \( \mu^k \) and assign a cost function like Equation 5.9. However, this influences the solution more than the approach we selected since route generation is now not only influenced by the need to schedule this maintenance, i.e. introduce slack, but also by how much scheduling this maintenance contributes to the objective function. Depending on the magnitude of the artificial cost, this may result in a solution for which the actual cost is unnecessarily higher.

The total cost of maintenance on an arc depends on which maintenance checks
5.2. PRICING PROBLEMS

the arc represents. We define the total cost of arc \((i, j, g) \in \hat{E}\) to be

\[
c^M_{ijg} = \sum_{m \in M_{ijg}} c^m_{ij}, \quad \forall (i, j, g) \in \hat{E}.
\] (5.10)

In addition to maintenance cost, dual information from the master problem has to be considered. Let \(\gamma, \alpha, \pi, \sigma\) be the dual values associated with constraints 5.1, 5.4, 5.6, and 5.7, respectively. Similar to the resource consumption of a flight, we assign the dual value \(\gamma_j\) of flight \(j \in N\) to all arcs terminating at node \(j\). Also, we assign \(\pi_{ij}, \forall (i, j) \in C_{Sh}\) and \(\sigma_{ij}, \forall (i, j) \in C_{Re}\) to all arcs representing these connections. The dual value \(\alpha_a\) associated with the convexity constraint of aircraft \(a\) is assigned to every arc in \(E^+\). Thus we define the reduced cost of arc \((i, j, g) \in \hat{E}\) as

\[
c_{ijg} = \begin{cases} 
  c^M_{ijg} - \alpha_a - \gamma_j & \text{if } (i, j, g) \in E^+, \\
  c^M_{ijg} - \gamma_j & \text{if } (i, j, g) \in E \text{ and } (i, j) \notin C_{Sh} \cup C_{Re}, \\
  c^M_{ijg} - \gamma_j + \pi_{ij} & \text{if } (i, j, g) \in E \text{ and } (i, j) \in C_{Sh}, \\
  c^M_{ijg} - \gamma_j + \sigma_{ij} & \text{if } (i, j, g) \in E \text{ and } (i, j) \in C_{Re}, \\
  c^M_{ijg} & \text{if } (i, j, g) \in E^-.
\end{cases}
\]

5.2.2 Crew Pairing Pricing Problem

A crew pairing pricing problem needs to be solved for each crew block \(b\) since we have to generate pairings specifically for the work-periods of the crews that are represented by the crew block. Several rules are considered (see Section 4.2.2), either explicitly by using resources or implicitly by restricting connections and flights.

Pairings for crew block \(b\) must not start before the crew block becomes available, \(\tau_s\), and pairings must not end after the crew becomes unavailable, \(\tau_t\). As Section 4.2.2 specifies, every duty in a pairing, including the last one, must finish with a de-briefing period. Due to our implementation (see below), we adjust \(\tau_t\) to reflect this time period. We set \(\tau_t\) to be equal to the actual time the crew block becomes unavailable minus the duration of the de-brief period. The last duty in a pairing then automatically finishes with this de-briefing period. Whenever we refer to the crew block becoming unavailable, we refer to this adjusted value.

Pairings must start at the location the crew is at when the crew becomes available. At the beginning of the planning horizon a crew may be on a pairing in which
case we say the crew is in category *Pairing Continues*. In this case the initial location of the crew may not be the crew base. Alternatively, a crew may be starting a new pairing during the planning horizon. Here the initial location is the crew base and the crew becomes available at or after the start of the planning horizon. We say the crew block is of type *Pairing Starts*. It should be noted that a crew block is either in category *Pairing Continues* or *Pairing Starts*.

The source node represents the initial location of the crew and when it becomes available. Thus, $C^+$ contains all connections $(s, j)$ that connect the source node $s$ to flight node $j \in N$ where flight $j$ departs from the crew’s location at the start of the planning horizon and departs after $\tau_s$.

The set of flight connections $C$ contains all connections $(i, j)$, $i \in N$, $j \in N$ for which the duration is equal to or larger than the minimum turn time and flight $i$ arrives at the same airport flight $j$ departs from. It should be noted that a connection with a duration such that $\text{MTT} \leq d_{ij} < \text{MST}$ is only legal if an aircraft is scheduled on the same connection, i.e., the crew does not change aircraft between flights $i$ and $j$. The alignment of aircraft and crew is handled by the master problem. In the pricing problem, we simply have to provide the opportunities for a crew to have such a short connection.

Unlike aircraft, crew need to return to a specific location at the end of a pairing, which is always the crew base. Therefore, $C^-$ contains only connections from flight $i \in N$ to sink node $t$ if flight $i$ terminates at the crew base and arrives before $\tau_t$.

In crew pairing, the activity (type of rest) on a connection solely depends on the duration of the connection and at what time of day the connection occurs. The activity is unambiguously defined by the connection. Therefore, only one arc is necessary to represent every connection. As a result, set $G_{ij}$ will always only contain one element.

With respect to rest types we distinguish between *duty connections* and *layover connections*. The latter are extended rest periods that are necessary between two duties. Let the set of all layover connections be denoted by $C_L$. This set can be divided further into a set of regular layovers, $C_{RL}$, and a set of long layovers, $C_{LL}$ (see Section 4.2.2), where $C_{RL} \cap C_{LL} = \emptyset$. For a crew block in category *Pairing Continues*, the duration and timing of a source connection may be such that it
fulfills the conditions of a regular layover or long layover. Therefore, we also let $C_L$, and thus $C_{RL}$ and $C_{LL}$, contain all connections in $C^+$ that fulfill the respective layover and long layover requirements.

The cost of a pairing depends on the cost of layovers in the pairing and the applicable credit accumulated during each duty. The cost of a layover can be associated with the connection. Let this cost be $c_{ij}, \forall (i, j) \in C_L$. Connections that are not layovers do not incur any cost, i.e. $c_{ij} = 0, \forall (i, j) \in \hat{C} \setminus C_L$.

The cost of a duty due to applicable credit, depends on which flights the duty consists of. This value cannot be calculated a priori as the duties are generated when the pricing problem is solved. Furthermore, the MinCredit rule described in Section 4.2.2 states that a duty will either incur a cost equal to the accumulated credit or a value $\zeta$, whichever is greater. For these reasons, we cannot simply assign a cost to a flight or connection that reflects the cost associated with applicable credit. Instead, we use a resource to model this cost. This is described below when discussing other resources used in the CPPP.

Similar to the ARPP, we have to consider dual information in the CPPP. Let $\theta, \lambda$, and $\beta$ be the dual values associated with constraints (5.2), (5.3), and (5.5), respectively.

Again the dual value $\theta_j$ of flight $j \in N^1$ is assigned to all arcs terminating at node $j$. Similarly, the dual value $\lambda_j$ of flight $j \in N^0$ is assigned to all arcs terminating at node $j \in N^0$. As was the case in the aircraft routing pricing problems, we assign $\pi_{ij}, \forall (i, j) \in C_{Sh}$ and $\sigma_{ij}, \forall (i, j) \in C_{Re}$ to all arcs representing short and restricted connections, respectively. The dual value $\beta_b$ associated with the convexity constraint of crew block $b$ is assigned to every arc in $E^+$. Thus we define the cost of arc $(i, j, g) \in \hat{E}$ as
\[ c_{ijg} = \begin{cases} 
    c_{ij} - \beta_b - \theta_j & \text{if } (i,j,g) \in E^+, j \in N^1 \\
    c_{ij} - \beta_b - \lambda_j & \text{if } (i,j,g) \in E^+, j \in N^0 \\
    c_{ij} - \theta_j & \text{if } (i,j,g) \in E, j \in N^1, (i,j) \notin C_{Sh} \cup C_{Re}, \\
    c_{ij} - \lambda_j & \text{if } (i,j,g) \in E, j \in N^0, (i,j) \notin C_{Sh} \cup C_{Re}, \\
    c_{ij} - \theta_j - \pi_{ij} & \text{if } (i,j,g) \in E, (i,j) \in C_{Sh}, j \in N^1 \\
    c_{ij} - \lambda_j - \pi_{ij} & \text{if } (i,j,g) \in E, (i,j) \in C_{Sh}, j \in N^0 \\
    c_{ij} - \theta_j - \sigma_{ij} & \text{if } (i,j,g) \in E, (i,j) \in C_{Re}, j \in N^1 \\
    c_{ij} - \lambda_j - \sigma_{ij} & \text{if } (i,j,g) \in E, (i,j) \in C_{Re}, j \in N^0 \\
    0 & \text{if } (i,j,g) \in E_a. 
\end{cases} \]

Several crewing rules are considered in the crew pairing pricing problem by using one or several resources. Table 5.5 lists the rules and all associated resources that we used in our experiments.

<table>
<thead>
<tr>
<th>Rule #</th>
<th>Rule</th>
<th>Resources</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Limits block hours in single duty</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>Limits duty hours in single duty</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>Limits block hours in consecutive duties</td>
<td>1, 3</td>
</tr>
<tr>
<td>4</td>
<td>Limits duty hours in consecutive duties</td>
<td>2, 4</td>
</tr>
<tr>
<td>5</td>
<td>Limits block hours in pairing</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>Accumulation of credit values</td>
<td>6</td>
</tr>
</tbody>
</table>

Table 5.5: Crew rules and respective resources considered in our problem.

A total of six resources is needed to model the six rules. In this section we will not discuss how the rules are implemented. See Section 7.1 for a detailed discussion of this. However, here we will define the weights the resources are based on. Rules 1, 3, and 5 restrict the accumulation of block hours, while Rules 2 and 4 are based on duty hours. It should be noted that in the following whenever we speak of Rule 1 we refer to the first rule in Table 5.5, i.e. the rule limiting the block hours in a single duty. Similarly for other rules.
When keeping track of duty hours we are interested in the duration of the briefing period prior to the first flight of the duty, the time between departure of the first flight in the duty and arrival of the last flight in the duty, and the duration of the de-briefing period after the last flight. Because every layover contains a de-briefing and briefing period, we need to make sure that we do not over-count de-briefing periods in the labelling algorithm. A way to overcome this is to explicitly count the briefing time but to not count the de-briefing time. Instead, the limit for the appropriate rules are adjusted. We set $U^k$ to be the airline-stipulated limit less the de-briefing time. By using a reduced limit, we will always only accumulate a number of duty hours that will still allow us to carry out a de-briefing period. Because a layover always contains a de-briefing period and because the crew block end time was adjusted by the de-briefing time, we are indeed able to carry out the de-briefing.

For all flights $j \in N$, the number of block hours and number of duty hours are equal to the duration of the flight, which was defined as $d_j$. Additionally, every flight $j \in N^1$ has a credit value $c_j$, $\forall j \in N^1$ associated with it that represents the amount of applicable credit a crew accumulates when it operates the flight. Since we do not consider any cost in the second period of the planning horizon, we have $c_j = 0$, $\forall j \in N^0$.

Then, the consumption of resource $k \in K$ on flight $j \in N$ is

$$u^k_j = \begin{cases} 
  d_j & \text{if } k \in \{1, 2, 3, 4, 5\}, \\
  c_j & \text{if } k \in \{6\}.
\end{cases}$$  \hfill (5.11)

On any connection the number of block hours and the credit value is 0 as no flying occurs. The number of duty hours on a flight connection depends on whether the connection is a duty or a layover connection. For the former, the duty hours are equal to the duration of the connection. For the latter, only the briefing time contributes to the duty that starts with the next flight. We have for all connections $(i, j) \in C$ and $k \in K$

$$u^k_{ij} = \begin{cases} 
  0 & \text{if } k \in \{1, 3, 5, 6\}, \\
  d_{ij} & \text{if } k \in \{2, 4\} \text{ and } (i, j) \notin C_L, \\
  \text{Brief} & \text{if } k \in \{2, 4\} \text{ and } (i, j) \in C_L.
\end{cases}$$  \hfill (5.12)
Then the usage of resource $k \in K$ along arc $(i,j,g) \in E$ is

$$u_{ijg}^k = u_{ij}^k + u_{j}^k, \quad \forall (i,j,g) \in E. \quad (5.13)$$

The consumption of duty hours on a source connection depends on the status of a crew at the beginning of the planning horizon. For a crew block in category Pairing Continues, the resource consumption again depends on whether the connection is a duty or a layover connection. Similar to above, a duty connection has a consumption equal to the duration of the connection. For a layover connection, only the briefing period is counted. A crew in category Pairing Starts always starts a new pairing. Therefore, we only have to count the briefing period. Thus, we have for all $k \in K$ and connections $(s,j) \in C^+$

$$u_{sj}^k = \begin{cases} 
0 & \text{if } k \in \{1,3,5,6\}, \\
|d_{sj}| & \text{if } k \in \{2,4\} \text{ and } (s,j) \notin C_L \text{ and in Pairing Continues}, \\
\text{Brief} & \text{if } k \in \{2,4\} \text{ and } (s,j) \notin C_L \text{ and in Pairing Starts, or if} \\
& k \in \{2,4\} \text{ and } (s,j) \in C_L,
\end{cases} \quad (5.14)$$

Since crews in category Pairing Continues are on a pairing at the start of the planning horizon, they have accumulated resources before the start of the planning horizon. This accumulation must be considered in the remainder of the pairing. Crews in category Pairing Starts, on the other hand, do not have any accumulation as the pairing is about to start. However, for superimposed pricing problems (see Chapter 9), we need to model resource start values for all crew blocks. Therefore, we defined the accumulation of resource $k \in K$ before the start of the planning horizon as $u_{s}^k$ for all crew blocks. For crew blocks $b \in B$ in category Pairing Starts, we set $u_{s}^k = 0$, $\forall k \in K$.

Similar to the aircraft routing pricing problem, we assign $u_{s}^k$ to source arcs. A crew block in Pairing Continues may have a replenishment on connection $(s,j) \in C^+$ (see below for replenishment sets). In this case, the accumulation prior to the planning horizon is reset (resources 1 and 2), or dealt with as described in Section 7.1 (resource 3, 4, and 6). In either case, the accumulation does not contribute to the usage of resource $k \in K \setminus \{5\}$ on arc $(i,j,g)$. Resource $k = 5$ is never reset, so the set of replenishment arcs for this resource is empty. The consumption prior
to the planning horizon for this resource is therefore never discarded. Thus, the
resource consumption along a source arc \((i, j, g) \in E^+\) is

\[
u^k_{sjg} = \begin{cases} 
    u^k_{sj} + u^k_j & \text{if } (s, j, g) \in \hat{E}^k, \\
    u^k + u^k_{sj} + u^k_j & \text{otherwise}.
\end{cases}
\] (5.15)

Arcs in set \(E^-\) terminate at the sink node, which does not consume resources. A sink connection does not consume block hours or credit as it represents the end of a pairing. Every pairing has to end with a de-briefing and this time counts toward duty hours. However, as was discussed above, by adjusting \(\tau_l\) and the limit for resources 2 and 4, we do not need to count the duration of the de-briefing period any more. We thus have the consumption of resource \(k \in K\) on sink arc \((i, t, g) \in E^-\) to be

\[
u^k_{itg} = 0, \forall k \in K, (i, t, g) \in E^-.
\]

For crew blocks in category Pairing Starts, it can easily be verified that every duty starts with a briefing and ends with a de-briefing period: every pairing and thus the first duty of the pairing starts with a connection \((i, j) \in C^+_b\) and therefore has a briefing period (Equation 5.14). Every pairing and thus final duty of a pairing ends at the sink node, which reflects a de-briefing period. Between two duties a layover is required, which includes one de-briefing and one briefing period. Hence, it is ensured that every duty starts with a briefing and ends with a de-briefing period.

The only difference for crew blocks in category Pairing Continues is the first connection in the planning horizon, i.e. connections \((s, j) \in C^+.\) If the connections is a duty connection, it continues the current duty. This duty will have had a briefing period before the start of the planning horizon and will have a de-briefing period at the end of the duty. If \((s, j) \in C^+\) is a layover connection, the last duty in the planning horizon of the previous day is ended with a de-briefing and the first duty in the current planning horizon is started with a briefing period.

It should be noted that if the first flight of a pairing starts some time after the crew block becomes available, this time is not counted towards the duty hours, and thus duration of the pairing. Similarly for when the last flight terminates before the
crew block becomes unavailable. It is therefore possible to generate pairings with a duration less than the availability of the crew block $d_{st}$.

Block hour and duty hour accumulation is reset by layover connections. The resetting formula depends on the crew rule (Table 5.5) and the type of layover (regular or long layover). Resources 1 and 2 are reset by any layover, thus $E^k = E_L$, $k \in \{1, 2\}$. Resources 3 and 4 are reset along long layovers and are set to equal the accumulation of resources 1 and 2, respectively, along regular layovers. Details can be found in Section 7.1. Here we define $E^k = E_L$, $k \in \{3, 4\}$, however, we note that the type of replenishment will depend on whether connection $(i, j) \in C_{RL}$ or $(i, j) \in C_{LL}$. Resource 5 is never reset, thus $E^k = \emptyset$, $k \in \{5\}$. The rule simply forbids that a pairing exceeds MTB hours of accumulated block hours. If a path exceeds this limit, it is infeasible and can be discarded. Resource 6 accumulates credit values for each duty. The value is therefore converted to cost and then reset at the end of each duty, which is the same as saying on every layover or when the pairing ends. Thus, $E^k = \{(i, j, g) : (i, j) \in C_L, (i, j, g) \in \hat{E}\} \cup E^-$. 
Chapter 6

A Branch-And-Price Algorithm

Section 2.2 gave a high level description of a column generation procedure. As mentioned there, many strategies that accelerate column generation (CG) and branch-and-price algorithms have been developed in the literature. Unfortunately, the effectiveness of these strategies and other implementation details are highly dependent on the problem that is to be solved. In this chapter, we develop a branch-and-price algorithm that is suitable to solve our problem, more specifically the restricted master problem (RMP) as presented in Section 5.1. Solving the pricing problems is discussed in Section 7.1.

We describe and discuss acceleration strategies that we identified to work well. We evaluate their impact on the performance of the overall algorithm by conducting thorough numerical experiments.

In the current chapter, we will focus on the following aspects of the algorithm.

• Solving individual linear programs during column generation. The discussion includes choice of LP solver and some insights into the structural properties of our master problem (Section 6.2).

• Initialisation of the restricted master problem (Section 6.3).

• How many columns to add in each CG iteration (Section 6.4).

• Removing columns from the RMP to reduce its size (Section 6.5).

• How to branch in order to obtain integer solutions. We discuss different strategies for selecting connections as well as issues and benefits of not solving every
node to optimality. Furthermore, we investigate the benefits and drawbacks of not allowing pricing to occur at every node (Section 6.6).

In the final section of this chapter (Section 6.7), we will summarise our findings and formally state our branch-and-price algorithm. It should be noted that we investigated several other aspects of column generation. These either did not improve the performance of the algorithm, and are therefore not included in this thesis, or they make up our main contributions and are presented in separate chapters. Heuristic pricing is discussed in Section 7.2, while the interaction between the restricted master problem and the pricing problems is discussed in Chapter 8 and Chapter 9.

The goal of this chapter is to develop an efficient algorithm that enables us to thoroughly investigate the challenges resulting from modeling individual tail numbers and crew blocks (Chapters 8 and 9). We therefore limit exposition of numerical results to a few key performance indicators such as run time, number of iterations, and number of branch-and-bound nodes, wherever applicable. Other statistical data, for example with respect to final solutions and individual pricing problems, may be of interest but would result in many additional tables and thus distract from the important findings.

We implemented our algorithm using the SCIP V2.1.1 framework (ZIB, 2013), in which we employ CPLEX V12.3 (IBM Corporation, 2011) as the LP solver. CPLEX is called with the default settings of SCIP, except for some tolerances, as we will discuss in Section 6.2. We conducted all numerical experiments on a Dell PowerEdge R410 with dual quad core 2.67GHz Intel Xeon X5550 processors and 32GB RAM running Windows Server 2008. Our algorithm does not, however, take advantage of the multi-threading capabilities. Only a single thread is used. We experienced variability in run times even if running the exact same experiment. We therefore conducted every experiment five times and use the average of these results when evaluating the performance with respect to solution times.
6.1 Description of Test Instances

In this section, we describe the test instances we use for all numerical experiments in this thesis. For each instance, we give information about the number of flights, aircraft, and crew blocks and the resulting sizes of the master problem and pricing problems. In Section 6.1.1, we describe maintenance requirements that are considered in the instances and, in Section 6.1.2, how the number of maintenance types can be reduced by preprocessing.

We generated several different test instances based on real world data provided by our industry partner Constraint Technologies International (Evans, 2013). The data was provided under a non-disclosure agreement. We are therefore not at liberty to discuss raw data in detail but rather give aggregated and/or range information instead.

The data represents the domestic flight schedule of an Australian airline. Despite three major metropolitan areas, Sydney, Melbourne, and Brisbane, the flight network is not of a hub-and-spoke structure. Many direct connections between several cities exist. Thus, the network is a point-to-point network (Akartunali et al., 2013). However, Sydney, Melbourne, and Brisbane, are of central importance. Due to the large population, many flights arrive and depart here. Furthermore, all crews are based in these cities and the airports are the major maintenance stations.

The size of the instances is determined by the duration of the planning horizon and the number of available aircraft. As discussed in Chapter 4, we believe a seven day planning horizon to be most beneficial for our integrated airline scheduling problem. However, in addition to seven-day instances, we also generate smaller five-day instances to facilitate evaluation of different parameter settings in our algorithm. All instances start on the same date. For instances of short duration, Period 1 is three days and Period 2 is two day. For the longer duration, Period 1 is four days and Period 2 is three days.

The flight schedule is for two different fleet types; the flights in the schedule are assigned to the fleet types. We will use the categorisation of the Airline Data Project (2012) to describe the aircraft types. One type can be categorised as small narrow body (SNB) aircraft, the other as large narrow body (LNB) aircraft. In the data, fleet SNB consists of 17 aircraft and LNB of 33 aircraft.
The flights considered in an instance are the flights that have to be operated by the respective fleet on the days in question. Thus, the number of flights in an instance is a function of the duration of the planning horizon and the size of the fleet. It should be noted that since we consider actual days in a schedule, we solve a dated problem.

The number of crew blocks in an instance also depends on the fleet size and the duration of the planning horizon. Naturally, the more aircraft in a fleet, the more crews are required to operate them. Additionally, since the start time of a crew block can be after the beginning of the planning horizon, longer planning horizons will result in more crew blocks. The crew blocks are generated according to the solution of CTI’s crew scheduling software, where this software solves the original non-integrated crew pairing problem, followed by a crew rostering problem.

In Table 6.1, we present real-world (top part) and semi-artificial (bottom part) instances. We provide an instance ID, the number of aircraft, crew blocks, flights, and arcs in each instance as well as information regarding the number of constraints in the resulting master problem. For real-world instances, we generate two instances for each combination of fleet type and number of days. The instance ID, e.g. 5-S-2, identifies the number of days (5), fleet type (S)NB, and which of the two instances (here 2).

In addition to the real-world instances, to evaluate the effectiveness of strategies developed in Chapters 8 and 9, we generated semi-artificial instances that have an even larger number of pricing problems. For these, we merged the flight schedules of fleet SNB and LNB and then generated two instances, one in which all aircraft are assumed to be of type SNB, one where all aircraft are of type LNB. The aircraft types have distinctive maintenance requirements and thus dissimilar resource accumulation, which will result in different routes and thus overall solutions. For the pair of instances, the underlying flight network is the same. If we change the fleet type of an aircraft, we assume a resource accumulation according to a uniform distribution between zero and limit $U^k$, for all $k \in K$. The initial location of the aircraft, however, remains unchanged.

Since these semi-artificial instances consider the combined flight schedule, a larger number of flights and thus cover constraints results. Moreover, a larger num-
<table>
<thead>
<tr>
<th>Inst. ID</th>
<th>Fleet</th>
<th>Days</th>
<th>Total</th>
<th>Aircraft</th>
<th>Cr. Blocks</th>
<th>Pricing Problems</th>
<th>Flights</th>
<th>MP Constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Total</td>
<td>Aircraft</td>
<td>Cr. Blocks</td>
<td>Total</td>
<td>Period 1</td>
<td>Period 2</td>
</tr>
<tr>
<td>5-S-1</td>
<td>SNB</td>
<td>5</td>
<td>64</td>
<td>17</td>
<td>47</td>
<td>456</td>
<td>275</td>
<td>181</td>
</tr>
<tr>
<td>5-S-2</td>
<td>SNB</td>
<td>5</td>
<td>70</td>
<td>17</td>
<td>53</td>
<td>449</td>
<td>268</td>
<td>181</td>
</tr>
<tr>
<td>5-L-1</td>
<td>LNB</td>
<td>5</td>
<td>104</td>
<td>33</td>
<td>71</td>
<td>659</td>
<td>408</td>
<td>251</td>
</tr>
<tr>
<td>5-L-2</td>
<td>LNB</td>
<td>5</td>
<td>96</td>
<td>33</td>
<td>63</td>
<td>659</td>
<td>408</td>
<td>251</td>
</tr>
<tr>
<td>7-S-1</td>
<td>SNB</td>
<td>7</td>
<td>51</td>
<td>17</td>
<td>34</td>
<td>641</td>
<td>376</td>
<td>265</td>
</tr>
<tr>
<td>7-S-2</td>
<td>SNB</td>
<td>7</td>
<td>57</td>
<td>17</td>
<td>40</td>
<td>646</td>
<td>369</td>
<td>277</td>
</tr>
<tr>
<td>7-L-1</td>
<td>LNB</td>
<td>7</td>
<td>97</td>
<td>33</td>
<td>64</td>
<td>961</td>
<td>559</td>
<td>402</td>
</tr>
<tr>
<td>7-L-2</td>
<td>LNB</td>
<td>7</td>
<td>97</td>
<td>33</td>
<td>64</td>
<td>968</td>
<td>559</td>
<td>409</td>
</tr>
<tr>
<td>3-S-A</td>
<td>SNB</td>
<td>3</td>
<td>118</td>
<td>50</td>
<td>68</td>
<td>620</td>
<td>426</td>
<td>194</td>
</tr>
<tr>
<td>3-L-A</td>
<td>LNB</td>
<td>3</td>
<td>118</td>
<td>50</td>
<td>68</td>
<td>614</td>
<td>421</td>
<td>193</td>
</tr>
<tr>
<td>5-S-A</td>
<td>SNB</td>
<td>5</td>
<td>132</td>
<td>50</td>
<td>82</td>
<td>1131</td>
<td>689</td>
<td>442</td>
</tr>
<tr>
<td>5-L-A</td>
<td>LNB</td>
<td>5</td>
<td>132</td>
<td>50</td>
<td>82</td>
<td>1130</td>
<td>689</td>
<td>441</td>
</tr>
</tbody>
</table>

Table 6.1: Sizes of real-world (top) and semi-artificial (bottom) instances.
ber of flights per day increases the number of short and restricted connections as these depend on the density of the flight network. Hence, the number of constraints in the RMP grows quickly. We therefore only generate artificial instances spanning three and five days. For the three day instances, Period 1 is two days, Period 2 is one day. The duration of these periods in a five day instance is as before. An artificial instance is identified by an “A” in the instance ID, e.g. 5-L-A.

The flight schedule is for domestic operations and, because destinations are served regularly, we anticipate that in an efficient schedule, no crew remains idle at a destination for a long period of time, i.e. they do not have layovers significantly longer than the required times. We therefore only generated crew connections that have a duration of less than 48 hours. This means that we include all possible duty connections and regular layover connections (recall that the type of crew connections depends on the duration of the connection), while limiting the number of long layovers to a reasonable number. A long layover has to be of a duration of at least 24 hours or 12 hours if it covers the time between 10pm and 6am local time. Generating all connections up to 48 hours should thus generate a sufficiently large number of long layovers.

In contrast, we only generate aircraft connections that have a duration of up to 16 hours. This ensures that a sufficient number of maintenance possibilities exist at appropriate maintenance stations. We did not include longer connections because in an efficient schedule, aircraft are never kept idle on the ground for a long period of time. Furthermore, we assume that the aircraft are used throughout the planning horizon, except when they undergo maintenance. We therefore only connect flights on the last day of Period 1 to the sink node, i.e. we limit the number of sink arcs.

The sizes of the pricing problems for each instance are presented in Table 6.2. We differentiate between aircraft routing pricing problems (ARPP) and crew pairing pricing problems (CPPP). Since our instances do not include any HEAM (see next section), the ARPPs always span the entire Period 1 of the planning horizon. The flights considered in an individual ARPP depend on the starting location of the aircraft as flights that cannot physically be reached before their departure do not need to be considered. Otherwise, the networks in the ARPPs are identical, and thus, the ARPPs are of similar size. The average number of flights and arcs as
6.1. DESCRIPTION OF TEST INSTANCES

The size of the CPPPs, on the other hand, varies significantly as they depend on the duration of the represented work periods and the starting location of the crews. The table gives the number of flights and arcs for the average and the largest CPPP in each instance.

Some of the instances take a very long time to solve, especially with certain parameter settings. We therefore do not use large instances to evaluate the performance of the strategies developed in this chapter. Instead, we will only conduct experiments on the real world instances spanning five days and the artificial instances spanning three days. In what follows, we will refer to these instances as small instances, the others as large instances.

6.1.1 Maintenance

As was described in Section 4.2.1, we do not follow the classical categorisation of A, B, C, and D checks but instead, consider the newer task-driven progressive maintenance checking approach. The solver has to schedule line maintenance (SLIM) for each aircraft whenever necessary and has to provide maintenance opportunities (MOPP) to absorb additional unscheduled maintenance that may arise on the day of operations. Heavy maintenance checks (HEAM) are not scheduled but their limits must be observed.

The maintenance types differ between fleets. In each fleet, four SLIMs have to be respected, while only one type of MOPP has to be provided. Our data does not contain any information about heavy maintenance checks, thus our instances do not contain any HEAM.

The maintenance checks that we consider in all instances are displayed in Table 6.3. It gives redacted maintenance IDs, the aircraft type they apply to, the maintenance category, the number of indicators that apply to the maintenance type, and the number of weeks between two such checks. It should be noted that the latter is merely an approximation to give the reader an idea of the time frames. Depending on the indicator, the “weeks between” really depends on the actual time elapsed, the flying time or the number of pressure cycles since the last maintenance of this type. The actual limits fall under the non-disclosure agreement and are therefore not provided. Maintenance types S1 and S2 are staggered, that is, a LNB aircraft will
Table 6.2: Sizes of pricing problems in each instance.

<table>
<thead>
<tr>
<th>Table ID</th>
<th>Nodes</th>
<th>Arcs</th>
<th>Repl.</th>
<th>Nodes</th>
<th>Arcs</th>
<th>Repl.</th>
</tr>
</thead>
<tbody>
<tr>
<td>5-S-1</td>
<td>1281</td>
<td>1049</td>
<td>398</td>
<td>1281</td>
<td>1049</td>
<td>398</td>
</tr>
<tr>
<td>5-S-2</td>
<td>1281</td>
<td>1049</td>
<td>398</td>
<td>1281</td>
<td>1049</td>
<td>398</td>
</tr>
<tr>
<td>5-L-1</td>
<td>1281</td>
<td>1049</td>
<td>398</td>
<td>1281</td>
<td>1049</td>
<td>398</td>
</tr>
<tr>
<td>5-L-2</td>
<td>1281</td>
<td>1049</td>
<td>398</td>
<td>1281</td>
<td>1049</td>
<td>398</td>
</tr>
<tr>
<td>7-S-1</td>
<td>1281</td>
<td>1049</td>
<td>398</td>
<td>1281</td>
<td>1049</td>
<td>398</td>
</tr>
<tr>
<td>7-S-2</td>
<td>1281</td>
<td>1049</td>
<td>398</td>
<td>1281</td>
<td>1049</td>
<td>398</td>
</tr>
<tr>
<td>3-S-A</td>
<td>1281</td>
<td>1049</td>
<td>398</td>
<td>1281</td>
<td>1049</td>
<td>398</td>
</tr>
<tr>
<td>3-L-A</td>
<td>1281</td>
<td>1049</td>
<td>398</td>
<td>1281</td>
<td>1049</td>
<td>398</td>
</tr>
<tr>
<td>5-S-A</td>
<td>1281</td>
<td>1049</td>
<td>398</td>
<td>1281</td>
<td>1049</td>
<td>398</td>
</tr>
<tr>
<td>5-L-A</td>
<td>1281</td>
<td>1049</td>
<td>398</td>
<td>1281</td>
<td>1049</td>
<td>398</td>
</tr>
<tr>
<td>7-S-S</td>
<td>1281</td>
<td>1049</td>
<td>398</td>
<td>1281</td>
<td>1049</td>
<td>398</td>
</tr>
<tr>
<td>7-L-S</td>
<td>1281</td>
<td>1049</td>
<td>398</td>
<td>1281</td>
<td>1049</td>
<td>398</td>
</tr>
<tr>
<td>3-S-S</td>
<td>1281</td>
<td>1049</td>
<td>398</td>
<td>1281</td>
<td>1049</td>
<td>398</td>
</tr>
<tr>
<td>3-L-S</td>
<td>1281</td>
<td>1049</td>
<td>398</td>
<td>1281</td>
<td>1049</td>
<td>398</td>
</tr>
</tbody>
</table>

Average ARPP

Average CPPP
have one of these approximately every two weeks. The same applies to maintenance
types S5 and S6 for an SNB aircraft.

<table>
<thead>
<tr>
<th>Name</th>
<th>Fleet</th>
<th>Type</th>
<th># Indicators</th>
<th>Weeks betw.</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>LNB</td>
<td>SLIM</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>S2</td>
<td>LNB</td>
<td>SLIM</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>S3</td>
<td>LNB</td>
<td>SLIM</td>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>S4</td>
<td>LNB</td>
<td>SLIM</td>
<td>1</td>
<td>14</td>
</tr>
<tr>
<td>M1</td>
<td>LNB</td>
<td>MOPP</td>
<td>1</td>
<td>0.3</td>
</tr>
<tr>
<td>S5</td>
<td>SNB</td>
<td>SLIM</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>S6</td>
<td>SNB</td>
<td>SLIM</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>S7</td>
<td>SNB</td>
<td>SLIM</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>S8</td>
<td>SNB</td>
<td>SLIM</td>
<td>1</td>
<td>14</td>
</tr>
<tr>
<td>M2</td>
<td>SNB</td>
<td>MOPP</td>
<td>1</td>
<td>0.3</td>
</tr>
</tbody>
</table>

Table 6.3: Considered maintenance types for each fleet.

As was discussed in Section 3.4 and 4.2.1, due to the rolling planning horizon, we
need to ensure that if an aircraft has to be maintained during the planning horizon,
the maintenance check is scheduled as late as possible. In Section 4.2.1 and 5.2.1,
we discussed how this can be achieved using appropriate cost functions for SLIMs:
a maintenance check that is scheduled early in the planning horizon incurs a higher
cost, i.e. penalty, than one that is scheduled later. Unfortunately, our industry
partner did not provide us with information regarding these penalties. We therefore
use values that are commensurate with crewing cost. A check that is carried out in
the beginning of the planning horizon incurs a cost that is about five times as high
as a layover. This seems to be a quite large penalty, however, since the decrease
is linear and we want the check to be scheduled in the last hours before a limit is
exceeded, these values seem to be reasonable.

6.1.2 Preprocessing of Maintenance Types and Indicators

Given the long intervals between two checks of the same SLIM type, which can be
several weeks, it is obvious that in a one-week planning horizon, we do not need to
consider every SLIM for each aircraft. We can remove any SLIM for an individual aircraft if we can prove that its limit cannot be exceeded during the planning horizon. This is trivial if the indicator is Actual Time between checks. For the other indicators, we calculate the resource usage of worst-case scenarios, i.e. the aircraft flies as much as possible, which means no other SLIM occurs, always flying the shortest or longest flight (depending on the indicator), and always using the minimum turn time between flights. If these worst-case resource usages in combination with the current resource accumulation of the aircraft do not exceed the resource limits of a SLIM, then the SLIM does not need to be considered for this aircraft. We can in fact conduct this analysis for each aircraft and maintenance type/indicator pair separately. Recall that each such pair is represented by a separate resource. We then remove the resource if its limit is never exceeded. Doing so does not affect other resources that relate to the same maintenance type.

We found that in our instances, even though SLIMs have up to three criterion, there always is one criteria that dominated the others, i.e. its limit is exceeded earlier. This is usually not due to the respective resource consumption during the planning horizon but because of the resource consumption in the weeks since the last maintenance check of the same type. For example, in our instances pressure cycles never need to be considered because the limits on flying time or actual time between are always exceeded more quickly. We conclude from the data that in an Australian domestic flight schedule, the average ratio of flying time and pressure cycles is very much in favour of flying time, which is not surprising considering the vastness of the Australian continent. Of course, we cannot generalise this statement. Flight schedules in other parts of the world, e.g. Europe, may have opposite tendencies. It should be noted that our model does not necessitate a single indicator per maintenance type. The pricing problem solver is capable of handling several resources relating to the same maintenance type.

In summary, in our instances every maintenance type only requires one indicator and thus resource. Therefore, for both fleet types we consider a total of five resources, four of which relate to the SLIMs, and one that relates to the MOPP. The advantage of removing resources is that less effort is required in the labelling algorithm (see Section 7.1.1) as a smaller number of labels results.
6.1. DESCRIPTION OF TEST INSTANCES

After preprocessing maintenance types and indicators, the ARPPs will have different resources even though the represented aircraft are of the same type. In our instances, we did not encounter an aircraft that required more than one SLIM in the planning horizon. In contrast, every aircraft requires the resource relating to the MOPP since it has a limit of three days. Thus, at most two resources are considered in each aircraft pricing problem. Table 6.4 shows the number of aircraft for each instance that require only a MOPP, e.g. M1, or a combination of a SLIM and a MOPP, e.g. S1+M1.

Assuming worst-case scenarios as described above leads to a significantly higher worse case resource consumption than the true worst case. For example, we repeat the same shortest or longest flight, respectively, over and over again even though this flight may not be operated several times. Furthermore, we assume unchanged flight operations throughout the night, which is not the case. The true worst-case consumption can be calculated by altering function FSP(\(\kappa\)), \(\kappa \in K\), of the preprocessing procedure (see Section 7.1.2) to calculate maximum instead of minimum weight paths. We did not do so because we wanted to generate instances in which the ARPPs consider multiple and varying resources since this can be expected in most real-world scenarios.
Table 6.4: Number of aircraft in each instance that are likely to require a check of the maintenance types indicated.
6.2 LP solving

The restricted master problem does not contain any integrality constraints and thus can be solved using any LP solution method. In Section 2.2, we already discussed the advantages and disadvantages of the primal and dual simplex as well as the interior point method in terms of warm start capabilities and effect on dual values. Since it is not clear which solver is the best choice for our problem, we investigated their impact on the overall performance of the branch-and-price algorithm.

CPLEX provides several LP solvers, however, not all of them are available when calling CPLEX from within the SCIP framework. Moreover, SCIP always specifies which LP solver CPLEX has to use. On computers capable of multi-threading, CPLEX also has the ability to run the primal and dual simplex concurrently to the interior point method (IBM Corporation, 2011). Unfortunately, this functionality is not available in SCIP.

We conducted experiments that compared the performance of our algorithm when using the SCIP default setting with always invoking primal simplex, dual simplex, or the interior point method. In the default setting, SCIP evaluates which simplex method to use based on the feasibility status of the current LP basis. For the interior point method, we do not perform the crossover step. By performing a crossover operation, an interior point solution is transformed to a basic solution. This operation can be expensive and, in column generation, is not necessary as only a dual feasible solution is required, not necessarily a basic one.

Table 6.5 shows the total run time in seconds of the branch-and-price algorithm when using the different LP solution methods while solving the root node. We would like to disclaim that all reported times in this thesis are measured in seconds. Furthermore, as we will discuss in Section 6.6, the algorithm performs only one dive in the tree, i.e. no backtracking is allowed.

The default setting and always using the primal simplex method performed best and almost equally well. This is not surprising as the default setting always invoked the primal method. The difference in time can be explained by SCIP evaluating which method to call. Dual simplex and especially the interior point method perform poorly with solution times of up to 3.5 times that of the default setting.

Table 6.6 displays the number of CG iterations required to solve the root node.
When using the default setting or always primal or dual simplex, the algorithm performs a somewhat similar number of iterations. For the interior point method, the number of CG iterations is on average reduced by 12.64% compared to the default setting. This relatively small reduction and the lack of warm-start capabilities explain the poor performance when using the interior point method. The longer LP solution times are not offset by a sufficiently faster convergence of the CG procedure.

Table 6.5: Overall run times (s) at root node when using different LP solution methods.

<table>
<thead>
<tr>
<th>Instance</th>
<th>Default</th>
<th>Primal</th>
<th>Dual</th>
<th>Int. Point</th>
</tr>
</thead>
<tbody>
<tr>
<td>5-S-1</td>
<td>645</td>
<td>659</td>
<td>1527</td>
<td>1986</td>
</tr>
<tr>
<td>5-S-2</td>
<td>968</td>
<td>856</td>
<td>1439</td>
<td>2811</td>
</tr>
<tr>
<td>5-L-1</td>
<td>2377</td>
<td>2353</td>
<td>6967</td>
<td>6281</td>
</tr>
<tr>
<td>5-L-2</td>
<td>2124</td>
<td>2039</td>
<td>7210</td>
<td>5656</td>
</tr>
<tr>
<td>3-S-A</td>
<td>1931</td>
<td>1721</td>
<td>2697</td>
<td>6704</td>
</tr>
<tr>
<td>3-L-A</td>
<td>1755</td>
<td>1786</td>
<td>2268</td>
<td>5496</td>
</tr>
</tbody>
</table>

Table 6.6: Number of CG iterations to solve the root node when using different LP solution methods.

<table>
<thead>
<tr>
<th>Instance</th>
<th>Default</th>
<th>Primal</th>
<th>Dual</th>
<th>Int. Point</th>
</tr>
</thead>
<tbody>
<tr>
<td>5-S-1</td>
<td>224</td>
<td>224</td>
<td>279</td>
<td>179</td>
</tr>
<tr>
<td>5-S-2</td>
<td>315</td>
<td>315</td>
<td>306</td>
<td>238</td>
</tr>
<tr>
<td>5-L-1</td>
<td>340</td>
<td>340</td>
<td>337</td>
<td>267</td>
</tr>
<tr>
<td>5-L-2</td>
<td>298</td>
<td>298</td>
<td>308</td>
<td>259</td>
</tr>
<tr>
<td>3-S-A</td>
<td>148</td>
<td>148</td>
<td>159</td>
<td>161</td>
</tr>
<tr>
<td>3-L-A</td>
<td>163</td>
<td>163</td>
<td>153</td>
<td>154</td>
</tr>
</tbody>
</table>

After the root node, when solving the first LP at a node, the SCIP default setting usually performs one dual simplex solve. Then, in subsequent LP solves at a node, most often the primal simplex method is invoked. In our instances, 5 – 15% of the LP solution time is spent in the dual simplex method, the remainder in the primal
6.2. LP SOLVING

We found that, when solving more than the root node, the default strategy outperforms always invoking the primal simplex by a small margin. We therefore elected to use SCIP’s default LP selection strategy for all our experiments.

We observed that in all our instances the solution process suffered from numerical issues. In about 20% of the LP solves, CPLEX takes measures that are described in the CPLEX User’s manual (2011) under section “Numeric difficulties”. Most often, the problems are perturbed or become infeasible after removing a shift. Markowitz thresholds are adjusted rarely and repeated singularities did not occur. In the remainder of this section, we attempt to investigate the causes of these numerical issues and try to alleviate them.

The data presented in the model is such that all cost and matrix coefficients are integral and have a relatively small variation in the order of magnitude. CPLEX provides the so-called condition number of unscaled basis to measure the sensitivity of a linear system to the problem data. If the “magnitude of the condition number exceeds 1e+13, numerical problems are more likely to occur”. However, the condition numbers for our instances are smaller than 1e+06, which indicates “virtually no chance of ill conditioning” (IBM Corporation, 2011).

Folklore in mathematical programming suggests several strategies that are easy to implement when dealing with numerical issues. We conducted experiments in which we altered the mathematical problem slightly by using set cover constraints instead of set partitioning constraints or using constraints of type $Ax \geq 1 - \epsilon$ instead of set partitioning constraints. We also manually perturbed costs that are otherwise equal to zero. Unfortunately, none of these had a noticeable effect.

CPLEX has the option to carry out calculations with an emphasis on numerical precision. Under this setting, each LP iteration is slower but a lower number of iterations may result if otherwise the numerical issues lead to a slow convergence of the LP solution value (IBM Corporation, 2011). Unfortunately, it is not possible to predict at which CG iteration numerical trouble will occur. We conducted experiments and found that resulting overall LP solution times did not warrant using an emphasis on numerical precision throughout the entire column generation algorithm.

Another strategy to alleviate numerical trouble is to alter tolerances (IBM Corporation, 2011). We increased the feasibility and the optimality tolerance as well as
the general integer comparison tolerance by a factor of ten. This reduced numerical issues slightly. A larger increase in the tolerances did not result in a further reduction of numerical issues. We thus use the slightly increased tolerances (factor of 10) for all experiments throughout this thesis.

If numerical issues occurred, CPLEX was usually able to overcome them by applying one or two counter-measures. However, even though this was rare, some LPs had significant numerical troubles. The problems had to be re-perturbed several times and repeatedly became infeasible after removing shifts. In these cases, CPLEX terminated with a solution status reporting infeasibilities after unscaling. SCIP then attempts to resolve the LP by calling CPLEX with several different parameter settings, including turning on/off FastMIP, unscaling, solving from scratch, and using the opposite simplex method. This often takes several hundred or even a few thousand seconds and does not always result in overcoming the numerical issues. In column generation, unless proving optimality, we only require feasible dual values, not necessarily optimal ones. SCIP has the ability to ignore unresolved numerical instabilities as long as the dual values are feasible, which is the case when CPLEX terminates with infeasibilities after unscaling. Thus, to avoid re-solving a single LP several times and thereby spending several hundred seconds, we chose to ignore numerical instabilities altogether. It should be noted though that this does not change an individual LP solve. CPLEX still takes longer to solve an LP when numerical troubles occur. Furthermore, since we are not using optimal dual values when unresolved numerical troubles occur, we cannot guarantee that a node in the branch-and-price algorithm is solved to optimality. However, as pointed out, this does not happen often, and, as will be discussed in Section 6.6, we do not solve all nodes to optimality in our algorithm anyway.

We analysed the master problem to identify where the numerical issues and resulting long LP solution times originate. For this, we solved the aircraft and the crew part of the integrated problem separately using the same algorithm. The aircraft RMP only contains the flight cover constraints (5.1) and convexity constraints (5.4). Similarly, the crew RMP contains only cover constraints (5.2) and convexity constraints (5.5). Thus, apart from the slightly larger number of crew convexity constraints, the number of constraints in the RMPs is equal. For the remainder of this
subsection we will call the respective problems crew problem and aircraft problem.

To enable a fair comparison, we must also generate a similar number of columns. We therefore add the same number of columns per CG iteration. Hence, for these experiments, we only solve one pricing problem per iteration and add up to ten columns per solve. The tailing-off effect in column generation usually results in the pricing problems generating fewer columns per iteration. To avoid any interference from this, we aborted the experiment after the first 500 iterations for all but instances 5-S-1, 5-S-2, and 7-S-1. The latter, we terminated after 450 iterations, the other two after 350 iterations.

Table 6.7 displays the total time in seconds spent solving the LPs for the aircraft and crew part separately. The fourth column shows the difference in percent. For the sake of comparison, we include the total number of columns that were generated in the experiment. Unlike other experiments in this chapter, we include results for larger instances as well since we are interested in comparing three, five, and seven day instances.

We observe a significantly lower total LP solution time for all instances when solving the crew problem despite generating a comparable number of columns. On average the total LP time was 39.79% lower, while only 3.16% fewer columns were generated. The slightly lower number of columns are due to some of the small crew pricing problems not returning ten columns in every iteration.

The main difference between the aircraft and crew problem is that on average a route contains many more flights than a pairing since crews need rest periods more frequently. Table 6.8 displays the average number of flights, and thus constraints, that were covered by a route and pairing, respectively, for each instance.

Not surprisingly, the number of flights per route depends on the duration of the planning horizon. Except for maintenance, aircraft are available throughout the planning horizon and thus will operate a relatively constant number of flights per day. This number depends on the fleet type, which is due to the data provided to us. Recall that the instances represent the flight schedules for an individual fleet. In the data, the small narrow body aircraft are used on shorter flights and therefore operate more flights per day. Since the instances are generated from the flight schedules of each fleet, our instances will show a similar behaviour, i.e. a larger number of flights
### Table 6.7: Total LP solutions times (s) and number of columns generated in the first 500 iterations when solving the crew and aircraft part separately.

<table>
<thead>
<tr>
<th>Instance</th>
<th>LP Time (s)</th>
<th>Columns</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Aircraft</td>
<td>Cr. Blocks</td>
</tr>
<tr>
<td>5-S-1</td>
<td>23.26</td>
<td>12.71</td>
</tr>
<tr>
<td>5-S-2</td>
<td>24.33</td>
<td>11.84</td>
</tr>
<tr>
<td>5-L-1</td>
<td>70.25</td>
<td>37.11</td>
</tr>
<tr>
<td>5-L-2</td>
<td>71.63</td>
<td>40.23</td>
</tr>
<tr>
<td>7-S-1</td>
<td>70.19</td>
<td>38.91</td>
</tr>
<tr>
<td>7-S-2</td>
<td>95.63</td>
<td>45.48</td>
</tr>
<tr>
<td>7-L-1</td>
<td>131.42</td>
<td>86.54</td>
</tr>
<tr>
<td>7-L-2</td>
<td>133.97</td>
<td>80.40</td>
</tr>
<tr>
<td>3-S-A</td>
<td>68.24</td>
<td>39.63</td>
</tr>
<tr>
<td>3-L-A</td>
<td>66.63</td>
<td>39.97</td>
</tr>
<tr>
<td>5-S-A</td>
<td>153.53</td>
<td>118.72</td>
</tr>
<tr>
<td>5-L-A</td>
<td>138.46</td>
<td>119.12</td>
</tr>
</tbody>
</table>

| Average  | -39.79      | -3.16     |
Table 6.8: Average number of flights covered per route or pairing for different instances.

<table>
<thead>
<tr>
<th>Instance</th>
<th>Flights/Route</th>
<th>Flights/Pairing</th>
</tr>
</thead>
<tbody>
<tr>
<td>5-L-1</td>
<td>13.31</td>
<td>6.16</td>
</tr>
<tr>
<td>5-L-2</td>
<td>13.32</td>
<td>6.15</td>
</tr>
<tr>
<td>5-S-1</td>
<td>15.97</td>
<td>6.17</td>
</tr>
<tr>
<td>5-S-2</td>
<td>15.74</td>
<td>5.98</td>
</tr>
<tr>
<td>7-S-1</td>
<td>22.13</td>
<td>8.00</td>
</tr>
<tr>
<td>7-S-2</td>
<td>21.74</td>
<td>7.25</td>
</tr>
<tr>
<td>7-L-1</td>
<td>18.38</td>
<td>7.06</td>
</tr>
<tr>
<td>7-L-2</td>
<td>18.37</td>
<td>7.15</td>
</tr>
<tr>
<td>3-S-A</td>
<td>9.87</td>
<td>6.19</td>
</tr>
<tr>
<td>3-L-A</td>
<td>9.71</td>
<td>6.04</td>
</tr>
<tr>
<td>5-S-A</td>
<td>16.50</td>
<td>7.11</td>
</tr>
<tr>
<td>5-L-A</td>
<td>16.08</td>
<td>6.96</td>
</tr>
</tbody>
</table>

per route for SNB instances. The four artificial instances contain the merged flight schedules and thus cover a number of flights per route equal to the weighted average of the number of flights per route of the individual fleets.

The average number of flights per pairing, on the other hand, is much lower. The main reasons are that crew blocks have a duration of at most five days and that crews need rest periods, i.e. layovers, during which they simply cannot cover any flights. The limited duration of each crew block is also the reason why the number of flights per pairing does not increase much as the planning horizon increases. The number of crew blocks that are on a pairing at the beginning of the horizon (category Pairing Continues) is invariant with respect to the duration of the planning horizons. Only the number of crew blocks that start throughout the horizon (category Pairing Starts) increases if the duration of the planning horizon is longer. However, recall that these crew blocks may have a duration of less than five days since the duration depends on the duration of the original work period (see Section 4.2).

It has been documented that when the number of non-zero entries of columns
approaches 15 or more, long LP solution times result because the problem is highly
degenerate, the simplex methods as well as the interior point method require more
iterations, and the column generation procedure converges only slowly (Oukil et al.,
2007), (IBM Corporation, 2011). It appears that our problem falls in line with these
findings.

However, the papers cited do not report numerical difficulties. In the previous
experiment, we found that the crew problem has virtually no numerical problems,
while the aircraft problem did suffer from them to an extent that is similar to the
integrated problem.

In summary, the aircraft aspect of the integrated problem causes long LP solution
times. This appears to be due to the high number of non-zero entries in the aircraft
columns and due to overcoming numerical difficulties during the LP solves. The
cause of the numerical issues remains unclear. However, since the only significant
difference between the crew and aircraft aspect is the number of non-zero entries, it
suggests that this is also the cause of the numerical issues.

We note that the difficulties when solving LPs most likely get exacerbated as col-
umn generation proceeds. When considering the entire solution process, we observe
that in total about three times as many columns are generated for aircraft than for
crew blocks. However, at this point we would like to note that while high LP solu-
tion times are mainly caused by the aircraft aspect of the problem, the majority of
the time spent in pricing is due to solving the harder crew pairing pricing problems
(see Section 8.1). It is therefore important not to lose focus of both aspects of the
problem.

6.3 Initialisation of the RMP

At the start of column generation, columns that yield a feasible LP are required
to initiate the procedure. Our restricted master problem is infeasible because Con-
straints (5.1) and (5.2) cannot be satisfied. A standard way to overcome this is to
add an artificial variable to each of these constraints (Lübbecke, 2010). The vari-
ables have high penalties associated with them, which ensures that they will not be
part of the solution once routes and pairings are available that satisfy the respective
constraints. The penalties have a real-world connotation; they can be understood
6.3. INITIALISATION OF THE RMP

as the cost of not assigning the flight in the schedule. A stand-by crew or aircraft will have to service this flight instead.

Due to the large penalties, the CG algorithm suffers from the heading-in effect (see Section 2.2). Barnhart and Cohn (2004) make the observation that it is often beneficial to start column generation with an initial set of columns representing a feasible solution. We conducted experiments in which we generate initial columns using two primal heuristics and compare the resulting performance to initialising the RMP purely with artificial variables. The heuristics typically do not generate a complete feasible solution, that is, some flights are left unassigned. We therefore, in addition to the generated columns, include all artificial variables in the RMP. This ensures feasibility not only at the start of the CG algorithm but also later on, when branching decisions result in temporarily re-introducing artificial variables to the solution of the RMP.

Gamache and Soumis (1998) propose disjoint column generation, that is generating columns in a CG iteration that do not cover the same tasks. Touati et al. (2010) note that disjoint columns may be more efficient in the first iterations but not so in later ones. We adapt this method in one of our initialisation methods: the first heuristic, Primal1, generates a set of columns of which some are disjoint (see Algorithm 6.1). The pricing problems are solved sequentially, where the ARPPs are ordered by decreasing order of resource consumption and the CPPPs are ordered by start date $d_{st}$. If feasible paths are found in a pricing problem, the flights in the best path are removed from the set of flights of all other pricing problems. Each pricing problem can generate up to three paths, for which the columns are added to the RMP. It should be noted that the algorithm is called for crew and aircraft pricing problems separately.

Often, the goal of initialisation methods is to simply provide an initial solution, regardless of its quality. Usually columns are generated that cover as many constraints as possible, which in our case is equivalent to a large amount of flying in the route or pairing. In our problem, this results in expensive columns. Recall that the cost of a column depends on whether a SLIM is scheduled or on the credit accumulation and occurrence of layovers, respectively. All these depend on the resource accumulation and therefore amount of flying. The expensive columns may not be
Algorithm 6.1: PrimalBasic()

Input: Set of all pricing problems $\mathcal{P}$.

Output: Set of columns $Q_p$, $\forall p \in \mathcal{P}$.

6.1.1 forall the $p \in \mathcal{P}$ do

6.1.2 Solve pricing problem $p$ giving set of columns $Q_p$.

6.1.3 $\mathcal{P} := \mathcal{P} \setminus \{p\}$

6.1.4 if $Q_p \neq \emptyset$ then

6.1.5 forall the $o \in \mathcal{P}$ do

6.1.6 Remove all flights that are in the optimal path $q_p \in Q_p$ from the

set of flights $N_o$ in pricing problem $o$.

useful after the initial phase of column generation once better columns are generated. In our algorithm, we therefore aim to, apart from covering as many constraints as possible in the initial solution, generate columns that are comparable in terms of cost to those generated later.

Without any changes to the objective function, the solver will generate routes and pairings that have a minimum amount of flying since the cost of a route or pairing depends on the amount of flying covered. We therefore need to alter the objective function in Primal1. We ignore all actual cost and instead associate a cost of -2 with 2/3 of all flights, while the remaining third has a cost of +1 associated with it. The solver then will avoid the flights with positive cost unless including one allows the path to cover another flight with negative cost. This cost function results in columns that cover a similar number of flights compared to columns generated during the actual CG procedure. It should be noted that the negative and positive cost are assigned randomly and this is done for each pricing problem separately. Doing so ensures that the method is not biased against particular flights.

Unfortunately, for a number of pricing problems, no feasible path can be found once several flights have been removed. This in turn often means that not all flights are covered by a route or pairing when Primal1 terminates. On average about 30% of all flights are not covered.

The second primal heuristic, Primal2, first calls Algorithm 6.1 and then generates columns that cover the remaining uncovered flights. For this, all pricing problems
are solved in sequence, where the ordering criteria is as before, except that we first solve all pricing problems for which no feasible path has been found yet. The network in each pricing problem again contains all flights, i.e. the full set $N^a$, $a \in A$ and $N^b$, $b \in B$, respectively.

We again modify the objective function to not consider actual cost but instead associate a cost of -1 with all flights that are not yet covered by any aircraft or crew block, respectively. All other flights have a cost of zero. When a pricing problem finds a path that includes a previously uncovered flight, the cost of that flight is set to zero in all other pricing problems. Note that the flight is not removed. The algorithm terminates if all flights are covered by at least one route and pairing or if all pricing problems were solved once.

The impact of the different initialisation methods will be most pronounced in the initial stage of the CG procedure. We therefore limited our experiments to solving the root node. Table 6.9 displays run times and the number of CG iterations when using the three different initialisation methods.

<table>
<thead>
<tr>
<th>Instance</th>
<th>Artificial</th>
<th>Primal1</th>
<th>Primal2</th>
</tr>
</thead>
<tbody>
<tr>
<td>5-S-1</td>
<td>907</td>
<td>867</td>
<td>859</td>
</tr>
<tr>
<td>5-S-2</td>
<td>1125</td>
<td>1235</td>
<td>1176</td>
</tr>
<tr>
<td>5-L-1</td>
<td>2234</td>
<td>2126</td>
<td>1896</td>
</tr>
<tr>
<td>5-L-2</td>
<td>2234</td>
<td>2126</td>
<td>1896</td>
</tr>
<tr>
<td>3-S-A</td>
<td>2610</td>
<td>2613</td>
<td>2353</td>
</tr>
<tr>
<td>3-L-A</td>
<td>2023</td>
<td>1932</td>
<td>1545</td>
</tr>
</tbody>
</table>

Table 6.9: Run time (s) and number of iterations required to solve the root node to optimality when using different restricted master problem initialisation methods.

Compared to initialising using only artificial variables, the run time can be reduced by an average of 1.45% when using Primal1, and 10.75% when using Primal2. The reduction is due to fewer CG iterations for all but instances 5-S-2 and 5-S-1. For the latter, the number increases by only a small amount, while for the former, the increase is 12.35 and 22.56% respectively. Interestingly, for instance 5-S-2, the
number of columns generated increases only by 0.72 and 5%, respectively. The additional iterations do not generate many additional columns. We find that for this instance, a longer tailing-off effect occurred. The same is true for instance 5-S-1. This suggests that the reduction due to Primal1 and Primal2 will be even more pronounced when performing early branching to avoid the tailing-off effect, which we describe in Section 6.6.1.

In summary, initialisation using Primal2 outperforms only using artificial variables and using Primal1. We therefore use this method in all subsequent experiments.

6.4 Adding Multiple Columns per Pricing Problem

In this section, we investigate how many columns should be added per pricing problem and iteration. It should be noted that we solve all pricing problems in every iteration. We investigate how many and which pricing problems to solve in an iteration in Chapter 8.

For column generation to proceed, only one column with negative reduced cost needs to be added per CG iteration. However, adding multiple negative cost columns per iteration is often beneficial and is thus a commonly used acceleration strategy (Barnhart and Cohn, 2004). The advantage is that several columns are available to enter the basis, thereby improving the objective function value. This usually results in carrying out fewer CG iterations; the procedure is said to converge more quickly. Of course, it cannot be guaranteed that several columns do in fact enter the basis as the new columns are generated using the same dual values and thus may be very similar.

The downside is that solving the LP may take longer as the size of the RMP increases more rapidly. In our algorithm, finding multiple columns is not more time consuming as the labelling algorithm stores multiple labels any way (see Section 7.1.1). The only additional time required is spent in physically creating and adding the columns to the master problem, although this is insignificant compared to the overall solution time.

It should be obvious that when generating many columns for a single pricing problem, the columns compete within the same convexity constraint, quite possibly
6.4. **ADDING MULTIPLE COLUMNS PER PRICING PROBLEM**

Reducing their likelihood of entering the basis. It seems compelling that generating and adding several columns for every pricing problem instead of adding the same number of columns for a single pricing problem is a superior strategy. We investigate this further in Chapter 8. Here, we simply use this notion and set a limit on the number of columns added per pricing problem in each iteration rather than using a global limit on the total number of columns added per iteration.

We conducted experiments in which we limited the number of columns added per pricing problem to 1, 2, 3, 4, 5, 7, and 10. It should be noted that due to the large number of pricing problems, this equates to an implicit global limit of 50 to 1300 columns per iteration, depending on the instance and individual limit. Table 6.10 shows the number of CG iterations performed when aborting the algorithm after a single dive, i.e. no backtracking is allowed.

<table>
<thead>
<tr>
<th>Instance</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>7</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>5-S-1</td>
<td>803</td>
<td>384</td>
<td>378</td>
<td>388</td>
<td>346</td>
<td>300</td>
<td>324</td>
</tr>
<tr>
<td>5-S-2</td>
<td>927</td>
<td>547</td>
<td>401</td>
<td>546</td>
<td>383</td>
<td>340</td>
<td>316</td>
</tr>
<tr>
<td>5-L-1</td>
<td>694</td>
<td>538</td>
<td>413</td>
<td>437</td>
<td>350</td>
<td>255</td>
<td>212</td>
</tr>
<tr>
<td>5-L-2</td>
<td>849</td>
<td>616</td>
<td>456</td>
<td>427</td>
<td>382</td>
<td>273</td>
<td>218</td>
</tr>
<tr>
<td>3-S-A</td>
<td>521</td>
<td>356</td>
<td>255</td>
<td>201</td>
<td>186</td>
<td>162</td>
<td>117</td>
</tr>
<tr>
<td>3-L-A</td>
<td>453</td>
<td>289</td>
<td>241</td>
<td>172</td>
<td>171</td>
<td>166</td>
<td>116</td>
</tr>
</tbody>
</table>

Table 6.10: Number of CG iterations required to solve each instance when using different limits on the number of columns added to the RMP per pricing problem.

Figure 6.1 displays the data graphically. We see a clear reduction in the number of iterations as the limit increases. The largest decrease occurs when increasing the limit from one to two, whereas the reduction is smaller for larger limits. For instances 5-L-1 and 5-S-2, we see an increase in the number of iterations when the limit is four compared to three. This can be attributed to the branching decisions made; a larger number of nodes was explored for these instances in this setting.

The run times (see Table 6.11 and Figure 6.2) show a reduction for most instances when increasing the limit from one to two, and an increase for larger limits. A limit
of four gives inconsistent results. The increase in run time for instances 5-L-1 and 5-S-2 are explained by the larger number of CG iterations (see above). For instance 5-L-2, the LPs were simply harder to solve, resulting in a disproportional increase in LP solution time. The decreases in run time for instances 3-S-A and 3-L-A when comparing a limit of four against three, are due to the smaller number of iterations not yet being offset by the increased LP solution times (which are due to a large number of columns in the RMP).

<table>
<thead>
<tr>
<th>Instance</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>7</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>5-S-1</td>
<td>1160</td>
<td>644</td>
<td>911</td>
<td>1113</td>
<td>1206</td>
<td>1270</td>
<td>2114</td>
</tr>
<tr>
<td>5-S-2</td>
<td>1129</td>
<td>965</td>
<td>986</td>
<td>1600</td>
<td>1269</td>
<td>1437</td>
<td>2220</td>
</tr>
<tr>
<td>5-L-1</td>
<td>2413</td>
<td>2664</td>
<td>2678</td>
<td>3671</td>
<td>3151</td>
<td>2699</td>
<td>3256</td>
</tr>
<tr>
<td>5-L-2</td>
<td>3011</td>
<td>3117</td>
<td>3126</td>
<td>3767</td>
<td>3804</td>
<td>3359</td>
<td>3556</td>
</tr>
<tr>
<td>3-S-A</td>
<td>3507</td>
<td>2513</td>
<td>2427</td>
<td>2089</td>
<td>2893</td>
<td>2948</td>
<td>2785</td>
</tr>
<tr>
<td>3-L-A</td>
<td>2224</td>
<td>2001</td>
<td>2022</td>
<td>1814</td>
<td>2142</td>
<td>2546</td>
<td>2637</td>
</tr>
</tbody>
</table>

Table 6.11: Run times (s) when using different limits on the number of columns added per pricing problem.
Except for two instances, the run times do not differ much for limits of two and three even though we observed a larger reduction in the number of iterations when the limit is three. The run times did not decrease as more columns are present in the LPs, making them harder to solve. In the following section, we devise strategies that reduce LP solution times that are due to a large number of columns in the RMP. We therefore expect the run times to decrease more for a limit of three than for a limit of two. We thus decided to use a limit of three in all subsequent experiments, unless stated otherwise. The reduction in the number of CG iterations flattens for limits larger than three and by using this limit, we stay clear of the unpredictable behaviour resulting from a limit of four.

6.5 Column Management

In the previous section, we observed that run times increase when a large number of columns per pricing problem is added to the master problem in each iteration. This is mainly due to the greater number of columns in the RMP. In this section, we discuss how columns may be removed from the RMP because they are, at least temporarily, not required. Deleting columns may lead to a reduction in LP solution times, however, the convergence of the CG procedure is not affected positively. In fact, more CG iterations may be required because it is necessary to re-introduce
columns. As a result, the overall solution time may be increased if columns are removed too aggressively.

The strategy we devised for removing columns is governed by a limitation of the SCIP framework: columns that were added at previous nodes cannot be removed at the current node. Before describing our strategy, we first examine the impact of this limitation and the options available in SCIP. The framework has three parameters that control column deletion: the column age limit $\psi$, a flag for removing all non-basic columns after the root node, and a flag for removing all non-basic columns after nodes other than the root node. The column age counts the number of consecutive LP solves for which a column has been non-basic. If the column age limit is reached, the column is removed. This is done every age limit halved number of iterations.

In general, any column that is removed by SCIP is stored in a column pool, called \emph{varpricer}. Whenever the RMP is solved, SCIP checks if any columns in the varpricer have negative reduced cost, adds a number of these to the RMP, and solves the RMP again. The pricing method implemented by the user is only called if no column is added by the varpricer. In the following, we refer to a column generation iteration in which the varpricer returns at least one column as a \emph{varpricer iteration}, while a \emph{user iteration} refers to an iteration in which the user's pricing method is called.

Removing all non-basic columns after the root node means that all non-basic columns, regardless of their age, are removed once solving of the root node is finished. Similar for removing all non-basic columns after any node other than the root node. Not using either of these two options, whilst also not taking advantage of the column age limit, means the LPs get fairly large and thus take a long time to solve. The advantage is that we never solve more than the necessary number of LPs. On the other hand, using both options is a very aggressive strategy that results in the LP being reduced to just $m$ variables at the beginning of every node, where $m$ is the number of constraints in the RMP. In this case, SCIP often performs many varpricer iterations before a user iteration. Furthermore, after columns were added by the user, SCIP may perform even more varpricer iterations before requesting additional columns from the user. Thus, even though each LP is easier to solve, a longer overall run time may result since many more LPs have to be solved. A
strategy in which all non-basic columns are removed after only some nodes has little effect as most columns were added in previous nodes and are therefore not permitted to be removed.

In preliminary experiments, we noted that a large number of user iterations are performed at the root node, while only few user iterations each were performed at other nodes. Thus, it seems appropriate to use a different age limit at the root node compared to other nodes.

We conducted experiments in which we varied the size of the column age limit between seven and 19, with increments of three. Table 6.12 shows the resulting run times for solving the root node using the different values of $\psi$.

<table>
<thead>
<tr>
<th>Instance</th>
<th>$\infty$</th>
<th>7</th>
<th>10</th>
<th>13</th>
<th>16</th>
<th>19</th>
</tr>
</thead>
<tbody>
<tr>
<td>5-S-1</td>
<td>363</td>
<td>444</td>
<td>356</td>
<td>361</td>
<td>369</td>
<td>331</td>
</tr>
<tr>
<td>5-S-2</td>
<td>448</td>
<td>574</td>
<td>362</td>
<td>397</td>
<td>357</td>
<td>467</td>
</tr>
<tr>
<td>5-L-1</td>
<td>1060</td>
<td>1417</td>
<td>914</td>
<td>805</td>
<td>919</td>
<td>959</td>
</tr>
<tr>
<td>5-L-2</td>
<td>1246</td>
<td>4342</td>
<td>1330</td>
<td>1351</td>
<td>901</td>
<td>880</td>
</tr>
<tr>
<td>3-S-A</td>
<td>670</td>
<td>614</td>
<td>657</td>
<td>608</td>
<td>604</td>
<td>656</td>
</tr>
<tr>
<td>3-L-A</td>
<td>572</td>
<td>455</td>
<td>480</td>
<td>499</td>
<td>499</td>
<td>496</td>
</tr>
</tbody>
</table>

Table 6.12: Root node solution times (s) when using different column age limits.

Compared to not removing columns ($\psi = \infty$), we observe an average increase of 50.66% for $\psi = 7$, and decreases of 7.71% ($\psi = 10$), 8.26% ($\psi = 13$), 13.69% ($\psi = 16$), and 9.78% ($\psi = 19$). The large increase for $\psi = 7$ is due to instance 5-L-2, which takes approximately 3.5 times as long to solve compared to not removing columns. The reason for this excessive run time is that SCIP carries out 3591 varpricer iterations between user iteration 203 and 204. This accounts for about half the run time, the remainder of the increase is due to the same issue happening many times over albeit at a much smaller scale.

However, even when excluding this instance, the increase is 11.13%. The overall run times are improved most when using an age limit of $\psi = 16$. We use this value for all subsequent experiments, including what follows in this section.
CHAPTER 6. A BRANCH-AND-PRICE ALGORITHM

To evaluate strategies for after the root node, we use several different settings:

Def: Default setting, in which we do not remove any columns at the end of or after the root node.

R/0/∞: Removing all non-basic columns at the end of the root node but not at other nodes and no removal due to ageing after the root node.

R/0/5: Removing all non-basic columns at the end of the root node but not at other nodes and setting an age limit of five after the root node.

R/A/ψ: Removing all non-basic columns at the end of the root node and after all other nodes and setting an age limit of ψ after the root node, where ψ ∈ {10, 13, 16, 19}.

We did not investigate age limits larger than five when removing all non-basic columns at the end of the root node but not at other nodes (setting R/0/ψ). Apart from user iterations, of which there are rarely more than four at each node, most varpricer iterations occur at the first few nodes after the root node (since all non-basic columns were removed after the root node). Larger values would therefore only have an effect at these initial nodes.

If, on the other hand, all non-basic columns are removed after every node, many more varpricer iterations result. In this case, we may want to use a larger age limit, one that is comparable to the limit used at the root node. We thus chose ψ ∈ {10, 13, 16, 19}.

The strategies investigated here have no effect on the root node. We therefore only present run times excluding the time spent at the root node in Table 6.13 for the different strategies. For R/0/∞, we observe an average reduction of 3.63% when compared to the default setting, while for R/0/5, we observe an increase of 4.71%. Regardless of the value of ψ, strategies R/A/ψ perform much worse for the instances containing a larger number of constraints (bottom four instances in the table). The increase is due to much larger overall LP solution times, resulting from a 56.88% to 65.52% increase in the number of solved LPs. Since the seven-day instances and the artificial five-day instances all contain even more constraints, we expect a similar behaviour for these instances.
6.6. Branching

In this section, we describe the branching scheme we developed to obtain integer solutions. We use follow-on branching and perform depth-first search without backtracking. In Section 6.6.1, we investigate how early branching can lead to large integrality gaps and show how this can be avoided by gradually tightening the early branching gap. This is followed by a section in which we develop a branching scheme that explicitly takes advantage of the replenishment structure in our problem. We investigate the effectiveness of this strategy and analyse why, for our problem, it is

<table>
<thead>
<tr>
<th>Inst.</th>
<th>R/0/∞</th>
<th>Def</th>
<th>R/0/5</th>
<th>R/A/10</th>
<th>R/A/13</th>
<th>R/A/16</th>
<th>R/A/19</th>
</tr>
</thead>
<tbody>
<tr>
<td>5-S-1</td>
<td>336.41</td>
<td>304.28</td>
<td>347.08</td>
<td>302.73</td>
<td>331.12</td>
<td>330.14</td>
<td>304.09</td>
</tr>
<tr>
<td>5-S-2</td>
<td>356.50</td>
<td>322.48</td>
<td>427.48</td>
<td>438.97</td>
<td>373.51</td>
<td>349.06</td>
<td>366.89</td>
</tr>
<tr>
<td>5-L-1</td>
<td>597.83</td>
<td>652.61</td>
<td>622.13</td>
<td>836.33</td>
<td>793.15</td>
<td>852.32</td>
<td>720.56</td>
</tr>
<tr>
<td>5-L-2</td>
<td>731.18</td>
<td>742.59</td>
<td>833.69</td>
<td>1059.94</td>
<td>998.52</td>
<td>891.78</td>
<td>881.57</td>
</tr>
<tr>
<td>3-S-A</td>
<td>686.02</td>
<td>665.23</td>
<td>650.83</td>
<td>1031.19</td>
<td>1024.36</td>
<td>995.13</td>
<td>985.49</td>
</tr>
<tr>
<td>3-L-A</td>
<td>649.00</td>
<td>581.59</td>
<td>598.50</td>
<td>920.06</td>
<td>938.81</td>
<td>883.30</td>
<td>951.69</td>
</tr>
</tbody>
</table>

Table 6.13: Run times (s) excluding time spent at the root node when using different column removal strategies.

In summary, due to the limitations of the SCIP framework, we remove columns at the root node that have been non-basic for 16 consecutive LP solves. After the root node, all non-basic columns are removed, regardless of their age. Thereafter, no columns are removed from the RMP at all.

If one is not using the SCIP framework, and thus has more control over column management, many other strategies are possible. For example columns could be removed based on the magnitude of positive reduced cost of non-basic columns as is done for example in (Grönkvist, 2005). Other options are to remove columns once a number of columns equal to a multiple of the number of constraints is exceeded or to always remove a percentage of all non-basic columns. Since implementing such strategies would involve significant modifications to the SCIP source code, we did not investigate them further.
not very efficient. Motivated by these findings, in the final section, we investigate the effect of suppressing the pricing step early on in the tree.

A branch-and-price algorithm obtains integer solutions by making branching decision based on fractional LP solutions and then carrying out additional column generation iterations at each node (Barnhart et al., 1998c). Many different algorithmic choices influence the performance of the algorithm. Perhaps the most important one is the choice of the branching decision. Branching on fractional columns in the RMP, as is done for example in (Cordeau et al., 2001) or (Mercier and Soumis, 2007), is a very natural choice. However, this is a very aggressive strategy which may cause infeasibility in deeper nodes because a large portion of the solution space is fixed with every branching decision. Furthermore, enforcing the 0-branch in the pricing problem is difficult as this often complicates the well structured pricing problem significantly (Vance, 1998).

Follow-on branching (Vance et al., 1997a), which is derived from the work of Ryan and Foster (1981), is a less aggressive branching scheme that, in its basic version, fixes a single connection between two flights. In either branch, this strategy is easily enforced in the pricing problem by removing appropriate arcs. All columns that are incompatible with this branching are removed from the master problem.

Due to the size of most airline scheduling problems, in practice, one is usually satisfied with finding a good solution quickly. Therefore, often only the 1-branch is explored and a depth-first search strategy with no or limited backtracking is employed. This branching scheme has been shown to be very successful for airline scheduling problems and is thus a widely used strategy (Gopalakrishnan and Johnson, 2005; Barnhart et al., 2003).

To obtain solutions even more quickly, it is possible to make several branching decisions at once. In these heuristic branch-and-price algorithms, all connections with a fractional value above a certain threshold are fixed to one (e.g. Mercier et al. (2005)). If none exists, only the connection with the highest fractional value is chosen. An even more aggressive strategy is to carry out a few rounds of branching on fractional columns early on in the tree and then revert to connection fixing above a threshold, as is done for example in (Grönkvist, 2005).

For our problem, preliminary results showed that the quality of integer solutions
obtained from only exploring the follow-on branch is generally very good even without backtracking. To avoid backtracking altogether, we have to be vigilant about branching decisions re-introducing artificial variables, especially late in the tree. We therefore chose a more conservative branching strategy, in which we employ follow-on branching, selecting the connection closest to one, but do not fix multiple connections at the same time. We always choose the branch in which the connection is fixed to one. We then terminate the algorithm once an integer solution is obtained.

To calculate the fractionality of every connection, we sum over all columns that contain the pair of flights that constitute the connection. We calculate these values for each aircraft and crew connection separately because we do not branch on aircraft and crew connections simultaneously. Formally, we define the fractionality of an aircraft connection \((i, j) \in \hat{C}\) as

\[
f_{ij}^A = \sum_{a \in A} \sum_{r \in R_a, (i, j) \in r} x^{ra},
\]

and the fractionality of a crew connection \((i, j) \in \hat{C}\) as

\[
f_{ij}^B = \sum_{b \in B} \sum_{p \in P_b, (i, j) \in p} y^{pb}.
\]

We then select the connection \((i, j) \in \hat{C}\) over all aircraft and crew connections for which \(f_{ij}^A\) or \(f_{ij}^B\) is closest but not equal to one. Depending on whether an aircraft or crew connection is chosen, we remove all arcs representing connections \((i, k) \in \hat{C}, k \neq j\) and \((k, j) \in \hat{C}, k \neq i\) from all pricing problems \(a \in A\) or \(b \in B\), respectively. In the RMP, all aircraft or crew columns, respectively, that contain \(i\) or \(j\) but not both are not feasible due to the branching decision; we say the columns are incompatible. All incompatible columns are fixed to zero (recall that we can’t actually remove them from the RMP because of the limitations of the SCIP framework).

Before presenting results and improving upon the branching scheme, we first make several general observations. Figure 6.3 shows the LP solution times and number of columns added in the first 100 iterations after the root node for instance 5-L-1. These are consecutive iterations that are not necessarily at the same node. We observe significant spikes whenever columns are added. The magnitude of these
spikes correlates to the number of columns added to a high degree. When many columns are added, the LP solver has to carry out a larger number of simplex iterations to regain LP optimality.

<table>
<thead>
<tr>
<th>Time (s)</th>
<th>Iteration Number</th>
<th>LP Time</th>
<th>Number of Columns Added</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>51</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>101</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>151</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>201</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>251</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>301</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>351</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>401</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>451</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>501</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>551</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>601</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>651</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 6.3: LP solution times and number of columns added per iteration in the first 100 iterations after the root node is solved for instance 5-L-1.

When considering the entire branch-and-price procedure, we see that the LP solution times also depend on when in the procedure an LP is solved. Figure 6.4 shows the LP times of the same run for all 664 iterations. These are consecutive iterations, which are scattered over all nodes. The first 266 are at the root node, whilst later iterations are deeper in the tree.

Figure 6.4: LP solution times in each iteration when solving instance 5-L-1.
6.6. BRANCHING

After an initial phase, we observe the largest LP times at the root node. During this phase many columns are added. As the procedure progresses, i.e. other nodes are explored, the average LP solution time decreases and fewer large spikes occur, except towards the end. This behaviour is mainly due to the branching decisions. The reduction in LP time is due to fewer compatible columns in the LP, effectively reducing the size of the LP. Toward the end of the procedure, relatively speaking, more new columns need to be generated because the branching decisions resulted in temporary infeasibility or a poor LP solution. Figure 6.5 displays the number of columns in the RMP and the number of incompatible columns. Columns representing pairings are mainly generated in the beginning. For aircraft columns, we observe that in this case every ARPP contributes the maximum number of columns at the root node except for the very end of the node. After the root node is solved, the increase is not as consistent or steep. For the number of incompatible columns, we observe the largest increase in the beginning of the tree (iterations 267 - 320). At deeper nodes, many branching decisions have already been made, resulting in fewer remaining compatible columns. New branching decisions thus have a smaller impact, at least with respect to the number of incompatible columns.

This general behaviour is hardly surprising. The magnitude of the LP solution times, however, do have implications for our algorithm. It seems prudent to not add many columns when the number of compatible columns is high. This is a key observation. In the remainder of this section, we will repeatedly return to this notion.

6.6.1 Early Branching

Column generation often suffers from the tailing-off effect (Lübbecke and Desrosiers, 2005). To prove LP-optimality, many similar columns are added which results in slow convergence of the LP solution value. In branch-and-price, LP optimality is not necessary, especially when we are content with finding a good instead of the optimal solution. We therefore branch as soon as the lower bound is close enough to the LP solution value (see Section 2.4). To calculate the lower bound in our problem, we need to consider the multitude of pricing problems and the fact that a crew block represents several crews. We define the lower bound as follows
\[ z_{LB} := z_{LP} + \sum_{a \in A} \bar{c}_a + \sum_{b \in B} \bar{c}_b \cdot n_b, \]  
\[ (6.1) \]

where \( z_{LP} \) is the current objective function value of the RMP, and \( \bar{c}_a \) and \( \bar{c}_b \) are the optimal negative reduced cost of aircraft routing pricing problem \( a \in A \) and crew pairing pricing problem \( b \in B \), respectively. Note that we multiply \( \bar{c}_b \) by \( n_b \), which is the number of crews in crew block \( b \). Representing identical work-periods of several crews in one crew block is equivalent to the symmetric case discussed in Section 2.1.3. In the following we will refer to \( z_{LB} \) as the local lower bound on \( z_{LP} \) as it is calculated at each iteration and thus node.

Since the local lower bound oscillates (see Section 2.3), we keep the maximum over all these bounds, called local dual bound \( z_{DB} \). For iteration \( v > 0 \), we have

\[ z_{DB}^v = \max\{z_{LB}^v, z_{DB}^{v-1}\}. \]

We then take a branching decision when the LP gap

\[ \text{LP gap} := \frac{z_{LP}^v - z_{DB}^{v-1}}{z_{DB}^{v-1}} < \epsilon, \]

\[ (6.2) \]
for some small value of $\epsilon$. It should be noted that in our algorithm, $z_{DB}^{\epsilon}$ is monotonically non-decreasing since no backtracking occurs. At this point we would like to make a distinction between local and global bounds. We use the term local dual bound (LDB) to refer to value $z_{DB}$ at the current node/iteration. In the first iteration of a node, $z_{DB}^{\epsilon - 1}$ is the local dual bound of the parent node. We use global dual bound (GDB) to refer to the smallest local dual bound $z_{GDB}$ among all open nodes. This value is used to calculate integrality gaps (IP gaps) for integer solutions as follows.

In branch-and-price, the local dual bound at the end of the root node becomes the local dual bound of the child nodes. Because we never backtrack in our algorithm, the not-selected child remains open and so the global dual bound $z_{GDB}$ does not improve. If we branch early at the root node, the final local dual bound at the root node may be significantly lower than the, unknown, optimal local dual bound of the root node. Using the weaker value for $z_{GDB}$ and thus to calculate IP gaps is technically correct but can be misleading in that it gives poor IP gaps although the solution quality may in fact be very good. To avoid this, we perform two separate runs. In the first, we abort the algorithm after solving the root node to optimality. This gives the optimal local dual bound of the root node, which we then use as $z_{GDB}$. In the second run, in which we perform early branching, we obtain an integer solution. We then evaluate the integer solution against the $z_{GDB}$ from the first run. It should be noted that the $z_{GDB}$ from the first run is not used in any form during the second run. In practice, if we do not need a measure for the solution quality, we do not need to carry out the first run.

The IP gaps that we report in this thesis are calculated as follows:

$$\text{IP gap} := \frac{z_{IP} - z_{GDB}}{z_{GDB}} \times 100,$$

where $z_{IP}$ is the objective function value of a given integer solution and, as described, $z_{GDB}$ is the optimal local dual bound of the root node, calculated in the first run.

We conducted experiments in which we tested several values for $\epsilon$. The behaviour of all instances was very similar. Thus, to lessen exposition of numerical results in this section, we only present averages. Table 6.14 gives the average changes in run times, the average IP gaps, the number of CG iterations, and the number of nodes in the branch-and-price tree, which, since we do not backtrack, is equal to its depth.
The reduction of run time is with respect to not branching early, i.e. $\epsilon = 0$. The averages are over test runs for the small instances.

<table>
<thead>
<tr>
<th>LP gap ($\epsilon$)</th>
<th>0.3</th>
<th>0.2</th>
<th>0.1</th>
<th>0.05</th>
<th>0.02</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run time (%)</td>
<td>-42.24</td>
<td>-41.36</td>
<td>-31.21</td>
<td>-28.70</td>
<td>-22.01</td>
<td>-</td>
</tr>
<tr>
<td>Avg. IP gap</td>
<td>7.92</td>
<td>4.42</td>
<td>3.13</td>
<td>1.95</td>
<td>0.86</td>
<td>0.46</td>
</tr>
<tr>
<td>Avg. Iter</td>
<td>275</td>
<td>315</td>
<td>379</td>
<td>400</td>
<td>384</td>
<td>431</td>
</tr>
<tr>
<td>Avg. Depth</td>
<td>400</td>
<td>374</td>
<td>350</td>
<td>301</td>
<td>246</td>
<td>144</td>
</tr>
</tbody>
</table>

Table 6.14: Average change in run time (%), average integrality gaps, average number of iterations, and average depth of the tree for different early branching values. The change in run time is with respect to not performing early branching.

We observe a significant average reduction in run times for large values of $\epsilon$. However, the resulting integrality gaps are unacceptable. In this thesis, we aim to achieve an average gap of 2%, thus we find gaps for $\epsilon > 0.05$ to be too large. The number of iterations is also reduced, whereas the number of nodes is increased considerably.

The reason for the large IP gaps is that the branching decisions are made based on fractional solutions of poor quality. Branching early means that we did not achieve a good LP solution yet, as evident by the higher LP solution values. Although there is no direct correlation between LP and IP solutions, the LP solution usually is somewhat indicative of the integer solution. A connection that has a high value in the LP solution is more likely to be part of an integer solution (Vance et al., 1997a). Consequently, a connection with a low fractional value, say 0.5, means that there is not much evidence that this connection should be part of an integer solution. Hence, branching on such a connection may result in a poor integer solution. In the following we refer to choosing a connection with a low fractional value as an inferior branching decision. Our experiments confirm this behaviour. Figure 6.6 shows the fractional value of the connection chosen to be branched on at each node for instance 3-S-A. For $\epsilon = 0.3$ (6.6b) the values are clearly worse, especially early on in the tree. We would like to point out that an LP gap of e.g. 30% does not
mean the LP solution value is 30% off of the optimal LP value. Usually the dual bound is, comparatively speaking, weak, meaning the true gap between the current LP value and the optimal LP value is more likely to be in the single percentages.

The poor quality of the fractional solutions that are used for branching are also the reason for the larger branch and bound tree when $\epsilon$ is large. As Figure 6.7 shows, the solutions are more fractional. Many more connections need to be fixed to one before fractionality is resolved.

We now turn our attention to the reduction in the number of CG iterations and run times. Naturally, at the root node, the number of iterations is smaller when branching early. For $\epsilon = 0.3$, the reduction is on average 35.46% compared to $\epsilon = 0$. Avoiding iterations at the end of the root node significantly reduces run times because at that point, solving an LP is computationally expensive as many compatible columns exist in the RMP. Additionally, we observe that the number of iterations after the root node is reduced by an average of 38.36%. More critical is, however, where in the tree these reductions occur. Early branching using the LP gap as defined above results in carrying out fewer iterations early on in the tree as follows. Due to the oscillating local lower bound, it is possible that when branching early, we achieve an LP gap that is in fact a fair amount less than $\epsilon$. In this case, a branching decision may not increase the LP solution value enough to result in an LP gap larger than $\epsilon$, resulting in further immediate branching, i.e. no CG iteration is carried out. The effect of this is that more branching decisions are made, and thus columns become incompatible, before new columns are generated. Then, adding new columns later in the tree results in comparatively lower LP solution times.

A promising strategy seems to be to combine the benefits of early branching using a large $\epsilon$ in the beginning with solving nodes to optimality later in the tree. This should avoid the tailing-off effect at the root node and reduce iterations early on in the tree while hopefully giving integer solutions of good quality. Determining factors in such a strategy are the initial gap and at what point to start solving to optimality. Instead of using a fixed point, we decided to gradually tighten the gap, thereby making branching decisions based on increasingly better LP solutions.

With this strategy several other questions remain. How should the gap be tightened and at what point should $\epsilon$ be driven to zero? We elected to linearly decrease
Figure 6.6: Number of fractional connections in the fractional solution at the end of each node in the branch-and-price tree. A connection is fractional if the sum over all columns that use this connection is larger than zero and smaller than one.
the gap and conduct experiments to identify an appropriate point at which the gap is zero. For this we introduce a value $\bar{D}$, which is a function of the number of flights in the instance, i.e. its size\(^1\). In our experiments, we reduced the gap to zero at depth $\chi \times \bar{D}$, where we varied $\chi \in \{0.25, 0.5, 0.75, 1\}$ and combined this with $\epsilon \in \{0.02, 0.05, 0.1, 0.2, 0.3\}$. As reference, we also solved without early branching, i.e. $\epsilon = 0$. Naturally, no tightening is required. These experiments produced a large amount of data. We therefore only present average IP gaps (Table 6.15) and the average reduction of run times in percent (Table 6.16).

We observe that the earlier we tighten (small values of $\chi$), the lower the IP gaps. Similarly, the smaller the initial value of $\epsilon$, the lower the gaps. Not surprisingly, the best gaps are achieved when both values are small, i.e. in the lower right corner of the table.

For the average run times, we observe significant reductions for all settings. The reduction is at least 20\%, which is mainly due to the decreased number of iterations at the root node. The reduction is lower for smaller $\epsilon$ and $\chi$ values, although not as consistent as for integrality gaps.

\(^1\)In all our experiments, value $\bar{D}$ was between 0.7 and 0.9 of the maximum depth of the branch-and-price tree. It should be noted that even if $\bar{D}$ is larger than the maximum depth, the method is still correct, it simply means that $\epsilon$ has not been driven to zero yet, quite possibly resulting in larger IP gaps (see above).
<table>
<thead>
<tr>
<th>$\epsilon$</th>
<th>0.3</th>
<th>0.2</th>
<th>0.1</th>
<th>0.05</th>
<th>0.02</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\chi$</td>
<td>1</td>
<td>3.55</td>
<td>2.29</td>
<td>1.61</td>
<td>b0.73</td>
<td>0.64</td>
</tr>
<tr>
<td>0.75</td>
<td>c2.10</td>
<td>1.86</td>
<td>1.38</td>
<td>0.67</td>
<td>0.59</td>
<td>-</td>
</tr>
<tr>
<td>0.5</td>
<td>7.99</td>
<td>1.53</td>
<td>1.00</td>
<td>a0.53</td>
<td>0.56</td>
<td>-</td>
</tr>
<tr>
<td>0.25</td>
<td>0.84</td>
<td>1.00</td>
<td>0.84</td>
<td>0.52</td>
<td>0.57</td>
<td>-</td>
</tr>
<tr>
<td>0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0.46</td>
</tr>
</tbody>
</table>

Table 6.15: Average integrality gaps when tightening early branching gaps.

<table>
<thead>
<tr>
<th>$\epsilon$</th>
<th>0.3</th>
<th>0.2</th>
<th>0.1</th>
<th>0.05</th>
<th>0.02</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\chi$</td>
<td>1</td>
<td>-43.96</td>
<td>-38.28</td>
<td>-29.65</td>
<td>b-30.86</td>
</tr>
<tr>
<td>0.75</td>
<td>c-43.59</td>
<td>-29.21</td>
<td>-31.63</td>
<td>-27.76</td>
<td>-26.48</td>
</tr>
<tr>
<td>0.5</td>
<td>-38.31</td>
<td>-31.87</td>
<td>-28.06</td>
<td>a-28.06</td>
<td>-26.05</td>
</tr>
</tbody>
</table>

Table 6.16: Average reduction in run time compared to solving all nodes to optimality when tightening early branching gaps.
6.6. BRANCHING

To select the best values for $\epsilon$ and $\chi$, we plot the average reduction in run time against the average integrality gaps for all settings in Figure 6.8. It is possible to identify several Pareto optimal strategies. These should be chosen based on the decision maker’s preferences. For example, an airline may select strategies a or b since they generate near optimal solutions (IP gaps of 0.53 and 0.73%, respectively) and yield a good reduction in run times. For the sake of this thesis, we elect strategy c as this roughly satisfies our requirements of an integrality gap of 2% while also providing the second largest reduction in run time.

![Figure 6.8: Relationship of average IP gaps and reduction in run time for different combinations of early branching values and tightening depths.](image)

To analyse and verify the behaviour of the procedure, we consider Figure 6.9, which displays the development of $z_{LP}$ and $z_{DB}$ after the root node for instance 3-S-A. Again, these iterations are over all nodes other than the root node, not necessarily at a single node. We observe that in the first half of the iterations, spikes in the LP value occur. Artificial variables are introduced, which is mainly due to the inferior branching decisions. Except for these spikes, $z_{LP}$ decreases, which is expected since $\epsilon$ is reduced as the depth increases. The increasing quality of the LP solution in the second half of the graph leads to not introducing any artificial variables after a branching decision. This cannot be guaranteed but we observed this behaviour in almost all our experiments. In iteration 96, the depth in the tree
is reached at which $\epsilon = 0$. At this point, the node must be solved to optimality for the first time, which is achieved within two iterations. Hence, the local dual bound is driven to equal to $z_{LP}$.

Figure 6.9: Behaviour of the LP value and the local dual bound over all iterations after the root node for instance 3-S-A.

Another experiment we performed in an attempt to reduce IP gaps was to allow backtracking instead of LP gap tightening. After exploring the 1-branch, we allowed backtracking to occur. We did not change the search strategy, i.e. continued with depth-first-search. The reason is that in say a breadth-first search strategy, a node high in the tree is selected, which would require a lot of branching decisions, and thus time, to reach another integer solution. The goal here was to spent only little extra time in backtracking; we opted for an additional 10% of the current run time. Of course, the cause of the large IP gaps may be higher up in the tree, which means we cannot revisit these decisions in such a strategy. In general, we achieved a slight reduction of IP gaps but this proved to be insufficient.

6.6.2 Branching on Replenishments

In this section, we present a variant of follow-on branching in which we focus on selecting connections during which replenishments can occur. The general idea is to
multiply the fractional values of such connections by a factor, while the fractional values of non-replenishment connections are not multiplied. We then select the connection with the highest resulting value. Since the replenishments are distributed throughout the network, we achieve a better distribution of branching decisions, i.e. fixed connections, in the network. In what follows, we give proper motivation for this idea. We present numerical results which show that this branching scheme obtains integer solutions in fewer branching steps, although, due to an increase in the number of CG iterations, takes longer to do so.

One of the key features of our problem is the existence of replenishments in the network. Pairings often have several replenishments as crews require layovers almost every night. Aircraft routes, on the other hand, have fewer replenishments. The majority here are MOPPs, which have to occur on average every two days, and, to a much lesser extent, SLIMs, which only few aircraft require, at most once during the planning horizon. A nice property of the replenishments is that, due to the temporal aspect, they are distributed regularly throughout a path in the network.

Let us consider for a moment what happens in follow-on branching. When a connection \((i, j)\) was set to one and another branching decisions has to be made, it is more likely that a connection terminating at \(i\) or a connection leaving \(j\) has a value equal to or close to one as well. If close to one, they may be chosen for branching next. The issue then is that this merely extends the same path and thus does not reduce the solution space by much. For example, if after \((i, j)\) connection \((j, k)\) is chosen, all other paths still may use any connection not involving \(i, j,\) or \(k\). Fixing parts of a different path, or even fixing a different part of the same path, is more beneficial as it reduces the solution space to a larger extend.

The idea in this branching scheme then is to focus on the replenishments and in doing so, hopefully fix more impactful connections, thus limiting the choices for the solver.

Another argument for focusing on replenishments is that while in our problem the cost of routes and pairings depend on the resource accumulation in the routes and pairings, the costs are in fact set by the replenishment arcs used in the paths representing the routes and pairings. In the ARPP, costs are only incurred along replenishments arcs that represent a SLIM. For pairings, credit is accumulated throughout
the duties but is only transformed to cost along layover connections. Furthermore, layovers are the only connections that have a fixed cost associated to them. From this point of view, branching on replenishments means that we focus on the aspect of the problem that is most impactful on the solution value, i.e. solution quality.

In this branching scheme, we use a variation of follow-on branching. If a connection represents a replenishment, we multiply $f_{ij}^A$ and $f_{ij}^B$ by a factor $\xi^A$ and $\xi^B$, respectively. For non-replenishment connections, $f_{ij}^A$ and $f_{ij}^B$ are not multiplied. Then, instead of simply selecting the connection for which $f_{ij}^A$ or $f_{ij}^B$ is closest to one, we select the connection that has the largest resulting value, where the original $f_{ij}^A$ or $f_{ij}^B$ is fractional. Depending on the values of $\xi^A$ and $\xi^B$, we can control how much emphasis we place on the replenishments, thereby allowing regular connections to be branched on as well.

It should be noted that we do not directly branch on replenishment arcs, but instead, on the connections representing the arcs. In crew pairing pricing problems the distinction is irrelevant as each connection is represented by exactly one arc. For aircraft routing pricing problems, on the other hand, this distinction is important.

Let us consider what the 1-branch in general follow-on branching represents for our problem: branching on connection $(i,j)$ means that if flight $i$ is chosen for an aircraft, the aircraft then has to connect to flight $j$. Now, if instead the branching decision is on a arc $(i,j,g)$, then, if flight $i$ is chosen for an aircraft, the aircraft has to connect to $j$ and activity $g$ is carried out between flights $i$ and $j$. Activity $g$ may represent a set of specific maintenances $m \in M_{ijg}$. As was described above, a branching decision will be enforced in all ARPPs. This means that for every aircraft, any route that covers flight $i$ will schedule all maintenance checks $m \in M_{ijg}$, regardless of whether the aircraft requires the checks or not. In contrast, by branching only on the representing connection, the pricing problem solver can decide for each individual aircraft whether and which maintenance checks to schedule or to simply connect from $i$ to $j$.

For crew pairing, this branching scheme can be seen as first locating good layover connections and then filling in the duties between the layovers. This is quite contrary

\[^2\text{In reality the maintenance checks would of course not be carried out or even entered in the maintenance schedule. In our solution process though, “scheduling” has an impact as it incurs cost and thus may lead to sub-optimal solutions.}\]
to the idea used in duty networks, in which duties are formed first (often a-priori) and then are concatenated to form pairings by selecting layovers (see Section 3.5).

We conducted experiments to test the effectiveness of this branching scheme. We varied the emphasis on replenishments but considered aircraft and crew to be equally important, thus $\xi^A = \xi^B$. To lessen exposition of numerical results, we only present averages. Table 6.17 shows the average change in percent of several performance measures. We provide the change in run times excluding the root node, overall LP time including the root node, nodes in the tree, and iterations after the root node. Furthermore, the table gives resulting average IP gaps. The comparisons are with respect to $\xi^A = \xi^B = 1$

<table>
<thead>
<tr>
<th>$\xi^A, \xi^B$</th>
<th>1.05</th>
<th>1.1</th>
<th>1.2</th>
<th>1.3</th>
<th>1.5</th>
<th>1.75</th>
<th>2.0</th>
<th>2.5</th>
<th>3.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>10.53</td>
<td>1.72</td>
<td>18.62</td>
<td>29.76</td>
<td>60.50</td>
<td>96.01</td>
<td>100.85</td>
<td>113.88</td>
<td>115.93</td>
</tr>
<tr>
<td>LP time</td>
<td>6.27</td>
<td>1.96</td>
<td>9.04</td>
<td>16.22</td>
<td>33.01</td>
<td>52.39</td>
<td>55.82</td>
<td>63.94</td>
<td>65.16</td>
</tr>
<tr>
<td>Iter.</td>
<td>15.41</td>
<td>1.96</td>
<td>9.42</td>
<td>20.97</td>
<td>12.35</td>
<td>26.84</td>
<td>25.86</td>
<td>29.35</td>
<td>26.66</td>
</tr>
<tr>
<td>IP gap</td>
<td>1.52</td>
<td>1.15</td>
<td>1.38</td>
<td>1.62</td>
<td>1.96</td>
<td>2.74</td>
<td>2.50</td>
<td>3.29</td>
<td>3.31</td>
</tr>
</tbody>
</table>

Table 6.17: Average difference (%) in run time excluding the root node, LP time including the root node, iterations after the root node, and number of nodes, i.e. depth of tree. The table also shows average IP gaps. The differences are with respect to $\xi^A = \xi^B = 1$.

The average run time excluding the root node more than doubles for large values of $\xi^A$ and $\xi^B$, while the average number of iterations increases by up to 30%. Furthermore, we observe average IP gaps of up to 3.31%. The average number of nodes in the tree, however, decreases. The increases are consistent with our previous findings. Multiplying the fractional values results in not choosing the connection with the highest fractional value, which may result in temporarily increasing the LP value by a larger amount, especially since costs are set by these connections as they represent replenishments. This in general requires more immediate iterations to again improve $z_{LP}$. Moreover, it often increases $z_{LP}$ such that the LP gap is larger than $\epsilon$. 

meaning we have to carry out an iteration instead of branching early. Thus, more iterations are required. More importantly, these are mostly early in the branch-and-price tree (when otherwise early branching would occur), thereby substantially increasing LP solution times and hence overall run times. The increase of up to 65.16% of the overall LP time supports this argument. It should be noted that this time includes the time spent solving LPs at the root node, which is constant for different values of $\xi^A$ and $\xi^B$ as they have no effect on the root node. Thus, the relative increase of the time spend solving LPs after the root node is most likely much higher. Generally, in our algorithm about half of the overall run time is spent at the root node. Assuming this ratio holds for LP solution times as well, we get an increase of LP times after the root node that is twice of what the table shows - and matches that of the overall run times excluding the root node.

The larger final IP gaps are explained by an observation we also made in the previous section: branching several times on connections for which the fractional values are not close to one results in weaker IP gaps. A lower fractional value means there is less evidence that this connection should be part of the integer solution.

The reduction in the number of nodes shows that this branching scheme is in fact effective at finding an integer solution more easily. By distributing the branched connections in the network, fewer branching decisions are required. It seems that the solver is able to find good integer subpaths between the replenishments more easily.

In conclusion, while finding integer solutions in fewer branching decisions, the branching scheme is not very efficient as it results in significantly increased run times. However, because the increase is mainly due to longer LP solution times, we believe that this scheme is beneficial in problems where the LP times are not as critical as in our case. In the remainder of this thesis, apart from the next section, we do not branch with an emphasis on replenishments, i.e. we use $\xi^A = \xi^B = 1$.

As concluding remarks, we would like to point out that further experiments are recommended to identify how much of the reduction in the number of nodes is due to the spatial distribution of the replenishments and how much is due to the cost being associated with the replenishments, and, additionally, do not depend on resources. We propose an experiment in which the costs of the problem are associated with
all connections, not just replenishments. Furthermore, it would be interesting to evaluate the effect of placing a higher emphasis on crew replenishments compared to aircraft replenishments and vice versa. We did not investigate this due to the discouraging LP solution times.

6.6.3 Suppressing the Pricing Step

The previous section introduced a branching scheme that emphasises branching on follow-ons that are replenishments. The branching scheme obtains integer solutions in fewer branching steps but is not efficient for our problem because of the long LP solution times resulting from carrying out more CG iterations early on in the tree. Motivated by this, in the present section, we investigate the effect of suppressing the pricing step early in the tree.

As before, we branch early to avoid the tailing off-effect at the root node. When the LP gap at the root node is below 0.3, we abort solving the root node and start branching. We then suppress pricing until a depth of \( \omega \times \chi \times \bar{D} \), where \( \chi = 0.75 \) and \( \bar{D} \) as defined in Section 6.6.1. We conduct experiments to evaluate different values of \( \omega \).

The LP solution value may deteriorate significantly if we branch several times without adding new columns. We therefore allow pricing to occur under certain circumstances. Pricing is permitted if the current \( z_{LP} \) is more than 2\% larger than the final \( z_{LP} \) of the parent node. This allows improving \( z_{LP} \) if the latest branching decision resulted in more than a small deterioration of the LP solution value. Additionally, we allow pricing if the current \( z_{LP} \) is more than 8\% larger than the final \( z_{LP} \) at the root node. This represents an upper bound on the LP solution quality we are willing to accept. It can also be seen as allowing pricing if the cumulative effect of several poor branching decisions resulted in a more substantial deterioration of the solution quality.

We conducted experiments in which we varied \( \omega \in \{ \frac{1}{3}, \frac{1}{2}, \frac{2}{3} \} \). Additionally, we investigated the effect of suppressing pricing when branching with an emphasis on replenishments. We first discuss results for \( \xi^A = \xi^B = 1 \), followed by \( \xi^A = \xi^B = 1.5 \) and \( \xi^A = \xi^B = 2 \).

Table 6.18 shows the change in run times excluding the root node in percent for
ξ^A = ξ^B = 1. The reduction is with respect to not suppressing the pricing step. We see a clear decrease for \( \omega = \frac{1}{2} \) and especially \( \omega = \frac{2}{3} \). It should be noted that this reduction is in addition to the reduction resulting from early branching (see Section 6.6.1).

<table>
<thead>
<tr>
<th>Instance</th>
<th>( \frac{1}{3} )</th>
<th>( \frac{1}{2} )</th>
<th>( \frac{2}{3} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>5-S-1</td>
<td>-34.61</td>
<td>-47.49</td>
<td>-61.59</td>
</tr>
<tr>
<td>5-S-2</td>
<td>21.51</td>
<td>-19.90</td>
<td>-38.47</td>
</tr>
<tr>
<td>5-L-1</td>
<td>20.02</td>
<td>-26.04</td>
<td>-30.58</td>
</tr>
<tr>
<td>5-L-2</td>
<td>-19.91</td>
<td>-28.63</td>
<td>-51.45</td>
</tr>
<tr>
<td>3-S-A</td>
<td>-20.89</td>
<td>-27.50</td>
<td>-31.46</td>
</tr>
<tr>
<td>3-L-A</td>
<td>0.04</td>
<td>-20.60</td>
<td>-24.12</td>
</tr>
</tbody>
</table>

Table 6.18: Change in run times in percent for different values of \( \omega \) compared to not suppressing the pricing step.

The difference in performance can be explained by considering what happens immediately after the suppression phase ends. Because the LP solution value deteriorated, many iterations will be carried out at this point. For \( \omega = \frac{1}{3} \), this happens at a depth at which the RMP still contains many compatible columns. Carrying out the iterations at this point is computationally expensive. We would like to point out that we likely exacerbated this issue since we tightened the \( \epsilon \) value as before. This means that at depth \( \frac{1}{3} \times 0.75 \times \tilde{D} \), \( \epsilon \) is equal to 0.2. Using \( \epsilon = 0.3 \) and tightening from here would require fewer iterations immediately following the suppression phase. However, during the suppression phase, we are likely to make several inferior branching decisions. We therefore want to increase LP solution quality quickly. We thus tighten throughout even though this has no effect during the suppression phase. Interestingly, for \( \omega = \frac{2}{3} \), \( \epsilon = 0.1 \), which means even more iterations are carried out following the suppression phase, the run times are reduced to a much larger extent. This again shows the importance of the number of incompatible columns; many more columns are incompatible at a depth of \( \frac{2}{3} \times 0.75 \times \tilde{D} \).

When branching on replenishments, i.e. \( \xi^A = \xi^B = 1.5 \) or \( \xi^A = \xi^B = 2 \), we
observe much smaller reductions. Table 6.19 shows the average reduction of run time excluding the root node in percent. The reduction is in comparison to the respective setting in which pricing is not suppressed.

<table>
<thead>
<tr>
<th>ω</th>
<th>Multiplier</th>
<th>(\frac{1}{3})</th>
<th>(\frac{1}{2})</th>
<th>(\frac{2}{3})</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\xi_A = \xi_B = 1.0)</td>
<td>-5.64</td>
<td>-28.36</td>
<td>-39.61</td>
<td></td>
</tr>
<tr>
<td>(\xi_A = \xi_B = 1.5)</td>
<td>-8.67</td>
<td>-15.50</td>
<td>-22.93</td>
<td></td>
</tr>
<tr>
<td>(\xi_A = \xi_B = 2.0)</td>
<td>-8.01</td>
<td>-15.00</td>
<td>-15.56</td>
<td></td>
</tr>
</tbody>
</table>

Table 6.19: Reduction in run times in percent for different values of \(\omega\) compared to not suppressing the pricing step.

As Table 6.20 shows, branching on replenishments results in more exceptions, i.e. CG iterations, during the suppression phase. This is time consuming, hence the smaller reductions. More exceptions are required because, again, inferior branching decisions are made and because fixing connections representing replenishments leads to larger increases of \(z_{LP}\) since costs are associated with these replenishments.

<table>
<thead>
<tr>
<th>ω</th>
<th>Multiplier</th>
<th>(\frac{1}{3})</th>
<th>(\frac{1}{2})</th>
<th>(\frac{2}{3})</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\xi_A = \xi_B = 1.0)</td>
<td>1</td>
<td>4</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>(\xi_A = \xi_B = 1.5)</td>
<td>15</td>
<td>32</td>
<td>43</td>
<td></td>
</tr>
<tr>
<td>(\xi_A = \xi_B = 2.0)</td>
<td>18</td>
<td>67</td>
<td>104</td>
<td></td>
</tr>
</tbody>
</table>

Table 6.20: Average number of CG iterations that were allowed despite suppressing pricing for different values of \(\xi^A\) and \(\xi^B\).

When suppressing the pricing step, we expect the IP gaps to be worse because despite allowing pricing in certain cases, we are more likely to use inferior LP solutions when making the branching decisions. Somewhat surprisingly, this did not occur. Table 6.21 shows the IP gaps for different values of \(\omega\) and when not suppressing the pricing step. As can be seen, there is no trend towards worse IP gaps.

We conclude that sufficient time is available after the suppression phase to gen-
A more aggressive strategy like fixing several connections above a certain threshold, as was discussed in the beginning of this main section, may be appropriate. On the other hand, evaluating each individual branching decision by re-solving the RMP even if no new columns were added, likely has a positive impact. Additional experiments are required to evaluate if a more aggressive branching scheme, that does not allow backtracking, is appropriate before making a final verdict.

Many other strategies for suppressing the pricing step are possible. A promising strategy is to take advantage of the decreasing number of columns that become incompatible due to the new branching decision. Later on in the tree, a branching decision will have a smaller impact since many columns are already incompatible due to previous branching decisions. We then could abort suppressing the pricing step if this number is smaller than a certain threshold. The number of new incompatible columns can be somewhat erratic, it is thus advisable to consider a moving average over the last few branching decisions. Naturally, the number of columns that become incompatible depends on the number of columns in the RMP and thus instance size. To avoid this dependency, we need to relate the moving average of the number of columns that become incompatible to the total number of columns in the RMP. We

<table>
<thead>
<tr>
<th>Instance</th>
<th>No suppr.</th>
<th>$\frac{1}{3}$</th>
<th>$\frac{1}{2}$</th>
<th>$\frac{2}{3}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>5-S-1</td>
<td>1.63</td>
<td>1.10</td>
<td>1.19</td>
<td>0.78</td>
</tr>
<tr>
<td>5-S-2</td>
<td>1.14</td>
<td>1.85</td>
<td>1.82</td>
<td>1.93</td>
</tr>
<tr>
<td>5-L-1</td>
<td>0.75</td>
<td>0.62</td>
<td>0.53</td>
<td>0.54</td>
</tr>
<tr>
<td>5-L-2</td>
<td>1.15</td>
<td>1.00</td>
<td>1.13</td>
<td>0.74</td>
</tr>
<tr>
<td>3-S-A</td>
<td>1.08</td>
<td>0.64</td>
<td>0.82</td>
<td>0.95</td>
</tr>
<tr>
<td>3-L-A</td>
<td>0.74</td>
<td>0.90</td>
<td>0.69</td>
<td>0.58</td>
</tr>
</tbody>
</table>

Table 6.21: IP gaps (%) for different values of $\omega$ compared to not suppressing the pricing step.
did not investigate this idea since the previously described strategy works well.

Another variation of our branching scheme we experimented with is to focus on aircraft connections, regardless of whether the connection represents a replenishment or not. Since the aircraft aspect of the problem causes the long LP solution times, increasing the number of incompatible aircraft columns more rapidly is a promising strategy. We conducted experiments in which we placed varying emphasis on branching on aircraft connections. In doing so, we observed an average reduction of up to 12.44% for run times excluding the root node. However, when combining this idea with LP gap tightening and suppressing the pricing step, the positive effect was lost.

In conclusion, suppressing the pricing step early in the branching tree is beneficial. We observed a 39.61% decrease in run time for \( \omega = \frac{2}{3} \) while the integer solution quality did not suffer. We therefore use this setting for all subsequent experiments.

6.7 Summary

We investigated the impact on the run times and number of CG iterations when using different LP solvers. We found that the interior point and dual simplex method performed poorly with respect to time. At the root node, primal simplex and the SCIP default setting, which may chose either of the simplex methods, performed equally well. After the root node though, the default setting outperformed always invoking the primal simplex method.

The LPs frequently suffered from numerical trouble. This increases LP solution times as CPLEX has to undertake countermeasures. To avoid excessive solution times of individual LPs, we take advantage of SCIP’s ability to ignore numerical instabilities. As a result, we cannot guarantee that a node is solved to optimality.

We analysed the cause of the numerical difficulties and long LP solution times by separately considering the crew and aircraft aspect of the problem. We found that the aircraft part is clearly the culprit of these challenges. The main difference between the crew and aircraft aspect is that routes have a much larger number of non-zero entries than pairings (up to 22 entries compared to just 8).

The RMP needs to be initialised at the beginning of the CG procedure. We developed a heuristic that generates disjoint and non-disjoint columns that have a
cost comparable to columns generated in the main algorithm. Furthermore, these columns cover a number of flights similar to those generated later. In a second step, we generate additional columns that contain previously uncovered flights. This initialisation method reduces the time to solve the root node on average by 10.75%.

To evaluate how many columns to add per iteration, we conducted experiments in which we set limits per pricing problem and iteration. We observed significant reductions in the number of CG iterations when adding more than one column per pricing problem. For larger limits, the solution times increased due to the growing size of the RMP.

The size of the restricted master problem increases quickly as columns are added. However, not all columns are necessary at all times. In removing columns we are restricted by the limitations of the SCIP framework. We therefore remove columns at the root node that have been non-basic for 16 consecutive LP solves. After the root node, all non-basic columns are removed, regardless of their age. Thereafter, no further columns are removed from the RMP. SCIP stores all removed columns and re-inserts them once they have negative reduced columns.

To obtain integer solutions, we use follow-on branching and only explore the 1-branch without backtracking. We investigated how early branching can lead to large integrality gaps and show how this can be avoided by gradually tightening the early branching gap. We found that the most beneficial strategy is to use an initial early branching gap of $\epsilon = 0.3$ and then drive this value to zero at depth $0.75 \times \bar{D}$, where $\bar{D}$ is a function of the size of the instance.

We develop a branching scheme that explicitly takes advantage of the existence of replenishments in our problem. Our experiments showed that this branching strategy reduces the number of branch-and-bound nodes needed in a dive but, due to the challenges in solving LPs, is inefficient in our case. We therefore do not use this strategy in our algorithm.

The long LP solution times due to adding columns, lead us to investigate the effect of suppressing the pricing step early on in the branching tree. We observed a 39.61% decrease in run time excluding the root node when suppressing pricing until a depth of $\frac{1}{2}\bar{D}$. In this scheme, we allow pricing if the LP solution value deteriorated by more than 2% compared to the parent node or more than 8% compared to the
root node. The integer solution quality did not suffer, which suggests that enough time is available to rectify the effects of the inferior branching decisions during the suppression phase.

We would like to acknowledge that we did not investigate all acceleration strategies that have been proposed in the literature. One of the main goals of this thesis is to develop strategies that can cope with the large number of pricing problems that results from modeling individual tail numbers and crew blocks. Our work therefore focusses on improving pricing problem solution times and on the interaction between the restricted master problem and the pricing problems. We envision that further performance improvements can be made, especially with respect to LP solution times and convergence of the procedure. In the past decade, significant improvements have been made in these areas by using dual value stabilisation and constraint aggregation, see Section 2.3.

After discussing the majority of the aspects of our algorithm, we now formally state the branch-and-price algorithm (6.2).
Algorithm 6.2: Branch–And–Price Algorithm

**Input:** An instance of the integrated aircraft routing, crew pairing, and tail assignment problem

**Output:** A solution $\bar{x}$, representing a schedule for all aircraft and crew that are part of the instance.

1. **Initialization**
   1.1 $v := 0$; // Iteration counter
   1.2 Finished = false, $z_{DB}^0 = -\infty$;
   1.3 Read data;
   1.4 Calculate $\bar{D}$; // Used to adjust early branching gap $\epsilon$.
   1.5 Run procedure PREPROCESSING with respect to all resources $\kappa \in K$ for all aircraft $a \in A$; // See Section 7.1.2
   1.6 Run Primal2 to initialise the RMP;

2. **Main Loop**
   2.1 while Finished = false do

3. **Enforce Branching Decisions**
   3.1 Enforce branching decisions in the RMP;
   3.2 $\epsilon = \max(0.3(1 - \frac{\text{CurrentDepth}}{0.75D}), 0)$; // Adjust early branching gap

4. **Start Column Generation**
   4.1 while do
   4.2 Solve RMP to get $\bar{x}$ and $z_{LP}$;
   4.3 if $\frac{z_{LP}^v - z_{DB}^{v-1}}{z_{DB}^0} < \epsilon$ then break; // Early branching?
   4.4 if Suppress pricing? then break; // See Section 6.6.3
   4.5 $v = v + 1$;
   4.6 $\Pi := \emptyset$;
   4.7 Select pricing problems to solve; // See Chapter 8
   4.8 Set dual values, enforce branching decisions in PPs, and solve selected PPs to obtain newly generated columns $\Pi$; // See Chapter 7
   4.9 Calculate $z_{DB}^v$; // Update bounds
   4.10 if $\Pi \neq \emptyset$ then add $\Pi$ to RMP;
   4.11 else break; // Solved node to optimality
   4.12 Calculate $f^A_{ij}$ and $f^B_{ij}$ from $\bar{x}$;
   4.13 Select connection (i,j) with value $f^A_{ij}$ or $f^B_{ij}$ closest to 1.; // Branching
   4.14 if no fractional connection exists then Finished = true;

5. **Return**
   5.1 return $\bar{x}$;
Chapter 7

Solving the Pricing Problems

In this chapter, we give a detailed description of the solution algorithm that we use to solve the pricing problems (Section 7.1). We conducted extensive numerical experiments, especially for the preprocessing procedure described in this chapter. The results of these experiments are presented in Section 7.2.

7.1 Pricing Problem Solver

In this section, we develop a solution algorithm that is suitable for solving the pricing problems defined in Sections 5.2.1 and 5.2.2. We extend the RCSPP-R as proposed by (Smith, 2011) to handle multi-arcs, a non-additive resource consumption, and cost that increases in non-linear fashion and depends on a resource. These extensions are a result of modeling the maintenance and crew rules described in Sections 4.2.1 and 4.2.2: multi-arcs result from the existence of parallel maintenance arcs, while the non-additive resource consumption is due to the accumulation of block and duty hours over consecutive duties. The cost of a duty depends on the amount of credit accumulated in the duty, which is modeled using a separate resource. Furthermore, a minimum value on the credit accumulation exists, which may result in a cost increase that is not linear in the credit associated with the flights that make up the duty. We refer to our airline related extension of the RCSPP-R as RCSPP-RAE.

In the following subsections, we first describe a labelling algorithm for the RCSPP-RAE and then give a preprocessing procedure that accelerates the labelling algorithm and often finds a feasible, if not the optimal, path in the network. In each
section, we define appropriate notation and discuss resource extension functions for all resources that are required to correctly model the rules considered in the pricing problems.

7.1.1 Labelling Algorithm

In this section, we describe the labelling algorithm that we use to solve the RCSPP-RAE. A general description of labelling algorithms can be found in (Desrochers and Soumis, 1988). We first formally define a label and then give resource extension functions (REF) for the cost and resource components of a label. We conclude the section by formally stating the labelling algorithm. The algorithm is very similar for the ARPP and the CPPP, the only difference being the resource extension functions. We therefore describe the algorithm in general terms and indicate which REF applies to what type of pricing problem whenever appropriate.

Let \( q \) be a path in network \( G = (\hat{N}, \hat{E}) \).\(^1\) Path \( q \) can be expressed as a sequence of nodes \( q = (\eta_1, \ldots, \eta_n) \), where \( \eta_\iota \in \hat{N}, \ \forall \iota = 1,\ldots,n \) and \( (\eta_\iota, \eta_{\iota+1}, g') \in \hat{E}, \ \exists g' \in G_{\eta_\iota,\eta_{\iota+1}}, \ \forall \iota = 1,\ldots,n-1 \). Let \( \vec{q}_i \) denote a forward path from source node \( s \in \hat{N} \) to node \( i \in \hat{N} \), i.e. \( \eta_1 = s \) and \( \eta_n = i \).

In a labelling algorithm several paths \( \vec{q}_i \) are generated, each represented by a label. The path that a label \( L \) at node \( i \) represents has cost \( \vec{c}_L \) and a resource consumption of \( \vec{w}_k^i \), \( k \in K \), associated with it, where \( K \) is the set of resources. While \( \vec{c}_L \) represents the cost since the beginning of the path, values \( \vec{w}_k^i \) represent the accumulation of resource \( k \) since the last replenishment arc of \( k \) in this path. Recall that we denoted replenishment arcs for resource \( k \) by \( \hat{E}_k \subset \hat{E} \). Formally, we define label \( L \) as a \((|K| + 1)\)-tuple \( L = (\vec{c}_L, \vec{w}_1^1, \ldots, \vec{w}_k^{|K|}) \).

**Forward Resource Extension Functions**

Unlike in the RCSPP-R in (Smith, 2011), which required only one resource extension function, we require a total of four different resource extension functions to extend the cost and resource components of a label. In accordance with the nomenclature in (Irnich, 2008), we use the term resource extension function for a cost extension

---

\(^1\)The majority of notation used in this section is defined in Section 5.2. A comprehensive list of mathematical notation used in this thesis can be found on page 369.
function as well. For an ARPP, we have one extension function, REF1, that extends cost and one resource extension function, REF2, that applies to all resources \( k \in K \) in ARPP. The CPPP requires a more complex resource extension function, REF3, for cost and two resource extension functions for resources. The first applies to resources \( k = \{1, 2, 5, 6\} \) and is identical to REF2. The second, REF4, applies to resources \( k = \{3, 4\} \), which represent the block and duty hour accumulation over consecutive duties.

In the following, we first discuss REFs for the ARPP, followed by those of the CPPP. We will use examples to illustrate complicated REFs. In these examples, we sometimes need to identify the cost and resource accumulation at a predecessor node along the path that label \( L \) represents. We therefore express REFs in general terms: given the cost \( \overrightarrow{c}_i \) of some path to node \( i \) and the resource consumption \( \overrightarrow{w}_k^i \), \( k \in K \) since the last replenishment of \( k \) along this path, what are these values when extending the path to node \( j \)?

For the ARPP, \( \overrightarrow{c}_i \) is extended by adding to it the cost of the arc along which the path is extended. We define REF1 as follows:

\[
\overrightarrow{f}_{ijg}(\overrightarrow{c}_i) = \overrightarrow{c}_i + c_{ijg}, \quad \forall (i, j, g) \in \hat{E}.
\] (7.1)

The extension of resources \( k \in K \) in ARPP depends on whether arc \((i, j, g) \in \hat{E} \) is a replenishment arc for \( k \) or not. For a non-replenishment arc, we add the usage of \( k \) on the arc to the current accumulation of resource \( k \). If, on the other hand, the arc is a replenishment arc for \( k \), the resource accumulation is set to equal just the usage of the arc itself. Recall that the arc resets the current accumulation (i.e. the replenishment) but also represents the usage of node \( j \), which technically occurs after the replenishment. We thus define REF2 to be

\[
\overrightarrow{f}_{ijg}(\overrightarrow{w}_k^i) = \begin{cases} 
\overrightarrow{w}_k^i + u_{ijg} & \text{if } (i, j, g) \in \hat{E} \setminus \hat{E}^k, \\
u_{ijg} & \text{if } (i, j, g) \in \hat{E}^k,
\end{cases}
\] (7.2)

where the first case is extending along a non-replenishment arc, while the second case represents extending along a replenishment arc.

Strictly speaking, REF2 is not entirely accurate for the aircraft routing pricing problem. Consider an arc on which two maintenance checks are carried out in sequence. Then the second check does not occur immediately after flight \( i \). If the
indicator is time between checks, the corresponding resource does accumulate and may in fact exceed the allowed limit. However, this is irrelevant since exceeding the limit in this context only means that no more flying may occur. This is satisfied because the aircraft is on the ground at a maintenance station and thus does not need to fly anywhere to carry out the second check.

Recall that the cost of a crew pairing depends on two cost components. The first is the cost of the connections used in the pairing, the second the accumulation of credit in the duties. For every duty in the pairing, the credit accumulation is compared to the minimum credit value $\zeta$, and the larger of the two is added to the cost (see Section 4.2.2). In a labelling algorithm we model this by using resource $k = 6$ to count the credit accumulation in the current duty. When the duty ends, i.e. extending along a layover or to sink node $t$, the value is compared to $\zeta$ and the greater value is added to the cost. After this, resource $k = 6$ is reset via REF2 to count the credit accumulation in the next duty. Thus, for cost in CPPP, we define REF3 as follows:

$$f_{ijg}(\overrightarrow{c_i}, \overrightarrow{w_i^6}) = \begin{cases} \overrightarrow{c_i} + c_{ijg} & \text{if } (i,j,g) \in \hat{E} \setminus \hat{E}^6, \\ \overrightarrow{c_i} + c_{ijg} + \max(\zeta, \overrightarrow{w_i^6}) & \text{if } (i,j,g) \in \hat{E}^6. \end{cases} \quad (7.3)$$

As was mentioned above, in CPPP resources $k = \{1, 2, 5, 6\}$ are extend via REF2. Resources $k = \{3, 4\}$, on the other hand, require a separate REF as their accumulation in some cases depends on the accumulation of resources $k = 1$ and $k = 2$, respectively. Before defining the REF, we review the rules resources 3 and 4 relate to (see Section 4.2.2 and Table 5.5) and give two examples that illustrate how the rules are modeled.

Rule 3 stipulates that if the total block hours in two consecutive duties exceeds a value $MB2$ and the layover between the two duties is not a long layover, then the pairing has to end or a long layover is required immediately following the two duties. Rule 4 requires the same for duty hours; the limit here is $MD2$. In the following we will only use Rule 3 to explain how these rules are modeled. All conclusions drawn from Rule 3 apply to Rule 4 similarly.

As an illustration of how Rule 3 is modeled, consider Figure 7.1 which shows a path with two duties. The first contains two duty connections (DC), the second,
which so far is a partial duty, starts with a regular layover (RL)\(^2\) and further contains a duty connection from node 3 to node 4. In the figure, the number below each arc is the block hour usage along the arc, which, according to our definition, is the usage of the following flight. In this example let MB2 = 6. A crew operating the two duties will have accumulated a total of seven block hours once it operated flight 4, i.e. reached node 4. The accumulation thus exceeded MB2. Rule 3 then requires that if the duty finishes, the pairing must finish as well, i.e. connection (4, \(t\)) is used, or that the crew has a long layover (LL), i.e. connection (4, 5). A regular layover is not sufficient, thus the crew cannot use connection (4, 6). It should be noted that the rule does not actually require that the duty must end as soon as MB2 is exceeded. It only specifies the type of rest that is required after the two consecutive duties. Therefore, in the absence of other rules, the crew can continue the second duty by using duty connection (4, 7). We would like to stress this point: the limit MB2 is a “soft limit”. It may be exceeded, in which case eventually an extended rest, i.e. long layover, is required or the pairing has to end.

Figure 7.1: Illustration of Rule 3. A crew working the two duties is required to either finish the pairing (connect to node \(t\)), have a long layover (connect to node 5), or continue the duty (node 7). It must not, however, have a regular layover (connect to node 6).

\(^2\)Technically the majority of a layover is not part of any duty as a layover is always between duties. Only the briefing period contained in the layover is part of the duty following the layover. However, due to our definition of arcs, we say that the arc representing the layover is the beginning of the duty.
Recall that Rule 3 contains the clause “and the layover between the two duties is not a long layover”. In other words, if a long layover exists between two consecutive duties, Rule 3 does not apply to this pair of duties. Therefore, in terms of a labelling algorithm, we have to reset the accumulation at the long layover so that we can count the accumulation in the next pair of consecutive duties. If for example in Figure 7.1, the regular layover between flight 2 and flight 3 is a long layover instead, the crew only accumulated four block hours once it finished operating flight 4. It thus would be allowed to have a regular layover, i.e. connection (4, 6).

To count the block hour accumulation in the last two consecutive duties, we use resource $k = 3$. Formally, we define $\overrightarrow{w}^3_i$ to be the block hour accumulation in the previous duty plus the accumulation in the current duty up to and including node $i$ (thus the last two duties), unless the layover between the previous duty and the current duty is a long layover, in which case $\overrightarrow{w}^3_i$ is only the accumulation in the current duty up to and including node $i$. Expressed differently, $\overrightarrow{w}^3_i$ is the accumulation of block hours since the second last layover or since the last long layover, whichever is less.$^3$

To illustrate how extending $\overrightarrow{w}^3_i$ is modeled, we consider the example in Figure 7.2, in which a partial path consists of three duties. In this example let us simultaneously calculate the block hour accumulation in a single duty, which is modeled using resource $k = 1$, and the accumulation in consecutive duties, i.e. $\overrightarrow{w}^3_i$.

The path starts with a long layover so any block hour accumulation prior to the long layover is irrelevant as both values were reset. Therefore, at node 3, we have $\overrightarrow{w}^1_3 = \overrightarrow{w}^3_3 = 1$. A duty connection simply increases both values, thus $\overrightarrow{w}^1_4 = 2$ and $\overrightarrow{w}^3_4 = 2$. The regular layover (4, 5) resets $\overrightarrow{w}^1_5$ to one because it ends the previous duty. Value $\overrightarrow{w}^3_5$ however is not reset because “since the last long layover or since the second last layover, whichever is less” means that both cases are equal and refer to connection (2, 3). Therefore the value is simply increased, i.e. $\overrightarrow{w}^3_5 = 3$. Further along we have $\overrightarrow{w}^1_6 = 2$ and $\overrightarrow{w}^3_6 = 4$. When extending along regular layover (6, 7), again $\overrightarrow{w}^1_7$ is reset to 1. However, $\overrightarrow{w}^3_7 = 3$ because now the clause “since the second

---

$^3$ Three cases are possible. If the last layover is a LL, then “since the last long layover” is obviously less than “since the second last layover”. If the last layover is a RL and the second last is a LL, then the two values are equal. If the last and the second last layover are RLs, then “since the second last layover” is less than “since the last long layover.”
7.1. PRICING PROBLEM SOLVER

Figure 7.2: A partial pairing consisting of three duties, which are separated by layover connections. The example is used to illustrate how the accumulation of a resource over consecutive duties is modeled.

last layover” is more restrictive than “since the last long layover”: the last layover is (6, 7), while the second last layover is (4, 5). So in this case, the accumulation at node 7 is equal to the current accumulation in the duty node 7 is part of plus the accumulation of the previous duty, i.e. Duty 2. The accumulation in the current duty is just the usage on (6, 7), whereas the accumulation of block hours in Duty 2 is precisely $\bar{w}_{ij}^3$. We therefore set $\bar{w}_{ij}^3 = \bar{w}_{ij}^1 + u_{ij}^3 = 2 + 1$.

Then formally, for CPPP, resources $k \in \{3, 4\}$ are extended according to REF4:

$$\tilde{f}_{ijg}^k(\bar{w}_{ij}^k, \bar{w}_{ij}^{k'}) = \begin{cases} \bar{w}_{ij}^k + u_{ijg}^k & \text{if } (i,j,g) \in \hat{E} \setminus \hat{E}^k, \\ \bar{w}_{ij}^{k'} + u_{ijg}^k & \text{if } (i,j,g) \in \hat{E}^k \text{ and } (i,j) \in C_{RL}, \\ u_{ijg}^k & \text{if } (i,j,g) \in \hat{E}^k \text{ and } (i,j) \in C_{LL}, \end{cases} \quad (7.4)$$

where $k'$ is the resource that accumulates block or duty hours for a single duty, respectively. So for $k = 3, k' = 1$ and for $k = 4, k' = 2$. Note that, as discussed earlier, if $\bar{w}_{ij}^k$ exceeds its soft limit (MB2 if $k = 3$ or MD2 if $k = 4$), then extension along an arc $(i,j,g) \in C_{RL}$ is not permitted under Rule 3, so the middle clause in (7.4) is only applied in the case $\bar{w}_{ij}^k$ does not exceed its corresponding soft limit.

The Labelling Procedure

The labelling algorithm maintains a list of non-dominated labels at each node and, according to the REFs discussed above, extends these labels along arcs to form new labels. A label $L$ is said to dominate label $\Gamma$ if both labels are at the same node...
and $\overrightarrow{c}_L \leq \overrightarrow{c}_\Gamma$ and $\overrightarrow{w}_k^L \leq \overrightarrow{w}_k^\Gamma$, $\forall k \in K$, where one of these inequalities has to be strictly less.

Let the set of non-dominated labels at node $i$ be denoted by $L_i$. The order in which labels in set $L_i$ are stored should facilitate quick insertion of new labels and reduce the number of times dominance has to be checked. We therefore keep $L_i$ in lexicographical order. A label $L$ is lexicographically less than a label $\Gamma$, $L \prec \Gamma$, if $\overrightarrow{c}_L < \overrightarrow{c}_\Gamma$ or, if $\overrightarrow{c}_L = \overrightarrow{c}_\Gamma$, then $\overrightarrow{w}_k^L = \overrightarrow{w}_k^\Gamma$, $\forall k = 1, \ldots, \bar{k}$ and $\overrightarrow{w}_{\bar{k}+1}^L < \overrightarrow{w}_{\bar{k}+1}^\Gamma$ for some $\bar{k} \in K$. Lexicographical ordering is a total ordering, i.e. either $L \prec \Gamma$ or $\Gamma \prec L$. Lexicographical ordering facilitates dominance checking because only labels lexicographically smaller than $L$ can dominate $L$ and only labels lexicographically larger than $L$ can be dominated by $L$.

The performance of a labelling algorithm largely depends on the number of labels that have to be generated. This in turn depends on the label treatment order. In (Smith et al., 2012), the authors investigate the performance of their labelling algorithm for the RCSPP-R when using different label treatment orders:

- choosing a node based on topological node ordering,
- choosing the node that has the minimum/maximum cost/weight label,
- and choosing the global minimum/maximum cost/weight label.

In treatment orders that choose a node, all labels at the node are extended before moving on to another node. Global orders choose the best label from a list that contains all existing labels. This label then is extended and possibly added before selecting the next label from the updated list.

The advantage of global orders is that a good label is extended, possibly reducing the number of labels that are generated. However, the authors argue that due to the replenishments, identifying a good label is not straightforward. Consider for example the minimum cost label. A low cost may mean that the accumulation of resources is high because no replenishments (which incurs cost) has occurred yet. Extending the path may result in replenishment cost further along the path. The instances tested in (Smith et al., 2012) only had one resource. In our instances several resources exist, which makes identifying a good candidate label even more difficult. The disadvantage of global orders is that storing and keeping the global list
in order requires additional computational effort as this list is in addition to every set $\mathcal{L}_i$. Smith et al. (2012) found that the additional computational effort offsets any benefits from extending a good label.

Choosing a node based on the minimum/maximum cost/weight label does not require an additional list. However, nodes and thus labels may be revisited, which requires additional effort. As with global orders, identifying a good label is not straightforward.

In (Smith et al., 2012), the most beneficial label treatment order for acyclic instances with positive and negative arc cost is based on topological node ordering. The advantage in such an ordering is that choosing a label is trivial and that no label will need to be re-visited once it has been treated. Due to the column generation formulation, we do encounter positive and negative arc cost in the pricing problems. Moreover, multiple resources exist, which suggests that a topological node ordering will be most beneficial for our problems. We therefore selected this strategy.

The labelling algorithm that we use to solve the RCSPP-RAE is given in Algorithm 7.1. Labels at a node $i$ are treated according to the lexicographical ordering of $\mathcal{L}_i$. A label $L \in \mathcal{L}_i$ is extended along every outgoing arc $(i, j, g) \in G_{ij}$, possibly generating a new label $\Gamma$ at node $j$.

In the CPPP, we have to check feasibility with respect to Rule 3 and 4 before extending a label. We are not permitted to extend along a regular layover if limit $U^k, k \in \{3, 4\}$ is exceeded and the last layover was a regular layover (line 7.1.7). It should be noted that all listed conditions have to be met to violate the respective rule. It is for example permitted to extend along a regular layover even if $\overline{\omega}_L^k > U^k$, as long as the last layover is a long layover, since in this case, Rule 3 and 4 do not apply. This illustrates that in order to check feasibility with respect to Rule 3 and 4, we need to know the type of layover of the last layover. The algorithm stores this information with every label.

If a label is extended, the components of the new label $\Gamma$ are calculated according to REF1, REF2, REF3, and REF4. In lines 7.1.11 through 7.1.12, we check feasibility with respect to all limits except for resources $k \in \{3, 4\}$ if the problem is a CPPP (since these are checked before extending a label, see above).

We differentiate between extending to a flight node $j \in \mathcal{N}$ (line 7.1.13) and
extending to the sink node \( t \) (line 7.1.16). For flight nodes, the algorithm stores only non-dominated labels, while for the sink node, it stores all labels that have negative reduced cost. Not checking dominance at the sink node will result in more columns that can be added to the master problem while only increasing the computational effort by very little if at all (more labels need to be added to the lexicographical set \( \mathcal{L}_t \) but no time is spend checking dominance).

**Algorithm 7.1:** Labelling algorithm for the RCSPP-RAE

**Input:** An instance \( G \) of type RCSPP-RAE.

**Output:** Set of negative cost paths including least cost path in \( G \).

7.1.1 \( \mathcal{L}_s = \{(0,0,0,\ldots,0)\} \);

7.1.2 \( \mathcal{L}_i = \emptyset, \forall i \in \hat{N} \setminus \{s\} \);

7.1.3 \( c^+ = 0 \);

7.1.4 for \( i \) in topological \( \hat{N} \) do

7.1.5 | for each label \( L \in \mathcal{L}_i \) do

7.1.6 | | for each arc \((i,j,g) \in \hat{E}\) do

7.1.7 | | if Instance is of type CPPP and \( \overrightarrow{w}_k^L > U^k \), \( k \in \{3,4\} \) and last layover was a regular layover and \((i,j) \in C_{RL}\) then

7.1.8 | | Do not extend label \( L \); // Violating Rule 3 or 4

7.1.9 | else

7.1.10 | | Extend \( L \) along \((i,j,g)\), giving label \( \Gamma \) at node \( j \);

7.1.11 | | if Instance is of type CPPP and \( \overrightarrow{w}_k^\Gamma \leq U^k \), \( \forall k \in \{1,2,5\} \) or

7.1.12 | | if Instance is of type ARPP and \( \overrightarrow{w}_k^\Gamma \leq U^k \), \( \forall k \in K \) then

7.1.13 | | if \( j \neq t \) and \( \Gamma \) is not dominated by any label in \( \mathcal{L}_j \) then

7.1.14 | | | \( \mathcal{L}_j := \mathcal{L}_j \cup \{\Gamma\} \);

7.1.15 | | Remove all labels from \( \mathcal{L}_j \) dominated by \( \Gamma \);

7.1.16 | | if \( j = t \) and \( \overrightarrow{c}_\Gamma < 0 \) then

7.1.17 | | | \( \mathcal{L}_t := \mathcal{L}_t \cup \{\Gamma\} \);

7.1.18 | | if \( \overrightarrow{c}_\Gamma < c^+ \) then \( c^+ := \overrightarrow{c}_\Gamma \);

7.1.19 return \( c^+ \) and all paths corresponding to labels in \( \mathcal{L}(t) \);

The algorithm stores \( c^+ \), the cost of the incumbent solution. The use of \( c^+ \) will
become more apparent when considering preprocessing information to accelerate the labelling algorithm (see Section 7.1.3).

Upon termination, the labelling algorithm returns set $L_t$. If the set is non-empty, we choose the best or a limited number (see Section 6.4) of paths and add them as columns to the master problem. We need to calculate their true cost as opposed to reduced cost as these values are the objective function coefficients in the master problem. If the pricing problem is a CPPP, the true cost of a path is calculated using REF3, while for ARPP, the true cost is calculated according to REF1, where in both cases the cost of the arcs $c_{ijg}$ in the path is taken to be either zero or equal to the layover cost or maintenance cost, respectively. In other words, the dual values are ignored.

### 7.1.2 Preprocessing

Preprocessing aims at reducing the computational effort required to find the optimal path in a network. Dumitrescu and Boland (2003) showed that preprocessing can be very effective for single resource constrained shortest path problems. In their work, nodes and arcs are removed if it can be proven that including the node or arc in a path results in an infeasible path or in a path that is inferior to an incumbent solution. Additionally, they show that the performance of a labelling algorithm can be improved by considering preprocessing information when extending a label.

These preprocessing methods have been developed further by Smith (2011) to handle multiple resources and resource replenishments. The algorithm presented therein solves the RCSPP-R, which has linear cost and an additive resource extension function. Irnich (2008) describes several resource extension functions and shows that even non-traditional REFs can be used in preprocessing. Due to the extension made in the RCSPP-RAE, we must take care when applying the above preprocessing ideas to our problem.

We first give a high level description of the preprocessing procedure. The algorithm is called for each pricing problem separately and iterates over the following three steps:

1. Several shortest path problems are solved to generate lower bounds on cost.
and resource consumption to reach every node \( i \in N \) from the source node \( s \) and to reach the sink node \( t \) from every node \( i \). By doing so, the algorithm generates forward paths from \( s \) to \( i \) and backwards paths from \( t \) to \( i \).

2. Forward paths and backward paths are concatenated to form \( s-t \) paths, which must be checked for feasibility.

3. Nodes and arcs are removed if it can be shown that including them results in an infeasible path or in a path with a cost that is not lower than that of the best \( s-t \) path found so far.

Before describing the three steps in more detail, we define notation and describe additional resource extension functions. Again resources \( k = 3 \) and \( k = 4 \) and the minimum credit value in the CPPP require more sophisticated REFs.

**Notation**

Let \( \overrightarrow{q}_{0i} \) be the minimum cost path from \( s \) to \( i \) and \( \overrightarrow{q}_{\kappa i} \) be the minimum weight path with respect to \( \kappa \in K \) from \( s \) to \( i \). The latter paths are not the paths with the smallest accumulation of \( \kappa \) since the beginning of the path but instead, are the paths with the smallest accumulation since the last replenishment of \( \kappa \). In the following, we will refer to these paths as \( \kappa \)-paths. For convenience of notation, let us define set \( \hat{K} = K \cup \{0\} \), where “0” indicates with respect to cost.

When generating a forward \( \kappa \)-path, \( \overrightarrow{q}_{\kappa i}, \kappa \in \hat{K} \), we keep track of all (other) resources \( k \in K, k \neq \kappa \) as well. Let \( \overrightarrow{w}_{ki}^k \) be the accumulation of resource \( k \in K \) since the last replenishment of \( k \) along the \( \kappa \)-path from \( s \) to \( i \).

When we generate a \( \kappa \)-path, only the resource limit on \( \kappa \in K \) is considered. Thus, the \( \kappa \)-path is feasible with respect to resource \( \kappa \) but may be infeasible overall since all other resource limits are ignored. We keep track of whether the path is infeasible due to any resource \( k \in K, k \neq \kappa \) at any point in time, in which case indicator \( \overrightarrow{v}_{ki}^\kappa \) is set to TRUE.

Then, value \( \overrightarrow{w}_{ki}^\kappa \) constitutes a lower bound on the accumulation of \( \kappa \) since the last replenishment of \( \kappa \) of any path from \( s \) to \( i \). In other words, any path that reaches \( i \) from \( s \) has accumulated at least \( \overrightarrow{w}_{ki}^\kappa \) of resource \( \kappa \).

Every path has a cost associated with it. Let the cost of each forward path
For the ARPP, the cost of the minimum cost path to \( i \), i.e. \( \overrightarrow{c}_{0i} \), is a lower bound on the cost of any path from \( s \) to \( i \). However, for the CPPP it is not, which will be explained below. Instead, we define a separate lower bound value \( \overrightarrow{c}_{0i}^* \). To be consistent, we use this notation to indicate the lower bound on cost in the ARPP as well, here \( \overrightarrow{c}_{0i} = \overrightarrow{c}_{0i}^* \).

The preprocessing method calculates similar paths and values from every \( i \) to \( t \). These paths are generated in a backward algorithm that starts at \( t \) and extends paths to every \( i \). Let these \emph{backward \( \kappa \)-paths} be \( \overleftarrow{q}_{ki} \), \( \kappa \in \hat{K} \). A more natural, i.e. forward, interpretation of a backward path is to view it as a path from node \( i \) to sink node \( t \), where the path starts with the arc leaving \( i \) but does not contain node \( i \) itself.

Similar to the forward case, we define values \( \overleftarrow{w}_{ki} \) as the accumulation of resource \( k \in K \) since the last replenishment of \( k \) along the \( \kappa \)-path from \( t \) to \( i \), \( \kappa \in \hat{K} \). In the forward interpretation, the clause “accumulation of resource \( k \) since the last replenishment of \( k \)” can be understood as the amount of resource \( k \) that is consumed between the current node \( i \) and the next replenishment of \( k \) along the path to \( t \). Since node \( i \) is not part of the backward path, the resource consumption of \( i \) is not included in \( \overleftarrow{w}_{ki} \). As for the forward calculations, we do not consider any resource limits other than for resource \( \kappa \in K \), hence we need an indicator \( \overleftarrow{v}_{ki} \) that records if the backward \( \kappa \)-path is infeasible with respect to a resource other than \( \kappa \) at any point during the path. Additionally, we define cost values \( \overrightarrow{c}_{ki} \), \( \kappa \in \hat{K} \) and lower bounds on cost \( \overrightarrow{c}_{0i}^* \).

Then, \( \overleftarrow{w}_{ki}^{\kappa} \) is the minimum accumulation of resource \( \kappa \in K \) that will be incurred to reach the nearest replenishment of \( \kappa \) along \emph{any} path from \( i \) to \( t \) (recall that the network is acyclic and directed). Furthermore, extending any forward path \( \overrightarrow{q}_{0i} \) further towards \( t \) will incur an additional cost of at least \( \overrightarrow{c}_{0i}^* \) before \( t \) is reached.

### Resource Extension Functions in Preprocessing

In Section 7.1.1, we defined several resource extension functions that describe how a label is extended along an arc. A label is nothing more than a forward path, thus we can use the REFs described there to generate the forward \( \kappa \)-paths \( \overrightarrow{q}_{ki} \), \( k \in K \). However, we require one additional REF to generate the minimum cost paths \( \overrightarrow{q}_{0i} \)
in the CPPP. We first give an example to illustrate why we cannot simply use the regular cost function \( \text{REF3} \) when calculating \( \vec{q}_{0i} \) and thus \( \vec{c}_{0i}^* \).

The issue arises from dealing with a maximum function when calculating the cost associated with credit values. For the example in Figure 7.3, let us calculate the cost of two paths according to \( \text{REF3} \) and thereby show how an incorrect lower bound results. For this example, let us assume a minimum credit value of \( \zeta = 3 \). Let a minimum cost path to node \( n \) have incurred a cost \( \vec{c}_{0n} = 1 \) and an accumulation of credit \( \vec{w}_{0n}^6 = 4 \). In the figure, the first number in the box is the cost of the path to the node, the second the accumulation of resource 6. We also have a path to node \( m \), which has a cost \( \vec{c}_{0m} = 2 \) and credit accumulation \( \vec{w}_{0m}^6 = 1 \). Both paths are extended to node \( i \). For the first path, we get \( \vec{c}_{0i} = 2 \) and \( \vec{w}_{0i}^6 = 5 \), while for the second path we have \( \vec{c}_{0i} = 3 \) and \( \vec{w}_{0i}^6 = 2 \). Connection \((i,j)\) is a layover, thus the credit accumulation must be converted to actual cost. We get \( \vec{c}_{0j} = \vec{c}_{0i} + c_{ijg} + \max(\zeta, \vec{w}_{0i}^6) = 2 + 1 + \max(3,5) = 8 \) for the first path and \( \vec{c}_{0j} = 3 + 1 + \max(3,2) = 7 \) for the second path. Therefore, the lower bound on cost for node \( j \) is seven. However, in a shortest path algorithm that is with respect to cost, a path is dominated if its cost is higher than that of another path. Thus, at node \( i \), the second path is dominated by the first path. It is discarded and is therefore not available for extension to node \( j \), which incorrectly results in a lower bound of eight instead of seven. We therefore cannot simply check dominance with respect to the current cost but have to factor in the credit values as well.

This could be addressed either by using 2-dimensional labels and retaining all that are not dominated, which would increase complexity of the problem, or finding an alternative valid lower bound that can be calculated with only a standard shortest path method. We have elected to do the latter, as follows. The cost of a pairing \( q \) is equal to the cost of the arcs in the pairing plus the accumulation of credit in the pairing, i.e.

\[
\begin{align*}
c_q &= \sum_{(i,j,g) \in q} c_{ijg} + \sum_{(i,j,g) \in q} w_{ijg}^6,
\end{align*}
\]

(7.5)

unless the accumulation of credit in an individual duty is less than the minimum credit value \( \zeta \), in which case \( \zeta \) is added instead of the credit accumulation of that

\footnote{Recall that resource \( k = 6 \) represents the accumulation of credit.}
Figure 7.3: The figure depicts a small network that illustrates how an incorrect lower bound on the cost to reach node $j$ results. The credit accumulation, which is converted to cost along layover connections, must be compared to the minimum credit value. The larger of the two is added to the current cost.
duty. Therefore, it should be obvious that (7.5) is a lower bound on cost. Thus, in
terms of resource extension functions, to obtain a lower bound on total cost at node
\( j \), we immediately convert \( u^6_{ijg} \) to cost and add this value and the regular cost of
the arc, \( c_{ijg} \), to the lower bound on cost at node \( i \), which we previously defined as \( \overrightarrow{c}^*_{0i} \). Formally, we have the following resource extension function \( \text{REF5} \):

\[
\overrightarrow{f}_{ijg}(\overrightarrow{c}^*_{0i}) = \overrightarrow{c}^*_{0i} + c_{ijg} + u^6_{ijg} \quad \forall \,(i,j,g) \in \hat{E}.
\]  

(7.6)

This REF is only used in the CPPP. The shortest path problem to generate the
minimum cost paths \( \overrightarrow{q}_{0i} \) is then with respect to \( \overrightarrow{c}^*_{0i} \) and not \( \overrightarrow{c}_{0i} \). However, we do
need to calculate \( \overrightarrow{c}_{0i} \) as well because later on, we require the actual cost of the path,
not just the lower bounds. As before, \( \overrightarrow{c}_{0i} \) is calculated according to \( \text{REF3} \).

In the above example, by using \( \text{REF5} \) to calculate lower bounds on cost, we have
\( \overrightarrow{c}^*_{0n} = 5 \) and \( \overrightarrow{c}^*_{0m} = 3 \). For node \( i \), we get \( \overrightarrow{c}^*_{0i} = 7 \) for the first path and \( \overrightarrow{c}^*_{0i} = 5 \) for
the second. Now the second path correctly dominates the first and thus is available
for extension to node \( j \), for which we get \( \overrightarrow{c}^*_{0j} = 7 \). It should be noted that \( \overrightarrow{c}^*_{0i} \) may
be larger than \( \overrightarrow{c}_{0i} \). This is correct since it already reflects the cost due to credit,
which \( \overrightarrow{c}_{0i} \) does not.

For the ARPP, \( \overrightarrow{c}^*_{0i} \) and \( \overrightarrow{c}_{0i} \) are equal and are both extended using \( \text{REF1} \).

In the following, we describe all resource extension functions that are used for
extension of backwards paths. We omit the first subscript, the \( \kappa \), since the REFs are
the same for all \( \kappa \)-paths, except for \( \overrightarrow{c}^*_{0i} \). We first describe the REF that extends
the cost in the ARPP, followed by the REF that extends all resources in the ARPP and
resources \( k = \{1, 2, 5, 6\} \) in the CPPP. The complicating cases in the CPPP, namely
extension of cost, extension of lower bounds on cost, and extension of resources
\( k = \{3, 4\} \), follow thereafter.

In the ARPP, the cost of the arc is simply added to the current accumulation of
cost. We therefore define \( \text{REF6} \) as

\[
\overrightarrow{f}_{ijg}(\overrightarrow{c}_j) = \overrightarrow{c}_j + c_{ijg}, \quad \forall \,(i,j,g) \in \hat{E}.
\]  

(7.7)

Recall that in the forward interpretation of a backward path, \( \overrightarrow{w}^k_i \) is the amount
of resource \( k \) that is consumed when going from node \( i \) to the next replenishment
arc of \( k \) along this backward path. Thus, when extending from \( j \) to \( i \) and \( (i,j,g) \) is a
replenishment arc of \( k \), \( \overrightarrow{w}^k_i \) is set to zero because no amount of resource \( k \) is required.
7.1. PRICING PROBLEM SOLVER

to get to the next replenishment of \( k \) since that replenishment occurs immediately
after the node. For non-replenishment arcs, we simply add the arc weight to the
previous accumulation. Thus, for all resources \( k \in K \) in the ARPP and resources
\( k = \{1, 2, 5, 6\} \) in the CPPP we define REF7 as follows:

\[
\overline{f}_{ijg}^{k}(\overline{w}_{j}^{k}) = \begin{cases} 
\overline{w}_{j}^{k} + u_{ijg}^{k} & \text{if } (i, j, g) \in \hat{E} \setminus \hat{E}^{k}, \\
0 & \text{if } (i, j, g) \in \hat{E}^{k}.
\end{cases}
\]  \hspace{1cm} (7.8)

In forward paths, credit cost is added to regular cost when extending along arcs
that are in set \( \hat{E}^{6} \). This set contains all sink arcs, i.e. \( E^{-} \), and all arcs that represent
layover connections, i.e. connections in set \( C_{L} \). In backward paths, however, a duty
ends along layover connections and when extending to the source node \( s \), i.e. \( E^{+} \)
but not along sink arcs as these in fact start the backward arcs. Thus, in backward
paths, arcs \((i, j, g) \in E^{+} \cup \hat{E}^{6} \setminus E^{-} \) end a duty. A further complication is that
when extending backwards along such an arc \((i, j, g) \in E^{+} \cup \hat{E}^{6} \setminus E^{-} \), we have to
be vigilant about the credit usage of flight \( j \). This flight is part of the duty, but its
credit, \( u_{ijg}^{6} \), is not reflected in \( \overline{w}_{j}^{6} \). Thus when comparing the minimum credit value
to the credit accumulation in the duty, we need to compare \( \zeta \) against the sum of \( \overline{w}_{j}^{6} \)
and \( u_{ijg}^{6} \). Formally, for backward paths, the cost in CPPP is extended by REF8:

\[
\overline{f}_{ijg}(\overline{c}_{j}, \overline{w}_{j}^{6}) = \begin{cases} 
\overline{c}_{j} + \overline{c}_{ijg} + \max(\zeta, u_{ijg}^{6} + \overline{w}_{j}^{6}) & \text{if } (i, j, g) \in E^{+} \cup \hat{E}^{6} \setminus E^{-}, \\
\overline{c}_{j} + \overline{c}_{ijg} & \text{otherwise}.
\end{cases}
\]  \hspace{1cm} (7.9)

As in the forward case, the maximum function may result in incorrect lower
bounds on cost. Again, we elect to solve a standard shortest path problem with
single-dimensional labels to calculate the lower bounds by immediately adding the
credit value of each arc to the cost. Thus, \( \overline{c}_{0i}^{*} \) is calculated via REF9:

\[
\overline{f}_{ijg}(\overline{c}_{0j}^{*}) = \overline{c}_{0j}^{*} + \overline{c}_{ijg} + u_{ijg}^{6}, \quad \forall (i, j, g) \in \hat{E}.
\]  \hspace{1cm} (7.9)

Recall that in the CPPP, \( \overline{w}_{i}^{3} \) is “the accumulation of block hours since the
second last layover or since the last long layover, whichever is less”. In the forward
interpretation of a backward path this is equivalent to saying \( \overline{w}_{i}^{3} \) is the accumulation
of block hours until the second next layover or until the next long layover, whichever
is less.
When extending resources \( k = \{3, 4\} \) in the CPPP, we have to differentiate between duty connections, long layovers, and regular layovers. Along duty connections, the accumulation is simply increased, while along long layovers, the accumulation is reset to zero. The complicating case is again extending along a regular layover. Let us once more consider Figure 7.2 (page 183). We simultaneously calculate \( \overrightarrow{w}^{1}_{i} \) and \( \overrightarrow{w}^{3}_{i} \) for this example. At node 8, \( \overrightarrow{w}^{1}_{8} = \overrightarrow{w}^{3}_{8} = 0 \) since no further resources are consumed to get to the next replenishment of \( k = 1 \) and \( k = 3 \) (the next long layover). Then, we have \( \overrightarrow{w}^{1}_{7} = \overrightarrow{w}^{3}_{7} = 2 \). Extending along regular layover (6, 7), we get \( \overrightarrow{w}^{1}_{7} = 0 \) since the duty ends, and \( \overrightarrow{w}^{3}_{6} = 3 \) since the amount of block hours required until the second next layover or until the next long layover, whichever is less, is 3. At node 5, we have \( \overrightarrow{w}^{1}_{5} = 1 \) and \( \overrightarrow{w}^{3}_{5} = 4 \). At node 4, \( \overrightarrow{w}^{3}_{4} \) is reset because now “until the second next layover or until the next long layover, whichever is less”, means that the first clause gives the smaller value. \( \overrightarrow{w}^{3}_{3} \) is equal to the usage of connection (4, 5) plus the block hour accumulation in Duty 2, which is \( \overrightarrow{w}^{1}_{3} \), so \( \overrightarrow{w}^{3}_{4} = 2 \). Formally, we define \( \text{REF}^{10} \) for resources \( k \in \{3, 4\} \) in the CPPP to be

\[
\text{REF}^{k}_{ijg}(\overrightarrow{w}^{k}_{j}, \overrightarrow{w}^{k'}_{j}) = \begin{cases} 
\overrightarrow{w}^{k}_{j} + u^{k}_{ijg} & \text{if } (i, j, g) \in \hat{E} \setminus \hat{E}^{k}, \\
\overrightarrow{w}^{k'}_{j} + u^{k}_{ijg} & \text{if } (i, j, g) \in \hat{E}^{k} \text{ and } (i, j) \in C_{RL}, \\
0 & \text{if } (i, j, g) \in \hat{E}^{k} \text{ and } (i, j) \in C_{LL},
\end{cases}
\]

(7.10)

where again, \( k' = 1 \) if \( k = 3 \) and \( k' = 2 \) if \( k = 4 \).

**Standard Shortest Path Calculations in Preprocessing**

The first step of the main preprocessing algorithm calls functions \( \text{FSP}(\kappa) \) and \( \text{BSP}(\kappa) \), \( \kappa \in \hat{K} \) other than \( \kappa \in \{3, 4, 6\} \) if the problem is a CPPPP to generate paths \( \overrightarrow{q}_{\kappa i} \) and \( \overrightarrow{q}_{\kappa i} \) and thus lower bounds \( \overrightarrow{c}^{0*}_{0i}, \overrightarrow{c}^{0*}_{0i}, \overrightarrow{w}^{\kappa}_{\kappa i}, \) and \( \overrightarrow{w}^{\kappa}_{\kappa i} \). These are standard forward (FSP) and backward (BSP) shortest path calculations, modulo the use of replenishment arcs. Since the network is acyclic, we achieve this by running a shortest path problem solver in which we minimise with respect to the \( \kappa \) in question and only consider the limits on the \( \kappa \) in question, all other resource limits are ignored. To solve these problems, we use a variant of our labelling algorithm (7.1) in which we only store a single label at each node, where the label is the best with respect to \( \kappa \). A label is extended using the appropriate REF for the cost or resource.
to be minimised, as described above. Since we are generating labels, FSP(\(\kappa\)) and BSP(\(\kappa\)) also keep track of the (actual) cost, \(\overrightarrow{c}_{\kappa i}\) and \(\overleftarrow{c}_{\kappa i}\), and resource consumption of all resources, \(\overrightarrow{w}_{\kappa i}\) and \(\overleftarrow{w}_{\kappa i}\), of the path the label represents. These costs and resources are calculated using the appropriate REFs described above.

To easily detect infeasibility, values \(\overrightarrow{c}^{*}_{0i}\), \(\overleftarrow{c}^{*}_{0i}\), \(\overrightarrow{w}_{\kappa i}\), and \(\overleftarrow{w}_{\kappa i}\), \(\kappa \in \hat{K}\), \(i \in \hat{N}\) are initialised as \(+\infty\). If for node \(i\) a value remains \(+\infty\) after FSP or BSP was called, it is clear that no feasible path to \(i\) exists. The node can be removed, which is described below.

While not imposing any limits on resources other than \(\kappa\), FSP(\(\kappa\)) and BSP(\(\kappa\)) do keep track of whether a hard limit of any resource \(k \neq \kappa\) is violated at any time, in which case \(\overrightarrow{\nu}_{\kappa i}\) or \(\overleftarrow{\nu}_{\kappa i}\) is set to TRUE, respectively. Infeasibility with respect to resources \(k \in \{3, 4\}\) in the CPPP is kept track of as well, which will be explained below. Here we only note that in order to do so, we need to know the type of layover of the last layover preceding node \(i\) and the type of layover of the next two layovers from \(i\) to \(t\). FSP(\(\kappa\)) and BSP(\(\kappa\)) store this information for every forward path \(\overrightarrow{q}_{\kappa i}\) and backward path \(\overleftarrow{q}_{\kappa i}\), respectively.

FSP() and BSP() can generate \(s-t\) and \(t-s\) paths when extending to \(t\) and \(s\), respectively. If \(\overrightarrow{\nu}_{\kappa t}\) = FALSE or \(\overleftarrow{\nu}_{\kappa s}\) = FALSE, a feasible path is found. In this case, the cost of the path, \(\overrightarrow{c}_{\kappa t}\) or \(\overleftarrow{c}_{\kappa s}\), respectively, is compared to the cost of an incumbent solution, \(c^+\). If the cost of the path is lower, \(c^+\) is updated and the corresponding path is stored as \(q^+\), where \(c^+\) is an upper bound on the cost of the optimal path.

**Concatenation of Paths**

In the second step of the preprocessing algorithm, all forward paths are concatenated with all backward paths in an attempt to find feasible \(s-t\) paths. Function ConcNode(\(\overrightarrow{q}_i, \overleftarrow{q}_i\)) concatenates a forward path \(\overrightarrow{q}_i = (s, \ldots, i)\) with a backward path \(\overleftarrow{q}_i = (i, \ldots, t)\), while function ConcArc(\(\overrightarrow{q}_i, (i, j, g), \overleftarrow{q}_j\)) concatenates a forward path \(\overrightarrow{q}_i = (s, \ldots, i)\) with an arc \((i, j, g)\) and a backward path \(\overleftarrow{q}_j = (j, \ldots, t)\). It should be noted that every forward path, e.g. a minimum cost path, can be concatenated with every backward path, e.g. a \(\kappa\)-path, where \(\kappa \in K\), however, the result needs to be checked for feasibility. Note that when using a minimum cost path for concate-
nation, we have to use the true cost, i.e. $\tilde{c}_{0i}$ or $\hat{c}_{0i}$, not the lower bounds $\tilde{c}_{0i}^*$ or $\hat{c}_{0i}^*$, respectively, to calculate the cost of the resulting $s-t$ path.

The $s-t$ path must be checked for feasibility with respect to all resources $k \in K$. In ConcNode(), equation

$$\tilde{w}_i^k + \hat{w}_i^k \leq U_k$$

(7.11)

has to hold for all resources $k \in K$ if the problem is an ARPP. For a CPPP the same equation has to hold for all resources $k \in \{1, 2, 5\}$. Resource 6 is not restricted by any limit so no check is necessary. Resource $k \in \{3, 4\}$ requires two separate checks, one for each pair of consecutive duties that node $i$ can be part of. For the pair in which $i$ is in the second of the two duties, we check that the next layover is a long layover if the two duties are separated by a regular layover and if

$$\tilde{w}_i^k + \hat{w}_i^{k'} > U_k$$

(7.12)

where $k' = 1$ if $k = 3$ and $k' = 2$ if $k = 4$. Recall that in this case, $\tilde{w}_i^k$ is the accumulation in the current duty up to and including node $i$ plus the accumulation in the previous duty, while $\hat{w}_i^{k'}$ is the usage in the remainder of the second duty, not including node $i$. For the pair where $i$ is in the first duty, if the layover between the first and the second duty is a regular layover and if

$$\tilde{w}_i^{k'} + \hat{w}_i^k > U_k$$

(7.13)

then the second next layover must not be a regular layover.

ConcArc() concatenates at an arc. Here we have to take into consideration what the arc represents. For all resources in the ARPP and resources $k \in \{1, 2, 5\}$ in the CPPP,

$$\tilde{w}_i^k + u_{ijg}^k + \hat{w}_i^k \leq U_k, \quad \text{if } (i, j, g) \notin E_k,$$

(7.14)

$$u_{ijg}^k + \hat{w}_i^k \leq U_k, \quad \text{if } (i, j, g) \in E_k,$$

(7.15)

must hold for a path to be feasible with respect to $k$. For $k \in \{3, 4\}$, we have to differentiate between duty connections, regular layovers, and long layovers. If connection $(i, j)$ is a duty connection, it can be either in the first duty of a pair of consecutive duties or in the second duty. In the latter case if the duties are separated
by a regular layover and if
\[ \overrightarrow{w}_{ik} + u_{ijg}^{k} + \overleftarrow{w}_{jk}^{k} > U^{k}, \] (7.16)
then the next layover must be a long layover. In the first case, if
\[ \overrightarrow{w}_{ik}^{k'} + u_{ijg}^{k} + \overleftarrow{w}_{jk}^{k} > U^{k} \] (7.17)
and the layover between the duties is a regular layover, then the second next layover must be a long layover. If \((i, j)\) is a regular layover, then it is either between the two consecutive duties or at the beginning of the first duty. For the first case, if
\[ \overrightarrow{w}_{ik}^{k'} + u_{ijg}^{k} + \overleftarrow{w}_{jk}^{k'} > U^{k}, \] (7.18)
then the next layover must be a long layover. Value \(\overrightarrow{w}_{ik}^{k'}\) represents the accumulation in the first duty, while \(u_{ijg}^{k} + \overleftarrow{w}_{jk}^{k'}\) is the accumulation in the second duty. If the layover is at the beginning of the first duty, then we need to verify two checks. The first is similar to what was described in the labelling algorithm: if the accumulation of block hours in the two duties prior to the “first” duty, i.e. prior to node \(i\), exceeds the limit and those two duties are separated by a regular layover, then layover \((i, j)\) cannot be a regular layover. Thus, if
\[ \overrightarrow{w}_{ik}^{k} > U^{k} \] (7.19)
and the previous layover is a regular layover, the resulting path is infeasible. The second check is to consider the usage along \((i, j, g)\) and the accumulation in the two duties following layover \((i, j)\). If
\[ u_{ijg}^{k} + \overleftarrow{w}_{jk}^{k} > U^{k} \] (7.20)
and the layover between the two duties is a regular layover, then the second next layover must be a long layover.

If \((i, j)\) is a long layover, it again can be between two consecutive duties or at the beginning of the first duty. In the first case, Rule 3 does not apply, while the second case is similar to that for a regular layover. However, we only need to check Equation (7.20), not Equation (7.19) because even if the limit is exceeded, the long layover \((i, j, g)\) means that Rule 3 is automatically satisfied.
Recall that e.g. $\overrightarrow{w}_i^k$ is the accumulation of $k$ since the last replenishment of $k$. Thus the above equations only check feasibility between the last replenishment of $k$ and the next replenishment of $k$. However, we must check feasibility before and after those replenishments as well. For this we defined $\overrightarrow{\nu}$ and $\overleftarrow{\nu}$. When concatenating at node $i$, we need $\overrightarrow{\nu}_i = \text{FALSE}$ and $\overleftarrow{\nu}_i = \text{FALSE}$ for the $s$–$t$ path to be feasible. When concatenating at an arc $(i, j, g)$, we required $\overrightarrow{\nu}_i = \text{FALSE}$ and $\overleftarrow{\nu}_j = \text{FALSE}$. If a path is feasible and has a cost that is less than that of an incumbent solution, a function is called to store the path (as $q^+$) and to update $c^+$. 

It should be noted that when applying ConcArc() to all arcs and pairs of forward and backward paths available, we implicitly cover all cases of ConcNode(). Finding good feasible $s$–$t$ paths is prudent as this may accelerate the labelling algorithm significantly (see Section 7.1.3). We chose to explore all possible combinations and thus only need to call ConcArc() in our preprocessing algorithm.

### Removing Arcs and Nodes

In the third step of the preprocessing algorithm, we attempt to remove nodes and arcs by using the lower bounds on cost and resource accumulation. To remove nodes, function RemNode$(a, b, \kappa)$ is called. The arguments are either the lower bounds on cost, i.e. $a = \overrightarrow{c}_0^*$ and $b = \overleftarrow{c}_{0i}$, or the lower bounds on each resource accumulation, i.e. $a = \overrightarrow{w}_i^k$ and $b = \overleftarrow{w}_{ki}$, for all $\kappa \in K$ except for $k \in \{3, 4, 6\}$ if the problem is a CPPP. The function checks if the sum of $a$ and $b$ exceeds the resource limit $U_k$ or, if the function was called with $\kappa = 0$, checks if the sum exceeds or is equal to the cost of the incumbent, i.e. $c^+$. If so, the node cannot be part of a feasible or lower cost path, respectively, and can be removed. All incoming and outgoing arcs at the node are removed as well.

In a similar way, arcs are removed in function RemArc$(a, b, c, \kappa)$. The function is called with either the lower bounds on cost, i.e. $a = \overrightarrow{c}_0^*$, $b = c_{ijg} + \overleftarrow{u}_{ijg}$, and $c = \overleftarrow{c}_0$, or the lower bounds on resource accumulation, i.e. $a = \overrightarrow{w}_i^k$, $b = \overleftarrow{u}_{ijg}$, and $c = \overleftarrow{w}_{kj}$, for all $\kappa \in K$ except for $k \in \{3, 4, 6\}$ if the problem is a CPPP. The sum of $a, b$, and $c$ again must not exceed the resource limit or be equal to or greater than the best known cost, respectively. If this fails, arc $(i, j, g)$ is removed. It should be noted that if any nodes or arcs are removed in RemNode() or RemArc(), all appropriate
sets are updated.

No restrictions to the accumulation of credit exists, i.e. in a CPPP, \( U^6 = \infty \). Therefore, nodes and arcs can never be removed due to this resource. We thus do not need values \( \overrightarrow{w}_{6i}^6 \) and \( \overleftarrow{w}_{6i}^6 \). On the other hand, the minimum weight paths with respect to resource 6 can be used for concatenation. However, calling FSP() and BSP() just to obtain additional forward and backward paths does not justify the computational effort. We therefore do not call FSP() and BSP() for resource 6 in the CPPP.

To explain why we do not attempt to find minimum resource usage paths with respect to resource \( k \in \{3, 4\} \), we observe that Rules 3 and 4 do not actually limit the number of block and duty hours in consecutive duties. They merely require a long layover at the end of the second duty if the soft limit is exceeded. We again use Rule 3 to illustrate the issue when removing nodes or arcs. To remove node \( i \), Rule 3 has to be checked for two pairs of consecutive duties, once for when \( i \) is in the first duty, once when it is in the second. For the first case, we need the lower bound on the accumulation of duty hours since the start of the first duty, i.e. \( \overrightarrow{w}_{1i}^1 \), and the lower bound on the accumulation of duty hours in the remainder of the first duty and the following duty, i.e. \( \overrightarrow{w}_{3i}^3 \). If \( \overrightarrow{w}_{1i}^1 + \overrightarrow{w}_{3i}^3 > U^3 \), then node \( i \) can be removed if we can prove that no path exists in which neither the first nor the following duty is followed by a long layover. However, we can hardly fulfill these requirements. The reason is that in function BSP(), we generate possibly infeasible minimum weight paths since limits on other resources are ignored. We therefore can, if the underlying network permits, almost always find a sequence of connections that avoids regular layovers and only contain duty connections until sink node \( t \). Since duty connections do not end a duty, the first duty does end with a long layover (recall that sink arcs have the same properties as long layovers) and thus the node cannot be removed.

A similar argument can be made for the second check and for when removing arcs. Values \( \overrightarrow{w}_{\kappa i}^\kappa \) and \( \overleftarrow{w}_{\kappa i}^\kappa \), \( \kappa \in \{3, 4\} \) are therefore of little use. Additionally, as is described in Section 7.1.4, we cannot use a standard shortest path problem solver to calculate these values. Therefore, we choose to not calculate lower bounds on resources 3 and 4, that is we do not call FSP() and BSP() for resources 3 and 4 in the CPPP.
If any node or arc is removed, new lower bounds and new forward and backward paths can be generated by calling functions FSP() and BSP(). We again can concatenate paths and possibly remove more nodes and arcs. This can be iterated until no more nodes and arcs are removed. It should be noted that arcs and nodes are removed if they cannot be part of a path with cost strictly less than the incumbent. Then, if the optimal solution was found in a previous iteration, the instance may be made infeasible by removing nodes and arcs of the optimal path. In this case, the algorithm terminates and returns the optimal path.

The Preprocessing Algorithm

Algorithm 7.2 displays the preprocessing procedure for the RCSPP-RAE. The algorithm first finds lower bounds on cost for forward paths and then updates the lower bound $c^-$. Note that if no feasible minimum cost path from $s$ to $t$ exists, the lower bound is set to $+\infty$ since $c^*_{st}$ is initiated as such. If, on the other hand, a feasible minimum cost $s$–$t$ path exists (line 7.2.5), the cost of the incumbent is updated if the true cost of the minimum cost path is lower than that of the incumbent. If the cost is updated, the new path is stored.

Line 7.2.7 checks if the lower bound is larger or equal to the upper bound, in which case the algorithm terminates. There are several cases how this can happen:

1. If $c^- = +\infty$, the problem is currently infeasible. If $c^+ = 0$, the problem is indeed infeasible or only solutions of non-negative cost exist. If $c^+ < 0$, then $q^+$, which was found in a previous iteration, is the optimal solution. Removing arcs and nodes in succeeding iterations made the remaining network infeasible.

2. If $0 \leq c^- < +\infty$, the problem is currently feasible, however, the algorithm can terminate since if $c^+ = 0$, no negative cost path exists, and if $c^+ < 0$, the optimal solution was found in a previous iteration. Removing nodes and arcs in successive iterations eliminated all other negative cost paths. In the latter case, path $q^+$ is optimal.

3. If $c^- < 0$ (and $c^+ < c^-$), the optimal solution was found in a previous iteration. Not all arcs or nodes that allow negative cost paths were removed, however, we do not continue as we are satisfied with having found the optimal path, i.e.
If \( c^- < c^+ \), better paths may exist and the algorithm continues by calling BSP(0) to calculate lower bounds on cost for backward paths. In lines 7.2.8 through 7.2.12 lower bounds on resource accumulation along forward and backward paths are calculated. The algorithm checks if any \( \kappa \)-path, \( \kappa \in K \), from \( s \) to \( t \) exceeds the resource limit. If so, the problem is currently infeasible. The lower bound is set to \( c^- = +\infty \). If \( c^+ = 0 \), the problem is infeasible. Otherwise, incumbent \( q^+ \) is optimal. Removing arcs and nodes in previous iterations has led to the remaining problem to be infeasible.

Lines 7.2.13 through 7.2.15 concatenate all available forward and backward paths, while lines 7.2.16 through 7.2.23 remove nodes and arcs. The algorithm starts over if any node or arc was removed. Otherwise, it returns the updated graph, lower bounds on cost and resource accumulation for every \( i \in N \), an upper and lower bound on the cost of the optimal \( s-t \) path, and all stored \( s-t \) paths. Upon termination, we can easily interpret the results: if \( c^- < c^+ \), the labelling algorithm (7.1) is called. Otherwise, the pricing problem was solved because either the optimal solution was found \( (c^+ < 0) \) or the problem is infeasible/no negative cost column exists \( (c^+ = 0) \).

### 7.1.3 Accelerating The Labelling Algorithm

If the preprocessing algorithm did not find the optimal solution, the labelling algorithm (7.1) is called. It can be accelerated by using preprocessing information to evaluate if extending label \( \Gamma \) further to \( t \) will always result in an infeasible or inferior \( s-t \) path. If so, the label can be removed.

We note that at node \( j \), at least \( \overleftarrow{w}_{\kappa j}^\kappa \) of resource \( \kappa \) will have to be accumulated until the closest replenishment of \( \kappa \). If this value plus the accumulation of resource \( \kappa \) since the last replenishment of \( \kappa \) along the path that label \( \Gamma \) represents exceeds the (hard) resource limit of \( \kappa \), no feasible path can result from extending \( \Gamma \). Thus, if \( \overleftarrow{w}_{\kappa j}^\kappa + \overrightarrow{w}_{\kappa j}^\kappa > U^\kappa \) for any resource \( \kappa \in K \) in the ARPP and for \( \kappa \in \{1, 2, 5\} \) in the CPPP, the label is not added to \( \mathcal{L}_j \), \( j \in N \). No such check is carried out for resources \( \kappa \in \{3, 4, 6\} \) in the CPPP because no useful values \( \overleftarrow{w}_{\kappa j}^\kappa \) can be calculated for these resources (see Section 7.1.2).
Algorithm 7.2: PREPROCESSING

Input: A RCSPP-RAE instance.

Output: A possibly reduced RCSPP-RAE instance, \( \overrightarrow{c}_{0i} \) and \( \overrightarrow{\nu}_{ki} \), \( \forall i \in \hat{N} \), \( \kappa \in K \) except for \( \kappa \in \{3, 4, 6\} \) if instance is a CPPP, \( c^+, c^- \), and feasible paths if available.

7.2.11 \( c^+ := 0 \), \( c^- := -\infty \), \( q^+ := \emptyset \);

7.2.12 while First iteration or any node or arc was removed do

7.2.13 Run FSP(0) to get \( \overrightarrow{c}^*, \overrightarrow{\nu}_0 \), and \( \overrightarrow{w}_0^k \), \( k \in K \);

7.2.14 \( c^- := \max(c^-, \overrightarrow{c}^*) \); // Update lower bound

7.2.15 if \( \overrightarrow{\nu}_0 = \) FALSE then // Feasible min cost path found

7.2.16 \( c^+ := \min(c^+, \overrightarrow{c}_{0k}) \), if \( c^+ = \overrightarrow{c}_{0k} \) then \( q^+ := \overrightarrow{q}_0 \);

7.2.17 if \( c^- \geq c^+ \) then go to 7.2.24; // Infeasible or optimal

7.2.18 Run BSP(0) to get \( \overrightarrow{c}_0^0, \overrightarrow{\nu}_0^0, \overrightarrow{c}_0^k, \overrightarrow{\nu}_k \), and \( \overrightarrow{w}_k^k \), \( k \in K \);

7.2.19 for \( \kappa \in K \) except for \( \kappa \in \{3, 4, 6\} \) if instance is a CPPP do

7.2.20 Run FSP(\( \kappa \)) to get \( \overrightarrow{c}_\kappa^\kappa, \overrightarrow{\nu}_\kappa \), and \( \overrightarrow{w}_\kappa^k \), \( k \in K \);

7.2.21 if \( \overrightarrow{w}_\kappa^k > U_k \) then \( c^- := +\infty \) and go to 7.2.24;

7.2.22 Run BSP(\( \kappa \)) to get \( \overrightarrow{q}_\kappa, \overrightarrow{c}_\kappa, \overrightarrow{\nu}_\kappa \), and \( \overrightarrow{w}_\kappa^k \), \( k \in K \);

7.2.23 for \( \kappa \in \hat{K} \), except for \( \kappa \in \{3, 4, 6\} \) if instance is a CPPP do

7.2.24 for \( (i,j,g) \in E \) do ConcArc(\( \overrightarrow{q}_{\kappa i}, (i,j,g), \overrightarrow{q}_{\kappa j} \));

7.2.25 for \( i \in N \) do

7.2.26 RemNode(\( \overrightarrow{c}_{0i}, \overrightarrow{\nu}_0^i, 0 \));

7.2.27 for \( \kappa \in K \), except for \( \kappa \in \{3, 4, 6\} \) if instance is a CPPP do

7.2.28 RemNode(\( \overrightarrow{w}_{\kappa i}, \overrightarrow{\nu}_{\kappa i}, \kappa \));

7.2.29 for \( (i,j,g) \in E \) do

7.2.30 RemArc(\( \overrightarrow{c}_{0i}, c_{ijg}, \overrightarrow{w}_0^i, 0 \));

7.2.31 for \( \kappa \in K \), except for \( \kappa \in \{3, 4, 6\} \) if instance is a CPPP do

7.2.32 RemArc(\( \overrightarrow{w}_{\kappa i}, u_{ijg}, \overrightarrow{w}_{\kappa j}, \kappa \));

7.2.33 return \( G, c^+, c^-, q^+ \), \( \overrightarrow{c}_0^*, \overrightarrow{\nu}_k^k \), \( \kappa \in K \) except for \( \kappa \in \{3, 4, 6\} \) if instance is a CPPP.
7.1. PRICING PROBLEM SOLVER

We can further reduce the number of labels at node $j \in N$ by considering the cost of a label, $\overrightarrow{c}_r$, and the lower bound on cost for node $j$, i.e. $\overrightarrow{c}_{0j}^s$. The cost of any $s$–$t$ path that is generated by extending label $\Gamma$ will be at least $\overrightarrow{c}_r + \overrightarrow{c}_{0j}^s$. We can compare this value to the cost of the incumbent, $c^+$. Recall that $c^+$ is initialised as zero or using the preprocessing algorithm (7.2) and is updated whenever a better path is found. We then chose to store label $\Gamma$ only if $\overrightarrow{c}_r + \overrightarrow{c}_{0j}^s < c^+$. This will never remove the optimal path but may remove paths that have negative reduced cost. On the other hand, depending on the quality of $c^+$, we can significantly reduce the number of labels generated in Algorithm 7.1. This clearly is a trade-off between generating a smaller number of labels and generating multiple columns that can be added to the master problem (see Section 6.4). We opted for speed since it is essential to solve the many pricing problems quickly. Furthermore, we expect that solving many pricing problems will result in multiple columns anyway.

It should be noted that if preprocessing did not return an upper bound, $c^+$ is initialised as zero in the labelling algorithm. As a result, we can only remove labels that will result in paths with cost larger or equal to zero. This highlights the importance of concatenating paths in preprocessing. If no upper bound or one of poor quality is found, a larger number of labels will result in the labelling algorithm.

The checks described here are carried out in function CheckFurtherExtension($\Gamma, j$) (7.3), which we call before checking dominance in line 7.1.13 of Algorithm 7.1. The function returns TRUE if the label should be added to $L_j$ and FALSE otherwise.

---

**Algorithm 7.3: CheckFurtherExtension()**

**Input:** Label $\Gamma$, and node $j$

**Output:** TRUE if label should be added, FALSE otherwise.

7.3.1 if Instance is of type ARPP and $\overrightarrow{c}_r + \overrightarrow{c}_{0j}^s < c^+$ and $\overrightarrow{w}_{\kappa r} + \overrightarrow{w}_{\kappa j} \leq U_\kappa$, $\forall \kappa \in K$, then

7.3.2 return TRUE else return FALSE

7.3.3 if Instance is of type CPPP and $\overrightarrow{c}_r + \overrightarrow{c}_{0j}^s < c^+$ and $\overrightarrow{w}_{\kappa r} + \overrightarrow{w}_{\kappa j} \leq U_\kappa$, $\forall \kappa \in \{1, 2, 5\}$, then

7.3.4 return TRUE else return FALSE
7.1.4 Consecutive Duty Rules with Hard Limits

In our problem, the consecutive duty rules require that if the limit is exceeded, a certain type of rest has to be scheduled. A less complicated case is to require a “hard” limit instead. This case, where generally speaking the accumulation of some resource is restricted over two consecutive periods, could be of relevance in other applications and is thus an interesting case in itself.

In this section, we will describe how, in the case of hard limits, lower bounds on the resource accumulation along backward paths can be calculated and used to eliminate arcs and nodes and to accelerate a labelling algorithm. Unfortunately, these backward calculations are not straightforward.

We will use a variant of Rule 3, which we refer to as Rule 3h, to illustrate the complicating issue. Rule 3h requires that the total block hours in two consecutive duties that are not separated by a long layover must not exceed MB2. In analogy to notation introduced in previous sections, we use $\kappa = 3$ to denote the accumulation across two duties (except when the duties are separated by a long layover), while $\kappa' = 1$ denotes the accumulation in a single duty. It should be noted that the underlying issue is neither due to the hard limit nor due to the existence of regular and long layovers. It is a consequence of accumulating across two periods and then selecting the minimum value at each node.

When calculating lower bounds on the accumulation of resource 3 for backward paths, we cannot use a standard shortest path problem solver because not all required paths are stored. Consider the example in Figure 7.4, in which we calculate values $\hat{w}_i^3$ for every depicted node. We will simultaneously calculate $\hat{w}_i^1$ to clarify calculations of $\hat{w}_i^3$. However, it should be noted that the critical values are $\hat{w}_i^3$, not $\hat{w}_i^1$. Then, a backward path to node 7 will have $\hat{w}_7^3 = 1$ because one block hour is required to reach the second next layover. $\hat{w}_7^1$ on the other hand is equal to zero because no block hours are required to get to the next layover since it immediately follows node 7. Extending this path to node 6 gives $\hat{w}_5^3 = 4$ and $\hat{w}_6^1 = 3$. Similarly, a path to node 8 has $\hat{w}_8^3 = 5$ and $\hat{w}_8^1 = 0$. For node 6, we get $\hat{w}_6^3 = 6$ and $\hat{w}_6^1 = 1$. Connection \((5, 6)\) is a regular layover, so when extending the paths to node 5, we have $\hat{w}_5^3 = w_{5y}^3 + \hat{w}_6^1$, i.e. the usage on the connection plus the accumulation in just this single duty. For the first path, $\hat{w}_5^3 = 1 + 3 = 4$ and $\hat{w}_5^1 = 0$ because again,
no block hours are required until the next layover. For the second path, we have 
\( \bar{w}_5^3 = 1+1 = 2 \) and \( \bar{w}_5^1 = 0 \). Therefore, the lower bound for node 5 is two. However, at node 6, in a standard shortest path problem algorithm that is with respect to resource 3, the second path is dominated by the first one and is thus not available for extension to node 5.

![Network Diagram](image)

Figure 7.4: A network that illustrates the issue when extending minimum 3-paths backwards. An incorrect lower bound of four results at node 5 because, in a basic shortest path algorithm, the top path dominates the bottom one at node 6.

It should be obvious that to get correct lower bounds on the accumulation of resource 3 at all nodes \( j \in N \), we could store all paths with non-dominated pairs \( (\bar{w}_j^1, \bar{w}_j^3) \). However, this is not necessary. In fact, we only need to store up to two paths at any node \( j \in N \). We always require the path that gives the lower bound for \( j \), i.e. \( \bar{w}_j^3 \), which is the least resource usage from \( j \) until the second next layover or next long layover, whichever is less. The additional path is the one with the least resource usage until the next layover, i.e. the path with the smallest \( \bar{w}_j^1 \). This path is required because when extending backwards along a regular layover \( (i, j) \), the previously next layover now becomes the second next layover. The second path may therefore give the lower bound at node \( i \). To illustrate why no other paths are required, we extend the example in Figure 7.4 to include a third path which is
extended from node 9 (see Figure 7.5). At node 6, the path has an accumulation of \( \overrightarrow{w}_6^3 = 5 \) and \( \overrightarrow{w}_6^1 = 2 \), and thus is not dominated when considering both, \( \overrightarrow{w}_j^1 \) and \( \overrightarrow{w}_j^3 \). We will show how this path can never be better with respect to resource 3 than the second path, i.e. the one with \( (\overrightarrow{w}_6^1 = 1, \overrightarrow{w}_6^3 = 6) \).

Extending any of the three paths along long layover \((3, 6)\) results in \( \overrightarrow{w}_3^3 = 0 \) and \( \overrightarrow{w}_3^1 = 0 \) for all of them. Since the goal of these calculations is to find lower bounds on the accumulation of resource 3, the three backward paths to node 3 are considered identical, thus storing any one of them will suffice. Extending along duty connection \((4, 6)\) will simply add the same value to \( \overrightarrow{w}_6^3 \) and \( \overrightarrow{w}_6^1 \) for any path, thus the order of the paths with respect to resource 3 at node 5 is the same as the order with respect to resource 3 at node 6. This is not the case when extending along regular layover \((5, 6)\). As was previously described, \( \overrightarrow{w}_5^1 = 0 \) for all paths, while \( \overrightarrow{w}_5^3 = w_{ij}^3 + \overrightarrow{w}_6^3 \). That is, the same value \( w_{ij}^3 \) is added to \( \overrightarrow{w}_6^3 \) of the respective path. Thus, at node 5, the order of the labels with respect to resource 3 is the same as the order at node 6 with respect to resource 1. The second path now dominates the third (and the first). Hence, at node 6, the only label that we need to store in addition to the label giving the minimum \( \overrightarrow{w}_6^3 \), is the label with the minimum \( \overrightarrow{w}_6^1 \).

Once the correct bounds are calculated we can call functions RemNode() and RemArc(). Node \( i \) is removed by again checking two cases, one where \( i \) is considered to be in the first duty, one where \( i \) is in the second duty. For the first case, if the duties are separated by a regular layover and if

\[
\overrightarrow{w}_{\kappa_i}^\kappa + \overrightarrow{w}_{\kappa_i}^\kappa > U^\kappa,
\]

then node \( i \) can be removed. In the second case, if the duties are separated by a regular layover and if

\[
\overrightarrow{w}_{\kappa_i}^\kappa + u_{ij}^\kappa + \overrightarrow{w}_{\kappa_i}^\kappa > U^\kappa,
\]

then node \( i \) can be removed.

In function ConcArc(), we again have to consider the type of connection that the arc represents. For arcs \((i, j, g)\) representing a duty connection, we remove the arc if the next layover is a regular layover and if

\[
\overrightarrow{w}_{\kappa_i}^\kappa + w_{ij}^\kappa + \overrightarrow{w}_{\kappa_i}^\kappa > U^\kappa,
\]

(7.21)
Figure 7.5: Extension to Figure 7.4. The example is used to show that, in order to calculate valid lower bounds with respect to resource 3, we need to store at most two paths, one giving the correct lower bound at the current node, one giving the lowest bound when extending backwards along a regular layover. For node 6, the top path gives the correct lower bound at node 6, while the middle path has to be stored so that we get the correct lower bound at node 5. The third path does not need to be stored at all.
or if the previous layover is a regular layover and if
\[ \overrightarrow{w}_{i'\kappa i} + u_{ijg}^\kappa + \overleftarrow{w}_{\kappa'j} > U^\kappa. \]  
(7.22)

An arc that represents a regular layover can be at the beginning of the first duty of a pair of duties. In this case, if the next layover, i.e. the one separating the duties, is a regular layover and if
\[ u_{ijg}^\kappa + \overleftarrow{w}_{\kappa'j} > U^\kappa, \]  
then arc \((i,j,g)\) can be removed. We also need to consider the case when the regular layover is between a pair of consecutive duties. The arc can be removed if
\[ \overrightarrow{w}_{i'\kappa i} + u_{ijg}^\kappa + \overleftarrow{w}_{\kappa'j} > U^\kappa. \]  
(7.24)

Similarly for long layovers: the arc can be removed if the long layover is at the beginning of the first duty and the next layover is a regular layover and if
\[ u_{ijg}^\kappa + \overleftarrow{w}_{\kappa'j} > U^\kappa. \]  
(7.25)

If the long layover is between the two duties, the rule doesn’t apply.

In the labelling algorithm we can use these bounds to evaluate if extending a label further will result in violating Rule 3h. This would be done in function \textit{CheckFurtherExtension}(). Again two checks are possible. The first assumes that node \(j\) is in the first duty, we thus need to consider \(\overrightarrow{w}_{\kappa'i'}\), the accumulation in the current duty up to node \(j\), and \(\overleftarrow{w}_{\kappa'j}\), which is the lower bound on the accumulation in the remainder of the current duty and the following duty. If the next layover along the backward path that gives lower bound \(\overleftarrow{w}_{\kappa'j}\) is a regular layover and if
\[ \overrightarrow{w}_{\kappa'i'} + u_{ijg}^\kappa + \overleftarrow{w}_{\kappa'j} > U^\kappa, \]  
then no feasible path can result from label \(\Gamma\). The second case assumes that \(j\) is in the second duty, thus if the previous layover is a regular layover and if
\[ \overrightarrow{w}_{\kappa'i'} + \overleftarrow{w}_{\kappa'j} > U^\kappa, \]  
(7.27)

then \(\Gamma\) can be discarded.
7.1.5 Problem Specific Accelerations Techniques

Our integrated problem exhibits certain features that can be exploited when solving the pricing problems. In this section, we discuss how this is implemented.

Among other goals, the preprocessing algorithm (7.2) finds minimum forward and backward \( \kappa \)-paths, \( \kappa \in K \), and eliminates arcs and nodes that cannot be part of a feasible path. Since the initial resource accumulation of each aircraft and crew block does not change during the CG procedure, we can carry out preprocessing with respect to resources at the start of the branch-and-price algorithm. The removed arcs and nodes are never considered in the remainder of the branch-and-price algorithm. Since this has an impact on all subsequent pricing problem solves, we spent more effort in the preprocessing procedure than during regular CG iterations (see next section). We continue iterating the preprocessing procedure (i.e. line 7.2.2 through 7.2.23) until no more nodes or arcs are removed. Since we are not interested in cost at this point, we call Algorithm 7.2 without carrying out any calculations with respect to cost. Furthermore, we do not concatenate any paths.

It should be noted that whenever we solve a pricing problem during the branch-and-price algorithm, we still remove nodes and arcs with respect to \( \kappa \). Calculations with respect to cost may have led to removing nodes and arcs, which leads to different forward and backward \( \kappa \)-paths, resulting in possibly additional removing due to \( \kappa \). A similar argument can be made for branching decisions. Again, different \( \kappa \)-paths may result after arcs have been removed.

The labelling algorithm (7.1) can be accelerated for ARPPs by considering that once a SLIM has been scheduled in a path, no additional checks of the same type need to be scheduled as usually several weeks pass between two checks of the same type\(^6\). This observation reduces the number of labels generated in the labelling algorithm in two ways as follows.

We redefine \( \text{REF2} \) in a way that along arc \((i, j, g) \in \hat{E}^k \), where \( k \) relates to a SLIM, the resource accumulation \( \overrightarrow{w}_{ij}^k \) is not set to equal the weight accumulation along this arc, i.e. \( u_{ijg}^k \), but to \(-1\) instead\(^7\). Furthermore, if the current accumulation \( \overrightarrow{w}_{i}^k = -1 \), we do not increase the accumulation of \( k \) along any arc, i.e. it will always

\(^6\)This is not true for the MOPPs, which are to be scheduled on average every two days.

\(^7\)Recall that \( u_{ijg}^k \geq 0 \), hence an accumulation of \(-1\) can never be achieved unless we deliberately set it to be \(-1\).
remain at -1. This reduces the number of non-dominated labels as follows: Let us assume that a label $L$ dominates label $\Gamma$ if resource $k$ is ignored. However, if $k$ is considered, $L$ does not dominate $\Gamma$ because $\overrightarrow{w}_L^k > \overrightarrow{w}_\Gamma^k$. From the observation, we know that if both labels have scheduled the SLIM that $k$ relates to, the accumulation of $k$ can be ignored when checking dominance. In our algorithm, an accumulation of $\overrightarrow{w}_L^k = \overrightarrow{w}_\Gamma^k = -1$ achieves just that.

The number of labels can be reduced further by noting that we do not need to extend a label along an arc $(i, j, g) \in \hat{E}^k$ if $\overrightarrow{w}_i^k = -1$, i.e. a replenishment for $k$ was scheduled previously where, again, $k$ relates to a SLIM. This seems trivial, however, to show that this is indeed true, we need to consider that another resource $k' \in K$, $k' \neq k$ which does not relate to the same SLIM may be reset along $(i, j, g)$ as well. This is the case when arc $(i, j, g) \in \hat{E}$ signifies that more than one SLIM is scheduled. However, as is described in Section 5.2.1, there always exists a parallel arc $(i, j, g') \in \hat{E}$ that does schedule the SLIM $k'$ relates to but not the SLIM $k$ relates to.

### 7.2 Preprocessing and Heuristic Pricing

The preprocessing algorithm (7.2) iterates until no more nodes and arcs are removed. Depending on the size of the network, this may be time-consuming and can be inefficient if only few nodes and arcs are removed, which is to be expected for later iterations. In this section, we investigate how many iterations, also called rounds, should be performed. Additionally, we show how the performance of the branch-and-price algorithm is affected if the pricing problems are solved heuristically, i.e. not to optimality. In our algorithm, these two goals are inter-linked as our main source of heuristic pricing is to abort solving pricing problems after preprocessing.

We first investigate the impact of preprocessing on solving individual pricing problems. We then examine how our preprocessing method impacts CG convergence. For this we analyse how the RMP initialisation method benefits from preprocessing and how preprocessing in general results in columns with fewer non-zero entries. Then, in Section 7.2.1, we investigate the effects of aborting the pricing problem solves after preprocessing based on different criteria. In the final Section (7.2.2), we provide an analysis of how solving pricing problems heuristically in our algorithm
impacts the quality of the dual bound of column generation.

In the first experiment, we carried out between one and five preprocessing iterations and compared the resulting performance with not carrying out any preprocessing. We first analyse the impact at a pricing problem level. Table 7.1 shows the average percentage of arcs remaining in the pricing problems for each instance and setting. It should be noted that for this, we only consider pricing problem solves in which the labelling algorithm was called, not those where the optimal solution was found in preprocessing (since we may not have called the RemNode() and RemArc() functions). We observe that the percentage of remaining arcs actually increases when carrying out more preprocessing. This somewhat counter-intuitive behaviour can be explained by simultaneously considering the percentage of pricing problems solved to optimality in preprocessing (Table 7.2). We note that this percentage increases for more preprocessing iterations. The reason is that the pricing problems for which a small number of preprocessing iterations results in a large decrease in the size of the network, can in fact be solved to optimality when carrying out more iterations. Thus, instead of reducing the percentage of remaining arcs, the percentage of pricing problems solved to optimality in preprocessing is increased.

<table>
<thead>
<tr>
<th>Instance</th>
<th>Number of Iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td>5-S-1</td>
<td>0.75</td>
</tr>
<tr>
<td>5-S-2</td>
<td>0.76</td>
</tr>
<tr>
<td>5-L-1</td>
<td>0.72</td>
</tr>
<tr>
<td>5-L-2</td>
<td>0.72</td>
</tr>
<tr>
<td>3-S-A</td>
<td>0.72</td>
</tr>
<tr>
<td>3-L-A</td>
<td>0.72</td>
</tr>
</tbody>
</table>

Table 7.1: Average fraction of arcs remaining in pricing problems after preprocessing when carrying out a different number of preprocessing iterations. These numbers do not reflect PP solves in which the optimal solution was found in preprocessing.

On the other hand, we observe that the average time spent solving all pricing problems per CG iteration mostly increased when performing more preprocessing
CHAPTER 7. SOLVING THE PRICING PROBLEMS

<table>
<thead>
<tr>
<th>Instance</th>
<th>Number of Iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td>5-S-1</td>
<td>5-S-2</td>
</tr>
<tr>
<td>5-S-1</td>
<td>0.17 0.21 0.23 0.31</td>
</tr>
<tr>
<td>5-S-2</td>
<td>0.20 0.21 0.24 0.29</td>
</tr>
<tr>
<td>5-L-1</td>
<td>0.18 0.21 0.30 0.31</td>
</tr>
<tr>
<td>5-L-2</td>
<td>0.19 0.22 0.33 0.38</td>
</tr>
<tr>
<td>3-S-A</td>
<td>0.33 0.35 0.44 0.45</td>
</tr>
<tr>
<td>3-L-A</td>
<td>0.36 0.37 0.45 0.48</td>
</tr>
</tbody>
</table>

Table 7.2: Fraction of pricing problems solved to optimality in preprocessing when carrying out a different number of preprocessing iterations.

Table 7.3: Average time (s) spent in pricing per CG iteration when performing different levels of preprocessing.

<table>
<thead>
<tr>
<th>Instance</th>
<th>Number of Iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td>5-S-1</td>
<td>5-S-2</td>
</tr>
<tr>
<td>5-S-1</td>
<td>0.80 0.64 0.74 0.77 0.77</td>
</tr>
<tr>
<td>5-S-2</td>
<td>0.66 0.56 0.64 0.77 0.74</td>
</tr>
<tr>
<td>5-L-1</td>
<td>1.18 1.26 1.46 1.66 1.91</td>
</tr>
<tr>
<td>5-L-2</td>
<td>1.30 1.23 1.36 1.54 1.60</td>
</tr>
<tr>
<td>3-S-A</td>
<td>2.23 1.91 2.12 2.38 2.49</td>
</tr>
<tr>
<td>3-L-A</td>
<td>2.09 1.57 1.96 2.20 2.15</td>
</tr>
</tbody>
</table>

Taking a broader perspective, we see, however, that preprocessing does have a significant impact on the overall run times of the branch-and-price algorithm (6.2), which is noteworthy especially when considering that we already made several ad-

---

8In the remainder of this thesis, “run time” will always refer to Algorithm 6.2. Unless specified, the run time is for solving the entire 1-branch of the branch-and-price tree.
vancements in Chapter 6. We therefore analyse the cause of this reduction in more
detail. Table 7.4 shows the run times for each of the six small instances. Figure 7.6
displays the results graphically. We observe a significant improvement when carry-
ing out preprocessing regardless of the number of iterations. In fact, no strategy
involving preprocessing performs much better than the others. The average reduc-
tion was 29.37% for one round of preprocessing, 29.31% for two rounds, 27.38% for
three, and 27.18% for five rounds.

<table>
<thead>
<tr>
<th>Instance</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>5-S-1</td>
<td>538</td>
<td>415</td>
<td>393</td>
<td>435</td>
<td>379</td>
</tr>
<tr>
<td>5-S-2</td>
<td>697</td>
<td>467</td>
<td>520</td>
<td>491</td>
<td>511</td>
</tr>
<tr>
<td>5-L-1</td>
<td>1439</td>
<td>914</td>
<td>938</td>
<td>941</td>
<td>970</td>
</tr>
<tr>
<td>5-L-2</td>
<td>1879</td>
<td>907</td>
<td>965</td>
<td>1048</td>
<td>1088</td>
</tr>
<tr>
<td>3-S-A</td>
<td>1198</td>
<td>1055</td>
<td>935</td>
<td>1010</td>
<td>1066</td>
</tr>
<tr>
<td>3-L-A</td>
<td>1099</td>
<td>879</td>
<td>902</td>
<td>869</td>
<td>868</td>
</tr>
</tbody>
</table>

Table 7.4: Run times (s) for the small instances when using different limits on the
number of preprocessing iterations.

As shown above, the majority of the reduction in run time is not due to smaller
networks in the labelling algorithm or due to finding the optimal path in the preprocessing procedure but instead, is due to a reduction in the number of column generation iterations at the root node (see Table 7.5). Recall that CG iterations at the root node are the most time consuming due to large LP solution times\(^9\). Thus, any reduction of CG iterations at the root node has a significant impact.

<table>
<thead>
<tr>
<th>Instance</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>5-S-1</td>
<td>145</td>
<td>107</td>
<td>101</td>
<td>101</td>
<td>99</td>
</tr>
<tr>
<td>5-S-2</td>
<td>179</td>
<td>122</td>
<td>132</td>
<td>115</td>
<td>129</td>
</tr>
<tr>
<td>5-L-1</td>
<td>173</td>
<td>105</td>
<td>99</td>
<td>101</td>
<td>100</td>
</tr>
<tr>
<td>5-L-2</td>
<td>209</td>
<td>100</td>
<td>92</td>
<td>97</td>
<td>96</td>
</tr>
<tr>
<td>3-S-A</td>
<td>74</td>
<td>59</td>
<td>52</td>
<td>55</td>
<td>54</td>
</tr>
<tr>
<td>3-L-A</td>
<td>81</td>
<td>57</td>
<td>55</td>
<td>51</td>
<td>50</td>
</tr>
</tbody>
</table>

Table 7.5: Number of column generation iterations at the root node for the small instances when using different limits on the number of preprocessing iterations. The solve was aborted once the early branching gap was reached.

Our results suggest that the reason for the reduction in the number of CG iterations is twofold, although the underlying cause is the same - as we will show in the following discussion. Firstly, preprocessing has a positive effect on our initialisation method (Section 6.3), and secondly, preprocessing generates columns with fewer non-zero entries during the CG procedure. We first analyse the effect on the initialisation method, then the effect on the CG procedure, followed by a discussion as to why these type of columns are generated.

As can be seen in Figure 7.7, the average number of flights covered in a route or pairing generated in the initialisation method is reduced consistently and falls in line with the general behaviour of the run time (see Figure 7.6). On average, the number of flights is reduced by 10.20% compared to not preprocessing. In the remainder of this thesis, we will speak of smaller paths or columns when we refer to

\(^9\)It should be noted that we perform early branching so the somewhat lower LP solution times toward the end of the tailing-off phase do not occur here.
7.2. PREPROCESSING AND HEURISTIC PRICING

paths or columns that cover fewer flights than the average path or column. Similarly, larger paths or columns refer to paths or columns that cover more flights than the average path or column. Furthermore, we define the size of a path or column as the number of flights covered. The type of path, i.e. route or pairing, depends on the context. If we have been talking about aircraft, paths will refer to routes. However, if we specifically talk about columns generated by different methods/algorithms, we compare the average number of flights covered by the columns generated in the respective methods/algorithms.

![Figure 7.7: Average number of flights covered in a route or pairing generated during Algorithm 6.1.](image)

Generating smaller paths in the initialisation method has several consequences. Recall that in Algorithm 6.1, the pricing problems are solved sequentially and that the flights in the optimal path found for a PP are removed from all other pricing problems. Each PP generates up to three columns, where the flights in the second and third best path are not removed from other pricing problems. Due to the way the flights are removed, some pricing problems are not able to find feasible paths through their remaining networks as these may be disconnected. Generating smaller paths then results in creating more paths because a larger number of pricing problems is able to find feasible paths as the networks are reduced less rapidly\(^\text{10}\). We found that with preprocessing, independent of how many iterations were performed,\[^\text{10}\]It should be noted though, that this does not automatically result in covering more flights with this set of columns as the columns are smaller.
the number of columns generated in Algorithm 6.1 was between 1.48 and 10.93% higher. However, we do not observe a correlation between an increased number of columns and the reduction in run times. For example, for the smallest decrease in run time, which was 11.02%, the number of columns was 6.07% larger, while for the greatest decrease in run time, 51.74%, only 5.06% more columns were generated. This suggests that the increase in the number of columns is in fact only a by-product, not the cause of the reduction in run times.

Having generated smaller, i.e. more flexible, and a larger number of columns, one may suggest that the LP objective function value of the initial RMP is lower, thereby reducing the heading-in effect and thus speeding up the entire CG procedure. Interestingly, we observed only a small average reduction of 1.29% of the initial LP objective function value when compared to not performing preprocessing. More importantly, as Figure 7.8 illustrates, there is no strong correlation between the quality of the initial LP value and the reduction in run time.

![Figure 7.8](image)

Figure 7.8: Relationship of the reduction in LP solution value of the initial RMP and the reduction in run time for the small instances and for different limits on the number of preprocessing iterations. The reductions are with respect to not carrying out preprocessing.

After eliminating these two arguments, the only remaining prominent explanation is that the CG procedure converges more quickly at the root node. Indeed, without exception, we observe this for all instances and all preprocessing limits.
7.2. PREPROCESSING AND HEURISTIC PRICING

Figure 7.9 shows the convergence for instance 5-S-1 when carrying out five iterations of preprocessing compared to not performing any preprocessing. As can be seen, even though the initial LP objective function value was much larger, it reduced more quickly compared to not preprocessing.

In an integer sense, it is plausible that the presence of some smaller columns facilitates finding a solution since the smaller columns “supplement” other columns more effectively. It seems that, at least in our application, this is true for the linear case as well. To support this argument, in the following, we will consider several algorithm settings (initialisation of the RMP and preprocessing) that result in smaller columns and evaluate their impact on the number of CG iterations at the root node. We focus on the aircraft aspect since routes have many more non-zero entries than crew pairings, so the impact of smaller columns should be largest here. Furthermore, since the duration of crew blocks is between one and five days, columns of varying number of non-zero entries are generated regardless.

We consider the columns generated in the first 20 CG iterations, which if executed, may include columns generated in the initialisation method. We focus on the columns of the first 20 CG iterations for two reasons. First, quicker convergences is,
to a large degree, dependent on reducing the heading-in effect. During this phase, the dual values are least stable, which may result in columns that are not very useful later on (Vanderbeck, 2005). Secondly, we will use the mean and standard deviation to measure the difference in the size of paths. Calculating these values with respect to all columns generated at the root node may camouflage what occurs during the heading-in phase since columns generated later are smaller than the ones generated early on: during the heading-in phase, due to the artificial variables, many flights have a high negative cost associated with them. Finding the most negative paths then means that the paths cover a larger number of flights. The instances behaved very similarly, we therefore again only present results for instance 5-S-1.

As a starting point to the discussion, we turn off the initialisation method and instead only use artificial variables to initialise the RMP (see Section 6.3). In the first 20 iterations we observe a mean route size, i.e. flights in route, of 18.17 with a standard deviation of 1.53. A total of 153 CG iterations were required to solve the root node to within an LP gap of 30%. As a reference, the average route size stabilised at about 15 flights some time after the heading-in effect.

Using the initialisation method but not preprocessing resulted in 145 CG iterations with a mean route size of 16.29 and a standard deviation of 2.4. These values are explained by noting that the columns generated in the initialisation method are much smaller (mean is 13.22) than during the first 20 iterations of the CG procedure (16.68). It should be noted that the latter value is lower than the mean in the previous setting, i.e. 18.17. It seems that the smaller columns generated in the initialisation method have a positive impact on the first 20 iterations of the CG procedure.

If, on the other hand, we perform five iterations of preprocessing but do not use the initialisation method, 113 CG iteration were required with a mean size of 17.6 and a standard deviation of 1.88. It seems that preprocessing has a larger impact, i.e. reduction in CG iterations, than the initialisation method, even though the average column size is not reduced as much. Here, the mean size is, however, misleading because that number does not reflect the entire root node: naturally, the initialisation method only generates few columns in the beginning of the algorithm,
and thus has limited effect\textsuperscript{11}, while preprocessing occurs in every CG iteration, thereby reducing column size throughout the entire root node solve. Calculating the mean over only the first 20 iterations does not accurately reflect this.

When using both preprocessing and initialisation, 99 CG iterations resulted, with a mean size of 15.63 and a standard deviation of 2.23. Thus, when combined, the reduction in the number of CG iterations is larger, albeit not by much, than the sum of the reductions of each individual setting. This is not that surprising when considering that preprocessing does have an impact on the initialisation method as well. The size of the columns generated by the initialisation method is further reduced from 13.22 to 12.67, see Figure 7.10.

Figure 7.10: Frequency polygon of the number of non-zero entries of columns generated in our initialisation method for instance 5-S-1 when carrying out preprocessing compared to no preprocessing.

Figure 7.11 shows the frequency polygons for the size of the columns generated in the first 20 CG iterations when using the basic setting, i.e. no preprocessing and no initialisation method, and the setting combining initialisation and preprocessing. To enable a fair comparison, we excluded the columns generated in the initialisation method\textsuperscript{12}. We observe a distribution that resembles a normal distribution. For the

\textsuperscript{11}Limited in the sense that it affects only the generation of columns during the initialisation method. Of course, the columns generated there do have an impact on the entire CG procedure.

\textsuperscript{12}Remember that the columns generated in the initialisation method do have an impact on the
combined setting, the mean is about two flights lower than for the basic setting. Furthermore, it is much closer to the value of 15, at which the column size stabilises after the heading-in phase. Additionally, very few large columns (size larger than 19) are generated. On the other hand, no very small columns (size less than ten) are generated in either of the settings.

To summarise our observations, by using the initialisation method and preprocessing, we generate columns during the heading-in phase that have a size more similar to those generated later at the root node. These columns should therefore be useful throughout the algorithm. The initialisation method itself generates columns that are much smaller than the average column. Our results suggest that both the slightly smaller columns due to preprocessing as well as the much smaller columns generated by the initialisation method allow the CG procedure to pass through the heading-in phase more quickly. We suspect that this is due to the fewer non-zero entries: the reduced cost of a column that enters the basis in the simplex method indicates by how much the objective function value reduces if the variable assumes a value of one (Bazaraa et al., 2008). However, the value of the entering variable is constrained by the interaction with the variables currently in the basis, which, loosely speaking, depends on the number of non-zero entries of the variables. Smaller columns therefore should be more likely to enter the basis at a higher value, thus reducing the LP value more even though their reduced cost are lower. We further hypothesise that the much smaller columns generated by the initialisation method have a positive effect because they allow larger, newly added columns to enter at a higher value as they can complement these columns more easily.

It remains to be shown why smaller columns are generated. We first consider the initialisation method, followed by the analysis of regular CG iterations. Recall that in Algorithm 6.1, two-thirds of the flights have a negative cost of two assigned to them, while the remainder has a positive cost of one. All other cost are ignored. Then, preprocessing removes a large proportion of flights with positive cost by showing that they can not be part of a better path. It is very effective at this due to the bi-polar cost structure. Furthermore, we observe that the vast majority of

---

columns generated in the first 20 iterations. Therefore, even though they are not represented by the polygon, they do have an impact.
7.2. PREPROCESSING AND HEURISTIC PRICING

Figure 7.11: Frequency polygon of the number of non-zero entries of columns generated in the first 20 iterations at the root node for instance 5-S-1.

flights are removed in the first round of preprocessing. This explains why additional preprocessing does not have a more significant impact at this point.

The difference in the column size then is due to the dominance checking in the labelling algorithm. Recall that all non-dominated labels are stored, where dominance is checked with respect to cost and all resources. Thus, due to the difference in resource accumulation, labels with high cost are stored as well. Extending these to the sink node means that columns covering many flights are generated.

Naturally, an optimal path covers a very similar number of flights regardless of whether preprocessing is carried out or not. However, the second and third best paths found often cover fewer flights when performing preprocessing simply because a lower number of flights exist in the remaining networks (recall that we add the three most negative columns for each pricing problem to the RMP, see Section 6.4).

We now turn our attention to the reduction in the column size during the CG procedure. We note that the labelling algorithm (7.1) usually finds many paths. We already pointed out that early in the CG procedure, due to the artificial variables, many flights have a high negative cost associated with them. Choosing the three

\footnote{Degenerate solutions may exist. Otherwise, the same number of flights is covered.}
most negative paths then means that the paths cover a larger number of flights. The preprocessing algorithm, on the other hand, often does not generate many paths. Recall that Algorithm 7.2 can abort after calling FSP(0) (Line 7.2.7). FSP(0) generates minimum forward paths and in doing so may find feasible paths from the source to the sink node. If such a path is optimal, the algorithm aborts. In this case, not many other paths may have been generated. Additionally, the concatenation function usually generates fewer columns than the labelling algorithm, although often more than three. More importantly, however, is that the paths generated in preprocessing are not as good, i.e. have lower negativity\textsuperscript{14}. This is important when the preprocessing algorithm finds the optimal path for the pricing problem. In this case, the optimal path covers the same, i.e. a large, number of flights, however, the second and third best column are smaller compared to the second and third best columns that would be generated by the labelling algorithm. It is these columns that improve the CG convergence.

The foregoing discussion highlights the importance of not finding paths that cover as many flights as possible in the initialisation method. Instead, it may even be advisable to find feasible paths that cover only a very small number of flights. These columns are likely not useful later in the CG procedure but may be very beneficial early on. In general, it may be advisable to not add the most negative column during the heading-in phase to avoid very large columns. Moreover, it may be desirable to deliberately generate very small columns during this phase since only few such columns are generated in the initialisation method, yet, seem to have a positive impact. We did not investigate any of these ideas further in this thesis, but suggest them for further research.

\subsection{Abort Pricing Problem Solving after Preprocessing}

To this point, we have not definitively answered the question of how many preprocessing iterations to perform. From the reduction of overall run times, it is clear that some preprocessing should occur. The time spent in pricing per CG iteration (Table 7.3) indicates that one or two rounds is a good choice. However, what we

\textsuperscript{14}It should be noted that this does not automatically mean that the columns are smaller; they may simply cover more flights that incur positive cost (dual value), although this does not seem to be the case very often.
have not considered yet, and also not fully taken advantage of, is the positive effect that preprocessing has on the concatenation function ConcArc() (Section 7.1.2). We investigate this in the present section.

Recall that the purpose of function ConcArc() is to find feasible paths, which gives an upper bound $c^+$, that is used to accelerate the labelling algorithm (see Section 7.1.3). However, if a feasible path was found, we may also abort solving the pricing problem even though the optimal path has not been found yet (recall that for column generation to proceed, only a negative reduced cost column needs to be added, not necessarily the optimal one). As was discussed in Section 7.1.2, ConcArc() generates feasible paths by concatenating forward and backward paths. When conducting multiple preprocessing iterations, the concatenation function is called more often and, in later iterations, works with forward and backward paths of better quality as arcs and nodes resulting in infeasible or inferior paths have been removed repeatedly. Thus, a larger number of paths and paths that have greater negativity result, which is of importance when aborting pricing problem solving after preprocessing. At this point, we need to state that we are now considering the impact of preprocessing on the entire branch-and-price tree, not just the root node. Generating less negative columns, i.e. smaller ones, may be beneficial early on at the root node but this is most likely not true for the remainder of the CG procedure. After the heading-in phase, the negativity of a column is much less of an indicator for the size of a column. Therefore, more negative columns may enter the basis at a high value as well, thus reducing the LP value more.

Therefore, to come to a final verdict on the number of preprocessing iterations, we conducted experiments to investigate if and based on what criteria we should abort solving after preprocessing. Recall that the preprocessing algorithm returns a lower bound, $c^-$, on the cost of the optimal path. Hence, a measure is available to evaluate the quality of the paths found so far. In the experiment, we aborted solving a pricing problem if the negative reduced cost of the best path found after preprocessing is within $\frac{1}{3}$, $\frac{1}{2}$, and $\frac{2}{3}$ of $c^-$, respectively. Additional strategies tested are to abort after preprocessing whenever a negative cost column is found (setting Neg), regardless of the magnitude, and the previously used strategy (Opt) of aborting only if the preprocessing procedure proves (via lower bound on the minimum cost path, see
Section 7.1.2) that the optimal path was found in preprocessing. These strategies are investigated while performing 1, 2, 3, or 5 rounds of preprocessing. Due to the large amount of data generated in this experiment, we only present the averages of the change (in %) of run time and number of CG iterations. These reductions are with respect to not performing any preprocessing at all. Table 7.6 shows the changes in run times for the entire branch-and-price tree. We observed significant reductions for all settings, which is mainly due to the previously described generation of smaller columns. However, when compared against the reduction of the Opt setting, we see that settings $\frac{1}{3}c^-$, $\frac{1}{2}c^-$, and $\frac{2}{3}c^-$ performed better, although there does not appear to be one that performed better than the others. Strategy Neg also performed better albeit not as much. When additionally considering the number of preprocessing iterations, we see that strategy $\frac{1}{2}c^-$ and performing only one round of preprocessing was the best setting.

<table>
<thead>
<tr>
<th>Prepr Iter.</th>
<th>Opt</th>
<th>$\frac{1}{3}c^-$</th>
<th>$\frac{1}{2}c^-$</th>
<th>$\frac{2}{3}c^-$</th>
<th>Neg</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-29.56</td>
<td>-39.23</td>
<td>-43.07</td>
<td>-40.75</td>
<td>-34.20</td>
</tr>
<tr>
<td>2</td>
<td>-31.18</td>
<td>-38.30</td>
<td>-37.90</td>
<td>-36.93</td>
<td>-32.85</td>
</tr>
<tr>
<td>3</td>
<td>-30.98</td>
<td>-40.55</td>
<td>-38.45</td>
<td>-39.30</td>
<td>-32.47</td>
</tr>
<tr>
<td>5</td>
<td>-27.68</td>
<td>-37.80</td>
<td>-38.80</td>
<td>-38.36</td>
<td>-30.45</td>
</tr>
</tbody>
</table>

Table 7.6: Average change of run time (%) for different heuristic pricing strategies when performing a different number of preprocessing iterations.

As was shown in the first experiment in this section, the reduction in the CG iterations at the root node is the main cause of the lower run times. Table 7.7 shows the average reduction (%) in CG iterations at the root node for the current experiment. The decrease is larger for strategies $\frac{1}{3}c^-$, $\frac{1}{2}c^-$, and $\frac{2}{3}c^-$ when compared to Opt, which is interesting since usually heuristic pricing does not have a positive effect on the convergence of column generation, at least not consistently - which is clearly the case here. The cause is once again generating smaller columns: when aborting without finding the optimal path, not only are the second and third best paths smaller, but so is the best path found. The reason why there is no clear winner among strategies $\frac{1}{3}c^-$, $\frac{1}{2}c^-$, and $\frac{2}{3}c^-$ is that the flexibility of the smaller columns is
off-set by their increasingly smaller negativity. However, this is not significant yet which explains the somewhat inconsistent results. For strategy Neg, on the other hand, this is clearly the case. Even though much smaller columns are generated, the substantially lower negativity results in more CG iterations.

<table>
<thead>
<tr>
<th>Prepr Iter.</th>
<th>Opt</th>
<th>$\frac{1}{3}c^-$</th>
<th>$\frac{1}{2}c^-$</th>
<th>$\frac{2}{3}c^-$</th>
<th>Neg</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>-37.34</td>
<td>-41.89</td>
<td>-38.91</td>
<td>-39.30</td>
<td>-15.22</td>
</tr>
<tr>
<td>5</td>
<td>-36.87</td>
<td>-42.21</td>
<td>-42.01</td>
<td>-39.33</td>
<td>-16.57</td>
</tr>
</tbody>
</table>

Table 7.7: Average change in number of CG iterations (%) for different heuristic pricing strategies when performing a different number of preprocessing iterations.

Nonetheless, for strategy Neg, despite the smaller reduction in CG iterations, we noticed that the overall run times were reduced quite significantly as well (see Table 7.6). This can be explained by noting that less time is spent in pricing. To enable a fair comparison, we again need to avoid distorting the total time spent in pricing by the reduction in CG iterations. Hence, we consider the time spent pricing per CG iteration, more precisely, we present the change (%) compared to not preprocessing. As Figure 7.12 shows, the greatest decreases are for strategy Neg, which is not surprising since this is the most aggressive strategy to abort after preprocessing. The figure also shows that more preprocessing consistently behaves worse, thereby confirming our previous findings.

Table 7.8 shows the percentage of pricing problem solves in which only the preprocessing procedure was executed but not the labelling algorithm. Confirming our previous findings, more rounds of preprocessing resulted in calling the labelling algorithm less often as more problems were solved to optimality in the preprocessing algorithm. Furthermore, we see that the highest ratios were achieved for strategy Neg and that these ratios are comparatively speaking much larger than those for strategies $\frac{1}{3}c^-$, $\frac{1}{2}c^-$, and $\frac{2}{3}c^-$, which explains why the reduction of CG iterations for Neg was quite different to that of the other three strategies.
Figure 7.12: Average change (%) of time spent pricing per CG iteration for different number of preprocessing iterations and aborting criteria. The changes are with respect to not performing preprocessing.

<table>
<thead>
<tr>
<th>Prepr Iter.</th>
<th>Opt</th>
<th>$\frac{1}{3}c^-$</th>
<th>$\frac{1}{2}c^-$</th>
<th>$\frac{2}{3}c^-$</th>
<th>Neg</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>23.97</td>
<td>33.67</td>
<td>37.76</td>
<td>37.69</td>
<td>48.20</td>
</tr>
<tr>
<td>2</td>
<td>26.21</td>
<td>36.62</td>
<td>37.89</td>
<td>38.89</td>
<td>49.10</td>
</tr>
<tr>
<td>3</td>
<td>33.14</td>
<td>39.48</td>
<td>41.29</td>
<td>41.87</td>
<td>51.05</td>
</tr>
<tr>
<td>5</td>
<td>36.99</td>
<td>39.92</td>
<td>41.55</td>
<td>41.67</td>
<td>50.59</td>
</tr>
</tbody>
</table>

Table 7.8: Percentage of pricing problems solves in which only the preprocessing procedure was executed but not the labelling algorithm for different aborting criteria and number of preprocessing iterations.
7.2. PREPROCESSING AND HEURISTIC PRICING

7.2.2 Heuristic Pricing and Quality of Local Lower Bound

When not solving pricing problems to optimality, the optimal negative reduced cost $\bar{c}_a$ and $\bar{c}_b$ of aircraft routing pricing problem $a \in A$ and crew pairing pricing problem $b \in B$, respectively, are not available to calculate the local lower bound $z_{LB}$ (Equation 6.1 in Section 6.6.1). Fortunately, Algorithm 7.2 provides a lower bound $c^-$ on the cost of the optimal negative reduced cost path of each pricing problem. We can use these values to calculate the lower bound. We do, however, expect the lower bound to be inferior. In this section, we investigate the usefulness of this bound with respect to early branching. For this, we compare the lower bounds resulting from using the optimal negative reduced cost and the one resulting from using $c^-$. 

We conducted experiments in which we solve all pricing problems to optimality, i.e. do not abort after preprocessing, unless preprocessing solved the pricing problem to optimality. While doing so, we store the $c^-$ after preprocessing. Then, in every CG iteration $\nu$, we calculate the local lower bound based on $c^-, a \in A$ and $c^-, b \in B$, which we define as $z_{LB}^-\nu$, as well as the local lower bound based on $c^a$ and $c^b$, in the remainder of the section referred to as $z_{LB}^b\nu$.

Until the dual values stabilise, the difference between $z_{LB}^-\nu$ and $z_{LB}^b\nu$ is relatively small compared to the gap between $z_{LP}$ and either of them. Furthermore, the LP gap is very large and is thus not very useful for early branching. Therefore, we only consider the development of the three values for later CG iterations at the root node. Figure 7.13 shows the behaviour for instance 5-S-1 in the last 80 iterations of the root node. We see that at around iteration 110, $z_{LB}^\nu$ stabilises, while $z_{LB}^-\nu$ does not. The reason for this is that preprocessing reports negative $c^-b$ values for most crew blocks $b \in B$, while no negative reduced cost columns exist, i.e. $\bar{c}_b = 0$\(^{15}\). Despite the instability, $z_{LB}^-\nu$ is occasionally close to $z_{LB}^\nu$, e.g. in iterations 118 and 138. In the column generation procedure, we use the upper envelope of the lower bound $z_{LB}^\nu$. Here, the upper envelope on $z_{LB}^-\nu$ is fairly good. However, unfortunately this is not always the case. Figure 7.14 shows a similar graph for instance 3-L-A. We see that $z_{LB}^-\nu$ is much lower than $z_{LB}^\nu$ and only gets close at the very end.

As was pointed out, our interest in $z_{LB}^-\nu$ is motivated by early branching. We

\(^{15}\)The minimum cost path that the preprocessing algorithm reports as $c^-$ is infeasible due to layover requirements.
CHAPTER 7. SOLVING THE PRICING PROBLEMS

Figure 7.13: Development of LP value and local lower bound in the final 80 iterations of the root node when using heuristic pricing compared to optimal pricing for instance 5-S-1.

Figure 7.14: Development of LP value and local lower bound in the final 80 iterations of the root node when using heuristic pricing compared to optimal pricing for instance 3-L-A.
therefore calculate in which iteration several LP gaps (see Section 6.6.1) are reached when using \( z_{\text{LB}}^- \). We compare this to using \( z_{\text{LB}}^* \). Table 7.9 gives the iteration number at which the LP gap is less than several \( \epsilon \) values for each instance. It also gives the number of iterations required to reach optimality (Opt).

<table>
<thead>
<tr>
<th>Instance</th>
<th>( \epsilon = 0.3 )</th>
<th>( \epsilon = 0.2 )</th>
<th>( \epsilon = 0.1 )</th>
<th>( \epsilon = 0.05 )</th>
<th>( \epsilon = 0.02 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>5-S-1</td>
<td>606 109 107 108 115 118 124 137 130 147 153</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5-S-2</td>
<td>121 123 122 128 131 139 140 157 158 187 208</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5-L-1</td>
<td>104 106 107 113 115 121 131 132 140 154 171</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5-L-2</td>
<td>99 104 99 104 106 119 116 123 125 148 157</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3-S-A</td>
<td>58 108 62 108 70 108 75 108 84 108 111</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3-L-A</td>
<td>56 79 57 90 64 99 70 99 76 99 100</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 7.9: Iteration at which certain LP gaps are reached at the root node when using \( z_{\text{LB}}^- \) compared to using \( z_{\text{LB}}^* \). The table also shows the number of iterations required to solve the root node to optimality.

We observe that for instance 3-S-A and 3-L-A, the gaps are reached much later when using \( z_{\text{LB}}^- \). For the other instances, the large gaps, i.e. \( \epsilon = 0.3 \) and \( \epsilon = 0.2 \), are reached in a similar number of iterations, while for smaller gaps, we again observe larger discrepancies.

Since we cannot rely on the quality of \( z_{\text{LB}}^- \), we decided to periodically calculate \( z_{\text{LB}}^* \) in our algorithm. If the relative decrease of \( z_{\text{LP}} \), i.e.

\[
\frac{z_{\text{LP}}^{v-5} - z_{\text{LP}}^{v}}{z_{\text{LP}}^{v}},
\]

over the last five iterations is smaller than 0.1, we perform a *full pricing* step, i.e. all pricing problems are solved to optimality. This gives \( z_{\text{LB}}^* \), and we can calculate the LP gap. If the early branching criteria is not met, we continue the CG procedure at this node. It should be noted that in the next CG iteration, the relative decrease of \( z_{\text{LP}} \) is likely to be less than 0.1 again. To avoid performing full pricing from here on out, we carry out up to ten iterations at the root node (five at non-root nodes) in which we only calculate \( z_{\text{LB}}^- \), i.e. do not perform full pricing. We would like to state
that in the experiments presented in this section, we used this strategy whenever we did not solve all pricing problems to optimality.

7.2.3 Summary of Numerical Experiments

The results of the experiments presented in this section showed that using the preprocessing algorithm (7.2) has a significant impact on the overall run times even though the time spent in pricing is not affected as much. The majority of the reduction is due to fewer CG iterations at the root node. It seems that the increased CG convergence is due to adding columns with fewer non-zero entries during preprocessing as well as during the improved initialisation method.

To take advantage of these columns, we conducted experiments in which we abort solving pricing problems after preprocessing if paths of a certain quality were found. The strategy that we found to work best is to carry out one iteration of preprocessing and abort solving a pricing problem after preprocessing if the reduced cost of the best path is larger than or equal to \( \frac{1}{2} c^- \), where \( c^- \) is the lower bound on the cost of the optimal path.

We investigated the quality of the local lower bound of column generation (\( z_{LB} \)) when solving pricing problems heuristically, i.e. using \( c^- \) instead of the optimal reduced cost of each pricing problem. For some instances, we saw that the resulting bound lead to many more iterations before the early branching criteria was met. We therefore do not solely rely on this bound but periodically solve all pricing problems to optimality. The first time we solve all pricing problems to optimality (full pricing) is when the relative gap (Equation 7.28) is smaller than 0.1. If the LP gap (Equation 6.2) is not within 30\%, we perform additional iterations at the current node. Full pricing is repeated after ten iterations if at the root node, or after five iterations if at any other node.

The majority of heuristic pricing in our algorithm is due to aborting after preprocessing. However, for the remainder of this thesis we also limit the number of labels that can be stored at each node in the labelling algorithm\(^{16}\). Preliminary results showed that ten is an appropriate value. If under this setting no negative cost path is found for any pricing problem, then the labelling algorithm is called again,

\(^{16}\)We did not use this rule in the experiments presented in this section.
this time without the limit. We did not investigate the impact of this strategy separately since the effect on the overall solution time was relatively small because the CG convergence was not affected much.

Many other heuristic pricing strategies are possible. For a survey the reader is referred to (Irnich and Desaulniers, 2005) and Sections 2.5 and 2.6. A strategy that may be of interest to our problem is to only extend along the outgoing arcs at each node that have the most negative cost associated with them. However, we need to consider our problem structure: replenishment arcs have cost associated with them and thus only few of them may be among the most negative cost arcs. Therefore, a more problem-specific variant of this strategy would be to extend along the most negative arc(s), where we chose one or several arcs from each connection type. So for example, we chose the most negative duty connection, regular layover, and long layover for each node. We leave this strategy for future work.
Chapter 8

Selection of Pricing Problems to Solve

In Chapter 5, we introduced a column generation based mathematical model for the integrated aircraft routing, crew pairing, and tail assignment problem. The restricted master problem is solved using our branch-and-price algorithm (Chapter 6), while the pricing problems are solved using the algorithms described in Chapter 7. Thus far, we have shown how solving the LPs, finding integer solutions, and solving individual pricing problems can be improved. What we have not investigated so far is how to deal with the multitude of pricing problems that result from our problem formulation. Apart from long solution times due to solving many pricing problems, a CG procedure usually only benefits little from adding columns from every PP in each iteration. The PPs use the same dual values, and thus tend to produce columns covering the same flights or tasks. However, in an integer solution, often only one aircraft or crew needs to be assigned to a task. Fortunately, similar to heuristic pricing, it is not necessary to solve all pricing problems in every CG iteration, unless LP optimality needs to be proven. This strategy is known as “partial pricing” (Desrosiers and Lübbecke, 2005) and has been shown to be very beneficial when a large number of pricing problems exist.

We devise new strategies for the challenge of solving many pricing problems in the present chapter. The goal is to reduce the time spent in pricing, which we strive to achieve by solving fewer pricing problems per iteration. Thus, at each iteration, we must select the (subset of) pricing problems to be solved. In this chapter, we
focus on strategies for doing so. We must not, however, ignore the effects on run time and convergence of the CG procedure, i.e. number of required CG iterations. We will therefore discuss strategies with respect to these three criteria.

We first give a brief literature review of problems that also give rise to multiple pricing problems and discuss the strategies others have used to address this issue. In Section 8.1, we discuss criteria that can be used to select the pricing problems to solve at each iteration. Experiments in Section 8.2.1 investigate how selecting a single or a subset of pricing problems (PP) to solve per iteration, and adding up to three columns per PP at each CG iteration, affects the branch-and-price algorithm. In Section 8.2.2, we present similar experiments, where this time the maximum number of columns to add per PP is such that the total number of columns that can be added per iteration is comparable to experiments in previous chapters. Then, in Section 8.2.3, we provide experiments in which we find a favourable limit on the number of columns to add per pricing problem. Finally, we summarise our findings and outline further challenges in the final section.

Solving an optimisation problem via column generation, where the formulation results in multiple pricing problems certainly is not new. After all, the original paper introducing Dantzig-Wolfe decomposition (Dantzig and Wolfe, 1960) already discusses the existence of multiple PPs - although they “collapse” into one PP as they are identical (also see Section 2.1.3). Today, many applications solved by column generation have more than one pricing problem. Examples in airline scheduling are the fleet assignment problem, in which usually one pricing problem exists per fleet type, e.g. Papadakos (2009), and the crew pairing problem, where the number of PPs is, amongst others, proportional to the number of crew bases, e.g. Saddoune et al. (2012). However, because these pricing problems represent some form of aggregation, e.g. all crews at a base or all aircraft of a certain type, only few PPs result. The situation is different when it is necessary to schedule at an individual level. For example, crew rostering problems usually give rise to many PPs: preassigned tasks and personal preferences have to be considered for each individual crew member. Gamache et al. (1998) heuristically solve a preferential bidding problem that has up to 108 crew members, and thus PPs. For the same problem, Achour et al. (2007) propose an exact solution method and successfully solve instances of up to 91 crew
members. Gamache et al. (1999) consider a large scale crew rostering problem, with instances of 380 pricing problems. The issue of Multiple PPs is not limited to airline related rostering. For example, Maenhout and Vanhoucke (2010) solve a nurse rostering problem, in which each of the 30 nurses is represented by a pricing problem.

Another example of individual scheduling is the tail assignment problem (Section 3.7). Because the problems are solved very close to the day of operations, routes for each individual aircraft are generated. This often results in one pricing problem per aircraft. Sarac et al. (2006) solve randomly generated instances, in which each of the 32 aircraft is represented by a PP, while Grönkvist (2006) and Gabteni and Grönkvist (2009) solve several instances of up to 33 aircraft.

In (Gamache et al., 1999) the pricing problems are solved in sequential order until a certain number of columns is generated. In the next CG iteration, the procedure continues where it left off. The authors find that adding 15 columns, where each PP adds at most one column, is a good choice. Maenhout and Vanhoucke (2010) use the same sequencing strategy for their pricing problems, although they do not specify how many pricing problems are solved per iteration. Freling et al. (2004) solve airline and railway crew planning problems, in which they solve five or number of crews divided by 20, whichever is greater, pricing problems per CG iteration. Potthoff et al. (2010) reschedule crews after a disruption. In their algorithm, they solve pricing problems sequentially and abort pricing when 30% of all PPs returned columns. Arbib and Marinelli (2009) solve an assortment problem, in which resource types are selected to create batches of product types. Each product type corresponds to a pricing problem. In every CG iteration, only the pricing problems that contributed columns in the previous iteration are solved, unless the dual value associated with the single constraint that restricts the resource types changes, in which case all pricing problems are solved.

In contrast, Michel and Vanderbeck (2012) only solve one PP per CG iteration, unless, of course, it does not return a negative reduced cost column. They solve a tactical inventory routing problem that has multiple pricing problems, which are formulated as multiple-choice knapsack problems. In their algorithm, they abort enumeration of pricing problems as soon as the first negative reduced cost column is
found. As a result only one column is added per CG iteration. The authors report a speed-up factor of 3.4 compared to when searching for the best column.

The majority of the previous references only consider how many pricing problems to solve but not which ones. In crew rostering problems, the pricing problems are often ordered by seniority of crew members, e.g. Gamache et al. (1998). In Sarac et al. (2006), the PP are sequenced in ascending order of remaining flying time of the aircraft. Gabteni and Grönkvist (2009) use a dual re-evaluation method for which they re-order the pricing problems in every CG iteration. In their “randomized ordering scheme”, the PPs are sequenced based on a score, which is a quadratic function of the old position in the sequence and the old score. Ropke and Pisinger (2006) develop an adaptive large neighbourhood search algorithm for a pick-up and delivery problem. The algorithm uses several heuristics, for which the past performance is kept track of using a score that is then used in a roulette wheel selection principle: the probability that a heuristic is called again is equal to its score divided by the sum of the scores over all heuristics. The scores are reset periodically. The roulette wheel selection strategy, also known as fitness proportionate selection, is a well known genetic algorithm selection scheme (Banzhaf and Reeves, 1999). Other related methods include rank selection or tournament selection. These selection schemes depend on fitness/previous performance of the individuals to be selected from a population of individuals. As we will discuss in the next section, it is not obvious how to measure the fitness/performance of a pricing problem.

8.1 Choice of Pricing Problem to Solve

In our problem, much fewer aircraft than crew blocks are available, yet the same number of flights have to be covered. In other words, aircraft are more scarce. Also, some crew blocks represent more crews than others. Therefore, pricing problems should not be considered equally important when generating columns. In this section, we investigate how many, which, and in what sequence we should solve pricing problems.

Before answering these questions, let us first consider the behaviour of the CG procedure when all pricing problems are solved in every iteration. A graphical representation for instance 5-S-1 is given in Figure 8.1, which shows in what iteration
a PP returned columns\(^1\). If a PP returned at least one column, the cell is coloured in dark grey, while if it did not, the cell is in a light grey. To the left of the vertical black line are aircraft routing pricing problems, to the right the crew pairing pricing problems. The latter are ordered by crew block start time \(\tau^b\), \(b \in B\), the ordering of the former does not matter here. The number of CG iterations is shown vertically, with the first being at the top. Note that these iterations are over all nodes that are solved in the branch-and-price tree. However, nodes in which no PP was solved - due to early branching or suppressing the pricing step - are not displayed. We indicate the iterations that are at the root node (top part), the ones that are during the early branching phase, i.e. \(\epsilon > 0\) (middle part, or early branching part), and the ones after \(\epsilon\) is driven to zero, i.e. no more early branching occurs (bottom part).

We observe that, except for after \(\epsilon = 0\), almost all ARPPs generate columns when they are solved. The reason for this are twofold. Firstly, as we already discussed in Section 6.6.1, when \(\epsilon > 0\), a branching decision may result in a fairly large increase in the LP value \(z_{LP}\). Usually, a larger number of columns need to be generated to sufficiently reduce \(z_{LP}\). Furthermore, it seems that the entire solution to the aircraft aspect is affected. Because of the large number of non-zero entries of aircraft columns, even a small change for one aircraft will cascade onto many others. Secondly, since the networks for the ARPPs are almost identical and the dual values tend to be larger after such an inferior branching decision, thereby “attracting” the paths, each one of the ARPPs will generate new columns. Conversely, after \(\epsilon = 0\), when dual values are smoother and smaller, generating columns is more selective, i.e. the pricing problems are not as “eager” to include a flight. Additionally, at this point the algorithm has to prove LP optimality, hence we expect some iterations in which no columns are generated for any PP.

The situation is very different for CPPPs. We see that the crew blocks just to the right of the black line, which are the ones in category Pairing Continues (i.e. are carried in partway through a current pairing), only generate few columns and only

\(^1\)95.42\% of the PPs either returned three or no columns. The intention here is not to identify whether or how many columns an individual PP returned in a specific iteration, but to grasp the general behaviour of the CG procedure. Hence, for the sake of this analysis, we do not differentiate between returning one, two, or three columns. Furthermore, we approximate the number of columns that were generated for a PP by how often the PP returned columns.
Figure 8.1: Graphical interpretation of the CG procedure. Vertically aligned are CG iterations while horizontally are the pricing problems. A dark grey cell signifies that between one and three columns were added for the pricing problem in that CG iteration. Light grey means it was solved but did not return any negative cost columns.
8.1. CHOICE OF PRICING PROBLEM TO SOLVE

early on. The crew blocks in the middle generate the largest number of pairings, especially in the beginning, but also scattered throughout the algorithm. The crew blocks to the right side generate columns mostly in the beginning, and only some later in the tree. The behaviour of the CPPPs can be explained by considering the duration of the crew block, $d_{bt}^b$, $b \in B$, the initial location, and the number of crews represented by the crew block, i.e. $n_b$. Crew blocks in category Pairing Continues have between one and four days left, $n_b$ is small, usually one or two, and they may start at a location from which only few flights depart since they are on a pairing at the beginning of the planning horizon. Hence only a smaller number of pairings needs to be, or even sometimes can be, generated. It seems that the algorithm finds good enough pairings already early on. Crew blocks in the middle and to the right start after the planning horizon. Their initial locations are Melbourne, Sydney, or Brisbane, which is where a large number of flights depart. Thus, more combinations of flights are possible, resulting in a larger number of possible pairings for these crew blocks. Furthermore, they represent up to 18 crews, which means more pairings need to be generated for these crew blocks. The difference between the middle and the right side is that the latter start later in the horizon and therefore cover fewer days, which results in a smaller number of possible pairings.

The branching decisions also have an impact. Consider branching on a crew connection that is early in the planning horizon, say day one. Crew blocks that start later are less likely to be affected and thus may not need to generate new columns after such a branching. Crew blocks in the middle, on the other hand, are affected more often since they span more days.

When comparing ARPPs and CPPPs, it is clear that many more columns are generated for ARPPs than for CPPPs. The aircraft aspect is harder to solve as fewer aircraft are available. Furthermore, it appears that the crew aspect can regain LP optimality much more easily than the aircraft aspect since a significantly smaller percentage of pairings are generated after the root node. We suspect that the reason for this is that, due to the smaller number of non-zero entries for pairings, branching on a crew connection has a much more limited effect, i.e. impacting a smaller number of crew blocks/pairings.

In summary, it is clear that we do not need to solve all pricing problems in every
iteration. Furthermore, ARPPs should be solved more often than CPPPs. For the latter, some PPs are more important than others, which depends on $d_{st}^b$, the starting location, and $n_b$, the number of crews represented by block $b$.

With this information at hand, we can devise appropriate selection strategies. Before stating the one we developed, we provide a discussion on the advantages and disadvantages of some of the influencing factors. Separating ARPPs and CPPPs, and possibly favouring ARPPs, has the advantage that more routes can be generated, as is desired. On the other hand, it is not clear when to solve which type. We found that in our instances, about 60 to 70% of the branchings are on crew connections. Nevertheless, from Figure 8.1 we deduce that even after branching on a crew connection, new aircraft routes need to be generated. Having a fixed ratio of ARPP and CPPP solves means that CPPPs will be solved throughout the algorithm, which may not be desired. On the other hand, the crew aspect of the problem determines the vast majority of the cost. Not generating new pairings when required (due to a fixed ratio) may mean that routes are generated based on the dual values of an LP solution of inferior quality. These routes, while somewhat beneficial at that point, may not be helpful later on. Any such ratio would require fine tuning, even more so if the ratio is to be adjusted dynamically throughout the algorithm in an attempt to remedy some of the challenges outlined here.

Letting $n_b$ influence the choice of CPPP may be misleading, especially after the root node. A branching decision most likely will not affect all pairings that have been generated/selected for a crew block with a large value of $n_b$. Furthermore, crew blocks that start late in the planning horizon have a large value of $n_b$ as well but most likely should not be solved at all if the branching connection is during the earlier days of the planning horizon.

The duration of pricing problems makes the strongest case to be an important factor in the choice of PP. Five-day CPPPs clearly generate more columns than CPPPs of shorter duration. This captures the larger number of possible pairings that result from the combinatorial combinations of flights in a five day period. Furthermore, crew blocks with a long duration are more likely to be affected by a branching decision. To support this argument, we note that there are some crew blocks in the middle part of Figure 8.1 that generate only few columns. These are
crew blocks that start after the beginning of the planning horizon (i.e. starting location is Sydney, Melbourne, or Brisbane) and have a large value of $n_b$. However, they only represent a short work-period of two or three days. This results in fewer possible combinations/pairings (less columns early on) and also not being as affected by branching decisions that are on e.g. the first or last day of the planning horizon (results in fewer columns later on). On the other hand, we once more have to differentiate between ARPPs and CPPPs. Both, the five day CPPPs and the ARPPs cover five days, yet the latter produce a lot more columns.

In addition to these input-dependent factors, we also should consider how a PP performed so far. Do we want to solve a PP again if it has already generated many columns or should it generate fewer? It is possible to argue both ways: if a PP contributed many columns, the RMP seems to have problems assigning the right flights to the aircraft or crew. On the other hand, too many columns for a single PP may not be sensible since the LP size is increased while only one (in case of ARPP) or up to $n_b$ (in case of CPPP) columns can be chosen in an integer solution. We probably need to relate the number of columns generated so far to the number of aircraft or crews represented by the PP. Additionally, such a performance-based strategy inevitably leads to the question of whether we should consider all columns from the beginning of the algorithm or only the ones generated in the last, say ten, iterations.

In the end, we could of course consider all these influencing factors when selecting pricing problems to solve. This would, however, require a lot of fine-tuning and may, despite the effort, still not give a good strategy for some instances.

Instead, we decided to rely on the dual values, as their flexibility is able to reflect the current state of the problem quite well. By setting appropriate dual values for the convexity constraints, the RMP is able to request more columns for specific PPs, including ones with large values of $n_b$. If more routes are required than pairings, the LP solver will reflect this accordingly via the dual values of the aircraft cover constraints (5.1) and crew cover constraints (5.1).

In our strategy, we sort and select the pricing problems based on the reduced cost of the best column found by the PP last time it was solved. This of course means that we are not truly selecting PPs based on the \textit{current} state of the RMP but,
instead, consider its most recent development. For example, if in the last iteration most ARPPs returned negative columns but CPPPs did not, the ARPPs will be chosen again. There is of course no guarantee that additional routes are required - after all, the newly added columns may suffice. Sequencing by reduced cost also captures that PPs spanning more days should generate more columns. If positive dual values, which result in negative reduced cost, are distributed throughout the flight network, a PP spanning more days is likely to generate columns of greater negativity. It thus will be solved more often.

Of course, it would be preferred to consider the true current state of RMP. However, in absence of a crystal ball, this can not be realised as it is unknown which pricing problem is going to find a column. It is, however, possible to consider the current dual value of the convexity constraint: for example, the networks of the ARPPs are almost identical and so the question of which ARPP will generate the most negative columns mainly depends on the dual value of the respective convexity constraint\(^2\). The order of the ARPPs could thus be adjusted by the current dual values of the convexity constraints. We leave this idea for future work.

In our selection strategy, we solve a subset of pricing problems in each iteration, where we solve at least a minimum number \(\Omega\) and at most a maximum number \(\Upsilon\) of pricing problems. If solving the maximum number of pricing problems did not return any negative reduced cost columns, we keep solving the PPs that have not been solved until the first one returns a column. We describe the procedure and its motivation in more detail in what follows.

We first solve all pricing problems and sequence them in a list based on a score \(\Theta_o \leq 0, o \in Q\), where \(Q = A \cup B\). At this point, we set \(\Theta_o\) to equal the negative reduced cost of the best column returned by pricing problem \(o\). If PP \(o\) did not return a column, \(\Theta_o\) is set to equal zero. At the next CG iteration, we solve and remove up to \(\Upsilon\) PPs from the top of the list. If none of these returns a negative reduced cost column, we solve and remove the next PP at the top of the list, repeating until a PP returns a column\(^3\). All solved pricing problems are then slotted back into the

\(^2\) Apart from the starting location and resource consumption at the beginning of the planning horizon.

\(^3\) It is therefore possible to solve more than \(\Upsilon\) PPs. In the following, we will not make this distinction.
8.1. **CHOICE OF PRICING PROBLEM TO SOLVE**

list based on the new $\Theta_o$ values, where, again, $\Theta_o$ equals the negative reduced cost of the best column returned by the PP. If a pricing problem has the same score, the PP to be added is placed after the incumbent PP. Note that this means that a PP that does not return a negative cost column, i.e. $\Theta_o = 0$, is added to the end of the list\(^4\).

It is possible that the same PP will be selected over and over again because it keeps returning columns of large negativity. However, other PPs might return columns as well but are not selected because, at some point, they returned columns of smaller negativity. To avoid this, we multiply the score of all PP that are not solved in the current iteration by 1.1.

Similarly, it is possible that at some point only a few PPs have a negative score, while the majority have a score of zero because, at some point, they did not return a column. This does not necessarily mean that the latter would not generate new columns given the current dual values. Preliminary results showed that it happens quite often in our problem that some PPs have a score of zero, while others continue to have non-zero scores. This is due to the integration of aircraft and crew: after an initial phase, the CPPPs may have generated enough columns, while the ARPPs have not. After several more iterations, the aircraft solution may have changed sufficiently for the RMP to request new pairings. Therefore, in order to not optimise the aircraft aspect completely while (temporarily) neglecting the crew part, we also solve a minimum number, i.e. $\Omega$ of PPs: if the number of PPs that currently have a negative score is less than $\Omega$, the difference is made up of PPs that have a score of zero, where again, we choose from the top of the list, which means we select the ones that have been solved least recently. As before, if these PPs do not return columns of negative reduced cost, they are added to the very end of the list. As a result, we only slowly work through the (sub)list of PPs with a score of zero (if other PPs have a negative score). Furthermore, this guarantees that, when solving to LP-optimality, we only solve every PP once. It should be noted that periodically, in order to evaluate if we can perform early branching, we also solve all pricing problems in the same iteration (see below). This updates/resets the scores and

\(^4\)It should be noted that there is only a single list. At the top, in increasing order of $\Theta_o$, are the PPs that returned a column last time they were solved. These are followed by the PPs with $\Theta_o = 0$, where the PP solved most recently is at the bottom.
thereby shuffles the sequence of PPs.

Similar to our algorithm, Gamache et al. (1999) also perform partial pricing and early branching. Their algorithm aborts if the LP value does not improve sufficiently in a given number of iterations. Such an approach is problematic when performing partial pricing. The LP value may simply not have improved much because the wrong pricing problems were solved. Throughout this thesis, we have used the dual bound resulting from Lagrangian theory as the indicator for early branching (see Equation 6.1 in Section 6.6.1). In Section 7.2, we already discussed the need to perform full pricing in order to calculate a valid, and good, lower bound $z_{LB}$. The motivation there was that the pricing problems need to be solved to optimality. Here, we note that, additionally, we must solve all pricing problems. We use the same strategy as described in Section 7.2: if the relative decrease of $z_{LP}$ over the last five iterations at a node is smaller than 0.1, we perform a full pricing step. If the early branching criteria is not met, we continue the CG procedure at the current node and do not perform full pricing again for ten iterations if at the root node, or five iterations if at other nodes. It should be noted that, unlike in Section 7.2, where $z_{LB}$ is a valid, albeit weaker, local bound, in partial pricing no such bound exists.

8.2 Numerical Experiments

In this section, we present numerical experiments in which we analyse how solving all pricing problems per CG iteration compares to solving only one PP and to solving only a subset. Comparing these strategies needs to be related to the number of columns generated per iteration and pricing problem as this affects the CG convergence and run time. In a first experiment (Section 8.2.1), we keep the maximum number of columns to add per solved PP as is, i.e. three. In Section 8.2.2, we adjust the limit of each PP such that the maximum number of columns added per iteration is constant for all investigated strategies. Then, in Section 8.2.3, we investigate how many columns should be added per PP and CG iteration for each selection strategy. In the ultimate section, we summarise our findings and outline remaining challenges.

Thus far, we always solved all pricing problems in every iteration and added up to three columns per PP. In the following, we will refer to this default strategy as $AllPP+3$. We now investigate how several different selection strategies compare to
8.2. NUMERICAL EXPERIMENTS

AllPP+3. Before presenting results, we introduce some notation. In general, we differentiate between two classes of selection strategies, SglPP and SetPP. In SglPP, we select and solve the PP at the very top of the list, where the ordering in the list is as described in the previous section. If it returns a negative reduced cost column\(^5\), we abort pricing, otherwise we solve the next in sequence. In SetPP, we solve between a minimum (Ω) and maximum (Υ) number of pricing problems. As described in the previous section, we chose up to Υ-many PPs that have a score \(\Theta_o < 0\). If the number of PP in the list with \(\Theta_o < 0\) is smaller than the minimum value Ω, the difference will be made up of PPs with \(\Theta_o = 0\). These minimum and maximum values should depend on the instance size because otherwise, the same strategy may have quite different implications: for example, for an instance with few PPs, a certain strategy may mean that a large portion of the PPs is solved, while for an instance with many PPs, the same strategy only solves a small fraction.

Therefore, for different SetPP strategies, we specify two values, the first is the divisor by which the number of pricing problems has to be divided to get the maximum number of PP to be solved in each iteration, i.e. Υ. The second value gives the divisor that provides the Ω. For example in strategy SetPP/5/10, we have \(\Upsilon = \lfloor |Q|/5 \rfloor\), while \(\Omega = \lfloor |Q|/10 \rfloor\), where \(|Q|\) is the number of PPs in the instance. For convenience, we present Table 8.1, which gives the resulting number of pricing problems for each instance when dividing by a certain value. For example, strategy SetPP/10/20 and instance 7-L-1 results in a maximum number of 14 and a minimum number of five PPs solved per iteration. It should be noted that smaller divisors give a larger number of PPs. A note on the SglPP strategy: we can of course express the strategy as SetPP/\(|Q|/|Q|\), however, to make the distinction more obvious, we chose the notation introduced above.

In this chapter, we will also investigate the number of columns to add per PP and iteration since the optimal such value may depend on the number of PP solved. For each strategy, we will specify the number of columns as follows: SglPP+3 means we add up to three columns for the single PP. Similarly, SetPP/5/10+2 means we add up to two columns for each of the \(\lfloor |Q|/5 \rfloor\) to \(\lfloor |Q|/10 \rfloor\) PPs that are solved.

\(^5\)In the following, when we say “returns a column”, we mean a negative reduced cost column was found.
Table 8.1: Number of resulting pricing problems when dividing by certain divisors for each instance.
8.2. NUMERICAL EXPERIMENTS

8.2.1 Adding up to Three Columns per Pricing Problem

In this experiment, we tested several selection strategies and kept the number of columns to add per PP and iteration as before, i.e. three. For this experiment, we provide an extensive analysis, while for subsequent experiments, to reduce the extent of exposition of numerical results, we simply highlight similar or opposite behaviour.

Table 8.2 shows the solution times for the small instances using different selection strategies while adding up to three columns per PP solve and CG iteration. We see that the SetPP strategies performed slightly better than AllPP+3, while SglPP+3 performed worse for most instances. On average, compared to AllPP+3, we achieved a reduction of 7.3% for SetPP/5/10+3, 10% (SetPP/7/15+3), 8.16% (SetPP/10/20+3), and 2.61% (SetPP/15/30+3), while the increase for SglPP+3 is 27.93%.

<table>
<thead>
<tr>
<th>Instance</th>
<th>AllPP+3</th>
<th>/5/10+3</th>
<th>/7/15+3</th>
<th>/10/20+3</th>
<th>/15/30+3</th>
<th>SglPP+3</th>
</tr>
</thead>
<tbody>
<tr>
<td>5-S-1</td>
<td>383</td>
<td>261</td>
<td>283</td>
<td>292</td>
<td>327</td>
<td>360</td>
</tr>
<tr>
<td>5-S-2</td>
<td>335</td>
<td>344</td>
<td>306</td>
<td>305</td>
<td>318</td>
<td>364</td>
</tr>
<tr>
<td>5-L-1</td>
<td>688</td>
<td>616</td>
<td>596</td>
<td>606</td>
<td>670</td>
<td>1034</td>
</tr>
<tr>
<td>5-L-2</td>
<td>822</td>
<td>653</td>
<td>599</td>
<td>682</td>
<td>694</td>
<td>964</td>
</tr>
<tr>
<td>3-S-A</td>
<td>815</td>
<td>855</td>
<td>852</td>
<td>855</td>
<td>939</td>
<td>1206</td>
</tr>
<tr>
<td>3-L-A</td>
<td>692</td>
<td>769</td>
<td>763</td>
<td>743</td>
<td>739</td>
<td>1036</td>
</tr>
</tbody>
</table>

Table 8.2: Solution times (s) for different selection strategies when adding three columns per pricing problem solve.

When considering just the time spent in pricing, from here on called *pricing time*, we observe significant reductions. Table 8.3 gives the change in percent with respect to AllPP+3. As can be seen, all selection strategies reduced the pricing time, with SetPP/10/20+3 and SetPP/15/30+3 performing better than the others.

The reductions are a direct result of solving fewer PPs (see Table 8.4). Again, SetPP/10/20+3 and SetPP/15/30+3 perform best, on average -55% and -59.9%, respectively, while for the others the average reduction was about 50%.
## CHAPTER 8. SELECTION OF PRICING PROBLEMS TO SOLVE

### Table 8.3: Change in total time spent pricing (in %) for different selection strategies when adding three columns per pricing problem and iteration. The change is with respect to strategy AllPP+3.

<table>
<thead>
<tr>
<th>Instance</th>
<th>SetPP</th>
<th>/5/10+3</th>
<th>/7/15+3</th>
<th>/10/20+3</th>
<th>/15/30+3</th>
<th>SglPP+3</th>
</tr>
</thead>
<tbody>
<tr>
<td>5-S-1</td>
<td>-50.14</td>
<td>-40.28</td>
<td>-46.85</td>
<td>-46.96</td>
<td>-46.16</td>
<td></td>
</tr>
<tr>
<td>5-S-2</td>
<td>-25.28</td>
<td>-31.28</td>
<td>-32.76</td>
<td>-38.95</td>
<td>-38.99</td>
<td></td>
</tr>
<tr>
<td>5-L-1</td>
<td>-41.11</td>
<td>-33.98</td>
<td>-50.12</td>
<td>-56.18</td>
<td>-25.69</td>
<td></td>
</tr>
<tr>
<td>5-L-2</td>
<td>-44.34</td>
<td>-47.49</td>
<td>-46.08</td>
<td>-52.28</td>
<td>-44.09</td>
<td></td>
</tr>
<tr>
<td>3-S-A</td>
<td>-20.76</td>
<td>-33.10</td>
<td>-36.28</td>
<td>-46.46</td>
<td>-23.87</td>
<td></td>
</tr>
<tr>
<td>3-L-A</td>
<td>-16.58</td>
<td>-33.12</td>
<td>-40.08</td>
<td>-41.39</td>
<td>-20.00</td>
<td></td>
</tr>
<tr>
<td>Avg.</td>
<td>-33.04</td>
<td>-36.54</td>
<td>-42.03</td>
<td>-47.04</td>
<td>-33.13</td>
<td></td>
</tr>
</tbody>
</table>

### Table 8.4: Total number of pricing problems solved for different selection strategies when adding three columns per pricing problem and iteration.

<table>
<thead>
<tr>
<th>Instance</th>
<th>SetPP</th>
<th>AllPP+3</th>
<th>/5/10+3</th>
<th>/7/15+3</th>
<th>/10/20+3</th>
<th>/15/30+3</th>
<th>SglPP+3</th>
</tr>
</thead>
<tbody>
<tr>
<td>5-S-1</td>
<td>19815</td>
<td><strong>6101</strong></td>
<td>8532</td>
<td>8421</td>
<td>7377</td>
<td>8337</td>
<td></td>
</tr>
<tr>
<td>5-S-2</td>
<td>20984</td>
<td>11630</td>
<td>10206</td>
<td>10717</td>
<td>9783</td>
<td><strong>9208</strong></td>
<td></td>
</tr>
<tr>
<td>5-L-1</td>
<td>22476</td>
<td>9131</td>
<td>11613</td>
<td>8154</td>
<td><strong>6583</strong></td>
<td>13018</td>
<td></td>
</tr>
<tr>
<td>5-L-2</td>
<td>24728</td>
<td><strong>9865</strong></td>
<td>10623</td>
<td>10507</td>
<td>10111</td>
<td>10909</td>
<td></td>
</tr>
<tr>
<td>3-S-A</td>
<td>17956</td>
<td>11871</td>
<td>9292</td>
<td>8609</td>
<td><strong>6525</strong></td>
<td>8479</td>
<td></td>
</tr>
<tr>
<td>3-L-A</td>
<td>16321</td>
<td>11486</td>
<td>9273</td>
<td><strong>8105</strong></td>
<td>8201</td>
<td>10113</td>
<td></td>
</tr>
</tbody>
</table>
On the other hand, as Table 8.2 shows, the change in overall solution time was nowhere near as favourable. The reason is that the time spent solving LPs was increased or did not change much: the average change compared to AllPP+3 was +1.45% for SetPP/5/10+3, -1.66% (SetPP/7/15+3), +2.84% (SetPP/10/20+3), +12.62% (SetPP/15/30+3), and +51.43 for SglPP+3. The changes in LP solving time impact the overall solution times much more than the reductions in pricing time since in our case, between 67% and 84% of the solution time is spent solving LPs.

The LP times are influenced by contradictory tendencies. On one hand, the size of the LPs was smaller since the total number of columns generated was reduced, ranging from -7.38% to -58.28% (compared to AllPP+3). On the other hand, as is to be expected, solving only a subset of PPs, i.e. partial pricing, does not reduce the LP objective value as quickly, hence, more CG iterations are required (see Table 8.5 and Figure 8.2). Recall that, in our case, solving an LP is computationally expensive, thus the LP times are increased disproportionally when carrying out more CG iterations.

<table>
<thead>
<tr>
<th>Instance</th>
<th>AllPP+3</th>
<th>/5/10+3</th>
<th>/7/15+3</th>
<th>/10/20+3</th>
<th>/15/30+3</th>
<th>SglPP+3</th>
</tr>
</thead>
<tbody>
<tr>
<td>5-S-1</td>
<td>300</td>
<td>380</td>
<td>488</td>
<td>587</td>
<td>706</td>
<td>994</td>
</tr>
<tr>
<td>5-S-2</td>
<td>285</td>
<td>512</td>
<td>549</td>
<td>604</td>
<td>692</td>
<td>929</td>
</tr>
<tr>
<td>5-L-1</td>
<td>212</td>
<td>330</td>
<td>520</td>
<td>462</td>
<td>573</td>
<td>1262</td>
</tr>
<tr>
<td>5-L-2</td>
<td>247</td>
<td>384</td>
<td>466</td>
<td>548</td>
<td>622</td>
<td>1182</td>
</tr>
<tr>
<td>3-S-A</td>
<td>150</td>
<td>378</td>
<td>395</td>
<td>480</td>
<td>553</td>
<td>1190</td>
</tr>
<tr>
<td>3-L-A</td>
<td>135</td>
<td>364</td>
<td>467</td>
<td>464</td>
<td>556</td>
<td>1126</td>
</tr>
</tbody>
</table>

Table 8.5: Total number of CG iterations for different selection strategies when adding three columns per pricing problem and iteration.

An interesting observation can be made when differentiating between the number of iterations at the root node and those at other nodes (Table 8.6). As can be seen, for strategies SetPP, the increase with respect to AllPP+3 was much larger after the root node. The reason is that, naturally, we do not actually know in advance
Figure 8.2: Convergence of LP value at the root node for instance 5-S-1 when using different pricing problem selection strategies while adding up to three columns per pricing problem and iteration.
8.2. NUMERICAL EXPERIMENTS

which PPs to solve. Ordering by previous reduced costs (see Section 8.1) can only
be an imperfect indicator. However, it seems that at the root node, this does not
matter as much. Quite often columns from any, especially any ARPP, will reduce
the LP value. Meanwhile, after the root node, especially deeper in the tree, we need
to solve the right PPs to improve \( z_{LP} \). Solving the “wrong” PPs may result in some
reduction (if they return some negative reduced cost column) but quite often not
enough, which requires additional iterations. SglPP+3 additionally has very poor
convergence at the root node (see Figure 8.2). The culprit here is that the heading-
in phase is much longer. During this phase, many iterations are performed in which
columns are generated that are not very useful later on (Vanderbeck, 2005).

<table>
<thead>
<tr>
<th>SetPP</th>
<th>/5/10+3</th>
<th>/7/15+3</th>
<th>/10/20+3</th>
<th>/15/30+3</th>
<th>SglPP+3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Root Node</td>
<td>61.28</td>
<td>76.17</td>
<td>109.18</td>
<td>160.09</td>
<td>442.78</td>
</tr>
<tr>
<td>After Root Node</td>
<td>108.68</td>
<td>172.72</td>
<td>181.35</td>
<td>222.63</td>
<td>471.94</td>
</tr>
</tbody>
</table>

Table 8.6: Average change in number of iterations (in %) for different selection
strategies when adding three columns per pricing problem and iteration. The change
is with respect to AllPP+3 and is differentiated by whether the iterations occurred
at the root node or after the root node.

Before summarising the results of this experiment, we provide a graphical anal-
ysis similar to the one in Figure 8.1 for strategy SetPP/5/10+3 and, once again,
instance 5-S-1. Figure 8.3 shows in which iteration a PP returned at least one col-
umn (dark grey), was solved but did not return a column (light grey), or was not
solved (white).

In this figure, the iterations in which we solve all PPs are easily identified (grey
horizontal lines). This may be due to performing full pricing (mainly in the top
part) or solving the RMP to optimality (mainly last third). Once again, although
this is somewhat coincidental, a third of the iterations are at the root, a third in the
early branching part, and a third when \( \epsilon = 0 \).

At the root node, after the initial phase to populate the list, the PP solves are
nicely distributed among the ARPPs and CPPPs, where, as desired, the ARPPs are
Figure 8.3: Graphical interpretation of the CG procedure. A dark grey cell signifies that between one and three columns were added for the pricing problem in that CG iteration. Light grey means it was solved but did not return any negative cost columns, while white represents not solved at all.
8.2. NUMERICAL EXPERIMENTS

clearly favoured. We also observe that the CPPPs are not clustered much, i.e. not many in consecutive iterations, then no CPPPs, then again in consecutive iterations, etc.

We observe a total of ten iterations in which we perform full pricing. In terms of early branching, we could have started full pricing later than we did, however, as was pointed out earlier, solving all PPs updates the list, which may be beneficial if otherwise some PPs are not solved for a while (multiplying a very small initial score by 1.1 may not have a sufficient effect). Moreover, full pricing periodically solves the entire crew part, which, at least at the root node, is beneficial since otherwise it may not be updated sufficiently often as the ARPPs return columns that are more negative (as they cover more flights (see Section 6.2)) and are therefore chosen more often.

We observe that the behaviour in the middle section of the early branching part resembles that of the root node. This is not surprising considering what occurred here: the iterations until the second full pricing (second full grey line in the middle part) are all at a single node, which is at a depth of 128. In other words, 128 branching decisions have been made to this point. The last branching decision then resulted in a significant increase in LP value, which triggered an exception to the suppressing rule (see Section 6.6.3). The PPs, here mostly ARPPs, that were affected the most by the branching decision were then solved several times. Once good enough columns have been found, other PPs needed to be solved since a vast part of the solution had changed. This involved solving some CPPPs, although good enough columns were found quickly (note the few dark cells scattered horizontally before the first full pricing). The algorithm then returned to solving ARPPs since the change in the crew part required new routes. The first full pricing reshuffled the list, which resulted in a distribution of PPs similar to that of the root node. ARPPs and CPPPs were solved iteratively, heavily favouring ARPPs, which is as desired. The second full pricing then proved that the LP gap satisfied the current \( \epsilon \)-value.

In the bottom part, where \( \epsilon = 0 \), the algorithm needs to prove LP optimality at a node. We see many iterations in which all or a large number of PPs were solved. It should be noted that while the number of iterations is fairly evenly distributed among the three parts, more PPs are solved in the bottom third. For this instance,
2098 PPs were solved at the root node, 1817 in the middle part, and 4617 in the bottom. On the other hand, 23.99s was spent solving the PPs at the root node, 13.62s in the middle part, and 22.75s in the bottom part. We observed that the distributions of PP solves and pricing time in the three parts is fairly consistent among all instances.

Finally, we provide a brief analysis on where the time is spent in pricing. We begin with AllPP+3 for instance 5-S-1, followed by SetPP/5/10+3 for the same instance. Figure 8.4 shows the time in seconds spent in pricing in each iteration for strategy AllPP+3. We observe somewhat stable times of about 0.3s before the first full pricing spike. In this case, the PPs were either solved in preprocessing or the labelling algorithm found paths while enforcing a limit of ten labels per node. Then, in full pricing, all pricing problems must be solved to optimality, which for some PP required up to 163000 labels. Later in the procedure, where we need to prove optimality again, we also must perform full pricing. However, we do not observe any such spikes. The reason is that at that point, due to the dual values, preprocessing is more effective at showing that no negative cost path exists or, if this fails, is able to remove many arcs and nodes from the network. The labelling algorithm then rarely has to generate more than several hundred to a few thousand labels, which is insignificant. In this instance, the largest number of labels generated in the third part was 12713.

For strategy SetPP/5/10+3 we observe much lower pricing times for most iterations before $\epsilon = 0$ (Figure 8.5). The reason is, of course, that only a small number of PPs were solved in most iterations. We also observe that the full pricing spikes are less pronounced, at least the early ones. For these early iterations, the dual values have not stabilised yet, which resulted in the preprocessing algorithm solving more PPs to optimality. This is mostly true for ARPPs. Finally, when $\epsilon = 0$, we have to prove LP optimality in several iterations. For these, we have to solve all pricing problems, and, as was the case for AllPP+3, we observe pricing times of about 0.3s.

**Summary**

To summarise our findings, when solving only a subset of PPs or only a single PP per iteration, the time spent in pricing can be reduced between 16.58 and 56.18% com-
8.2. NUMERICAL EXPERIMENTS

Figure 8.4: Time spent pricing in each iteration when solving instance 5-S-1 while solving every pricing problems per iteration.

Figure 8.5: Time spent pricing in each iteration when solving instance 5-S-1 while solving only a subset of pricing problems per iteration (strategy SetPP/5/10+3).
pared to solving all pricing problems in every iteration. Strategy SetPP/15/30+3 performed best in this regard with an average reduction of 47.02%. However, when considering the run times, SetPP/7/15+3 and SetPP/10/20+3 were superior with average reductions of 10% and 8.16%, respectively, compared to only 2.61% for SetPP/15/30+3. The impact on the run times was not more significant because the overall LP solution times were not reduced much if at all. It is actually interesting that the LP times did not increase more considering how many more CG iterations are necessary under the selection strategies. Compared to AllPP+3, between 26.67% and 724.07% more iterations were required. Clearly, the reduced number of columns generated overall, ranging from -7.38% to -58.28%, balance the increase in CG iterations. In other words, smaller LPs and adding fewer columns per iteration results in less time required per LP solve, while on the other hand, many more LPs need to be solved.

This experiment further showed that we do not need to generate such a large number of columns as we did under the AllPP strategy. Even though the number of columns was reduced by an average of 43.02%, where the average is over all selection strategies, the average IP gap was 1.24%, while that for AllPP+3 is 1.00%.

At this point, we would like to mention that more substantial reductions in run times may be achievable for other applications. Recall that in our case, the restricted master problem is difficult to solve - partly due to numerical troubles (see Section 6.2). In applications where this does not occur, the decrease in pricing time may have a larger impact.

8.2.2 Fixed Upper Limit on the Number of Columns per Iteration

In the previous section, we did not change the number of columns that may be added per pricing problem and iteration. As a result, a sometimes very small number of columns was added per iteration. For example, under SglPP+3, only up to three columns are added per iteration - except in a full pricing step where we add up to $3|Q|$ columns. As a result, CG convergence suffered significantly. In this section, we set an upper limit on the number of columns added per iteration that is equal to the implicit upper limit under the AllPP+3 strategy. As before, we do not set a global limit but instead set a limit for individual PPs such that the overall limit is equal to
that under AllPP+3. Thus, for the SetPP strategies, the number of columns added per PP is $3|Q|$ divided by the maximum number of PPs solved per iteration, i.e. $\Upsilon$. We will indicate these strategies as for example Set/5/10+$3|Q|/\Upsilon$.

Table 8.7 gives the average reduction in run times, time spent in pricing, and time spent solving LPs for different selection strategies. The reductions are in percent and with respect to AllPP+3. We observed run times that are comparable to that of the experiments presented in Section 8.2.1. Again, SetPP/5/10+$3|Q|/\Upsilon$, SetPP/7/15+$3|Q|/\Upsilon$, and SetPP/10/20+$3|Q|/\Upsilon$ performed better than SetPP/15/30+$3|Q|/\Upsilon$ and especially SglPP+$3|Q|/\Upsilon$. However, we observed much more substantial reductions in pricing time compared to the previous experiment. An additional reduction of 20.01% to 31.53% was achieved. Again, the overall LP times countered these savings. The LP times were increased because, unlike in the previous experiment, more columns were generated overall.

<table>
<thead>
<tr>
<th>SetPP</th>
<th>/5/10</th>
<th>/7/15</th>
<th>/10/20</th>
<th>/15/30</th>
<th>SglPP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run Time</td>
<td>-9.77</td>
<td>-9.61</td>
<td>-7.71</td>
<td>-0.42</td>
<td>24.60</td>
</tr>
<tr>
<td>Pricing</td>
<td>-53.05</td>
<td>-59.99</td>
<td>-64.47</td>
<td>-67.82</td>
<td>-64.67</td>
</tr>
<tr>
<td>LP Time</td>
<td>2.87</td>
<td>4.89</td>
<td>9.12</td>
<td>20.08</td>
<td>53.06</td>
</tr>
<tr>
<td>Iter</td>
<td>24.01</td>
<td>25.05</td>
<td>29.07</td>
<td>43.02</td>
<td>148.34</td>
</tr>
<tr>
<td>Columns</td>
<td>56.47</td>
<td>59.23</td>
<td>67.57</td>
<td>84.09</td>
<td>122.52</td>
</tr>
</tbody>
</table>

Table 8.7: Average change in run time, time spent in pricing, and time spent solving LPs for different selection strategies. All changes are in percent and with respect to strategy AllPP+3. In each of the strategies shown in the table, up to $3|Q|/\Upsilon$ columns were added in each pricing problem.

The most surprising result is the increase in columns. Remember that the change is with respect to AllPP+3, i.e. all pricing problems are solved in each iteration. Intuitively, one would not expect more columns, when solving only a limited number of PPs, unless, the number of iterations is increased drastically - which is not the case here. The reason for the larger number of columns is that under AllPP, the implicit upper limit on the total number of columns added per iteration is rarely
CHAPTER 8. SELECTION OF PRICING PROBLEMS TO SOLVE

tight. Apart from the beginning, the CPPPs hardly generate any columns. Thus, at
least at the root node, for most iterations, the number of columns added per iteration
tends to be more like $3|A|$. After the root node, the behaviour is less predictable as
even ARPPs do not always generate columns; usually though, we observe an average
that is far less than even $3|A|$. Under the SetPP strategies, on average, we generate
more columns per PP and iteration. If $3|A|$ is larger than the maximum number
of pricing problems allowed to solve, $\Upsilon$, we generate a total of $\Upsilon \cdot 3|Q|/\Upsilon = 3|Q|
columns in iterations in which only ARPPs are solved, where, for this argument, we
assume that each ARPP returns the maximum number of columns. We observed
that the average number of columns per iteration was increased by between 29.85%
and 37.71% for the SetPP strategies when compared to AllPP+3.

Of course, it is possible to additionally impose a global limit on the number of
columns generated, where this limit is smaller than $3|Q|$ or even $3|A|$. This would,
however, not allow a valid comparison, unless such a limit is used in strategy AllPP
as well. Additional experiments would be required to identify a satisfactory global
limit. We leave these experiment for future work.

In summary, adding up to $3|Q|/\Upsilon$ per PP, results in significant reductions in
pricing time. For the strategies tested here, the average reductions with respect
to AllPP+3 were between 53.05% and 67.82%, where the latter is for strategy
SetPP/15/30+3$|Q|/\Upsilon$. The strategies required an increasing number of CG iter-
ations and columns, which resulted in more time spent solving LPs. The increases
correlate to the number of PPs solved per iteration, the fewer the larger the in-
creases. The additional time spent solving LPs is the reason why the reductions
in pricing time do not translate to larger savings in run time. With respect to the
latter, the best strategies are SetPP/5/10+3$|Q|/\Upsilon$ and SetPP/7/15+3$|Q|/\Upsilon$ with
reductions of 9.77% and 9.61%, respectively, when compared to AllPP+3.

We also attempted to reverse-engineer the number of columns to add per pricing
problem in a way that the average number of columns per iteration is actually similar
to that of AllPP+3. We already noted that the number of columns added during an
iteration after the root node is highly unpredictable. However, even at the root node,
the number of columns added is highly irregular as many columns are added early on
and then again during full pricing. Short of adjusting the number of columns to add
per PP dynamically, we did not succeed in achieving an average number of columns per iteration that enables a satisfactory comparison. We therefore abstained from presenting results for this experiment.

### 8.2.3 Optimal Number of Columns added per Pricing Problem for Different Selection Strategies

In the previous two sections, we tested several selection strategies to evaluate how many pricing problems should be solved per CG iteration. To enable a fair comparison to the strategy used in previous chapters, i.e. strategy AllPP+3, the maximum number of columns added per PP in those experiments was set to be commensurate with the respective strategy. In these experiments, we covered the two extreme cases, where in one we added very few columns per CG iteration, while in the other, we added many. In this section, we investigate what the best limit on the number of generated columns per PP is for the different selection strategies.

In this experiment, we selected the three strategies that performed best in the previous two experiments, i.e. SetPP/5/10, SetPP/7/15, and SetPP/10/20 and varied the maximum number of columns to add per PP, testing each value in \{3, 5, 10, 15, 20, 30\}. Additionally, we tested strategy SglPP while adding up to \{10, 25, 50, 100, 150\} columns for the single PP. We included the latter strategy because in the previous experiments, the limit on the number of columns constituted two very extreme cases. In Section 8.2.1, adding only up to three columns in the majority of iterations (except for full pricing), resulted in very slow CG convergence, while in Section 8.2.2, adding up to 3|Q| columns seemed to be too many columns for a single PP.

The present experiment resulted in a large amount of data. In this section, to make the results more digestible, we only give averages and present these using diagrams. Detailed results can be found in Appendix A. To make the diagrams more readable, we introduce a key for each setting, see Table 8.8.

Figure 8.6 shows the average change (%) in run time (top), pricing time (middle), and LP time (bottom). The changes are with respect to AllPP+3. We observe fairly consistent behaviour when comparing the SetPP strategies with each other. In general, SetPP/10/20 performed better than SetPP/7/15, which again was superior
### Table 8.8: Keys to identify SetPP and SglPP strategies.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Key</th>
<th>Setting</th>
<th>Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>SetPP/5/10+3</td>
<td>a</td>
<td>SetPP/7/15+3</td>
<td>g</td>
</tr>
<tr>
<td>SetPP/5/10+5</td>
<td>b</td>
<td>SetPP/7/15+5</td>
<td>h</td>
</tr>
<tr>
<td>SetPP/5/10+10</td>
<td>c</td>
<td>SetPP/7/15+10</td>
<td>i</td>
</tr>
<tr>
<td>SetPP/5/10+15</td>
<td>d</td>
<td>SetPP/7/15+15</td>
<td>j</td>
</tr>
<tr>
<td>SetPP/5/10+20</td>
<td>e</td>
<td>SetPP/7/15+20</td>
<td>k</td>
</tr>
<tr>
<td>SetPP/5/10+30</td>
<td>f</td>
<td>SetPP/7/15+30</td>
<td>l</td>
</tr>
<tr>
<td>SetPP/10/20+3</td>
<td>m</td>
<td>SglPP+10</td>
<td>s</td>
</tr>
<tr>
<td>SetPP/10/20+5</td>
<td>n</td>
<td>SglPP+25</td>
<td>t</td>
</tr>
<tr>
<td>SetPP/10/20+10</td>
<td>o</td>
<td>SglPP+50</td>
<td>u</td>
</tr>
<tr>
<td>SetPP/10/20+15</td>
<td>p</td>
<td>SglPP+100</td>
<td>v</td>
</tr>
<tr>
<td>SetPP/10/20+20</td>
<td>q</td>
<td>SglPP+150</td>
<td>w</td>
</tr>
<tr>
<td>SetPP/10/20+30</td>
<td>r</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
8.2. NUMERICAL EXPERIMENTS

to SetPP/5/10. The differences were, however, rather small. In terms of run time, all settings, except when adding up to 30 columns, performed better than AllPP+3. Adding 30 columns performed worse because the LP times increased substantially for this setting. For the run time, we see that adding up to five, ten or 15 columns was more beneficial than adding three or 20 columns. In general, the run time for SetPP/10/20 was superior compared to the other two SetPP strategies because the LP times were more favourable. Additionally, the pricing time was reduced slightly.

When considering the run time, strategy SglPP was clearly inferior to AllPP+3 and all SetPP strategies, regardless of the number of columns added. SglPP was, however, comparable if not better than the best SetPP strategy when considering the pricing time. Unfortunately, the LP times were increased substantially for all settings of SglPP.

Figure 8.7 shows the average change (%) in the number of iterations and the average change (%) in the number of columns generated for the same experiment. Again, the changes are with respect to AllPP+3. In general, Strategy SetPP/10/20 required more iterations than SetPP/7/15 and SetPP/5/10, while, as expected, all of them required more than strategy AllPP+3.

The number of columns generated for the SetPP strategies was very consistent when considering different limits on the number of columns to add. Smaller limits resulted in reductions, while larger values gave increases. When comparing the SetPP strategies, it shows that the fewest columns were generated for SetPP/10/20, although the differences are not very large.

We observe that the difference in the reduction of columns between the SetPP strategies was not as distinctive as the difference in the number of iterations (compare e.g. setting a, g, and m). Therefore, in addition to fewer columns, another reason must exist why SetPP/10/20 required more iterations. Recall that e.g. SetPP/5/10 solves more PPs per iteration than SetPP/10/20. It therefore has a higher chance of solving the PP that gives the best negative reduced cost column at any given iteration. This results in quicker convergence, i.e. fewer iterations. In other words, the columns generated are more useful. Our experiments confirm this. For example, for instance 5-L-2, the average reduced cost of the columns generated in strategy SetPP/5/10+3 was -671, while for SetPP/10/20+3 the cost was -477.
Figure 8.6: Average change (%) in run time (top), pricing time (middle), and LP time (bottom) for different selection strategies as identified by Table 8.8. The changes are with respect to AllPP+3
Figure 8.7: Average change (%) in the number of iterations (top) and average change (%) in the number of columns generated (bottom) for different selection strategies as identified by Table 8.8. The changes are with respect to AllPP+3.
For strategy SglPP, we observe that the number of columns generated is very similar to SetPP/10/20. Yet, it required a drastically larger number of iterations. Again, we see that the columns generated are not as useful, leading to more iterations. For the same instance, the average reduced cost of the columns generated in SglPP+10 was only -299.

In summary, the SetPP strategies behaved very similarly and gave impressive reduction with respect to pricing time, where adding up to 15, 20 or 30 columns per PP performed best with an average reduction of 54.69%. Adding this many columns does, however, have a negative impact on the LP solution times. Therefore, in our case, strategy SetPP/10/20+5 performed best in terms of run time with a reduction of 15.07%. Overall, strategy SetPP/10/20 required more CG iterations compared to SetPP/5/10 and SetPP/7/15, while the number of columns was slightly in favour of the former. By far, strategy SglPP performed worst in all criteria except for pricing time and number of columns generated, for which it was comparable to SetPP/10/20.

8.2.4 Summary of Results

In this chapter, we presented experiments in which we tested several strategies that only solve one or a subset of PPs. We compare the resulting performance of the branch-and-price algorithm to that when using the strategy employed in previous chapters, i.e. solving all PPs in every CG iteration.

In the first experiment we investigated how several selection strategies behave when adding up to three columns per PP. We observed substantial reductions in the time spent in pricing, with average reductions between 33.04% and 47.04%. Unfortunately, these gains were offset by increases in LP solution times, which, most of the time, were actually fairly small but, at least in our case, are more impacting on run times than pricing time. Overall, we found that selecting a subset of PPs, where the maximum number of PPs solved per iteration is equal to \(|Q|/7\) and the minimum number is equal to \(|Q|/15\), i.e. strategy SetPP/7/15+3, performed best in terms of run time. The average reduction with respect to the default strategy AllPP+3 was 10%. The selection strategies resulted in a much larger number of CG iterations, with increases between 26.67 and 724.07% when compared to AllPP+3.
On the other hand, the number of columns generated overall was reduced by between 7.38% and 58.28%.

In the second experiment, we add up to a number of columns per PP so that the total number of columns that can be added per iteration is comparable to strategy AllPP+3. We observe savings in pricing time that are even larger than in the first experiment. The average reductions range between 53.05% and 67.82%. Unfortunately, adding a large number of columns per PP results in more columns generated overall, which increases LP solution time, offsetting the gains in pricing time. With respect to run time, the best strategies are SetPP/5/10+3|Q|/Υ and SetPP/7/15+3|Q|/Υ with reduction of 9.77% and 9.61%, respectively.

In the final experiment, we investigated several limits on the number of columns to add per PP in order to find good such limits. We observed average reductions in pricing time of between 33.04% and 58.4%. The average reductions in LP time were reduced slightly for a few settings, while for most, it was increased again. We found that strategy SetPP/10/20+5 performed best in terms of run time with a reduction of 15.07%. The pricing time for this setting was reduced by 53.19%. Smaller limits (>10) reduce the number of columns generated, while larger limits (>15) result in an increase. In the remainder of this thesis, we will use strategy SetPP/10/20+5 whenever we solve a subset of pricing problems.

In all three experiments we found that solving only a single PP per iteration is a clearly inferior strategy, regardless of the number of columns added. While the reductions in pricing time are excellent, the excessive increase in the number of CG iterations increases the overall LP solution times significantly and thus offsets the gains in pricing time.

Overall it seems that selecting the pricing problems to be solved based on reduced cost (see Section 8.1) seems to work well. We saw that most of the time the selected PPs are distributed nicely among ARPPs, i.e. we avoid solving the same PPs over and over again. CPPPs are solved less often, which is as desired since they do not need to generate as many columns.

Figure 8.5 showed that the majority of the remaining time spent in pricing is spent in iterations in which we perform full pricing or attempt to prove LP optimality. Avoiding these spikes is a major motivation for the following chapter.
CHAPTER 8. SELECTION OF PRICING PROBLEMS TO SOLVE
Chapter 9

Superimposed Pricing Problems

In the previous chapter, we developed strategies that successfully address the challenges arising from the multitude of pricing problems in our integrated problem. In the strategies developed there, we select pricing problems based on a score, which depends on the reduced cost of the best path found last time the pricing problem was solved. This means that the choice of pricing problem does not in fact depend on the current state of the RMP. Additionally, we found that while for most iterations, the pricing time was reduced significantly, some iterations, namely those in which we had to perform full pricing or prove LP optimality, still required solving all PPs and were thus computationally expensive.

In this chapter, we develop special pricing problems, each of which can be solved instead of several original pricing problems. We call these new pricing problems superimposed pricing problems (SUPP) as they represent aggregations of the networks of the original pricing problems (OPP). By solving SUPPs, we address both of the remaining challenges: (i) solving a SUPP implicitly solves the original pricing problems that are represented by the SUPP, and thus avoids having to select specific pricing problems (although we do combine the two ideas later in this chapter, i.e. we select and solve a subset of the SUPPs), and (ii) solving fewer SUPPs avoids the excessive pricing time when solving all OPPs in iterations where we have to perform full pricing or prove LP optimality.

Superimposed pricing problems fulfill the same function as the original pricing problems. They either generate at least one feasible negative reduced cost column or prove that no such column exists: as we will describe in subsequent sections,
CHAPTER 9. SUPERIMPOSED PRICING PROBLEMS

the cost of the most negative reduced cost column obtained from solving a SUPP to optimality is equivalent to the cost of the most negative reduced cost column generated by the entire set of OPPs represented by the SUPP. We cannot, however, guarantee that the most negative column for each of the represented OPPs is found, in fact, this is rather unlikely.

The structure of this chapter is as follows. We first introduce superimposed pricing problems for aircraft (Section 9.1), followed by superimposed pricing problems for crew blocks (Section 9.2). We then provide a brief discussion about the types of columns generated by SUPPs as well as the impact on the lower bound of the LP objective function value when using SUPPs in column generation (Section 9.3). We provide extensive numerical experiments in Section 9.4. Finally, we summarise our findings in Section 9.5.

9.1 Superimposed Pricing Problems for Aircraft

In this section, we describe superimposed pricing problems that may be solved instead of multiple original aircraft routing pricing problems. We first describe how the network of such a superimposed pricing problem for aircraft (ASUPP) is constructed, followed by an illustration of how paths for different aircraft are generated in the labelling algorithm when solving a ASUPP. We list conditions that have to be met to aggregate aircraft routing pricing problems in an ASUPP. Finally, we discuss limitations of ASUPPs.

As discussed in Section 5.2.1, the original ARPPs are very similar. The sets of nodes representing flights, i.e. $N^a, a \in A$, are almost identical except for some flights early in the planning horizon: consider a flight that departs Melbourne for Sydney at 5 am on the first day of the planning horizon. A node representing this flight will be in the network of a PP representing an aircraft that starts the planning horizon in Melbourne. However, no such node will be in the networks of aircraft that start at other locations (unless an earlier flight or sequence of flights exist that allow the aircraft to be flown to Melbourne to operate the 5 am flight). In the following, we will use the expression early nodes to refer to nodes that represent flights that cannot be operated by all aircraft to be combined in an ASUPP because the aircraft cannot reach the flight departure location before the departure time.
We recall that all aircraft are of the same type, no aircraft needs to return to a specific location at the end of the planning horizon, and all aircraft become unavailable at the same time, i.e. at the end of the planning horizon\(^1\). This implies that if two aircraft are able to operate a flight (here: reach the departure location before the departure time) and a legal sequence of connections to a subsequent flight exists, then both aircraft are able to operate the subsequent flight. Hence, apart from the early nodes, sets \(N^a, a \in A\), are identical.

It follows that the sets of flight connections are identical as well (except for connections involving early nodes). Moreover, the activities that can be carried out along the flight connections are identical for all aircraft (since they are of the same fleet type). When assuming that the arcs representing these activities are indexed in the same order, we have that the cost and resource usage along arc \((i, j, g) \in E^a\) are identical for all \(a \in A\) as they depend on the flights, not the aircraft: the cost \(c_{ijg}\) depends on the maintenance cost of the connection represented by the arc and the dual value \(\gamma_j\) of the flight \(j \in N^a\) following the connection. The resource consumption depends on the flying time and the time between flights. All these values are independent of the aircraft. The same argument can be made for arc sets \(E^{-a}\) which are identical for all \(a \in A\), except if some of the early nodes in the planning horizon connect directly to sink node \(t^a\).

The main difference between the networks of the ARPPs \(a \in A\) are the source node \(s^a\) and source arcs \((s^a, j, g) \in E^{+a}\), where \(E^{+a}\) was defined as the set of source arcs of \(a\). The source node represents the initial location of the aircraft, while the source arcs identify flights that depart from that location. The resource usage along a source arc accounts for the resource accumulation of the aircraft prior to the planning horizon, which may differ by aircraft. Additionally, the cost of a source arc not only represents possible maintenance cost and the dual value of the flights following the arc but also reflects the dual value \(\alpha^a\) associated with the convexity constraint of \(a\) (Constraint (5.4)).

Figure 9.1 shows the hypothetical networks of three aircraft, AC1, AC2, and AC3, the first two starting the planning horizon in Melbourne, the third in Brisbane.

\(^1\)Unless the aircraft has a heavy maintenance check during the planning horizon. This case is similar to the crew pairing pricing problems (see following section).
In this example, we only consider the reduced cost and a single resource. Notice how for the first and second aircraft, the only difference is the cost and resource usage of the source arcs. Otherwise, the networks are identical, including cost and resource usage on any arc other than the source arcs. The third network differs from the other two in the flight that can be reached via the source arc. We again see that the cost and resource usage along the source arc is aircraft-specific.

Now, when superimposing the first and second network, we arrive at the network shown in Figure 9.2. In this network, the global source node \( s \) represents \( s^1 \) and \( s^2 \), while the global sink node \( t \) represents \( t^1 \) and \( t^2 \). All identical nodes and arcs are represented by a single node or arc, respectively. We see that the only difference to the original two networks is the parallel source arcs. Notice how there is one arc for each original source arc and that the cost and resource usage on this arc is as before.

When superimposing all three original networks, we arrive at the network depicted in Figure 9.3. Notice how the source node \( s \) now also represents \( s^3 \), although the third aircraft starts at a different location. However, this is accurately modeled by only connecting \( s \) to Node 2 via the single source arc \((s, 2, 1)\), which corresponds to aircraft AC3.

Let us now consider how a labelling algorithm works on this aggregated network. A label \( L = (c_i^\alpha, w_1^\beta, \text{Aircraft ID}) \) at node \( i \) is extended according to the resource extension function described in Section 7.1.1. In this example, we will use a label notation that includes the aircraft number, i.e. \( L = (c_i^\alpha, w_1^\beta, \text{Aircraft ID}) \). This is purely for illustration purposes\(^2\). The aircraft number is not a resource and dominance is not applied to it. The algorithm starts with a default label \((0, 0, -)\) at the source node. This label is then extended along all outgoing source arcs. Therefore, at Node 1, two labels exist, \((-100, 90, AC1)\) and \((-90, 80, AC2)\). Extending from \( s \) to Node 2, we get a label \((-30, 200, AC3)\) on Node 2. From Node 1, we extend to Node 4 and get labels \((-110, 95, AC1)\) and \((-100, 85, AC2)\), while extending from Node 1 to Node 3 gives labels \((-120, 98, AC1)\) and \((-110, 88, AC2)\). Extending the single label at Node 2 to Nodes 3 or 4 gives labels that are dominated by the existing labels at those nodes. Hence, no new label is added. Finally, extending from Node 3 to \( t \)

---

\(^2\)In our implementation, we do not actually store the aircraft ID with the label. When converting a path to a column, we can always identify the correct aircraft by the first arc in the path.

\(^3\)Notice how the individual aircraft is identified by the source arc along which we extended.
Figure 9.1: Networks of three different aircraft routing pricing problems. The initial location of two aircraft (top and middle) is Melbourne (MEL), while for the third (bottom), it is Brisbane (BNE).
Figure 9.2: Aggregated network of aircraft AC1 and AC2. The arcs leaving source node $s$ will identify the respective aircraft.

Figure 9.3: Aggregated network of aircraft AC1, AC2, and AC3. The arcs leaving source node $s$ will identify the respective aircraft.
we get (-120, 98, AC1) and (-110, 88, A2). Similarly, extending from 4 to t, we get (-110, 95, AC1) and (-100, 85, AC2). As described in Section 7.1.1, we do not check dominance at the sink node because all of these paths represent feasible routes and simply storing and converting them to columns is computationally inexpensive. As a result, we generate two paths (s, 1, 3, t) and (s, 1, 4, t) for each aircraft AC1 and AC2, while we do not generate any paths for aircraft AC3.

This example highlights some of the strengths and weaknesses of the superimposed pricing problems. Not generating columns for an aircraft may be desired if the columns have negative reduced cost much closer to zero than those of the columns of other aircraft. Thus dominance in the SUPP acts naturally as a pricing problem selection mechanism based on the (current) reduced cost. On the other hand, generating no negative reduced cost column for an aircraft while some exist, may not be beneficial either.

The other major drawback is that for some aircraft, negative reduced cost columns may be generated in many iterations, i.e. the aircraft’s PP is very often “selected” in the SUPP. The reason is that dominance is checked with respect to cost and resource accumulation. In the example above, a label for aircraft AC2 will not be dominated by a label for aircraft AC1 or AC3 (regardless of the current values of the dual variables) since it always has the lowest resource consumption. Thus, the algorithm will generate a column for the second aircraft in every CG iteration in which the dual value $\alpha^{AC2}$ is such that a negative cost path results. For aircraft AC1, on the other hand, columns are only generated if the dual values of the convexity constraints are such that for cost of the source arcs, we have $c_{s11} < c_{s12}$ because otherwise, the labels of aircraft AC1 are always dominated by those of aircraft AC2. It should be noted though that the networks in the example above do not contain any replenishments. Once resources are reset by a replenishment, the order of dominance may be changed. If for example the arc between Node 1 and Node 4 is a replenishment for the single resource, extending labels (-100, 90, AC1) and (-90, 80, AC2) along this arc gives (-110, 5, AC1) and (-100, 5, AC2), in which case the first label dominates the second.

In summary, we require several conditions to be fulfilled when aggregating aircraft routing pricing problems in a superimposed pricing problem in order to ensure
that any feasible solution to the aggregated problem corresponds to a feasible solution for some aircraft and that no minimum reduced cost feasible solution for an individual pricing problem can have lower reduced cost than the minimum reduced cost value of the aggregated problem.

- Apart from the nodes early in the planning horizon, the node sets representing flights have to be identical.

- For any flight node, and any aircraft with that flight node in its OPP, the arc sets out of that node must be identical. The cost and resource usage along these arcs must be identical as well.

- The aircraft, and thus paths, do not have to end at a specific location. An exception is if all aircraft that are aggregated in the SUPP end at the same location.

- The time at which aircraft become unavailable is identical (but not necessarily when they become available).

- Maintenance costs are not dependent on the aircraft (otherwise, the assumptions for applying dominance are violated).

Features that may apply without limiting our ability to aggregate OPPs are:

- The beginning of the networks can be different. This includes the source nodes (start time and location), source arcs, the cost of the source arcs, the resource usage of the source arcs, and nodes early in the planning horizon.

- The cost associated with maintenance may depend on the resource accumulation of an aircraft.

- The set of resources considered for the aircraft may be different (as will be discussed below).

Before stating more precise conditions for superimposing networks of ARPPs, we repeat some notation that is of central importance at this point.

- \( C \subseteq N \times N \) denotes the set of all possible connections between flights to be considered.
9.1. SUPERIMPOSED PRICING PROBLEMS FOR AIRCRAFT

- $N^a \subseteq N$ is the set of flights to be considered in the OPP for $a \in A$.

- $C^a \subseteq (N^a \times N^a) \cap C$ is the set of connections to be considered in the OPP for $a \in A$.

- $N^{+a} \subseteq N^a$ is the set of “first flights” for aircraft $a \in A$, i.e. those reachable from the position of aircraft $a$ at the start of the planning period.

- $N^{-a} \subseteq N^a$ is the set of “last flights” for aircraft $a \in A$, i.e. those that can end any sequence of flying planned for aircraft $a$.

- $(\hat{N}^a, \hat{E}^a)$ denotes the directed multigraph for the OPP of aircraft $a \in A$, with nodes $\hat{N}^a = \{s^a\} \cup N^a \cup \{t^a\}$, where $s^a$ and $t^a$ are dummy source and sink nodes, respectively, and $\hat{E}^a$ is the set of directed multiarcs, with each multiarc of the form $(i, j, g)$ where one of $i = s^a$ and $j \in N^{+a}$, $j = t^a$ and $i \in N^{-a}$, or $(i, j) \in C^a$ hold, and $g \in G_{ij}^a$, the index set of multiarcs from $i$ to $j$ for aircraft $a$, in particular $G_{ij}^a = \{1, 2, \ldots, |G_{ij}^a|\}$.

- In the case that $(i, j) \notin (\{s^a\} \times N^{+a}) \cup C_n \cup (N^{-a} \times \{t^a\})$ we define $G_{ij}^a = \phi$.

**Definition 9.1.1** A set of aircraft $\tilde{A} \subseteq A$ is superimposable if for every pair of distinct aircraft $a, a' \in \tilde{A}$, $a \neq a'$, the following two properties hold:

(i) for all $i \in N^a \cap N^{a'}$, and for all $j \in N$, $G_{ij}^a = G_{ij}^{a'}$, and for all $g \in G_{ij}^a = G_{ij}^{a'}$ we have that $c_{ij}^{ag} = c_{ij}^{a'g}$ and the vector $u_{ij}^{ag} = u_{ij}^{a'g}$, i.e. $u_{ij}^{ak} = u_{ij}^{a'k}$ for all $k \in K$, and

(ii) $N^{-a} \cap N^{a'} = N^{-a'} \cap N^a$, and furthermore for all $i \in N^{-a} \cap N^{-a'}$, $G_{it}^a = G_{it}^{a'}$, and for all $g \in G_{it}^a = G_{it}^{a'}$, we have that $c_{it}^{ag} = c_{it}^{a'g}$ and (componentwise) $u_{it}^{ag} = u_{it}^{a'g}$.

A consequence of Properties (i) and (ii) above is that if flight $i$ occurs in both of the OPP networks for two distinct aircraft $a$ and $a'$, then for any path in the OPP network for $a$ from $i$ to the sink node $t^a$ there is a path in the OPP network for $a'$ from $i$ to the sink node $t^{a'}$ using an identical sequence of flight nodes and having identical cost and resource accumulation vector.
In order to give a formal definition, we introduce some notation for superimposed pricing problems. Let $\iota$ be an ASUPP and $A^\iota$ be the set of OPPs represented by $\iota$. Then for an ASUPP $\iota$, we have network $G^\iota = \{\hat{N}^\iota, \hat{E}^\iota\}$ as follows.

**Definition 9.1.2** The superposition network for a superimposable set of aircraft $A^\iota \subseteq A$ is defined to have

- nodes $\hat{N}^\iota = \{s^i\} \cup N^i \cup \{t^i\}$ where $N^i = \bigcup_{a \in A^i} N^a$ is the set of flight nodes and $s^i$ and $t^i$ are the dummy source and sink nodes, respectively;
- first flights $N^{+i} = \bigcup_{a \in A^i} N^{+a}$;
- for every source arc in the network of each OPP, we have one source arc in the superposition network. Formally, we have source arcs $E^{+i} = \{(s^i, i, g) : i \in N^{+i}, g \in G_{s^i i}^a\}$ where $G_{s^i i}^a = \{1, 2, \ldots, \sum_{a \in A^i} |G_{s^i i}^a|\}$ and costs $c_{s^i i g}^a$ and resource vectors $u_{s^i i g}^a$ defined so that for each $i \in N^{+i},$

\[\{(c_{s^i i g}^a, u_{s^i i g}^a) : g \in G_{s^i i}^a\} = \bigcup_{a \in A^i} \{(c_{s^i i g}^a, u_{s^i i g}^a) : g \in G_{s^i i}^a\},\]

i.e. for every source arc in the superposition network for $A^i$ there is a source arc in the OPP network for an $a \in A^i$ having the same flight node, cost and resource vector, and vice versa, where we write $\alpha(i, g) \in A^i$ to denote the aircraft with a source arc having flight node $i$ and the same cost and resource vector as $(s^i, i, g) \in E^{+i}$ and write $g^+(i, g) \in G^{\alpha(i, g)}_{\alpha(i, g) i}$ to denote the multiarc index of that source arc in the network for $\alpha(i, g)$;
- flight connections $C^i = \bigcup_{a \in A^i} C^{\alpha}$;
- flight multiarcs $E^i = \bigcup_{a \in A^i} E^{\alpha}$ with multiarc index set

\[G_{ij}^a = \{1, \ldots, \max_{a \in A^i} |G_{ij}^a|\}\]

for each $(i, j) \in C^{\alpha}$ and with cost and resource vector $(c_{ij g}^a, u_{ij g}^a) = (c_{ij g}^a, u_{ij g}^a)$ for any $a \in A^i$ with $(i, j) \in C^{\alpha}$, for each $g \in G_{ij}^a$ (which is well defined since $A^i$ is superimposable and hence satisfies Property (i), so all aircraft $a \in A^i$ with $(i, j) \in C^{\alpha}$ have identical multiarc index sets and identical arc cost and
resource vectors and those aircraft \( a \in A^i \) without \((i, j) \in C^a \) have empty multiarc index sets);

- last flights \( N^{-\i} = \bigcup_{a \in A^i} N^{-a} \);

- sink arcs \( E^{-\i} = \{(i, t^\i, g) : i \in N^{-\i}, g \in G^a_{i, t^\i}\} \) defined similarly to flight connections, i.e. where

\[
G^a_{i, t^\i} = \{1, \ldots, \max_{a \in A^i} |G^a_{i, t^\i}|\}
\]

for each \( i \in N^{-\i} \) and with cost and resource vector \((c^a_{i, t^\i, g}, u^a_{i, t^\i, g}) = (c^a_{i, t^\i, g}, u^a_{i, t^\i, g})\) for any \( a \in A^i \) with \( i \in N^{-a} \), for each \( g \in G^a_{i, t^\i} \) (which is well defined since \( A^i \) is superimposable and hence satisfies Property (ii)); and

- all arcs \( \hat{E}^i = E^{+i} \cup E^{-i} \cup E^{-\i} \).

Note that before solving the constraint shortest path problem in the superposition network, any source arc in the superposition network that is dominated by the cost and resource vector of another with the same flight node can safely be removed by preprocessing, however for ease of manipulation of multiarc indices, we assume these are kept.

We now state a central proposition that, for the purpose of column generation, enables us to solve \( \text{ASUPP} \) \( \i \) instead of all individual OPPs \( a \in A^i \).

**Proposition 9.1.3** Let \( A^i \subseteq A \) be superimposable and let \( q^* \) be a minimum cost feasible path in its superposition network with cost \( \bar{c} \) and arcs \((i_\i, i_{\i+1}, g_\i) \in \hat{E}^i \) for \( \kappa = 0, 1, \ldots, g \) with \( i_0 = s^i \) and \( i_{\i+1} = t^i \). Then there exists an \( a \in A^i \) for which the arc sequence \((s^a, i_1, g_0^a), (i_1, i_2, g_1), \ldots, (i_\i, t^a, g_\i)\) is a feasible path in its OPP network, for some \( g_0^a \in G^a_{s^a i_1} \). Furthermore \( \bar{c} = \min_{a \in A^i} \bar{c}^a \) where \( \bar{c}^a \) denotes the optimal value of the OPP for \( a \).

**Proof** We will prove the first part by induction. We first observe that by the construction of the superposition network for \( A^i \), the aircraft \( a = \alpha(i_1, g_0) \in A^i \) has source arc with index \( g^+(i_1, g_0) \in G^a_{s^a i_1} \) in its OPP network having cost and resource vector identical to that of \((s^i, i_1, g_0)\). We will now show that the arc sequence
\[
(s^a, i_1, g_0^a), (i_1, i_2, g_1), \ldots, (i_\i, t^a, g_\i)
\]

is a path in the OPP network for \( a = \alpha(i_1, g_0) \) having identical cost and resource consumption as \( q^* \), hence we may take \( g_0^a = \)
$g^+(i_1, g_0)$. We thus have the first step of the induction: we have that for this choice of $a \in A'$ and $g_0$, the path $(s^a, i_1, g_0^a)$ from $s^a$ to $i_1$ in the OPP network for $a$ has identical cost and resource consumption vector as does $q^*$ up to node $i_1$. Our inductive assumption is that, for some $\kappa \in \{1, \ldots, q\}$, the arc sequence \((s^a, i_1, g_0^a), \ldots, (i_{\kappa-1}, i_\kappa, g_{\kappa-1})\) is a path in the OPP network for $a$ from $s^a$ to $i_\kappa$ and has identical cost and resource consumption vector as $q^*$ up to node $i_\kappa$. Thus $i_\kappa \in N^a$. Now by Property (i) in the case $\kappa < q$ (and superimposability of $A'$), as well as by construction of the superposition network for $A'$, it must be that $(i_\kappa, i_{\kappa+1}) \in C^a$, and furthermore that $g_\kappa \in G_{i_\kappa,i_{\kappa+1}}^a$ with arc $(i_\kappa, i_{\kappa+1}, g_\kappa) \in E^a$ in the OPP network for $a$ having identical cost and resource vector to arc $(i_\kappa, i_{\kappa+1}, g_\kappa) \in E^a$ in the superposition network. In the case that $\kappa = q$, since $i_\kappa \in N^{q'}$ it must be that for some $a' \in A'$, $i_\kappa \in N^{a'}$. It follows from Property (ii) (and superimposability of $A'$) that since $i_\kappa \in N^{a'} \cap N^a$ it must be that $i_\kappa \in N^{a}$. Thus by construction of the superposition network for $A'$ it must be that $g_\kappa \in G_{i_\kappa,t^a}^a$ with arc $(i_\kappa, t'^a, g_\kappa) \in E^{-a}$ in the OPP network for $a$ having identical cost and resource vector to arc $(i_\kappa, t'^a, g_\kappa) \in E^{-a}$ in the superposition network. Thus by induction on $\kappa$, the arc sequence $(s^a, i_1, g_0^a), (i_1, i_2, g_1), \ldots, (i_q, t'^a, g_\kappa)$ is a path in the OPP network for $a = \alpha(i_1, g_0)$ with identical cost and resource consumption as $q^*$. Since $q^*$ is feasible with identical resource consumption vectors on every arc, this arc sequence in the OPP network for $a$ must also be feasible.

A consequence of this first part of the proof is that $\bar{c}^a \geq \bar{c}^{\alpha(i_1, g_0)} \geq \min_{a \in A'} \bar{c}^a$. To prove the final part of the proof (equality in this relationship), we show that for every $a \in A'$ and every path in the OPP network for $a$ from $s^a$ to $t^a$ there is a path in the superposition network for $A'$ from $s'$ to $t'$ having identical flight node sequence and identical sequence of arc costs and resource consumption vectors.

Let the arc sequence $(s^a, i_1, g_0^a), (i_1, i_2, g_1), \ldots, (i_q, t'^a, g_\kappa)$ represent a feasible path $q_a$ in the network of OPP $a \in A'$. Since $N^{a} \subseteq N^{+a}$, we have that node $i_1 \in N^{+a}$. Given the construction of index set $G_{s'i_1}^a$, there exists exactly one arc $(s'^a, i_1, g_0^a) \in E^{+a}$ corresponding to the first arc $(s^a, i_1, g_0^a) \in E^{+a}$ of path $q^a$ (which is the arc for which $g^+(i_1, g_0) = g_0^a$). Furthermore, this arc has the same cost and resource vector as arc $(s^a, i_1, g_0^a)$, i.e. $c_{s'i_1} = c_{s'11}^{s'1i, g_0}$ and (componentwise) $u'_{s'i_1} = u_{s'1i, g_0}^a$. Now given that $C^a \subseteq C'$ and the definition of set $E'$, which implies that the mul-
9.1. SUPERIMPOSED PRICING PROBLEMS FOR AIRCRAFT

tiarc sets \( G^a_{ij} \) and \( G^a_{ij} \) must be identical for \((i,j) \in C^a\), we have that for each arc \((i_{\kappa-1},i_{\kappa},g_{\kappa-1}) \in E^a\), for all \( \kappa \in \{2,\ldots,\varrho\} \) of path \( q^a \), there must exist an arc \((i_{\kappa-1},i_{\kappa},g_{\kappa-1}) \in E^a\) with identical cost and resource vector. The same argument can be made for \((i_\varrho,t^a,g_\varrho) \in E^{-a}\) since \( N^{-a} \subseteq N^{-t} \) and \( G^a_{i_\varrho t^a} = G^a_{i_\varrho t^a} \). Therefore we have that the arc sequence \((s^t,i_1,g_0),(i_1,i_2,g_1),\ldots,(i_\varrho,t^a,g_\varrho)\) constitutes a feasible path in the network of ASUPP \( t \) and has the same cost and resource consumption vector as \( q^a \). Hence, since all feasible paths in the networks of all OPPs \( a \in A^t \) are also feasible in the superposition network and have the same cost and resource consumption, we have \( \bar{c}^a \leq \bar{c}^a \) for all \( a \in A^t \) and the result follows.

It follows that we can prove no negative reduced cost column exists for any aircraft in \( A^t \) for the price of one ASUPP solution.

**Corollary 9.1.4** \( \bar{c}^j \geq 0 \) if and only if \( \bar{c}^j \geq 0 \), for all \( j \in A^t \).

In what follows, we explain how aircraft with quite different maintenance requirements may still be aggregated in the same ASUPP. In Section 6.1.2, we described how preprocessing can eliminate resources from individual ARPPs if it can be shown that the corresponding limits can never be exceeded. In the individual ARPP, this can be viewed as setting the start values of these resources to zero and never increasing the accumulation of the resources in question during the labelling algorithm (see Section 7.1.5).

We can use this observation when aggregating ARPPs. For example, as Table 6.4 shows, instance 5-S-1 contains five aircraft that may require maintenance checks S1 and M1. The same instance contains four aircraft that may require S2 and M1. Then, in a SUPP that represents all of these nine aircraft, a label \( L \) is a 4-tuple, i.e. \( L = (\bar{c}_L^1, \bar{w}_L^1, \bar{w}_L^2, \bar{w}_L^5) \), where resource 1 corresponds to S1, resource 2 to S2, and resource 5 to M1. For the first five aircraft we set \( \bar{w}_L^2 = 0 \), while for the others \( \bar{w}_L^1 = 0 \). As before, dominance is checked with respect to all resources. Then, the labels for an aircraft in the first group may not only dominate the labels for aircraft of the same group but also those of the second. Hence, a smaller number of labels may result. Note how combining these aircraft in a single SUPP never results in a number of non-dominated labels that is larger than the sum of the number of non-dominated labels over the individual aircraft. Of course, more memory is required to store the labels and comparing the labels is computationally more expensive.
In our implementation, however, we elected not to aggregate aircraft that have different maintenance requirements. Initial experiments showed that, apart from the slightly increased computational effort, aggregating ARPPs too much at times results in not generating many columns (due to dominance). This proved to be detrimental to the CG convergence (see Section 9.4). Instead, we aggregate according to the common maintenance requirements (6.4). So for instance 5-S-1, one SUPP, represents the seven aircraft requiring just M1, another SUPP represents all five aircraft that require S1 and M1, and so forth. As a result, the largest number of resources, including cost, considered in any ASUPP in our instances is three. In summary, a total of five ASUPPs are generated in every instance, except for 5-S-1 and 5-S-2, which do not have any aircraft that require maintenance S4 and M1. For these instances, the aircraft can be aggregated to only four ASUPPs. We would like to emphasise this point: regardless of the number of aircraft in an instance, we can always solve all of them implicitly by solving at most five ASUPPs.

9.2 Superimposed Pricing Problems for Crew Blocks

In comparison to the ARPPs, the crew pairing pricing problems are more dissimilar. Apart from their different initial locations and initial resource consumptions, the crews also have to return to their crew base and have to do so before a certain time expires, which we defined as the crew block end time $\tau$ (see Section 5.2.2). We need to consider these differences when aggregating CPPPs in superimposed pricing problems for crew blocks (CSUPP). We first address the issue of different crew bases, followed by varying crew block end times.

Since the sink node $t^b$ represents the crew base, we cannot simply represent all crew blocks in one CSUPP as they may result in infeasible pairings. To illustrate why, let us consider the example in Figure 9.4. The top part shows the network of a crew block CB1, for which the starting location is Perth (PER) and the crew base is Brisbane (BNE). The optimal path is $(s^1, 1, 4, t^1)$ and has reduced cost of -50. The second path in the network $(s^1, 3, 4, t^1)$ is dominated by the first at Node 4.

The middle part of the figure shows the network of crew block CB2, which starts in and is based at Melbourne (MEL). The only path through the network has a reduced cost of +205. Hence, this pricing problem does not generate any negative
Figure 9.4: The top part of the figure shows a crew block CB1 that starts the planning horizon in Perth and must return to Brisbane. The middle part shows a crew block CB2 that starts and ends in Melbourne. The bottom part shows an aggregated network in which the sink nodes are not aggregated.
The bottom shows the aggregated network of these two pricing problems. As for the ASUPP, we can aggregate $s^1$ and $s^2$ in the global source node $s$. The source arcs will correctly identify the crew block, connect to the correct flights, and set the current resource consumption. It should be noted, however, that the sink nodes are not aggregated. The reason for this will become apparent as we discuss a labelling procedure on this network. As in the previous example, we again encode the crew block ID in the label although this is not necessary. Starting with a default label $(c_i, \text{Crew Block ID}) = (0, -)$ and extending along all source arcs, we get label $(-60, \text{CB1})$ at Node 1, $(-100, \text{CB2})$ at Node 2, and $(-50, \text{CB1})$ at Node 3. By extending from Node 1 to Node 4, we get label $(-50, \text{CB1})$. Extending from Node 2 to Node 3 gives label $(-95, \text{CB2})$ which dominates the label currently stored at Node 3. Then, extending this new label to Node 4 gives $(-85, \text{CB2})$, which dominates label $(-50, \text{CB1})$. The label is deleted and is thus not available for extension to sink node $t^1$. Now, $(-85, \text{CB2})$ is not allowed to be extended to $t^1$ because the pairings of crew block CB2 have to end in Melbourne, not Brisbane. On the other hand, extending from Node 3 to Node 4 and further to $t^2$ correctly gives a path of cost +205, which does not result in a negative cost column. We see that this network does not generate any negative cost column at all, although a negative cost path exists in the top part of the network. In conclusion, we should not have extended any label for crew block CB2 to Node 4 or, in general, to any node from which we cannot reach the corresponding sink node. This information, however, is not available in the network.

We could, of course, run a procedure to identify all sink nodes that can be reached from every node and then store this information with the node. Unfortunately, the issue is even more complicated. Crew blocks have different end times, which means that the same crew base may be reachable from a certain node for a crew block that has a late end date but not for one with a sooner end date. Therefore, we would have to store for each node which crew block can reach its crew base in the remaining available time. This is, however, computationally expensive since it amounts to finding a path that is feasible with respect to all resources for every crew block from every node in the network to the respective sink node.

We thus chose to aggregate by crew base: any crew block for which the repre-
9.2. **SUPERIMPOSED PRICING PROBLEMS FOR CREW BLOCKS**

Presented crews are based at the same crew base are aggregated in one CSUPP. Note that this includes crew blocks that start at different locations\(^4\). The CSUPP then will only include sink arcs that start at flights that terminate at the crew base. Since the only crew bases in our instances are Sydney, Melbourne, and Brisbane, we always form three CSUPPs in every instance.

When aggregating CPPPs, we must also respect the time periods during which they are available. As an illustration, let us consider Figure 9.5, which shows two crew blocks that are both based in Melbourne but have different periods of availability. The first (top part of the figure) is available during the first two days of the planning horizon, while the second (middle part) starts in the morning of Day 2 and ends in the evening of Day 3. As can be verified easily, no negative cost path exists for either of the crew blocks.

The bottom part of the figure shows the aggregated network. Again, the source arcs identify the crew block for which a path would be generated. Now, as this network illustrates, a label at Node 2 corresponding to path \((s, 1, 2)\) could be extended to Node 3 and further to sink node \(t\), resulting in a path with a negative cost of -8. However, this is not a feasible path. According to the initial arc, the path is generated for crew block CB1, which must not have any flights on Day 3.

Even if a crew block exists that spans these three days, the path would not be correct. Let a third crew block have a source arc \((s, 1, 2)\) between \(s\) and Node 1 with an arc cost of +15, otherwise the network is identical to the one in the bottom of the figure. Then, the path \((s, 1, 2, 3, t)\) is feasible, however, its cost is -3 not -8 since we must use \((s, 1, 2)\), not \((s, 1, 1)\) as only the former corresponds to the third crew block. The issue now is that the labels of the correct path \((s, 1, 2, 3, t)\) will be dominated by those of the infeasible version at Nodes 1 and 2 since the cost of the latter are smaller. As a result, this aggregated network would not generate any feasible negative cost path even though one exists.

We can avoid this issue by not aggregating any CPPPs that have different end times (as we showed for ASUPPs, different start times are permissible). The networks then implicitly will make sure that any path has the correct duration. How-

\(^4\)Recall that, due to the rolling planning horizon, some crews may start the planning horizon at an airport other than their crew base.
Figure 9.5: The top part of the figure shows a crew block CB1 that spans the first two days in the planning horizon. The middle part shows a crew block CB2 that covers Day 2 and Day 3 of the same planning horizon. The bottom part shows an aggregation of the networks.
ever, this does not result in a much smaller number of pricing problems, especially when additionally considering the different crew bases. Instead, we introduce a single resource that, loosely speaking, keeps track of the remaining available time of a crew block and thereby forbids extension to nodes that are beyond the crew block end time. The definition of this resource is slightly cumbersome, we therefore first introduce the concept in less formal terms.

Consider a resource for which the initial accumulation of each crew block is zero and that is increased along arcs according to the actual duration of the connection and the flight represented by the arc. If the resource has a limit equal to the duration of crew block $b$, then no label corresponding to $b$ would be extended past the crew block end time because this would violate the resource limit. For example, in the aggregated network in Figure 9.5, a path $(s, 1, 2)$ corresponding to crew block CB1, would not be extended to Node 3 as this flight is after the crew block end time. The issue now is that the limit depends on crew block end time, and thus on each crew block, which would require modifications to our pricing problem solver. However, we can avoid this by setting a resource start value for each crew block that is equal to some large value $LV$ minus the duration of the crew block. Then, as before, the resource is accumulated along all arcs but forbids extension to nodes if limit $LV$ is exceeded.

Formally, we define resource 7 to capture this “remaining available time”. Then, for source arcs $(s, j, g) \in E^+$, we define $u_{sjg}^7 = LV - d_{st}^b + u_{sj}^7 + u_j^7$, where $d_{st}^b$ is the duration of the crew block that arc $(s, j, g)$ corresponds to, $u_{sj}^7$ is the duration of connection $(s, j)$, and $u_j^7$ is the duration of flight $j$. We define $LV$ to be the maximum over the $d_{st}^b$ values of all crew blocks represented by the CSUPP, for our instances, $LV$ usually is five days. It should be noted that any large value suffices as long as it is larger than the maximum duration of the crew blocks represented by the CSUPP. Then, the usage\(^5\) along an arc $(i, j, g) \in E$ is $u_{ijg}^7 = u_{ij}^7 + u_j^7$. Along sink arcs $(i, t, g) \in E^-$, we define $u_{itg}^7 = 0$. The resource limit is $U^7 = LV$. Then, in

\(^5\)It is possible to not use the actual duration of connections and flights but instead only consider the time slices in which crew blocks start or end. In our problem, a time slice is one day, thus, an arc can increment the resource accumulation for every midnight it crosses. Of course, the resource start value and $LV$ have to be set accordingly. It should be noted that this does not have an influence on the number of labels generated in the labelling algorithm.
CHAPTER 9. SUPERIMPOSED PRICING PROBLEMS

As an illustration, let us once more consider the example introduced above, which we now extend to show resource 7 (Figure 9.6). In this example, we will use minutes as the timing counter and use a value $LV = 3 \text{ days} \times 1440 = 4320$. Flight 1 departs at 11:40 am, i.e. 700 minutes after midnight, and has a duration of 60 minutes. Connection (1,2) has a duration of 1280 minutes, which means Flight 2 departs at 10:00 am, or 600 minutes after midnight. The flight has a duration of 100 minutes. The duration of connection (2,3) is 1000 minutes, and Flight 3 has a duration of 100 minutes. Then, since CG1 has a duration of two days (2880 minutes), the start value of resource 7 for crew block CB1 is $4320 - 2880 = 1440$, that for crew block CB2 is $4320 - 2880 = 1440$, while for crew block CB3, we have $4320 - 4320 = 0$. For the resource usage on arc $(s, 1, 1)$, which corresponds to crew block CB1, we have $u^7_{s11} = LV - d^1_{st} + u^7_{s1} + u^7_t = 4320 - 2880 + 700 + 60 = 2200$. Notice how, for these calculations, we assume the crew block to become available at midnight, hence the 700 minutes until the start of Flight 1. Similarly, we have for $u^7_{s12} = 4320 - 4320 + 700 + 60 = 760$, where the arc corresponds to crew block CB3. Arc $(s, 2, 1)$ corresponds to the second crew block and connects to a flight on the second day. Remember that crew block CB2 starts on the second day. Here we again start counting from midnight on, however, from midnight of the second day. Hence we have $u^7_{s21} = 4320 - 2880 + 600 + 100 = 2140$. Then, in a labelling algorithm we start with the default label $(0, 0, -)$ and extend along all source arcs. At Node 1, this gives two labels, $(10, 2200, CB1)$ and $(15, 760, CB2)$. Notice how now, unlike in the previous example, the first label does not dominate the second because the accumulation of resource 7 is lower for the second label (because it can be extended further in the network). Extending both labels to Node 2, we get $(2, 3580, CB1)$ and $(7, 2140, CB2)$, which are added to the previously existing label $(11, 2140, CB3)$\(^6\). The label is correctly dominated by $(7, 2140, CB2)$ and is therefore deleted. It should be noted, however, that, due to resource 7, the deleted label is not dominated by $(2, 3580, CB1)$. Then, the label for the first crew block cannot be extended to Node 3 because $U^7 = LV = 4320$ is exceeded otherwise. Label $(7,$

\(^6\)Notice how label two and three have the same accumulation of resource 7. This is by design since both corresponding crew blocks are available until the end of Day 3, hence have the same “remaining available time”.

9.2. SUPERIMPOSED PRICING PROBLEMS FOR CREW BLOCKS

Figure 9.6: The figure shows an extension to Figure 9.5. In this CSUPP, three crew blocks of varying duration are represented. Resource 7 was introduced to forbid generating pairings for crew blocks that include flights which depart after the crew block end time.

2140, CB2), on the other hand, can be extended to Node 4 and then \( t \), giving the optimal path of cost -3.

It should be noted that, for our instances, due to resource 7, no path of duration longer than five days is generated, even if the CSUPP spans more than five days.

In summary, similar to the ASUPP, several conditions and criteria have to be fulfilled when aggregating pricing problems in a CSUPP.

- Crew blocks may have different start times and initial locations. This is reflected by the source arcs which are crew block dependent. The early nodes, that can be reached from the source node, may therefore be different.

- The crews have to return to their bases. Hence, the paths must end at the sink node representing the correct base. We fulfill this implicitly by only aggregating crew blocks that have the same crew base.

- The time at which crew blocks become unavailable may be heterogeneous, in which case we introduce an additional resource to the CSUPP. The resource
prevents generating paths for a crew block that include flights that are past
the crew block end time.

- No flight can be assigned to an individual crew block, unless an entire sequence
  of flights is assigned as well. The sequence has to either start with the source
  node or end with the sink node. This in effect moves the start or end time,
  respectively, as well as the start or end location, respectively, of the crew block.

- The arc sets between flight nodes has to be identical. The cost and resource
  usage along these arcs must be identical as well.

- The sets of resources considered for the crew blocks aggregated in a CSUPP
  are, in our case, identical, although this is not required, as was shown for the
  ASUPP.

- Replenishment costs are not dependent on the crew block. However, as be-
  fore, the replenishment costs do depend on the resource accumulation, e.g.
  credit accumulation (the difference is that the cost is path but not crew block
  dependent).

We define additional notation before stating two propositions. Let $\delta$ be a CSUPP
and $B^\delta$ be the set of OPPs represented by $\delta$. Furthermore, let $\bar{c}^\delta$ be the cost of the
most negative reduced cost column generated by CSUPP $\delta$.

Then, if the original pricing problems $B^\delta$ represented by CSUPP $\delta$ fulfill the
preceding conditions, we arrive at the following proposition.

**Proposition 9.2.1** If CSUPP $\delta$ is solved to optimality, we have $\bar{c}^\delta = \min_{j \in B^\delta}(\bar{c}^j)$.

It follows that we can prove no negative reduced cost column exists for any crew
block in $B^\delta$ for the price of one CSUPP solution.

**Corollary 9.2.2** $\bar{c}^\delta \geq 0$ if and only if $\bar{c}^j \geq 0$, for all $j \in B^\delta$.

### 9.3 General Discussion and Lower Bound

In the previous two sections, we described how any number of ARPPs can be ag-
gregated in at most, in our case, five aircraft SUPPs. Similarly, we aggregate any
number of CPPPs to just three CSUPPs. To respect the different periods for which crew blocks are available, we had to introduce an additional resource in the CSUPPs. In this section, we discuss some of the advantages and disadvantages of superimposed pricing problems.

The most obvious advantage is that we only need to solve eight SUPPs instead of $|Q|$-many original pricing problems whenever we must solve all pricing problems (full pricing or proving LP optimality). It remains to be seen if this indeed saves time as each SUPP may require more time to solve.

An issue when solving many OPPs in one iteration is that, due to the dual values, many columns may be generated that represent the same route or pairing. Solving SUPPs reduces this phenomenon: as stated before, the information which aircraft or crew block a label corresponds to is not stored with the label. Therefore, at any node in the network, the number of labels that represent identical paths is determined by how many non-dominated such labels exist. For example, in an ASUPP that has only one resource (and the single cost), at most two labels representing identical paths can exist (one having the minimum cost value, the other the minimum resource accumulation). This is, of course, not true if more than one resource is considered. In the worst case scenario, the number of non-dominated labels representing identical paths at a node in the SUPP is the same as the number of non-dominated labels representing identical paths at the corresponding nodes when considering all individual represented OPPs. However, this is rarely the case.

Due to the convexity constraints, we may only generate columns for few represented OPPs when solving a SUPP. The advantage is that the RMP does not get overloaded with too many columns. The disadvantage is that we may need more CG iterations as the currently dominated columns may prove to be helpful later on. Of course, this is an issue that we already encountered with heuristic pricing (see Section 7.2) and partial pricing (see Chapter 8).

We already established that, if solved to optimality, a SUPP will always generate the most negative column among its represented OPPs. However, it may not generate the most negative reduced cost column for each of them. This has to be considered when calculating the lower bound $z_{LB}$ on the objective function value $z_{LP}$. Let $\mathcal{A}$ be the set of ASUPPs, $A^i$, $i \in \mathcal{A}$ was defined as the set of aircraft
represented by ASUPP \( \iota \), where we have \( \cap_{\iota \in \mathcal{A}} A^\iota = \emptyset \) and \( \cup_{\iota \in \mathcal{A}} A^\iota = A \). Similarly, we define \( \mathcal{B} \) as the set of CSUPPs and \( \mathcal{B}^\delta, \delta \in \mathcal{B} \), as the set of crew blocks represented by CSUPP \( \delta \), where we have \( \cap_{\delta \in \mathcal{B}} \mathcal{B}^\delta = \emptyset \) and \( \cup_{\delta \in \mathcal{B}} \mathcal{B}^\delta = \mathcal{B} \). Then, instead of summing up the cost of the most negative reduced cost columns over all original pricing problems, as was done in Equation (6.1), we replace the negative reduced cost values of the original PPs by the minimum negative reduced cost value of the representing SUPPs. We thus have

\[
 z_{\text{SUPP}}^{\text{LB}} := z_{\text{LP}} + \sum_{\iota \in \mathcal{A}} \bar{c}^\iota |A^\iota| + \sum_{\delta \in \mathcal{B}} \bar{c}^\delta (\sum_{b \in \mathcal{B}^\delta} n_b), \quad (9.1)
\]

where \( |A^\iota| \) is the number of aircraft represented by ASUPP \( \iota \). Value \( z_{\text{SUPP}}^{\text{LB}} \) is a valid lower bound since \( \bar{c}^\iota \leq \bar{c}^a, \forall a \in A^\iota \) and \( \bar{c}^\delta \leq \bar{c}^b, \forall b \in \mathcal{B}^\delta \).

When solving a SUPP, several negative reduced cost columns may be generated, which may or may not be generated for several different OPPs. It is possible to add just the best such column, add up to a certain number, or simply add all of them to the RMP. As we have done throughout this thesis, we add up to a certain number of columns for each individual OPP. For this, we store the labels that are extended to the common sink node for each corresponding OPP separately and then chose the e.g. three most negative columns for every OPP.

A limitation of SUPPs is that branching is restricted to more general decisions. For example, we cannot assign a specific flight or connection to an individual aircraft or crew block without singling out the crew block or aircraft, i.e. not represented by a SUPP any more. To avoid this, any decisions have to be made for at least all the aircraft or crew blocks in a SUPP. On the other hand, the common branching techniques, follow-on and variable branching (see Section 6.6), are applicable to SUPPs. In follow-on branching, removing or fixing connections is applied to all aircraft or crew, respectively, anyway. In the 1-branch in variable branching, we assign an entire column and thus path to an aircraft or crew. As before we remove all flights and appropriate connections from all networks. Additionally, in the case of an aircraft, the source arcs corresponding to the aircraft are removed from the representing SUPP. For crew blocks, we do the same if this crew was the last unassigned crew in the crew block. Otherwise, we need to continue generating columns in the CSUPP for the other crews in the crew block.
9.4 Computational Experiments

We conduct several numerical experiments in which we solved superimposed pricing problems instead of the original pricing problems. In all experiments, we compare the performance to previously derived best strategies, i.e. AllPP+3, which is solving all OPPs in every iteration and adding up to three columns, and SetPP/10/20+5, which is solving a subset of OPPs and adding up to five columns per OPP.

A challenge we faced when using superimposed pricing problems is that our preprocessing algorithm (7.2) was not developed with these aggregated networks in mind. The algorithm results in generating columns for the same OPPs over and over again, which is detrimental to CG convergence. This is partly due to the actual implementation, partly due to the design of the algorithm (see Section 9.4.4). We therefore turned off Algorithm 7.1.2 for some of the experiments, which we will indicate in the description of each experiment. To enable comparable results, we chose to switch off preprocessing for AllPP+3 and SetPP/10/20+5 as well whenever we do so for SUPPs. Providing results for both cases, i.e. performing and not performing preprocessing, should allow for more general conclusions - which is important when evaluating the suitability of the SUPP approach to other applications.

In Section 9.4.1 we solve all SUPPs in every iteration while not utilising the preprocessing algorithm (7.2). In Section 9.4.2, we compare the lower bound on the objective function value as it results from solving SUPPs to the one resulting from solving OPPs. Then, in Section 9.4.3, we solve a subset of SUPPs and compare the performance to the strategies of the first experiment. Finally, in Section 9.4.4, we employ the preprocessing algorithm and develop a mixed strategy that first utilises SetPP/10/20+5 and then solves SUPPs.

The goal of the previous chapter was to develop strategies that avoid solving all pricing problems in every iteration. We measured the effectiveness of these strategies by using the time spent in pricing while also considering the effect on overall run time and convergence of the CG procedure. The reason why we did not simply compare the number of pricing problem solves is that this may have been misleading. For example, as we saw in Figure 8.1, the majority of CPPPs do not generate a negative cost column in most iterations; the preprocessing algorithm (7.2) can usually prove this in a tenth or hundredth of a second. Not solving these PPs in
CHAPTER 9. SUPERIMPOSED PRICING PROBLEMS

an iteration does not save much in terms of solution time. In the present section, we investigate the effectiveness of superimposed pricing problems. Solving SUPPs will most certainly result in fewer pricing problem solves as each of them represent several OPPs. However, it may take longer to solve a SUPP. Therefore, once again, we use time spent in pricing, which we called pricing time, as the primary indicator. As before, we additionally discuss the effects on other common performance measurements, i.e. run time, number of iterations, number of columns generated, IP gaps, etc.

9.4.1 Solving all SUPPs per Iteration without Preprocessing

In this section, we present experiments in which we solved all superimposed pricing problems (AllSUPP) in every iteration and added up to a certain number of columns per original PP to the RMP, i.e. impose limits on the number of columns generated for each OPP\(^7\). We compare the resulting performance to the best strategies found in previous chapters, i.e. solving all original pricing problems while adding up to three negative reduced cost columns (strategy AllPP+3) and solving a subset of OPPs and adding up to five columns per OPP (strategy SetPP/10/20+5). For all these settings, we do not perform preprocessing. We then analyse why solving SUPPs results in faster convergence than AllPP+3, followed by a graphical interpretation of the CG procedure when solving SUPPs. In the following, in order to preserve space, we will use SetPP+5 in tables to refer to strategy SetPP/10/20+5.

In the first experiment, for strategy AllSUPP, we tested adding up to 1, 2, 3, 4, or 5 columns per OPP. Table 9.1 shows the change in pricing time (%) for each small instance when solving all SUPPs. It also shows the change for SetPP/10/20+5. All changes are with respect to AllPP+3 without preprocessing. For the AllSUPP strategies, we observed average reductions by a factor of up to five, whereas for SetPP/10/20+5, the pricing time was only halved.

When we calculate the time spent in pricing per iteration and relate the resulting values to those of AllPP+3, we observe average reductions of between 78.84% and 80.85% for the AllSUPP strategies and 72.6% for SetPP/10/20+5. In other words,\(^7\)It should be noted that this does not mean that we perform full pricing in every iteration. A limit of ten labels is imposed on each node, except for iterations in which we perform full pricing to evaluate early branching or prove LP-optimality.
Table 9.1: Change in pricing time (%) when solving the small instances using strategy AllSUPP with several limits on the number of columns to add per OPP. The change is with respect to strategy AllPP+3 without preprocessing. The table also shows the change for SetPP/10/20+5.

<table>
<thead>
<tr>
<th>Inst.</th>
<th>+1</th>
<th>+2</th>
<th>+3</th>
<th>+4</th>
<th>+5</th>
<th>SetPP+5</th>
</tr>
</thead>
<tbody>
<tr>
<td>5-S-1</td>
<td>-65.09</td>
<td>-66.79</td>
<td>-73.19</td>
<td>-70.91</td>
<td>-73.37</td>
<td>-51.06</td>
</tr>
<tr>
<td>5-S-2</td>
<td>-64.43</td>
<td>-69.32</td>
<td>-70.68</td>
<td>-75.36</td>
<td>-71.15</td>
<td>-38.74</td>
</tr>
<tr>
<td>5-L-1</td>
<td>-77.51</td>
<td>-82.10</td>
<td>-84.46</td>
<td>-84.81</td>
<td>-83.04</td>
<td>-60.77</td>
</tr>
<tr>
<td>5-L-2</td>
<td>-77.43</td>
<td>-82.90</td>
<td>-82.30</td>
<td>-82.98</td>
<td>-82.93</td>
<td>-59.34</td>
</tr>
<tr>
<td>3-S-A</td>
<td>-77.79</td>
<td>-84.40</td>
<td>-83.75</td>
<td>-83.66</td>
<td>-86.78</td>
<td>-45.16</td>
</tr>
<tr>
<td>3-L-A</td>
<td>-80.99</td>
<td>-83.28</td>
<td>-86.29</td>
<td>-85.31</td>
<td>-86.35</td>
<td>-50.85</td>
</tr>
<tr>
<td>Average</td>
<td>-73.87</td>
<td>-78.13</td>
<td>-80.11</td>
<td>-80.50</td>
<td>-80.60</td>
<td>-50.99</td>
</tr>
</tbody>
</table>

Table 9.1: Change in pricing time (%) when solving the small instances using strategy AllSUPP with several limits on the number of columns to add per OPP. The change is with respect to strategy AllPP+3 without preprocessing. The table also shows the change for SetPP/10/20+5.

on a per iteration basis, the effect of SetPP/10/20+5 and AllSUPP on the pricing time is fairly similar.

Table 9.2 shows the run time in seconds of Algorithm 6.2, i.e. exploring the follow-on branch, for each instance and parameter setting. When compared to AllPP+3 without preprocessing, we observed average reductions of 46.71% for AllSUPP+1, 48.77% (AllSUPP+2), 49.49% (AllSUPP+3), 48.22% (AllSUPP+4), and 47.41% for AllSUPP+5. The average reduction for SetPP/10/20+5, on the other hand, was only 28.34%.

These reductions were a result not only of the significant decreases in pricing time but also in LP time. The average change in LP time (%) is shown in Table 9.3. The table also shows the average change (%) in the number of iterations, number of columns generated, and number of nodes, i.e. depth, of the tree that was explored. We see that the reduction in the LP time was somewhat similar for all AllSUPP settings, however, the reasons are vastly different. In AllSUPP+1, a larger number of iterations resulted, which was countered by a highly reduced number of columns. The LPs were smaller, and thus faster to solve, but more iterations were required because a limited number of columns was available in each iteration. In contrast,
Table 9.2: Run time (s) for the small instances. Several limits on the number of columns to add per OPP are tested when solving all superimposed pricing problems per iteration. The performance is compared to strategy AllPP+3 and SetPP/10/20+5. Preprocessing was not performed for any of the settings. The table also gives the average change in percent compared to AllPP+3.

AllSUPP+5 added more columns, although still significantly less than AllPP+3, and hence needed fewer iterations than AllSUPP+1. The strategies adding 2, 3, or 4 columns fell in-between these two extremes. The reduction in the number of columns generated for SetPP/10/20+5 is comparable to that of AllSUPP+5, however, the reduction in LP time was only slightly more than half of the reduction in LP time for AllSUPP+5. The reason is that, while we have a reduction of 7.71% in the number of CG iterations for AllSUPP+5, we observed an increase of 84.76% for SetPP/10/20+5. What we observe here is that sometimes the “wrong” pricing problems were solved under SetPP/10/20+5, resulting in more iterations. Clearly, implicitly solving the OPPs via SUPPs is, at least in this case, better than explicitly selecting and solving a subset of OPPs.

Table 9.3 also shows the average IP gaps. We observed that they were slightly larger than the average IP gap of 0.88% for AllPP+3, although this is very much within our goal of 2%. The depth of the tree did not vary much, we attribute the differences to normal fluctuations due to columns added, branching decisions, etc.

The decreases in the number of iterations compared to AllPP+3 was mainly due

<table>
<thead>
<tr>
<th>Inst.</th>
<th>AllPP+3</th>
<th>+1</th>
<th>+2</th>
<th>+3</th>
<th>+4</th>
<th>+5</th>
<th>SetPP+5</th>
</tr>
</thead>
<tbody>
<tr>
<td>5-S-1</td>
<td>466</td>
<td>262</td>
<td>270</td>
<td><strong>250</strong></td>
<td>278</td>
<td>280</td>
<td>335</td>
</tr>
<tr>
<td>5-S-2</td>
<td>613</td>
<td>307</td>
<td>299</td>
<td>315</td>
<td><strong>278</strong></td>
<td>301</td>
<td>458</td>
</tr>
<tr>
<td>5-L-1</td>
<td>1496</td>
<td>653</td>
<td>641</td>
<td><strong>602</strong></td>
<td>636</td>
<td>659</td>
<td>845</td>
</tr>
<tr>
<td>5-L-2</td>
<td>1680</td>
<td>742</td>
<td><strong>662</strong></td>
<td>699</td>
<td>691</td>
<td>726</td>
<td>897</td>
</tr>
<tr>
<td>3-S-A</td>
<td>1189</td>
<td>774</td>
<td>701</td>
<td>727</td>
<td>733</td>
<td><strong>690</strong></td>
<td>1014</td>
</tr>
<tr>
<td>3-L-A</td>
<td>1049</td>
<td>635</td>
<td>624</td>
<td><strong>577</strong></td>
<td>631</td>
<td>640</td>
<td>925</td>
</tr>
</tbody>
</table>


AllSUPP+5 added more columns, although still significantly less than AllPP+3, and hence needed fewer iterations than AllSUPP+1. The strategies adding 2, 3, or 4 columns fell in-between these two extremes. The reduction in the number of columns generated for SetPP/10/20+5 is comparable to that of AllSUPP+5, however, the reduction in LP time was only slightly more than half of the reduction in LP time for AllSUPP+5. The reason is that, while we have a reduction of 7.71% in the number of CG iterations for AllSUPP+5, we observed an increase of 84.76% for SetPP/10/20+5. What we observe here is that sometimes the “wrong” pricing problems were solved under SetPP/10/20+5, resulting in more iterations. Clearly, implicitly solving the OPPs via SUPPs is, at least in this case, better than explicitly selecting and solving a subset of OPPs.

Table 9.3 also shows the average IP gaps. We observed that they were slightly larger than the average IP gap of 0.88% for AllPP+3, although this is very much within our goal of 2%. The depth of the tree did not vary much, we attribute the differences to normal fluctuations due to columns added, branching decisions, etc.

The decreases in the number of iterations compared to AllPP+3 was mainly due
9.4. COMPUTATIONAL EXPERIMENTS

Table 9.3: Average change (%) in LP time, number of iterations, number of columns generated, and depth of the branch-and-price tree. The changes are over the small instances and are with respect to AllPP+3. The table also gives the average IP gap (%) for each setting. The average IP gap for AllPP+3 (not shown) is 0.88%.

to decreases at the root node. Table 9.4 displays the change in percent in the number of iterations at the root node. As we can see, bar instance 3-L-A, we consistently achieved reductions when adding a larger number of columns per OPP. We will investigate this somewhat counter-intuitive behaviour in what follows.

Table 9.4: Change (%) in number of iterations at the root node to reach the early branching criteria. The change is with respect to strategy AllPP+3.

In this analysis, we will describe the general behaviour of the CG procedure at the root node when solving instance 5-S-1 using settings AllSUPP+3 and AllPP+3. We will highlight the behaviour during different phases of the root node. After
concluding the discussion of the root node, we will also describe the behaviour during the remainder of the branch-and-price tree. Here, we provide an extensive analysis, while in later experiments, we will only refer and compare to the findings described in the present section.

Choosing settings AllSUPP+3 and AllPP+3 should enable a fair comparison as in both cases, we add up to three columns per OPP. This means an implicit global limit of $3|Q|$ columns per iteration exists, where $|Q|$ was defined as the number of OPPs in the instance. More importantly, in both settings, the contribution of each original pricing problem to improving the LP value is constrained to three columns as well.

Figure 9.7a compares the convergence of the LP objective function value at the root node for instance 5-S-1. Figure 9.7b provides a close up for iterations 20 through 162. We observed that initially, $z_{LP}$ converged more quickly for AllPP+3 (first 10 iterations) and then flattened out faster than AllSUPP+3. (iterations 20+). The heading-in phase for AllSUPP+3 is longer, it requires about 20 iterations. Figure 9.7b shows that between iterations 20 and 90, and more so between iterations 40 and 90, we still achieved improvements for AllPP+3, however, they were not as good as for AllSUPP+3. In other words, the rate of decrease for $z_{LP}$ was larger for AllSUPP+3 than for AllPP+3. The rate also did not decrease as quickly as for AllPP+3. It is this period that resulted in a smaller number of iterations for AllSUPP+3 because, while in both cases the tailing-off started at about iteration 90, it started with a lower $z_{LP}$ value for AllSUPP+3.

The behaviours of $z_{LP}$ can, to a large degree, be explained by considering the number of columns generated in each iteration (Figure 9.8). We see that, in general but especially early on, AllSUPP+3 generated fewer columns. This explains why $z_{LP}$ converged less quickly in the first 20 iterations for AllSUPP+3; a smaller number of columns was available to enter the basis. When considering just the first 20 iterations, we observed that, in contrast to AllPP+3, AllSUPP+3 initially generated fewer, only around 50, columns, which then increased to 87 columns in iteration 15. To explain this, we need to remember that early on in the CG procedure, many artificial variables are present in the LP solutions, thus, the dual values can be quite extreme. Given such a cost structure, a SUPP will generate fewer columns
Figure 9.7: Convergence of LP objective function value at the root node when solving instance 5-S-1 using setting AllSUPP+3 and AllPP+3.
because paths are dominated more often. The figure also illustrates that in some iterations, starting at iteration 52, many more columns are generated. This is due to performing full pricing in these iterations. It is noteworthy that in full pricing, AllSUPP+3 generates a similar number of columns as AllPP+3, while in others, it does not. Recall that in non-full pricing iterations, we may only store up to ten labels at each node (Section 8.2.4). These labels theoretically can all be for the same ARPP, resulting - overall - in much fewer columns per iteration. Without such a limit, i.e. in full pricing, labels of other aircraft are stored as well, resulting in more columns overall.

![Figure 9.8: Number of columns that were generated in each iteration at the root node when solving instance 5-S-1 using strategy AllPP+3 and AllSUPP+3](image)

Figure 9.8 shows the cumulative number of crew columns, i.e. pairings, that were generated for each iteration. As we observed in the previous chapter, AllPP+3 generated pairings mostly early on, while the number of routes generated throughout the root node was constant. For AllSUPP+3, the increase in the number of routes is likewise mostly linear, however, pairings are also generated more evenly throughout the root node (Figure 9.9). Of interest is that starting at about iteration 40, the number of pairings generated per iteration is larger for AllSUPP+3 than for AllPP+3 (slope of the graphs). This period coincides with the part of the root node in which \( z_{LP} \) converges more quickly.
for AllSUPP+3.

<table>
<thead>
<tr>
<th>Iteration</th>
<th>AllPP+3</th>
<th>AllSUPP+3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2000</td>
<td>5000</td>
</tr>
<tr>
<td>1000</td>
<td>3000</td>
<td>6000</td>
</tr>
<tr>
<td>2000</td>
<td>4000</td>
<td>7000</td>
</tr>
<tr>
<td>3000</td>
<td>5000</td>
<td>8000</td>
</tr>
<tr>
<td>4000</td>
<td>6000</td>
<td>9000</td>
</tr>
<tr>
<td>5000</td>
<td>7000</td>
<td>10000</td>
</tr>
<tr>
<td>6000</td>
<td>8000</td>
<td>11000</td>
</tr>
</tbody>
</table>

Figure 9.9: Number of crew columns that were present in the RMP in each iteration at the root node when solving instance 5-S-1 using strategy AllPP+3 and AllSUPP+3.

As Figure 9.11 shows, the number of aircraft columns generated per iteration was, unlike for AllPP+3, not constant for AllSUPP+3. The algorithm generated less than $3|\mathcal{A}|$ columns per iteration for ASUPP $\iota$, which is a result of the dominance checking in the labelling algorithm. Hence, in total, less than $3|\mathcal{A}|$ columns were generated per iteration. In contrast, in AllPP+3, every ARPP always generated three columns (at the root node).

For AllSUPP+3, we observed that overall a larger portion of crew pairings were generated. Of the columns generated at the root node, under setting AllPP+3, 37.24% were pairings, while for AllSUPP+3, 47.35% were pairings. We observed a similar increase in all instances. It should be noted though, that this increase is not due to generating more pairings but, instead, is due to a larger decrease in the number of aircraft columns. For instance 5-S-1, the number of pairings generated at the root node decreased from 4948 to 3611, while the number of routes decreased from 8337 to 4015. It seems that, in general, under setting AllPP+3, the ARPPs generate columns too often. More precisely, too many ARPPs generate columns too often. This is not at all that surprising, in fact, we have used this observation in the
Figure 9.10: Number of aircraft columns that were present in the RMP in each iteration at the root node when solving instance 5-S-1 using strategy AllPP+3 and AllSUPP+3.

Figure 9.11: Number of aircraft columns that were generated in each iteration at the root node when solving instance 5-S-1 using strategy AllPP+3 and AllSUPP+3.
previous chapter: since the networks are very similar, the ARPPs often generate the same or very similar columns. Solving only a subset, i.e. strategy SetPP/10/20+5, decreases the LP value sufficiently without generating too many identical columns. We achieve a similar result when solving SUPPs: as discussed above, the dominance checking reduces the number of identical columns added per iteration.

Before summarising our findings to explain the faster convergence of AllSUPP+3 between iterations 40 and 90, we make one final observation about the size of the aircraft columns generated under the different settings. Figure 9.12 shows the cumulative difference between the average aircraft column size per iteration as generated under AllPP+3 and AllSUPP+3. An increase means that the average size of the columns for AllPP+3 was larger, and vice versa. We see that for the first 50 iterations, AllPP+3 generated larger columns, between iteration 50 and 90, the average size was about the same, and, finally, from iteration 90 on, AllPP+3 generated smaller columns. This is in fact an accurate description. The average size of the columns generated for AllSUPP+3 was fairly constant, while for AllPP+3, they were larger early on and smaller later.

The reason for the difference in size lies in a combination of the network structure and scheduling maintenance as late as possible. In anticipation of utilising aircraft as much as possible, we only included connections to the sink node from flight nodes that are on the last day of the first period. Consider two aircraft, AC1 and AC2, and a node \( i \), which is on the last day of the first period. Suppose two labels exist at \( i \), one for each aircraft, and that \( i \) is the 10th flight covered (the labels may or may not represent the same path). Let the labels be \((-10, 100, AC1)\) and \((-20, 101, AC2)\), where the first value is the cost of the label and the second the accumulation of a single resource, e.g. flying time. Hence, at node \( i \), the labels do not dominate each other. Since the flight is on the last day and aircraft do not have to return to specific locations, both labels can be extended to the sink node, resulting in a column of size 10. In our implementation, we then store the final labels for each aircraft separately. We can also extend the labels from \( i \) to a node \( j \). Let us assume that doing so would exceed the flying time in both cases, thus a replenishment must occur. Extending along a replenishment arc, the resource accumulation of both labels is reset to the

---

8Recall that aircraft only need to cover the flights in the first period.
same value. As a result, the first label is now dominated by the second. Let us assume the cost of extending from $i$ to $j$ is -1. Then, extending from $j$ to the sink node now only results in a path for aircraft AC2, which is of cost -21. This path covers more flights than the previously found paths, hence, the column is of size 11. Now, under AllPP+3 each pricing problem is solved separately and we get two columns that include flight $j$, and thus two columns of size 11.

The difference in size during the root node then is explained by the different dual values of AllPP+3 and ALLSUPP+3. In the first 50 iterations, more flights $j$ have a dual value such that it is worthwhile extending to $j$, however, as we have seen AllSUPP+3 does not do so for all aircraft. It seems that between iteration 50 and 90, the behaviour of the dual values is inconsistent so that the averages are similar. Then, after iteration 90, AllPP+3 has fewer flights $j$ worth including in a path than AllSUPP+3. Admittedly, we do not have a conclusive explanation why this is the case. However, it seems that the behaviour early on is of more importance.

![Cumulative difference between the average aircraft column size per iteration as generated under AllPP+3 and AllSUPP+3. An increase means that the average column for AllPP+3 was larger, and vice versa.](image)

**Figure 9.12:** Cumulative difference between the average aircraft column size per iteration as generated under AllPP+3 and AllSUPP+3. An increase means that the average column for AllPP+3 was larger, and vice versa.

In summary, we observed that, when compared to AllPP+3, setting AllSUPP+3 required more iterations to pass through the heading-in phase. On the other hand, between iterations 20 and 90 it had better decreases in the LP objective function.
9.4. COMPUTATIONAL EXPERIMENTS

value. As a result, the tailing-off phase started at a lower $z_{LP}$ value and was therefore shorter because the early branching criteria was met earlier. We found that the longer heading-in phase is due to adding much fewer columns in early iterations. We identified several plausible causes for the quicker convergence between iterations 20 and 90, although, none of them was conclusive. We do believe that it is a combined effect of the following reasons:

- AllPP+3 generates the vast majority of pairings in the first 20 iterations and only few after. AllSUPP+3, in contrast, generated pairings throughout the root node (only tailing off towards the end). We theorise that AllPP+3 finds a good solution to the crew part early on - this solution is optimal/almost optimal given the current aircraft solution, hence only very few pairings are generated. The algorithm then tries to improve the aircraft solution based on the current crew solution. However, achieving large decreases in $z_{LP}$ is not easy because of the large number of non-zero entries of the aircraft columns. In a way, the crew solution is, however, waiting for the aircraft solution to change so that the crew solution is not optimal any more, and thus requires new crew columns. Solving SUPPs defers generating crew columns and thus finding a good crew solution. Optimising both problems concurrently seems to allow for larger gains in the objective function value as the crew solution does not “get stuck” after the heading-in phase.

- The average size of the aircraft columns generated under AllPP+3 in the first 50 iterations was larger than for AllSUPP+3. We believe that the detrimental effect of the larger columns did not show in the first 40 iterations because simply many more columns were generated during this phase (Figure 9.11), countering the negative effects. Then, during the middle phase, about iteration 40 to 90, the greater interaction of the larger columns restricts entering variables more, resulting in slower convergence. In other words, the LPs under AllPP+3 are more “rigid”.

- SUPPs generate more dissimilar columns (see Section 9.3). We expect that this has a positive effect as more of the generated columns may enter the basis or do so at a larger value.
We now provide a graphical analysis of the CG procedure when solving 5-S-1 using settings AllSUPP+3 (Figure 9.13) and AllPP+3 (Figure 9.14). In the former, a dark grey cell signifies that at least one column was found by the representing SUPP for this OPP, a white cell means no column was found. It should be noted that this diagram does not show the SUPPs. Doing so would not display valuable information as they were all solved in every iteration and almost always generated some columns. As before, a dark cell in Figure 9.13 means the OPP was solved and returned a column, while a light grey cell signifies the OPP was solved but did not return a column.

Two distinctions are immediately obvious when comparing the two figures. We observed both in the foregoing analysis. Firstly, the ASUPPs did not generate columns for every aircraft in each iteration, and secondly, crew columns were generated throughout the root node. We observe that this was also true for iterations after the root node.

When taking a closer look, we observe that for some aircraft, more columns were generated than for others. This is due to the issue described in Section 9.1: aircraft that have a lower resource consumption at the beginning of the planning horizon do generate columns more often as the corresponding labels are dominated less often, i.e. only if the resource is replenished at some point.

At first glance, it seems that under AllSUPP+3, more crew columns were generated at the root node than under AllPP+3. However, this merely appears so. Under AllSUPP+3, 3331 pairings were generated at the root node, 942 when $\epsilon > 0$, and 150 when $\epsilon = 0$. For AllPP+3, we have 4639 (root node), 902 ($\epsilon > 0$), and 313 when $\epsilon = 0$. We note that the generation of columns for the crew blocks just to the right of the black line was very similar under both strategies. However, for all other crew blocks, we observe how in early iterations, AllSUPP+3 did not generate columns for all pricing problems (most obvious in upper right corner). Instead, more columns were generated for these crew blocks later on.

One could have assumed that, due to the dominance, the generation of columns is fairly clustered, i.e. columns are generated for the same OPP in several consecutive iterations. This does not seem to happen, except of course, for the OPPs for which we generate columns more often due to the lower initial resource accumulation.
Figure 9.13: Graphical interpretation of the CG procedure. Instance 5-L-1 was solved using AllSUPP+3. A dark grey cell signifies that at least one column was found by the representing SUPP for this OPP, a white cell means no column was found.
Figure 9.14: Graphical interpretation of the CG procedure. Instance 5-L-1 was solved using AllPP+3. A dark cell means the OPP was solved and returned a column, while a light grey cell signifies that the OPP was solved but did not return a column.
9.4.2 Quality of Lower Bound

In this experiment, we investigated the quality of the lower bound as it results from solving all SUPPs (Equation 9.1) compared to solving all OPPs (Equation 6.1). For this, we solved to optimality all SUPPs and all OPPs in every iteration in the same run. We solved the entire root node, i.e. did not perform early branching. It should be noted that we only added the columns generated by the SUPPs to the RMP, although this choice is arbitrary. We first analyse how the two lower bounds behave in each iteration before investigating in which iteration different early branching criteria were met. We then compare the time it took to solve pricing problems.

Figure 9.15 shows the behaviour of $z_{\text{LP}}$ and the two lower bounds $z_{\text{SUPP}}^{\text{LB}}$ and $z_{\text{OPP}}^{\text{LB}}$ for the first 117 iterations at the root node when solving instance 5-S-2. Figure 9.16 shows the same for the remaining iterations at the root node.

![Graph showing convergence of LP objective function value and lower bounds](image)

We observed that initially, there was a large gap between the two bounds. The reason is that the best path found in each CSUPP had high negativity due to the artificial variables being present in the LP solution. This path contained many
flights and corresponds to an OPP that covers five days. As we discussed in Section 9.3, we cannot guarantee that we additionally found the best paths for the other represented OPPs. We therefore have to multiply this best column by the number of crews represented in the CSUPP $\delta$, i.e. $\sum_{b \in B^\delta} n_b$. This results in a very poor bound as the optimal reduced cost for the smaller CPPPs are grossly overestimated. The reduced cost of the ARPPs, on the other hand, are usually not overestimated much because they all cover the same number of days, i.e. the networks are all fairly similar to the network of the representing ASUPP.

![Figure 9.16: Convergence of the LP objective function value and the lower bounds resulting from solving all SUPPs and all OPP in the same iteration. Displaying iteration 118 through 221, which are all at the root node of instance 5-S-2.](image)

Table 9.5 displays in which iteration the LP gaps we investigated in Section 6.6.1 are reached when using the reduced cost as generated by the SUPPs compared to those generated by the OPPs.
9.4. COMPUTATIONAL EXPERIMENTS

<table>
<thead>
<tr>
<th>Inst.</th>
<th>$\epsilon = 0.3$</th>
<th>$\epsilon = 0.2$</th>
<th>$\epsilon = 0.1$</th>
<th>$\epsilon = 0.05$</th>
<th>$\epsilon = 0.02$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SUPP</td>
<td>OPP</td>
<td>SUPP</td>
<td>OPP</td>
<td>SUPP</td>
</tr>
<tr>
<td>5-S-1</td>
<td>127</td>
<td>126</td>
<td>129</td>
<td>127</td>
<td>136</td>
</tr>
<tr>
<td>5-S-2</td>
<td>140</td>
<td>140</td>
<td>142</td>
<td>141</td>
<td>156</td>
</tr>
<tr>
<td>5-L-1</td>
<td>119</td>
<td>119</td>
<td>125</td>
<td>123</td>
<td>139</td>
</tr>
<tr>
<td>5-L-2</td>
<td>143</td>
<td>141</td>
<td>144</td>
<td>143</td>
<td>157</td>
</tr>
<tr>
<td>3-S-A</td>
<td>57</td>
<td>51</td>
<td>63</td>
<td>55</td>
<td>71</td>
</tr>
<tr>
<td>3-L-A</td>
<td>62</td>
<td>55</td>
<td>65</td>
<td>60</td>
<td>75</td>
</tr>
</tbody>
</table>

Table 9.5: Iteration at which certain LP gaps are reached at the root node when solving all SUPPs compared to solving all OPPs. We added up to three columns for each pricing problem. The columns added were those generated by the SUPPs.

We observed that except for 3-S-A and 3-L-A, the number of iterations did not differ much, at least for larger $\epsilon$ values. For instances 3-S-A and 3-L-A, the differences were noticeable but we believe these to be within an acceptable range. In summary, the lower bound resulting when solving SUPPs is acceptable, i.e. we do not expect much longer tailing-off phases to occur because of this strategy. Furthermore, the results suggest that in general, the quality of the lower bound resulting from solving SUPPs is better the more similar the original pricing problems are.

We now turn our attention to the time it took to solve SUPPs. Figure 9.17 shows the time it took to solve the five ASUPPs and the time it took to solve five original ARPPs (for this comparison we took the five largest solution time among all ARPPs in each iterations). Figure 9.18 shows the same for the three CSUPPs and the three most time consuming CPPPs. The instance shown here is 5-L-1.

We see that for aircraft, solving the five ASUPPs and five OPPs required only minimal time up until about iteration 120. Only then did the pricing problem solution time increase. We observe that it takes longer to solve the OPPs than the ASUPPs, although not by much. This is due to choosing the five ARPPs that take the longest to solve. They may all be represented by the same ASUPP, which results in a smaller overall time for the ASUPPs.

For crew blocks we observe the opposite: early on, more time is required than
in later iterations. This is not surprising as crew blocks generate columns mostly in the beginning. We clearly see that, unlike for aircraft, it takes longer to solve the CSUPPs that the OPPs. This difference can be explained by recalling that we had to introduce an additional resource, which adds computational effort when comparing labels but also results in more non-dominated labels. Additionally, each SUPP represents many CPPPs, which results in a much larger number of labels in a CSUPP compared to an original CPPP.

It is important to note that in this comparison, we did not consider the time it took to solve all OPPs. Figure 9.19 shows this time and compares it to the pricing time for all eight SUPPs. We see that significantly less time was spent when solving the SUPPs. It seems that at an individual level, solving a SUPP takes more time - although not much - than a single OPP of the same type, i.e. aircraft or crew. However, because this implicitly solves many OPPs, overall significant time is saved.

It should be noted that in this experiment, we solved all SUPPs and OPPs to optimality in every iteration, i.e. performed full pricing in every iteration. We observed (results not shown) that when employing the preprocessing algorithm (Algorithm 7.2), we achieved similar savings.

In Algorithm 6.2, when at the root node (other node), we do not repeat full
9.4. COMPUTATIONAL EXPERIMENTS

Figure 9.18: Time (s) it took to solve in each iteration the three CSUPPs and the three original CPPPs that took the longest to solve. The iterations are at the root node of instance 5-L-1.

Figure 9.19: Time (s) it took to solve in each iteration the eight SUPPs compared to the time it took to solve all OPPs. The iterations are at the root node of instance 5-L-1.
pricing for ten (five) iterations if the early branching criteria was not met. We do so to avoid solving all original pricing problems to optimality too often as this is very time consuming, especially when otherwise we solve only a subset of pricing problems, as for example in strategy SetPP/10/20+5. Under an AllSUPP strategy, the difference is not that large as the same number of pricing problems are solved in full pricing. The only difference is that now they have to be solved to optimality. It is therefore reasonable to repeat full pricing at shorter intervals, for example every seven instead of ten iteration. We did, however, not implement this to enable a fair comparison.

9.4.3 Selecting Subsets of SUPPs without Preprocessing

In this section, we apply the ideas used in Chapter 8 to superimposed pricing problems: we solve a subset of the eight SUPPs in each iteration, except when performing full pricing or proving optimality, in which case we solve all SUPPs.

For this, we introduce some notation. In general, let SetSUPP indicate strategies in which a subset of SUPPs is solved. Then, for specific settings, we will indicate the maximum number of SUPPs to be solved, the minimum number to be solved, and the number of columns to add per OPP. For example, SetSUPP#3#2+5 indicates that at most three and at least two SUPPs are solved per iteration and that every original pricing problem may add up to five columns.

In the experiments presented in this section, we use strategy SetSUPP#3#2, SetSUPP#2#1, and SglSUPP, where the latter solves only one SUPP per iteration. For SetSUPP#3#2 and SetSUPP#2#1, we add up to 1, 2, 3, 4, 5, or 7 columns per OPP, while for SglSUPP, we add up to 1, 2, 3, 4, 5, 7, or 10 columns per OPP. We compare the resulting performance to those of the previous experiment (AllPP+3, SetPP/10/20+5, and all AllSUPP settings). For this experiment, we turned off our preprocessing algorithm for all settings. The experiment generated a large amount of data. As we will see below, the best strategies are those discussed in Section 9.4.1. We will therefore only provide averages over the results of the small instances.

\textsuperscript{9}Notice the difference in notation. In SetPP/10/20+5, the maximum number is \(|Q|\) divided by 10 because this number should depend on the instance size. When solving SUPPS, in our problem, we almost always encounter eight SUPPs. We therefore explicitly specify the number (hence ‘#’) of SUPPs to solve.
The top of Figure 9.20 shows the average change in percent of the run time, pricing time, and LP time of all settings. All changes are with respect to AllPP+3. Similarly, the bottom shows the average change in percent of the total number of iterations, of the number of iterations at the root node, of the total number of columns generated, and of the number of columns generated at the root node. Note that here, Strategy SetPP/10/20+5 is indicated by “PP” (left-most setting in the table).

The graphs illustrate several observations. First and foremost, solving all SUPPs per iteration was the best strategy with respect to almost all criteria. The performance deteriorated as fewer SUPPS were solved. AllSUPP and SetSUPP#3#2 were clearly superior in comparison to the best strategy developed in the previous chapter, i.e. SetPP/10/20+5. SetSUPP#2#1 is on par with said strategy in terms of run time but performed much better with respect to pricing time. The increasing run times when solving fewer SUPPs were a direct result of increased LP times, which in turn were caused by a larger number of iterations.

For the total number of iterations and total number of columns generated, we observed that the behaviour in the changes, i.e. slopes, was very much dependent on the change at the root node. In other words, the root node is by far the most important part of the algorithm in terms of these criteria. It is here where we achieve the largest gains.

When considering individual limits on the number of columns to add, we observed that, regardless of how many SUPPs were solved, adding more columns once again lead to fewer iterations and a reduction in pricing time. However, the changes in number of columns and iterations balanced each other such that the LP time was fairly constant - although it was slightly less for larger limits on the number of columns to add per OPP. As a result, the overall run time did not differ much for varying limits on the number of columns to add when considering an individual strategy.

In summary, this experiment showed that, at least without our preprocessing algorithm, solving all SUPPs in every iteration is clearly the superior strategy compared to explicitly selecting and solving OPPs or SUPPs. Under AllSUPP strategies, this choice is made implicitly, and because it considers the current state of the RMP,
Figure 9.20: Average change (%) of the run time, pricing time, and LP time of several settings (top). Average change (%) of the total number of iterations, of the number of iterations at the root node, of the total number of columns generated, and of the number of columns generated at the root node. All changes are with respect to AllPP+3. Strategy SetPP/10/20+5 is indicated by “PP”. The number of columns added per OPP in the strategies is indicated on the horizontal axis.
9.4. COMPUTATIONAL EXPERIMENTS

is better at doing so. Overall, AllSUPP+3 performed best, with average reductions in run time of 49.49% and pricing time of 80.11% when compared to AllPP+3.

9.4.4 SUPPs while Performing Preprocessing

In this section, we present an experiment in which we used the same settings as in Sections 9.4.1 and 9.4.3, except that this time we employ the preprocessing algorithm (7.2). We once more compare to AllPP+3 and SetPP/10/20+5, for which we also performed preprocessing.

Similar to the experiments in the previous sections, i.e. without preprocessing, the average reduction of pricing time compared to AllPP+3 (with preprocessing) was again very good, ranging from 43.73% to 79.72%. In comparison, the reduction of pricing time for SetPP/10/20+5 was 53.54%. However, the results for the run times were not as encouraging. The change in average run time was between -12.22% and +22.95%, while the change for SetPP/10/20+5 was -15.43%. Recall that, when not performing preprocessing, the AllSUPP strategies performed much better in terms of run time than settings SetPP/10/20+5 and AllPP+3. The reason was a significant reduction in LP time, which was a result of adding much fewer columns while not or only minimally increasing the number of CG iterations (when compared to AllPP+3 without preprocessing). In contrast, when performing preprocessing, the opposite is the case. The LP times were increased, which was a result of a substantially larger number of CG iterations. For the SUPP strategies, this increase was between 40.1% and 367.49% (compared to AllPP+3 with preprocessing). We will give an explanation for this behaviour in what follows.

Unlike what one might have expected, the reason for the increase in iterations is actually not that our preprocessing algorithm generates too few columns in the SUPPs but that it generates columns for the same OPPs over and over again. This becomes apparent when considering Figure 9.21. The figure shows in which iteration at least one column was generated for an OPP when solving all SUPPs and adding up to three columns per OPP, i.e. strategy AllSUPP+3. The Figure shows the behaviour for instance 5-L-1 and only the iterations that are at the root node.

We observe that until the first full pricing iteration, which is after about two-thirds of the iterations, columns for the same ARPPs were generated over and over
Aircraft Crew Blocks

Figure 9.21: Graphical interpretation of the CG procedure. Instance 5-L-1 was solved using AllSUPP+3 while performing preprocessing. The figure only shows the iterations at the root node. A dark grey cell signifies that at least one column was found by the representing SUPP for this OPP, a white cell means no column was found.

again. For crew blocks, on the other hand, this did not occur. The cause for this discrepancy is that the labelling algorithm was almost always executed for the CSUPPs, while, up until the first full pricing iteration, the ASUPPs were solved in the preprocessing algorithm. CSUPPs have more resources and are thus not easily solved in Algorithm 7.2.

The reason why ASUPPs generate columns several times for the same OPPs is partly implementation based, partly due to the design of our preprocessing algorithm. For example, when generating minimum forward $\kappa$-paths in function FSP() (see Section 7.1.2), only the label giving the minimum $\kappa$-value is stored. The corresponding aircraft is always the same since the initial resource accumulation does not change over the course of the branch-and-price algorithm. Since these paths are used in the concatenation functions, columns for only few aircraft result.

Our results showed that after adding columns for other aircraft in the first full pricing iteration, the aircraft solution resulted in dual values that do not allow the pricing problem solver to terminate after preprocessing (see Section 7.2.1). Instead, the labelling algorithm was called for the ASUPPs as well, resulting in columns for
There are several ways to overcome this challenge. The most obvious is to develop a preprocessing procedure that caters to the SUPPs. Alternatively, we can force the ASUPPs to execute the labelling algorithm. The problem with this is that our preprocessing algorithm removes arcs and nodes based on the forward and backward paths. It may be that all or the majority of source arcs for the same aircraft are removed in many iterations. To avoid this, we can choose to not execute our preprocessing algorithm for ASUPPs. A different strategy is to solve SUPPs for the crew blocks but employ a selection strategy, as described in Chapter 8 for the ARPPs. We leave these ideas for future research. Instead, we chose to switch strategies during the root node solve.

The solution process starts with the SetPP/10/20+5 strategy and then switches to AllSUPP+3 in the first iteration in which full pricing is performed. Additionally, we tested switching in the fifth such iteration, or at the end of the root node if less than five full pricing iterations are performed - which was not the case in our instances. In the following, we will denote these hybrid strategies as SwSUPP@1 and SwSUPP@5, respectively. Simply switching strategies may not give the optimal strategy, however, it enables an easy comparison to the SetPP/10/20+5 strategy. In the following experiment, we perform preprocessing in all settings.

We did achieve improvements for the small instances, however, for the larger instances, the effect of switching strategies is more noticeable because they require more iterations to prove that the early branching criteria is met. We therefore present results for the large instances. Table 9.6 shows the pricing time in seconds for the six large instances when solving using setting AllPP+3, SwSUPP@1, SwSUPP@5, and SetPP/10/20+5.

The average reduction in pricing time when compared to AllPP+3 was 63.73% for SwSUPP@1 and 67.15% for SwSUPP@5, while for SetPP/10/20+5, the reduction was 50.32%. We would like to point out that for the largest two instances, i.e. 5-S-A and 5-L-A, the gains were substantial. By switching to an AllSUPP strategy, we were able to halve the already reduced pricing time of SetPP/10/20+5.

Table 9.7 shows the run time in seconds for the six large instances using the same settings. When compared to AllPP+3, the run time was reduced by an average of
CHAPTER 9. SUPERIMPOSED PRICING PROBLEMS

### Table 9.6: Pricing time (s) for the six large instances when solving using settings AllPP+3, SetPP/10/20+5, and when switching from SetPP/10/20+5 to AllSUPP+3 in the first or fifth full pricing iteration.

<table>
<thead>
<tr>
<th>Inst.</th>
<th>AllPP+3</th>
<th>SwSUPP@1</th>
<th>SwSUPP@5</th>
<th>SetPP+5</th>
</tr>
</thead>
<tbody>
<tr>
<td>7-S-1</td>
<td>353</td>
<td>191</td>
<td><strong>158</strong></td>
<td>181</td>
</tr>
<tr>
<td>7-S-2</td>
<td>400</td>
<td>198</td>
<td><strong>171</strong></td>
<td>227</td>
</tr>
<tr>
<td>7-L-1</td>
<td>870</td>
<td>224</td>
<td>249</td>
<td>301</td>
</tr>
<tr>
<td>7-L-2</td>
<td>783</td>
<td>262</td>
<td><strong>227</strong></td>
<td>367</td>
</tr>
<tr>
<td>5-S-A</td>
<td>2140</td>
<td>565</td>
<td><strong>519</strong></td>
<td>1133</td>
</tr>
<tr>
<td>5-L-A</td>
<td>1395</td>
<td>396</td>
<td><strong>384</strong></td>
<td>773</td>
</tr>
</tbody>
</table>

30.43% for SwSUPP@1 and 38.02% for SwSUPP@5. For SetPP/10/20+5, we only observed an average reduction of 23.07%.

### Table 9.7: Run time (s) for the six large instances when solving using setting AllPP+3, SetPP/10/20+5, and when switching from SetPP/10/20+5 to AllSUPP+3 in the first or fifth full pricing iteration.

<table>
<thead>
<tr>
<th></th>
<th>AllPP+3</th>
<th>SwSUPP@1</th>
<th>SwSUPP@5</th>
<th>SetPP+5</th>
</tr>
</thead>
<tbody>
<tr>
<td>7-S-1</td>
<td>1838</td>
<td>1373</td>
<td><strong>1206</strong></td>
<td>1532</td>
</tr>
<tr>
<td>7-S-2</td>
<td>2018</td>
<td>1388</td>
<td><strong>1329</strong></td>
<td>1693</td>
</tr>
<tr>
<td>7-L-1</td>
<td>6492</td>
<td>2889</td>
<td><strong>2867</strong></td>
<td>3536</td>
</tr>
<tr>
<td>7-L-2</td>
<td>5001</td>
<td><strong>2921</strong></td>
<td>2960</td>
<td>3452</td>
</tr>
<tr>
<td>5-S-A</td>
<td>15084</td>
<td>13104</td>
<td><strong>9638</strong></td>
<td>13984</td>
</tr>
<tr>
<td>5-L-A</td>
<td>9699</td>
<td>8165</td>
<td><strong>7096</strong></td>
<td>7580</td>
</tr>
</tbody>
</table>

These reductions are once more due to the changes in the number of iterations and number of columns generated. Table 9.8 shows the average change in percent of LP time, number of iterations, and number of columns generated under SwSUPP@1, SwSUPP@5, and SetPP/10/20+5 when compared to AllPP+3. Important to note is the difference in the reduction of iterations between the SwSUPP strategies and SetPP/10/20+5. Switching the strategy clearly has benefits. Recall that in general,
for the selection strategies developed in Chapter 8, choosing the right OPPs to solve is not as important early on in the CG procedure as any OPP, especially any ARPP, returns valuable columns. However, later in the procedure, the “right” OPPs have to be solved to generate the most negative reduced cost columns. Columns of small negativity do improve the objective function value but result in more iterations. Now, switching to an AllSUPP strategy avoids this choice, hence the significant reductions compared to SetPP/10/20+5.

<table>
<thead>
<tr>
<th></th>
<th>SwSUPP@1</th>
<th>SwSUPP@5</th>
<th>SetPP+5</th>
</tr>
</thead>
<tbody>
<tr>
<td>LP Time</td>
<td>-24.79</td>
<td>-32.99</td>
<td>-17.83</td>
</tr>
<tr>
<td>Iterations</td>
<td>58.04</td>
<td>47.21</td>
<td>194.64</td>
</tr>
<tr>
<td>Columns</td>
<td>-21.01</td>
<td>-33.41</td>
<td>-49.61</td>
</tr>
</tbody>
</table>

Table 9.8: Average change (%) in LP time, number of iterations, and number of columns generated. The changes are with respect to setting AllPP+3. The averages are over the six large instances.

9.5 Summary

In this chapter, we introduced superimposed pricing problems that represent aggregations of several original pricing problems. Solving these SUPPs implicitly solves all represented OPPs and thereby fulfills the requirements of column generation, that is to either generate a negative reduced cost column or prove optimality if none exists.

Two different versions of superimposed pricing problems were introduced. One aggregates aircraft routing pricing problems, the other aggregates crew pairing pricing problems. Several requirements must be fulfilled in order to aggregate OPPs, see Sections 9.1 and 9.2, respectively. The requirements for the CSUPP are more restrictive as the original CPPPs not only have varying start times and start locations but also different end times and end locations. This is considered in the CSUPP by introducing an additional resource and only aggregating CPPPs for which the end locations are identical.

We conducted several numerical experiments to verify the effectiveness of using
SUPPs instead of OPPs. When not performing preprocessing (7.2) for any setting, we saw that SUPPs performed much better than OPPs. For the best strategy, which was AllSUPP+3, we achieved an average reduction in pricing time of 80.11%, while the average reduction in run time was 49.49%. We observed that this was clearly better than the reductions achieved by the best strategy developed in Chapter 8, i.e. strategy SetPP/10/20+5. These reductions were 50.99% and 28.34%, respectively. Solving SUPPs is a superior strategy because we avoid explicitly selecting OPPs. This selection was made based on reduced cost of pricing problem solves in previous iterations. Solving SUPPs, on the other hand, considers the current state of the RMP, and thus often generates more beneficial columns because in a selection strategy, the OPP providing the best columns may not be selected. We also combined the ideas of the present and the previous chapters by selecting a subset of SUPPs to solve. We observed that this performed worse than solving all SUPPs in every iteration. Again, explicitly selecting pricing problems is inferior to letting the current dual values implicitly make this decision.

When employing Algorithm 7.2, the ASUPPs generate negative reduced cost columns for the same ARPPs over and over again until the first full pricing iteration. This results in a significantly increased number of iterations and hence run time. The reason is that the algorithm was not designed to consider multiple aircraft or crew blocks in the same pricing problem. To avoid this behaviour, we experimented with first employing strategy AllPP/10/20+5 and then switching to solving SUPPs. We found that the best setting was to switch to AllSUPP+3 in the fifth full pricing iteration. For the large instances, when compared to AllPP+3, we observed average reduction in pricing time of 67.15% and average reductions of run times of 38.02%. The original AllPP/10/20+5 gave average reductions of only 50.32% and 23.07%, respectively.

We also investigated the quality of the lower bound on $z_{LP}$ when solving SUPPs. We found that early in the CG procedure the bound is significantly worse than when solving the original pricing problems. However, by the time the bounds become of interest to evaluate if early branching should be performed, the bound resulting from the SUPPs is sufficiently close the the bound resulting from the OPPs.
Chapter 10

Comparison of Approaches and Conclusions

In this chapter, we first summarise the paradigm described in this thesis and then review the advances we made to the solution algorithm throughout the previous chapters. After this, we provide numerical experiments for the large instances, which enables a direct comparison of the solution strategies. Finally, we highlight some interesting future research directions with respect to our integrated problem as well as to the solution algorithm.

10.1 Summary

In this thesis, we provided a detailed description of a new paradigm that enables solving an integrated aircraft routing, crew pairing, and tail assignment problem very close to the day of operations. Under this new paradigm, routes and pairings are generated only few days before the day of operations. This represents a break from the sequential approach currently employed by airlines. Implementing the new paradigm therefore will require re-engineering of some business processes (see Chapter 4). The most significant change is that after the rostering phase was completed, crews are only told when they work but not which flights they are scheduled on as these decisions will be revisited in our integrated problem. From these work-periods, we build so-called crew blocks that aggregate all work-periods for which the start time, end time, and crew base are the same. The solver will generate an
appropriate number of pairings for each crew block, which can then be assigned to individual crews. Since the work periods differ between crew blocks, we have to generate pairings for each individual crew block.

Solving the problem only days before the day of operations enables generating routes specifically for each aircraft, thereby eliminating the need to later assign routes to aircraft. These routes consider the location and maintenance history of the individual aircraft. Unlike standard aircraft routing, we consider all maintenance requirements that the aircraft has during the planning horizon.

In Chapter 5, we presented a column generation formulation in which the master problem ensures that all flights are serviced by one aircraft and one crew. Furthermore, the master problem contains constraints that model the interaction of aircraft and crew along short and restricted connections. Along the former, crews may not change aircraft, while along the latter, doing so incurs a penalty. Restricted connections are of a duration that makes it likely to propagate delays; staying with an aircraft avoids spreading the delay, thus increasing schedule robustness.

All maintenance and crewing rules are considered in the pricing problems, which we modeled as resource constrained shortest path problems with replenishments, see Sections 5.2.1 and 5.2.2, respectively. The replenishments represent maintenance checks for aircraft or rest periods for crews. The rules we considered in our problem required several extensions to the standard resource constrained shortest path problems with replenishments. We extend the RCSPP-R as proposed by (Smith, 2011) to handle multi-arcs, a non-additive resource consumption, and cost that increases in non-linear fashion and depends on a resource. In Chapter 7, we presented a labelling algorithm that is capable of handling these extensions. Furthermore, we developed a preprocessing procedure that reduces the computational effort in the labelling algorithm.

In Chapter 6, we developed a branch-and-price algorithm for the mathematical problem described in Chapter 5. Through numerical testing, we investigated the choice of LP solver, evaluated the benefit of two initialisation methods for the restricted master problem, considered how many columns to add per iteration, and evaluated if columns should be removed from the restricted master problem if they have been non-basic for a number of iterations.
To obtain integer solutions, we use follow-on branching and only explore the
1-branch without backtracking. Numerical results showed that branching early can
lead to substantial savings in run time. However, without adjusting the early branch-
ing criteria, large integrality gaps result. We therefore gradually tighten the branch-
ing criteria, until at some point, we do not perform early branching any more. To
avoid excessive LP solution times, we suppress the pricing step early on in the
branching tree, unless the LP objective function value deteriorated significantly due
to branching decisions.

We found that our preprocessing algorithm (7.2) has a significant impact on
the overall run time. The majority of the reduction is due to fewer CG iterations
at the root node. Numerical results suggest that the increased CG convergence is
due to adding columns with fewer non-zero entries during preprocessing as well as
during the improved initialisation method. Additionally, we use the preprocessing
algorithm as a major source of heuristic pricing: we abort solving a pricing problem
after preprocessing if columns of a certain quality were found.

Since we generate routes specifically for each aircraft and pairings specifically for
each crew block, we must solve a large number of pricing problems. This results in
spending a great deal of time in the pricing step and also in adding many columns
to the RMP in each iteration, which deteriorates LP solution times. In this thesis,
several strategies were developed to address the challenge of many pricing problems.

In Chapter 8, we select and solve a subset of the pricing problems in each itera-
tion. We experimented with different sizes of subsets, where the size depends on the
number of pricing problems in an instance. The choice of pricing problems to solve
is determined by a score, which depends on when a pricing problem was solved last
and the negativity of the best column returned in that solve. This reflects the state
of the RMP and whether a PP should add further columns. We found that these
selection strategies drastically reduced the time spent in pricing. The run time of
the entire algorithm was reduced as well, while the number of iterations generally
increased\(^1\).

In Chapter 9, we developed an alternative strategy to tackle the multitude of pric-
ing problems. We introduced so-called superimposed pricing problems (SUPP) that

\(^1\)Numerical results are shown in the following section.
represent aggregations of several original pricing problems. Solving these SUPPs implicitly solves all represented original pricing problems (OPP) and thereby fulfills the requirements of column generation, that is to either generate a negative reduced cost column or prove optimality if none exists. We discussed the requirements that have to be fulfilled by the original pricing problems in order to be aggregated in a SUPP.

Numerical results showed that employing SUPPs in the branch-and-price algorithm significantly outperforms selecting and solving the original pricing problems when neither method uses preprocessing (Algorithm 7.2). Solving SUPPs is a superior strategy because we avoid explicitly selecting OPPs. This selection is made based on reduced cost of pricing problem solves in previous iterations. Solving SUPPs, on the other hand, considers the current state of the RMP, and thus often generates more beneficial columns because, in a selection strategy, the OPPs providing the best columns may not be selected.

When employing Algorithm 7.2 (preprocessing), the SUPPs representing aircraft generate negative reduced cost columns for the same aircraft over and over again. This results in a significantly increased number of iterations and hence run time. The reason is that our processing algorithm was not designed to consider multiple aircraft or crew blocks in the same pricing problem. To avoid this behaviour, which occurs mostly early on, we switch from selecting and solving a subset of original problems to solving superimposed pricing problems at some point during the root node. We found that this strategy gives the best strategy when employing Algorithm 7.2.

### 10.2 Numerical Results for Large Instances

Over the course of Chapters 6 through 9, we developed an ever-improving solution algorithm. In this section, we will compare the performance improvements by considering several criteria. We present results for solving the large instances using the best settings of each chapter. We first present a table (10.1) that shows the parameters used in each setting.

Table 10.2 shows the resulting run times in seconds for all six large instances. In general, we observed that each setting improved the run times. Setting Basic
## Numerical Results for Large Instances

<table>
<thead>
<tr>
<th>Inst.</th>
<th>Basic</th>
<th>B&amp;P</th>
<th>HeurPr</th>
<th>SetPP</th>
<th>SwSUPP</th>
</tr>
</thead>
<tbody>
<tr>
<td>LP solver</td>
<td>Primal</td>
<td>Primal</td>
<td>Primal</td>
<td>Primal</td>
<td>Primal</td>
</tr>
<tr>
<td>Increased tolerances</td>
<td>$\times 10$</td>
<td>$\times 10$</td>
<td>$\times 10$</td>
<td>$\times 10$</td>
<td>$\times 10$</td>
</tr>
<tr>
<td>Initialisation RMP</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Number columns per PP</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>Column agelimit root node</td>
<td>$\infty$</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>Column agelimit other nodes</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>$\infty$</td>
</tr>
<tr>
<td>Remove columns after root</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Follow-on branching</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Branch on replenishments</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Early branching</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Suppressing pricing</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Preprocessing</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Abort PP solve after prepr.</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Limit labels at no node</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Subset of PP</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Superimposed PP</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Table 10.1: Parameters used in each setting.
performed rather poorly and was not able to even solve the root node for instances 5-S-A and 5-L-A within the time limit of 12 hours. B&P more than halved the run time for the first four instances, while for the latter, it was not able to explore the entire follow-on branch within the given time limit (see Table B.2). Setting HeurPr again halved the run times, while for SetPP and SwSUPP the relative decreases were smaller. The average reduction of SetPP compared to HeurPr was 23.07%, and that of SwSUPP compared to SetPP was 18.91%. Of course, it should be noted that, as the algorithm becomes better, it is harder to make further improvements. When comparing SwSUPP to Basic, we observed an average overall reduction by a factor of 10, where we exclude instances 5-S-A and 5-L-A as those numbers are not reliable.

<table>
<thead>
<tr>
<th>Inst.</th>
<th>Basic</th>
<th>B&amp;P</th>
<th>HeurPr</th>
<th>SetPP</th>
<th>SwSUPP</th>
</tr>
</thead>
<tbody>
<tr>
<td>7-S-1</td>
<td>17040</td>
<td>3067</td>
<td>1838</td>
<td>1532</td>
<td>1206</td>
</tr>
<tr>
<td>7-S-2</td>
<td>18004</td>
<td>4378</td>
<td>2018</td>
<td>1693</td>
<td>1329</td>
</tr>
<tr>
<td>7-H-1</td>
<td>25366</td>
<td>12191</td>
<td>6492</td>
<td>3536</td>
<td>2867</td>
</tr>
<tr>
<td>7-H-2</td>
<td>22873</td>
<td>10988</td>
<td>5001</td>
<td>3452</td>
<td>2960</td>
</tr>
<tr>
<td>5-S-A</td>
<td>43325*</td>
<td>43246*</td>
<td>15084</td>
<td>13984</td>
<td>9638</td>
</tr>
<tr>
<td>5-L-A</td>
<td>43248*</td>
<td>43249*</td>
<td>9699</td>
<td>7580</td>
<td>7096</td>
</tr>
</tbody>
</table>

Table 10.2: Run times (s) when solving the large instances using the best settings of each chapter. A * indicates that solving was aborted due to reaching the time limit of 12 hours.

The reductions in time spent in pricing, which was our main goal in the numerical part of this thesis, were very substantial. Compared to Basic, the pricing time in SwSUPP was reduced by an average factor of 64, again excluding instances 5-S-A and 5-L-A. For these two instances, the reductions will be much larger. Even though the root node has not finished solving, the total pricing time is already 54 and 71 times that under SwSUPP, respectively\(^2\). Overall, we again observe that every setting improved the pricing time achieved by the previous setting. With

\(^2\)For the four instances that were successfully solved under Basic, the root node accounted for between 40% and 61% of the total run time. Thus, it can easily be seen how the total pricing time will increase when solving more than the root node for instances 5-S-A and 5-L-A.
respect to the main strategies developed here, i.e. SetPP and SwSUPP, we observe an average reduction of 50.32% when comparing SetPP to HeurPP and a further 32.85% when comparing SwSUPP to SetPP.

<table>
<thead>
<tr>
<th>Inst.</th>
<th>Basic</th>
<th>B&amp;P</th>
<th>HeurPr</th>
<th>SetPP</th>
<th>SwSUPP</th>
</tr>
</thead>
<tbody>
<tr>
<td>7-S-1</td>
<td>13130</td>
<td>1171</td>
<td>353</td>
<td>181</td>
<td>158</td>
</tr>
<tr>
<td>7-S-2</td>
<td>13831</td>
<td>1899</td>
<td>400</td>
<td>227</td>
<td>171</td>
</tr>
<tr>
<td>7-H-1</td>
<td>11071</td>
<td>1976</td>
<td>870</td>
<td>301</td>
<td>249</td>
</tr>
<tr>
<td>7-H-2</td>
<td>10456</td>
<td>1672</td>
<td>783</td>
<td>367</td>
<td>227</td>
</tr>
<tr>
<td>5-S-A</td>
<td>28075</td>
<td>12790</td>
<td>2140</td>
<td>1133</td>
<td>519</td>
</tr>
<tr>
<td>5-L-A</td>
<td>27328</td>
<td>9973</td>
<td>773</td>
<td>384</td>
<td></td>
</tr>
</tbody>
</table>

Table 10.3: Time spent pricing (s) when solving the large instances using the best settings of each chapter. A * indicates that solving was aborted due to reaching the time limit of 12 hours.

Setting SwSUPP achieved the lowest LP times for all instances (see Table 10.3 in Appendix B). It provides the best balance between the number of iterations and the number of columns added. SetPP adds fewer columns but requires more iterations, while settings B&P and HeurPr add more columns but need fewer iterations (see Tables 10.5 and 10.6). Of more interest at this point is the ratio of pricing time and LP time, which is shown in Table 10.4. We see that under the Basic setting, a large proportion of time is spent in pricing. As we make improvements to the algorithm, the pricing time is reduced much more than the LP time - which is not surprising as this was our goal. Under SwSUPP, the average ratio is 1/10.

When considering the number of CG iterations required to solve the instances, we observed vastly different results, which is a consequence of how a strategy achieves its performance improvements. B&P reduces the number of iterations significantly when compared to Basic. The causes of this are early branching, adding multiple columns, and suppressing the pricing step. HeurPr gives inconsistent results compared to B&P. The smaller columns, especially early on at the root node, can increase convergence, while adding less negative columns in general requires additional iterations. SetPP clearly increases the number of iterations because only a
## CHAPTER 10. COMPARISON OF APPROACHES AND CONCLUSIONS

### Table 10.4: Ratio of pricing time and LP time when solving the large instances using the best settings of each chapter. A * indicates that solving was aborted due to reaching the time limit of 12 hours.

<table>
<thead>
<tr>
<th>Inst.</th>
<th>Basic</th>
<th>B&amp;P</th>
<th>HeurPr</th>
<th>SetPP</th>
<th>SwSUPP</th>
</tr>
</thead>
<tbody>
<tr>
<td>7-S-1</td>
<td>3.39</td>
<td>0.64</td>
<td>0.25</td>
<td>0.14</td>
<td>0.16</td>
</tr>
<tr>
<td>7-S-2</td>
<td>3.35</td>
<td>0.79</td>
<td>0.25</td>
<td>0.16</td>
<td>0.15</td>
</tr>
<tr>
<td>7-H-1</td>
<td>0.78</td>
<td>0.20</td>
<td>0.16</td>
<td>0.10</td>
<td>0.10</td>
</tr>
<tr>
<td>7-H-2</td>
<td>0.85</td>
<td>0.18</td>
<td>0.19</td>
<td>0.12</td>
<td>0.09</td>
</tr>
<tr>
<td>5-S-A</td>
<td>1.85*</td>
<td>0.42*</td>
<td>0.17</td>
<td>0.09</td>
<td>0.06</td>
</tr>
<tr>
<td>5-L-A</td>
<td>1.72*</td>
<td>0.30*</td>
<td>0.17</td>
<td>0.12</td>
<td>0.06</td>
</tr>
</tbody>
</table>

### Table 10.5: Number of CG iterations required when solving the large instances using the best settings of each chapter. A * indicates that solving was aborted due to reaching the time limit of 12 hours.

<table>
<thead>
<tr>
<th>Inst.</th>
<th>Basic</th>
<th>B&amp;P</th>
<th>HeurPr</th>
<th>SetPP</th>
<th>SwSUPP</th>
</tr>
</thead>
<tbody>
<tr>
<td>7-S-1</td>
<td>3261</td>
<td>694</td>
<td>688</td>
<td>1730</td>
<td>958</td>
</tr>
<tr>
<td>7-S-2</td>
<td>3080</td>
<td>770</td>
<td>745</td>
<td>1589</td>
<td>948</td>
</tr>
<tr>
<td>7-H-1</td>
<td>3310</td>
<td>858</td>
<td>645</td>
<td>1671</td>
<td>962</td>
</tr>
<tr>
<td>7-H-2</td>
<td>2403</td>
<td>815</td>
<td>554</td>
<td>1686</td>
<td>816</td>
</tr>
<tr>
<td>5-S-A</td>
<td>614*</td>
<td>371*</td>
<td>726</td>
<td>2550</td>
<td>1076</td>
</tr>
<tr>
<td>5-L-A</td>
<td>774*</td>
<td>424*</td>
<td>538</td>
<td>2090</td>
<td>926</td>
</tr>
</tbody>
</table>

The number of columns that are generated increased when comparing B&P to Basic (Figure 10.6). Adding three columns per PP and iteration offsets the decrease in the number of iterations. We see that many iterations in Basic hardly generate any columns as the algorithm attempts to prove LP optimality at every node.
whereas B&P often does not have to do so due to early branching. HeurPr reduced the number of columns, which, as we pointed out in Section 7.2, is somewhat counter-intuitive. The smaller columns resulted in fewer iterations, which generally means a reduced number of columns. Additionally, the preprocessing algorithm (7.2) does not always generate three columns (recall that we may abort solving a PP after preprocessing). SetPP reduced the number of columns because fewer PPs were solved, while SwSUPP increased the number again since more columns were generated for several PPs that are represented by a SUPP.

<table>
<thead>
<tr>
<th>Inst.</th>
<th>Basic</th>
<th>B&amp;P</th>
<th>HeurPr</th>
<th>SetPP</th>
<th>SwSUPP</th>
</tr>
</thead>
<tbody>
<tr>
<td>7-S-1</td>
<td>35520</td>
<td>39330</td>
<td>35113</td>
<td>19966</td>
<td>24753</td>
</tr>
<tr>
<td>7-S-2</td>
<td>35486</td>
<td>44128</td>
<td>38401</td>
<td>21116</td>
<td>26434</td>
</tr>
<tr>
<td>7-H-1</td>
<td>77964</td>
<td>89144</td>
<td>67553</td>
<td>28608</td>
<td>40237</td>
</tr>
<tr>
<td>7-H-2</td>
<td>66861</td>
<td>93762</td>
<td>63406</td>
<td>33481</td>
<td>40095</td>
</tr>
<tr>
<td>5-S-A</td>
<td>49202*</td>
<td>81649*</td>
<td>105093</td>
<td>50037</td>
<td>73975</td>
</tr>
<tr>
<td>5-L-A</td>
<td>56573*</td>
<td>90293*</td>
<td>90673</td>
<td>43286</td>
<td>60752</td>
</tr>
</tbody>
</table>

Table 10.6: Number of columns generated when solving the large instances using the best settings of each chapter. A * indicates that solving was aborted due to reaching the time limit of 12 hours.

Table 10.7 shows the number of pricing problems that were solved under the different settings. For Basic, B&P, and HeurPr, this number is linear in the number of iterations. Thus, we solved a very large number of pricing problems in the Basic setting. Compared to HeurPr, SetPP on average halved the number of PP solved. Under strategy SwSUPP, the fewest pricing problems needed to be solved. The reason is that only eight SUPPs exist. At an individual level, solving a SUPP was more time consuming than solving an original pricing problem. However, as Table 10.3 showed, we achieved reductions in pricing time because, in comparison to SetPP, fewer iterations were necessary, while, compared to HeurPr, solving all SUPPs was less time-consuming than solving all original PPs.

The number of branch-and-price nodes, which, since we only explore the follow-on branch, is equal to the depth of the tree, was not much different for SetPP and
CHAPTER 10. COMPARISON OF APPROACHES AND CONCLUSIONS

<table>
<thead>
<tr>
<th>Inst.</th>
<th>Basic</th>
<th>B&amp;P</th>
<th>HeurPr</th>
<th>SetPP</th>
<th>SwSUPP</th>
</tr>
</thead>
<tbody>
<tr>
<td>7-S-1</td>
<td>166311</td>
<td>35394</td>
<td>36101</td>
<td>16113</td>
<td>7371</td>
</tr>
<tr>
<td>7-S-2</td>
<td>175560</td>
<td>43890</td>
<td>43879</td>
<td>19250</td>
<td>7284</td>
</tr>
<tr>
<td>7-H-1</td>
<td>321070</td>
<td>83226</td>
<td>64057</td>
<td>19381</td>
<td>8212</td>
</tr>
<tr>
<td>7-H-2</td>
<td>233091</td>
<td>79055</td>
<td>54778</td>
<td>25976</td>
<td>6781</td>
</tr>
<tr>
<td>5-S-A</td>
<td>81048*</td>
<td>48972*</td>
<td>98986</td>
<td>39910</td>
<td>9680</td>
</tr>
<tr>
<td>5-L-A</td>
<td>102168*</td>
<td>55968*</td>
<td>73602</td>
<td>34555</td>
<td>8510</td>
</tr>
</tbody>
</table>

Table 10.7: Number of solved pricing problems when solving the large instances using the best settings of each chapter. A * indicates that solving was aborted due to reaching the time limit of 12 hours.

SwSUPP when compared to HeurPr (see Table B.2 in Appendix B). The choice of PPs to solve has little influence on the branching decisions. However, it seems that SwSUPP requires fewer nodes than SetPP. This behaviour was consistent except for one instance, for which they were almost equal; the average reduction was 6.92%. It is not obvious why such a reduction should result. Additional experiments are required to verify this observation.

The IP gaps, as defined by Equation 6.3, are calculated with respect to the externally generated optimal LP value of the root node. Thus, we are able to compare the IP gaps that result for the different settings. We observed that no strategy performed consistently better (see Table B.3 in Appendix B). We conclude that the pricing selection strategies have no or very limited impact on the quality of the integer solution. Noteworthy are the IP gaps for instance 7-S-1. Under the Basic setting, an artificial variable was present in the final integer solution. For SetPP this was not the case, the solution was simply not of a good quality (IP gap was 5.34%). As the IP gap for instance 7-S-1 under the Basic strategy shows, despite the cautious branching strategy (Section 6.6), we cannot guarantee a feasible solution when not allowing backtracking to occur. On the other hand, this simply means that when implementing the schedule, a stand-by aircraft or crew has to cover the unassigned flight.
Summary

Our goal was to show that the mathematical model resulting from the paradigm described in this thesis can be solved in reasonable time. The largest test instances, which contain 50 aircraft and 83 crew blocks while covering five days, can be solved in less than 2.5 hours. Seven-day instances with 33 aircraft and 64 crew blocks can be solved in less than one hour. This is well under our goal of four hours.

The best strategy we developed is a hybrid strategy called SwSUPP@5, which is a combination of SetPP/10/20+5 and AllSUPP+3. Initially this strategy solves between $\left\lfloor \frac{|Q|}{20} \right\rfloor$ and $\left\lfloor \frac{|Q|}{10} \right\rfloor$ many original pricing problems, where $|Q|$ is the sum of the number of aircraft and crew blocks in an instance. Up to five columns are added per pricing problem and iteration. Then, in the fifth iteration in which we perform full pricing, we switch to solving all eight superimposed pricing problems. This implicitly solves all original pricing problems. We add up to three columns per original pricing problem and iteration.

As we saw in Table 10.3, the new method achieved substantial reductions in the time spent in pricing. Table 10.4 shows that of the remaining run time, the vast majority is spent solving LPs. To further reduce overall run times, and thus enable solving even larger instances, solving the restricted master problem needs to be improved. We have not put a great deal of attention to this task and therefore expect further significant improvements by applying some of the techniques described in Section 2.3, i.e. stabilization of dual values and constraint aggregation.

10.3 Future Work

Both the features considered in the integrated problem and the solution method for the resulting mathematical model provide ample opportunity for future research. The problem can be extended to consider deadheading by including deadheading arcs or nodes in the pricing problems. Deadheading crews may spill passengers, which can be considered in the master problem by including constraints that result in a penalty for each passenger that gets spilled by a crew member.

The problem can be extended to consider fleet assignment. Including a flight in a route of an individual aircraft automatically assigns the fleet type. Then, via
additional constraints, the master problem has to ensure that the correct crew type is assigned to each flight.

Before implementing our paradigm airline wide, thorough verification is required as the new approach represents a significant change in business processes and because of the shorter reaction time of only four days before plans are put into action. This may involve developing a simulation tool to evaluate the robustness of the new schedules under several disruption scenarios and possibly applying the new paradigm to only a small fleet so as to limit any negative effects.

Algorithmically, several questions are worthwhile investigating. We saw that, in our case, columns which have a lower number of non-zero entries than the average column can speed up CG convergence, especially early on in the algorithm. Specifically generating such smaller columns during the heading-in phase may be beneficial. It would be interesting to investigate if these findings translate to other problems.

Throughout this thesis, we limited the number of columns to add per pricing problem. We believe this to be a better strategy than imposing a global limit as this may result in adding many columns for just a few or even only a single pricing problem. In this case, we expect the LP value to not reduce as much. Additional computational experiments are required to verify this assumption. Of course, it is possible to have a global limit in addition to an individual limit. We can for example chose the most negative reduced cost column of each pricing problem and then make up the difference to the global limit by choosing the most negative columns not yet added.

We only explored very basic heuristic pricing in the labelling algorithm as we abort solving a pricing problem if the preprocessing algorithm generated a column of a certain quality. It may be worthwhile to investigate additional heuristic pricing methods, especially since our preprocessing algorithm does not work well for superimposed pricing problems. A problem-specific strategy we already outlined is to extend labels along the most negative arc(s), where we chose one or several arcs from each connection type. So for example, we may chose the most negative duty connection, regular layover, and long layover for each node.

When selecting a subset of pricing problems to solve, we order the pricing prob-
lems by a score, which depends on when the pricing problem was solved last and the negativity of the best column returned in that solve. This, of course, does not reflect the true current state of the RMP. We may adjust the order, and hence selection, of the pricing problems by the dual value of the corresponding convexity constraints.

For the SUPPs, we aggregated the original pricing problems as much as possible, except for when ARPPs had different resource requirements, as preliminary experiments showed that this is not beneficial. It may be worthwhile investigating aggregations that have different objectives. For example, we can aggregate by duration of crew block, which likely gives a better lower bound (see discussion in Section 9.3). Another different goal would be to generate more columns per iteration. In this case, fewer OPPs should be represented by a single SUPP, resulting in fewer dominated labels and hence more columns (when considering all SUPPs).

We can implement a heuristic pricing method specifically for SUPPs. Instead of including all parallel source arcs between source node $s$ and a flight node $i$, we only include one such arc, where we ensure that this arc corresponds to a different OPP for each different flight node $i$.

In our numerical experiments, we observed that our preprocessing algorithm (7.2) is not very suitable for the SUPPs. Developing a preprocessing procedure that caters to the SUPPs should provide ample opportunity for improvement.

The best strategy developed in this thesis, at least when using our preprocessing algorithm, is a hybrid strategy in which we simply switch from a SetPP to an AllSUPP strategy. A more sophisticated hybrid strategy would be to only perform SetPP on the aircraft part as this is where the preprocessing algorithm is most detrimental when used in conjunction with SUPPs.

The major remaining obstacle to solving larger instances is the time spent solving LPs. The RMPs in the largest instances have up to 3346 constraints, which makes them too large to be solved efficiently in a column generation framework. Several strategies have been developed in the literature to reduce the size of the RMP. Constraints can be aggregated, e.g. (Elhallaoui et al., 2010) or (Saddoune et al., 2011), or the problem can be decomposed using Bender’s decomposition, e.g. (Sandhu and Klabjan) or (Mercier and Soumis, 2007).
Appendix A

Numerical Results for Pricing Problem Selection Strategies
### Appendix A. Numerical Results for Pricing Problem Selection Strategies

<table>
<thead>
<tr>
<th>Inst.</th>
<th>AllPP+3</th>
<th>5-S-1</th>
<th>5-S-2</th>
<th>5-L-1</th>
<th>5-L2</th>
<th>3-S-A</th>
<th>3-L-A</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>+3</td>
<td>+5</td>
<td>+10</td>
<td>+15</td>
<td>+20</td>
<td>+30</td>
<td></td>
</tr>
<tr>
<td>5-S-1</td>
<td>383</td>
<td>261</td>
<td>281</td>
<td>273</td>
<td>358</td>
<td>309</td>
<td>352</td>
</tr>
<tr>
<td>5-S-2</td>
<td>335</td>
<td>344</td>
<td>328</td>
<td>325</td>
<td>292</td>
<td>409</td>
<td>403</td>
</tr>
<tr>
<td>5-L-1</td>
<td>688</td>
<td>616</td>
<td>619</td>
<td>617</td>
<td>633</td>
<td>683</td>
<td>792</td>
</tr>
<tr>
<td>5-L2</td>
<td>822</td>
<td>653</td>
<td>617</td>
<td>616</td>
<td>663</td>
<td>669</td>
<td>810</td>
</tr>
<tr>
<td>3-S-A</td>
<td>815</td>
<td>855</td>
<td>905</td>
<td>917</td>
<td>762</td>
<td>966</td>
<td>1004</td>
</tr>
<tr>
<td>3-L-A</td>
<td>692</td>
<td>769</td>
<td>683</td>
<td>743</td>
<td>679</td>
<td>790</td>
<td>861</td>
</tr>
<tr>
<td></td>
<td>5-S-1</td>
<td>93</td>
<td>46</td>
<td>39</td>
<td>32</td>
<td>47</td>
<td>36</td>
</tr>
<tr>
<td></td>
<td>5-S-2</td>
<td>87</td>
<td>65</td>
<td>67</td>
<td>53</td>
<td>37</td>
<td>49</td>
</tr>
<tr>
<td></td>
<td>5-L-1</td>
<td>175</td>
<td>103</td>
<td>93</td>
<td>86</td>
<td>69</td>
<td>91</td>
</tr>
<tr>
<td></td>
<td>5-L2</td>
<td>201</td>
<td>112</td>
<td>91</td>
<td>85</td>
<td>77</td>
<td>74</td>
</tr>
<tr>
<td></td>
<td>3-S-A</td>
<td>147</td>
<td>116</td>
<td>115</td>
<td>122</td>
<td>64</td>
<td>101</td>
</tr>
<tr>
<td></td>
<td>3-L-A</td>
<td>136</td>
<td>113</td>
<td>85</td>
<td>72</td>
<td>62</td>
<td>85</td>
</tr>
<tr>
<td></td>
<td>5-S-1</td>
<td>270</td>
<td>199</td>
<td>225</td>
<td>223</td>
<td>288</td>
<td>252</td>
</tr>
<tr>
<td></td>
<td>5-S-2</td>
<td>229</td>
<td>260</td>
<td>247</td>
<td>301</td>
<td>175</td>
<td>235</td>
</tr>
<tr>
<td></td>
<td>5-L-1</td>
<td>462</td>
<td>467</td>
<td>477</td>
<td>476</td>
<td>509</td>
<td>535</td>
</tr>
<tr>
<td></td>
<td>5-L2</td>
<td>571</td>
<td>493</td>
<td>478</td>
<td>481</td>
<td>532</td>
<td>540</td>
</tr>
<tr>
<td></td>
<td>3-S-A</td>
<td>592</td>
<td>664</td>
<td>716</td>
<td>715</td>
<td>524</td>
<td>779</td>
</tr>
<tr>
<td></td>
<td>3-L-A</td>
<td>481</td>
<td>584</td>
<td>526</td>
<td>596</td>
<td>544</td>
<td>625</td>
</tr>
<tr>
<td></td>
<td>5-S-1</td>
<td>300</td>
<td>380</td>
<td>310</td>
<td>274</td>
<td>315</td>
<td>255</td>
</tr>
<tr>
<td></td>
<td>5-S-2</td>
<td>285</td>
<td>512</td>
<td>497</td>
<td>391</td>
<td>265</td>
<td>300</td>
</tr>
<tr>
<td></td>
<td>5-L-1</td>
<td>212</td>
<td>330</td>
<td>336</td>
<td>279</td>
<td>236</td>
<td>241</td>
</tr>
<tr>
<td></td>
<td>5-L2</td>
<td>247</td>
<td>384</td>
<td>311</td>
<td>288</td>
<td>254</td>
<td>235</td>
</tr>
<tr>
<td></td>
<td>3-S-A</td>
<td>150</td>
<td>378</td>
<td>410</td>
<td>333</td>
<td>175</td>
<td>262</td>
</tr>
<tr>
<td></td>
<td>3-L-A</td>
<td>135</td>
<td>364</td>
<td>270</td>
<td>239</td>
<td>183</td>
<td>235</td>
</tr>
<tr>
<td></td>
<td>5-S-1</td>
<td>18699</td>
<td>12290</td>
<td>16398</td>
<td>24853</td>
<td>34391</td>
<td>36265</td>
</tr>
<tr>
<td></td>
<td>5-S-2</td>
<td>16951</td>
<td>15700</td>
<td>18279</td>
<td>26113</td>
<td>29843</td>
<td>44850</td>
</tr>
<tr>
<td></td>
<td>5-L-1</td>
<td>28353</td>
<td>17738</td>
<td>22925</td>
<td>31668</td>
<td>42521</td>
<td>50471</td>
</tr>
<tr>
<td></td>
<td>5-L2</td>
<td>30912</td>
<td>18069</td>
<td>23371</td>
<td>31387</td>
<td>39437</td>
<td>47607</td>
</tr>
<tr>
<td></td>
<td>3-S-A</td>
<td>30592</td>
<td>19400</td>
<td>26224</td>
<td>34015</td>
<td>39788</td>
<td>50549</td>
</tr>
<tr>
<td></td>
<td>3-L-A</td>
<td>28635</td>
<td>20399</td>
<td>22912</td>
<td>34166</td>
<td>39828</td>
<td>48541</td>
</tr>
</tbody>
</table>

Table A.1: Run times in seconds (top), time spent pricing (s) (2nd from top), time spent solving LPs (s) (3rd from top), number of CG iterations (4th from top), and number of columns generated (bottom) when solving the small instances using setting AllPP+3 or the SetPP/5/10 strategy while adding 3, 5, 10, 15, 20, or 30 columns.
<table>
<thead>
<tr>
<th>Inst.</th>
<th>+3</th>
<th>+5</th>
<th>+10</th>
<th>+15</th>
<th>+20</th>
<th>+30</th>
</tr>
</thead>
<tbody>
<tr>
<td>5-S-1</td>
<td>283</td>
<td>256</td>
<td>267</td>
<td>271</td>
<td>297</td>
<td>376</td>
</tr>
<tr>
<td>5-S-2</td>
<td>306</td>
<td>305</td>
<td>309</td>
<td>318</td>
<td>353</td>
<td>400</td>
</tr>
<tr>
<td>5-L-1</td>
<td>596</td>
<td>585</td>
<td>622</td>
<td>639</td>
<td>581</td>
<td>738</td>
</tr>
<tr>
<td>5-L2</td>
<td>599</td>
<td>609</td>
<td>618</td>
<td>538</td>
<td>674</td>
<td>797</td>
</tr>
<tr>
<td>3-S-A</td>
<td>852</td>
<td>782</td>
<td>821</td>
<td>780</td>
<td>827</td>
<td>847</td>
</tr>
<tr>
<td>3-L-A</td>
<td>763</td>
<td>752</td>
<td>711</td>
<td>702</td>
<td>689</td>
<td>763</td>
</tr>
<tr>
<td>5-S-1</td>
<td>56</td>
<td>34</td>
<td>45</td>
<td>37</td>
<td>27</td>
<td>36</td>
</tr>
<tr>
<td>5-S-2</td>
<td>60</td>
<td>55</td>
<td>49</td>
<td>42</td>
<td>48</td>
<td>43</td>
</tr>
<tr>
<td>5-L-1</td>
<td>116</td>
<td>89</td>
<td>98</td>
<td>86</td>
<td>66</td>
<td>74</td>
</tr>
<tr>
<td>5-L2</td>
<td>105</td>
<td>99</td>
<td>70</td>
<td>45</td>
<td>86</td>
<td>86</td>
</tr>
<tr>
<td>3-S-A</td>
<td>98</td>
<td>76</td>
<td>77</td>
<td>72</td>
<td>105</td>
<td>65</td>
</tr>
<tr>
<td>3-L-A</td>
<td>91</td>
<td>96</td>
<td>95</td>
<td>67</td>
<td>52</td>
<td>52</td>
</tr>
<tr>
<td>5-S-1</td>
<td>211</td>
<td>206</td>
<td>203</td>
<td>214</td>
<td>250</td>
<td>315</td>
</tr>
<tr>
<td>5-S-2</td>
<td>230</td>
<td>231</td>
<td>241</td>
<td>255</td>
<td>282</td>
<td>331</td>
</tr>
<tr>
<td>5-L-1</td>
<td>432</td>
<td>443</td>
<td>470</td>
<td>499</td>
<td>463</td>
<td>604</td>
</tr>
<tr>
<td>5-L2</td>
<td>448</td>
<td>463</td>
<td>500</td>
<td>446</td>
<td>535</td>
<td>651</td>
</tr>
<tr>
<td>3-S-A</td>
<td>680</td>
<td>635</td>
<td>667</td>
<td>630</td>
<td>640</td>
<td>704</td>
</tr>
<tr>
<td>3-L-A</td>
<td>600</td>
<td>582</td>
<td>538</td>
<td>559</td>
<td>571</td>
<td>633</td>
</tr>
<tr>
<td>5-S-1</td>
<td>488</td>
<td>346</td>
<td>426</td>
<td>349</td>
<td>238</td>
<td>265</td>
</tr>
<tr>
<td>5-S-2</td>
<td>549</td>
<td>489</td>
<td>438</td>
<td>362</td>
<td>430</td>
<td>341</td>
</tr>
<tr>
<td>5-L-1</td>
<td>520</td>
<td>387</td>
<td>372</td>
<td>322</td>
<td>232</td>
<td>275</td>
</tr>
<tr>
<td>5-L2</td>
<td>466</td>
<td>378</td>
<td>285</td>
<td>206</td>
<td>346</td>
<td>333</td>
</tr>
<tr>
<td>3-S-A</td>
<td>395</td>
<td>302</td>
<td>258</td>
<td>237</td>
<td>328</td>
<td>207</td>
</tr>
<tr>
<td>3-L-A</td>
<td>467</td>
<td>379</td>
<td>331</td>
<td>226</td>
<td>199</td>
<td>199</td>
</tr>
<tr>
<td>5-S-1</td>
<td>11084</td>
<td>12820</td>
<td>19188</td>
<td>25090</td>
<td>30926</td>
<td>48487</td>
</tr>
<tr>
<td>5-S-2</td>
<td>13167</td>
<td>16178</td>
<td>21683</td>
<td>29933</td>
<td>35333</td>
<td>48283</td>
</tr>
<tr>
<td>5-L-1</td>
<td>15565</td>
<td>18623</td>
<td>33592</td>
<td>36298</td>
<td>39838</td>
<td>58038</td>
</tr>
<tr>
<td>5-L2</td>
<td>15209</td>
<td>18902</td>
<td>26766</td>
<td>30627</td>
<td>40186</td>
<td>61355</td>
</tr>
<tr>
<td>3-S-A</td>
<td>17732</td>
<td>19883</td>
<td>28203</td>
<td>33027</td>
<td>41579</td>
<td>49706</td>
</tr>
<tr>
<td>3-L-A</td>
<td>16350</td>
<td>20736</td>
<td>25936</td>
<td>32933</td>
<td>40755</td>
<td>52127</td>
</tr>
</tbody>
</table>

Table A.2: Run times in seconds (top), time spent pricing (s) (2nd from top), time spent solving LPs (s) (3rd from top), number of CG iterations (4th from top), and number of columns generated (bottom) when solving the small instances using the SetPP/7/15 strategy while adding 3, 5, 10, 15, 20, or 30 columns.
### Table A.3: Run times in seconds (top), time spent pricing (s) (2nd from top), time spent solving LPs (s) (3rd from top), number of CG iterations (4th from top), and number of columns generated (bottom) when solving the small instances using the SetPP/10/20 strategy while adding 3, 5, 10, 15, 20, or 30 columns.
<table>
<thead>
<tr>
<th>Instance</th>
<th>SglPP+10</th>
<th>SglPP+25</th>
<th>SglPP+50</th>
<th>SglPP+100</th>
<th>SglPP+150</th>
</tr>
</thead>
<tbody>
<tr>
<td>5-S-1</td>
<td>334</td>
<td>335</td>
<td>323</td>
<td>307</td>
<td>374</td>
</tr>
<tr>
<td>5-S-2</td>
<td>337</td>
<td>339</td>
<td>365</td>
<td>347</td>
<td>376</td>
</tr>
<tr>
<td>5-L-1</td>
<td>843</td>
<td>865</td>
<td>826</td>
<td>837</td>
<td>791</td>
</tr>
<tr>
<td>5-L2</td>
<td>882</td>
<td>881</td>
<td>956</td>
<td>829</td>
<td>941</td>
</tr>
<tr>
<td>3-S-A</td>
<td>1169</td>
<td>1263</td>
<td>1162</td>
<td>1112</td>
<td>1306</td>
</tr>
<tr>
<td>3-L-A</td>
<td>968</td>
<td>922</td>
<td>970</td>
<td>991</td>
<td>1134</td>
</tr>
<tr>
<td>5-S-1</td>
<td>41</td>
<td>37</td>
<td>27</td>
<td>23</td>
<td>33</td>
</tr>
<tr>
<td>5-S-2</td>
<td>42</td>
<td>41</td>
<td>41</td>
<td>38</td>
<td>35</td>
</tr>
<tr>
<td>5-L-1</td>
<td>84</td>
<td>78</td>
<td>79</td>
<td>70</td>
<td>59</td>
</tr>
<tr>
<td>5-L2</td>
<td>86</td>
<td>99</td>
<td>53</td>
<td>46</td>
<td>71</td>
</tr>
<tr>
<td>3-S-A</td>
<td>92</td>
<td>118</td>
<td>67</td>
<td>59</td>
<td>88</td>
</tr>
<tr>
<td>3-L-A</td>
<td>85</td>
<td>62</td>
<td>53</td>
<td>54</td>
<td>115</td>
</tr>
<tr>
<td>5-S-1</td>
<td>276</td>
<td>278</td>
<td>277</td>
<td>265</td>
<td>317</td>
</tr>
<tr>
<td>5-S-2</td>
<td>276</td>
<td>278</td>
<td>303</td>
<td>287</td>
<td>316</td>
</tr>
<tr>
<td>5-L-1</td>
<td>706</td>
<td>733</td>
<td>690</td>
<td>708</td>
<td>673</td>
</tr>
<tr>
<td>5-L2</td>
<td>746</td>
<td>726</td>
<td>848</td>
<td>730</td>
<td>811</td>
</tr>
<tr>
<td>3-S-A</td>
<td>998</td>
<td>1062</td>
<td>1016</td>
<td>969</td>
<td>1127</td>
</tr>
<tr>
<td>3-L-A</td>
<td>806</td>
<td>786</td>
<td>841</td>
<td>855</td>
<td>929</td>
</tr>
<tr>
<td>5-S-1</td>
<td>804</td>
<td>753</td>
<td>569</td>
<td>457</td>
<td>548</td>
</tr>
<tr>
<td>5-S-2</td>
<td>850</td>
<td>752</td>
<td>678</td>
<td>607</td>
<td>612</td>
</tr>
<tr>
<td>5-L-1</td>
<td>938</td>
<td>793</td>
<td>750</td>
<td>597</td>
<td>524</td>
</tr>
<tr>
<td>5-L2</td>
<td>1009</td>
<td>954</td>
<td>750</td>
<td>615</td>
<td>658</td>
</tr>
<tr>
<td>3-S-A</td>
<td>858</td>
<td>878</td>
<td>642</td>
<td>588</td>
<td>580</td>
</tr>
<tr>
<td>3-L-A</td>
<td>981</td>
<td>838</td>
<td>621</td>
<td>544</td>
<td>657</td>
</tr>
<tr>
<td>5-S-1</td>
<td>11545</td>
<td>15728</td>
<td>19753</td>
<td>27052</td>
<td>35029</td>
</tr>
<tr>
<td>5-S-2</td>
<td>11716</td>
<td>16277</td>
<td>21810</td>
<td>27131</td>
<td>35721</td>
</tr>
<tr>
<td>5-L-1</td>
<td>15372</td>
<td>20971</td>
<td>27533</td>
<td>35707</td>
<td>41175</td>
</tr>
<tr>
<td>5-L2</td>
<td>16393</td>
<td>21476</td>
<td>30057</td>
<td>37763</td>
<td>47391</td>
</tr>
<tr>
<td>3-S-A</td>
<td>17772</td>
<td>24143</td>
<td>30348</td>
<td>43910</td>
<td>52019</td>
</tr>
<tr>
<td>3-L-A</td>
<td>16100</td>
<td>23763</td>
<td>28446</td>
<td>42177</td>
<td>55052</td>
</tr>
</tbody>
</table>

Table A.4: Run times in seconds (top), time spent pricing (s) (2nd from top), time spent solving LPs (s) (3rd from top), number of CG iterations (4th from top), and number of columns generated (bottom) when solving the small instances using the SglPP strategy while adding 10, 25, 50, 100, or 150 columns.


## Appendix B

### Numerical Results for Large Instances

<table>
<thead>
<tr>
<th>Inst.</th>
<th>Basic</th>
<th>B&amp;P</th>
<th>HeurPr</th>
<th>SetPP</th>
<th>SwSUPP</th>
</tr>
</thead>
<tbody>
<tr>
<td>7-S-1</td>
<td>3871</td>
<td>1843</td>
<td>1439</td>
<td>1313</td>
<td>1010</td>
</tr>
<tr>
<td>7-S-2</td>
<td>4130</td>
<td>2417</td>
<td>1569</td>
<td>1422</td>
<td>1117</td>
</tr>
<tr>
<td>7-H-1</td>
<td>14177</td>
<td>10023</td>
<td>5479</td>
<td>3130</td>
<td>2505</td>
</tr>
<tr>
<td>7-H-2</td>
<td>12340</td>
<td>9128</td>
<td>4088</td>
<td>2969</td>
<td>2626</td>
</tr>
<tr>
<td>5-S-A</td>
<td>15199*</td>
<td>30242*</td>
<td>12602</td>
<td>12539</td>
<td>8837</td>
</tr>
<tr>
<td>5-L-A</td>
<td>15873*</td>
<td>32997*</td>
<td>7996</td>
<td>6543</td>
<td>6443</td>
</tr>
</tbody>
</table>

Table B.1: Time spent solving LPs (s) when solving the large instances using the best settings of each chapter. A * indicates that solving was aborted due to reaching the time limit of 12 hours.

<table>
<thead>
<tr>
<th>Inst.</th>
<th>Basic</th>
<th>B&amp;P</th>
<th>HeurPr</th>
<th>SetPP</th>
<th>SwSUPP</th>
</tr>
</thead>
<tbody>
<tr>
<td>7-S-1</td>
<td>225</td>
<td>348</td>
<td>346</td>
<td>379</td>
<td>339</td>
</tr>
<tr>
<td>7-S-2</td>
<td>243</td>
<td>370</td>
<td>370</td>
<td>386</td>
<td>355</td>
</tr>
<tr>
<td>7-H-1</td>
<td>377</td>
<td>605</td>
<td>587</td>
<td>582</td>
<td>583</td>
</tr>
<tr>
<td>7-H-2</td>
<td>240</td>
<td>574</td>
<td>535</td>
<td>572</td>
<td>519</td>
</tr>
<tr>
<td>5-S-A</td>
<td>1*</td>
<td>65*</td>
<td>760</td>
<td>769</td>
<td>723</td>
</tr>
<tr>
<td>5-L-A</td>
<td>1*</td>
<td>215*</td>
<td>730</td>
<td>748</td>
<td>689</td>
</tr>
</tbody>
</table>

Table B.2: Number of branch-and-price nodes evaluated when solving the large instances using the best settings of each chapter. A * indicates that solving was aborted due to reaching the time limit of 12 hours.
APPENDIX B. NUMERICAL RESULTS FOR LARGE INSTANCES

<table>
<thead>
<tr>
<th>Inst.</th>
<th>Basic</th>
<th>B&amp;P</th>
<th>HeurPr</th>
<th>SetPP</th>
<th>SwSUPP</th>
</tr>
</thead>
<tbody>
<tr>
<td>7-S-1</td>
<td>25.42</td>
<td>2.29</td>
<td>2.25</td>
<td>5.24</td>
<td>3.03</td>
</tr>
<tr>
<td>7-S-2</td>
<td>3.50</td>
<td>1.17</td>
<td>2.34</td>
<td>1.68</td>
<td>1.83</td>
</tr>
<tr>
<td>7-H-1</td>
<td>1.53</td>
<td>1.75</td>
<td>1.32</td>
<td>1.01</td>
<td>1.23</td>
</tr>
<tr>
<td>7-H-2</td>
<td>0.16</td>
<td>2.01</td>
<td>0.50</td>
<td>1.45</td>
<td>1.13</td>
</tr>
<tr>
<td>5-S-A</td>
<td>*</td>
<td>*</td>
<td>1.07</td>
<td>1.46</td>
<td>1.09</td>
</tr>
<tr>
<td>5-L-A</td>
<td>*</td>
<td>*</td>
<td>1.12</td>
<td>1.93</td>
<td>1.13</td>
</tr>
</tbody>
</table>

Table B.3: IP gaps when solving the large instances using the best settings of each chapter. A * indicates that solving was aborted due to reaching the time limit of 12 hours.
Bibliography


343


CASA. Civil Aviation Orders, 2012. retrieved 03.10.2012., 94


ZIB. *Solving Constraint Integer Programs, Konrad-Zuse-Zentrum für Informationstechnik Berlin*, 2013. 122
Glossary

A-check  Type of maintenance that has to be carried out frequently.

Actual time (AT)  Indicator for a maintenance type. The corresponding maintenance check is triggered if the limit for this indicator/maintenance pair is exceeded otherwise.

AllPP  Strategy that solves all original pricing problems in every iteration.

AllSUPP  Strategy that solves all superimposed pricing problems in every iteration.

Aircraft routing problem  Generates generic aircraft routes such that all flights are covered.

Applicable credit  Amount of pay that a crew member receives per duty/flight.

ARPP  Aircraft routing pricing problem.

ASUPP  Superimposed pricing problem for aircraft.

Backward path  Path generated from the sink node $t$ to a preceding node.

Block hours  Time spend flying plus taxi time, which is the time spent at an airport once all aircraft doors are closed.

BSP($\kappa$)  Standard backward shortest path calculations used in the preprocessing algorithm.

C-check  Type of maintenance that has to be carried very infrequently.

Concatenation  Joining a forward and a backward path.
Connection network  Type of representation for a flight schedule. A node represents a flight, an arc between two nodes indicates that a feasible connection between the corresponding flights exists.

CPPP  Crew pairing pricing problems.

Crew base  Airport that a crew is based at. The crew must end its pairing here.

Crew block  A crew block represents all crews that have the same original Work-Periods and for which the same rules apply. The start and end times and the start and end locations have to be the same.

Crew pairing problem  Generates generic pairings such that all flights are covered.

Crew rostering problem  Pairings are assigned to actual crew members.

CSUPP  Superimposed pricing problem for crews.

D-check  Type of maintenance that has to be carried very rarely.

DOO  Day of operations. The day a schedule is put into action.

Feasible connection  A sequence of two flights where the first flight ends at the same airport the second departs from. Sufficient time must be allowed between the two flights.

Duty connection (DC)  A connection that does not qualify as a rest period for crews.

Duty hours  Duration of a duty including briefing and de-briefing periods.

Fleet assignment problem  Assigns an aircraft type to every flight in the schedule.

Flight connection  Connection between two flights.

Flying time (FT)  Indicator for a maintenance type. The corresponding maintenance check is triggered if the limit for this indicator/maintenance pair is exceeded otherwise.
**FSP(κ)** Standard forward shortest path calculations used in the preprocessing algorithm.

**HEAM, Heavy maintenance** Type of maintenance. Requires a hangar and is fixed in time and location. Is scheduled externally. The correct tail has to be at the correct maintenance station before the HEAM commences.

**Hub-and-spoke networks** Type of flight network. Low demand airports (spokes) are connected to larger airports (hubs) via direct flights but usually no direct flight between two spokes exists.

**Layover connection** A connection that is of sufficient duration so that a crew can sleep. Further differentiated into regular layovers (RL) and long layovers (LL).

**Minimum credit** Each duty either incurs a cost equal to the accumulated credit value or a minimum credit value \( \zeta \).

**MOPP, Maintenance opportunities** Provides slack in the schedule to absorb unscheduled maintenance. Scheduling this maintenance check does not incur any cost.

**MST, Minimum sit time** The minimum connection time for a crew connection.

**Restricted connection** A connection with a duration between MST and MST + 45 minutes. Such a connection is likely to propagate delay if the first flight is delayed.

**MTT, Minimum turn time** The minimum time for an aircraft connection. Also known as turnaround time.

**OPP** Original pricing problem.

**Pairing Continues** Crews in this crew block are on a pairing at the beginning of the planning horizon. Their initial location is different from the crew base.

**Pairing Starts** The crews represented by a crew block in this category start a new pairing during the planning horizon. The initial location is the crew base and the crew becomes available at or after the start of the planning horizon.
**Period 1** First part of the planning horizon. Exactly one aircraft and one crew must be assigned to each flight in this period.

**Period 2** Second part of the planning horizon. Up to one crew can be assigned to each flight in this period.

**Point-to-point networks** Type of flight network. No centralised hub exists. Instead, many connections between various airports are offered.

**Pressure cycles (PC)** Indicator for a maintenance type. The corresponding maintenance check is triggered if the limit for this indicator/maintenance pair is exceeded otherwise.

**Progressive maintenance checks** Newer, task-driven maintenance approach. Used especially for new generation aircraft. Resulting maintenance checks are much shorter.

**RCSPP** Resource constrained shortest path problem.

**RCSPP-R** Resource constrained shortest path problem with replenishments.

**RCSPP-RAE** Airline related extension of the RCSPP-R. The pricing problems in this thesis.

**REF** Resource extension function.

**Rule 3** Block hour accumulation over consecutive duties. Defined in Table 5.5.

**Schedule design problem** Uses demand forecast to determine which markets are serviced with what frequency.

**SetPP** Strategy that solves a subset of original pricing problems in every iteration.

**SetSUPP** Strategy that solves a subset of superimposed pricing problems in every iteration.

**SglPP** Strategy that solves one original pricing problem in every iteration.

**SglSUPP** Strategy that solves one superimposed pricing problem in every iteration.

**Sink connections** A connection from a flight node to the sink node $t$. 
SLIM, Scheduled line maintenance  Maintenance with high or medium frequency and a duration of several hours. The solver schedules these maintenance checks whenever necessary.

Source connections  A connection from source node $s$ to a flight node.

SUPP  Superimposed pricing problem. Aggregation of original pricing problems.

Tail assignment problem  The generic routes are assigned to individual aircraft.

Time-line network  Type of representation for a flight schedule. Also called time-space network. Each airport is represented by a time line. Nodes along the time line correspond to arrival and departure times of flights.

Work-Period  Period of time a crew is scheduled to work according to the original roster.

$k$-path  A path from either $s$ to node $i$ or from $t$ to $i$. This path gives the lowest consumption of resource $k$, where the accumulation of $k$ is since the last replenishment of $k$. 
# Mathematical Notation

The following is a list of mathematical notation used in Chapters 5 through 10. The column labelled Page gives the page on which the variable is defined while Description gives a short description that may not suffice as a definition.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Page</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A$</td>
<td>98</td>
<td>Set of all aircraft.</td>
</tr>
<tr>
<td>$a$</td>
<td>98</td>
<td>Index of single aircraft.</td>
</tr>
<tr>
<td>$A$</td>
<td>289</td>
<td>Set of superimposed pricing problems for aircraft.</td>
</tr>
<tr>
<td>$B$</td>
<td>99</td>
<td>Set of all crew blocks.</td>
</tr>
<tr>
<td>$b$</td>
<td>99</td>
<td>Index of single crew block.</td>
</tr>
<tr>
<td>$B$</td>
<td>290</td>
<td>Set of superimposed pricing problems for crew blocks.</td>
</tr>
<tr>
<td>Brief</td>
<td>94</td>
<td>Duration of briefing period.</td>
</tr>
<tr>
<td>$\dot{C}$</td>
<td>104</td>
<td>Set of all connections for pricing problem.</td>
</tr>
<tr>
<td>$C$</td>
<td>104</td>
<td>Set of connections between two flights.</td>
</tr>
<tr>
<td>$C^+$</td>
<td>104</td>
<td>Set of connections between $s$ and flight nodes $j$.</td>
</tr>
<tr>
<td>$C^-$</td>
<td>104</td>
<td>Set of connections between flight $i$ and sink node $t$.</td>
</tr>
<tr>
<td>$C_{Sh}$</td>
<td>99</td>
<td>Set of all short connections.</td>
</tr>
<tr>
<td>$C_{Re}$</td>
<td>99</td>
<td>Set of all restricted connections.</td>
</tr>
<tr>
<td>Symbol</td>
<td>Page</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>------</td>
<td>-------------</td>
</tr>
<tr>
<td>$c^{pb}$</td>
<td>99</td>
<td>Cost of pairing $p$.</td>
</tr>
<tr>
<td>$c^{ra}$</td>
<td>99</td>
<td>Cost of route $r$.</td>
</tr>
<tr>
<td>$c_j$</td>
<td>117</td>
<td>Credit value of flight $j$.</td>
</tr>
<tr>
<td>$c_{ij}$</td>
<td>115</td>
<td>Cost of a connection. Only layover connections actually have a cost associated with them.</td>
</tr>
<tr>
<td>$c_{ijg}$</td>
<td>103</td>
<td>Cost of arc $(i,j,g)$.</td>
</tr>
<tr>
<td>$C_L$</td>
<td>114</td>
<td>Set of all layover connections.</td>
</tr>
<tr>
<td>$C_{RL}$</td>
<td>114</td>
<td>Set of all regular layover connections.</td>
</tr>
<tr>
<td>$C_{LL}$</td>
<td>114</td>
<td>Set of all layover connections.</td>
</tr>
<tr>
<td>$c_m$</td>
<td>112</td>
<td>Maximum cost to schedule maintenance type $m$.</td>
</tr>
<tr>
<td>$c^m_{ij}$</td>
<td>112</td>
<td>Cost of scheduling maintenance type $m$ on connection $(i,j)$.</td>
</tr>
<tr>
<td>$c^M_{ijg}$</td>
<td>113</td>
<td>Total maintenance cost along arc $(i,j,g)$.</td>
</tr>
<tr>
<td>$\overleftarrow{c}_L$</td>
<td>178</td>
<td>Cost of the forward path that label $L$ represents</td>
</tr>
<tr>
<td>$\overrightarrow{c}_i$</td>
<td>179</td>
<td>Cost of a forward path to node $i$.</td>
</tr>
<tr>
<td>$\overrightarrow{c}_{\kappa i}$</td>
<td>189</td>
<td>Cost of least $\kappa$-path from $s$ to $i$.</td>
</tr>
<tr>
<td>$\overleftarrow{c}_{0i}$</td>
<td>189</td>
<td>Lower bound on cost to reach node $i$ from the source node $s$.</td>
</tr>
<tr>
<td>$\overrightarrow{c}_{s i}$</td>
<td>189</td>
<td>Cost of least $\kappa$-path from $t$ to $i$.</td>
</tr>
<tr>
<td>$\overleftarrow{c}_{0i}$</td>
<td>189</td>
<td>Lower bound on cost to reach node $i$ from the sink node $t$.</td>
</tr>
<tr>
<td>$d_{st}$</td>
<td>103</td>
<td>Duration the aircraft or crew block is available.</td>
</tr>
<tr>
<td>$d_i$</td>
<td>103</td>
<td>Duration of flight $i$.</td>
</tr>
<tr>
<td>Symbol</td>
<td>Page</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>------</td>
<td>-------------</td>
</tr>
<tr>
<td>$d_{ij}$</td>
<td>99</td>
<td>Duration of connection $(i,j)$.</td>
</tr>
<tr>
<td>$d_{ij}^k$</td>
<td>110</td>
<td>Time between end of a maintenance check (that $k$ relate to) and departure of flight $j$.</td>
</tr>
<tr>
<td>$\bar{D}$</td>
<td>161</td>
<td>Value used for tightening of LP gaps and suppressing pricing. Function of the number of flights in an instance.</td>
</tr>
<tr>
<td>De-Brief</td>
<td>94</td>
<td>Duration of de-briefing period.</td>
</tr>
<tr>
<td>$\hat{E}$</td>
<td>103</td>
<td>Set of all arcs in pricing problem.</td>
</tr>
<tr>
<td>$E$</td>
<td>103</td>
<td>Set of arcs in pricing problem that are between two flight nodes.</td>
</tr>
<tr>
<td>$E^+$</td>
<td>103</td>
<td>Set of source arcs in pricing problem.</td>
</tr>
<tr>
<td>$E^-$</td>
<td>103</td>
<td>Set of sink arcs in pricing problem.</td>
</tr>
<tr>
<td>$\hat{E}_k$</td>
<td>103</td>
<td>Replenishment arcs for resource $k$.</td>
</tr>
<tr>
<td>$f^A_{ij}$</td>
<td>153</td>
<td>Fractional value of aircraft connection $(i,j)$.</td>
</tr>
<tr>
<td>$f^B_{ij}$</td>
<td>153</td>
<td>Fractional value of crew connection $(i,j)$.</td>
</tr>
<tr>
<td>$G = (\hat{N}, \hat{E})$</td>
<td>103</td>
<td>Graph of pricing problem.</td>
</tr>
<tr>
<td>$G_{ij}$</td>
<td>102</td>
<td>Set of arcs for connection $(i,j)$.</td>
</tr>
<tr>
<td>$g$</td>
<td>102</td>
<td>Index over set of arcs for connection $(i,j)$. $g \in G_{ij}$.</td>
</tr>
<tr>
<td>$h^{ph}_{ij}$</td>
<td>100</td>
<td>1 if connection $(i,j) \in C_{Sh} \cup C_{Re}$ is part of pairing $p$.</td>
</tr>
<tr>
<td>$h^{ra}_{ij}$</td>
<td>99</td>
<td>1 if connection $(i,j) \in C_{Sh} \cup C_{Re}$ is part of route $r$.</td>
</tr>
<tr>
<td>$i$</td>
<td>98</td>
<td>Index over set of nodes.</td>
</tr>
<tr>
<td>$j$</td>
<td>98</td>
<td>Index over set of nodes.</td>
</tr>
<tr>
<td>Symbol</td>
<td>Page</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>------</td>
<td>-------------</td>
</tr>
<tr>
<td>$K$</td>
<td>103</td>
<td>Set of resources in the pricing problem.</td>
</tr>
<tr>
<td>$\hat{K}$</td>
<td>188</td>
<td>A path is always calculated w.r.t. $\kappa \in \hat{K}$, $\hat{K} = K \cup 0$, where 0 indicates cost.</td>
</tr>
<tr>
<td>$k$</td>
<td>103</td>
<td>Index over weight set $K$.</td>
</tr>
<tr>
<td>$L$</td>
<td>178</td>
<td>A label.</td>
</tr>
<tr>
<td>$L_i$</td>
<td>184</td>
<td>Set of non-dominated labels at node $i$.</td>
</tr>
<tr>
<td>$M$</td>
<td>105</td>
<td>Set of maintenance types that apply to aircraft $a$.</td>
</tr>
<tr>
<td>$m$</td>
<td>105</td>
<td>Index over set of maintenance types.</td>
</tr>
<tr>
<td>$M_{ijg}$</td>
<td>106</td>
<td>Set of maintenance checks that are carried out on arc $(i, j, g)$.</td>
</tr>
<tr>
<td>$M^H$</td>
<td>105</td>
<td>Set of heavy maintenance types for aircraft $a$.</td>
</tr>
<tr>
<td>$M^S$</td>
<td>105</td>
<td>Set of scheduled line maintenance types for aircraft $a$.</td>
</tr>
<tr>
<td>$M^O$</td>
<td>105</td>
<td>Set of maintenance opportunities for aircraft $a$. Contains only one element.</td>
</tr>
<tr>
<td>MB1</td>
<td>95</td>
<td>Limit on the number of block hours in a duty.</td>
</tr>
<tr>
<td>MB2</td>
<td>95</td>
<td>Soft limit on the number of block hours in consecutive duties.</td>
</tr>
<tr>
<td>MD1</td>
<td>95</td>
<td>Limit on the number of duty hours in a duty.</td>
</tr>
<tr>
<td>MD2</td>
<td>95</td>
<td>Soft limit on the number of duty hours in consecutive duties.</td>
</tr>
<tr>
<td>MST</td>
<td>34</td>
<td>Minimum sit time.</td>
</tr>
<tr>
<td>MTB</td>
<td>95</td>
<td>Limit on the total number of block hours in a pairing.</td>
</tr>
<tr>
<td>Symbol</td>
<td>Page</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>------</td>
<td>-------------</td>
</tr>
<tr>
<td>MTT</td>
<td>34</td>
<td>Minimum turn time.</td>
</tr>
<tr>
<td>$N$</td>
<td>98</td>
<td>Set of all flights in planning horizon. Also set of flight nodes in a pricing problem.</td>
</tr>
<tr>
<td>$\hat{N}$</td>
<td>102</td>
<td>Set of all nodes in pricing problem. $\hat{N} = N \cup s \cup t$.</td>
</tr>
<tr>
<td>$N^0$</td>
<td>98</td>
<td>Set of flights that do not need to be covered.</td>
</tr>
<tr>
<td>$N^1$</td>
<td>98</td>
<td>Set of flights that do need to be covered.</td>
</tr>
<tr>
<td>$n_b$</td>
<td>99</td>
<td>Number of crews in crew block $b$.</td>
</tr>
<tr>
<td>$o$</td>
<td>242</td>
<td>Index for general pricing problem.</td>
</tr>
<tr>
<td>$p$</td>
<td>99</td>
<td>Pairing $p \in P_b$.</td>
</tr>
<tr>
<td>$P_b$</td>
<td>99</td>
<td>Set of feasible pairings for crew block $b$.</td>
</tr>
<tr>
<td>$q$</td>
<td>103</td>
<td>Path in a network/pricing problem.</td>
</tr>
<tr>
<td>$\vec{q}_i$</td>
<td>178</td>
<td>Forward path from $s$ to node $i$.</td>
</tr>
<tr>
<td>$Q$</td>
<td>242</td>
<td>Set of all original pricing problems.</td>
</tr>
<tr>
<td>$\vec{q}_{\kappa i}$</td>
<td>188</td>
<td>Subpath w.r.t. to $\kappa \in \hat{K}$ from $s$ to $i$.</td>
</tr>
<tr>
<td>$\vec{q}_{\kappa i}$</td>
<td>189</td>
<td>Subpath w.r.t. to $\kappa \in \hat{K}$ from $t$ to $i$.</td>
</tr>
<tr>
<td>$r$</td>
<td>98</td>
<td>Route $r \in R_a$.</td>
</tr>
<tr>
<td>$R_a$</td>
<td>98</td>
<td>Set of routes for aircraft $a$.</td>
</tr>
<tr>
<td>$s$</td>
<td>102</td>
<td>Source node of pricing problem.</td>
</tr>
<tr>
<td>$t$</td>
<td>102</td>
<td>Sink node of pricing problem.</td>
</tr>
<tr>
<td>$T_{RC}$</td>
<td>87</td>
<td>Maximum duration of a connection to be considered a restricted connection.</td>
</tr>
<tr>
<td>Symbol</td>
<td>Page</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>------</td>
<td>-------------</td>
</tr>
<tr>
<td>$U^k$</td>
<td>103</td>
<td>Limit of resource $k$ in pricing problem.</td>
</tr>
<tr>
<td>$u^k_i$</td>
<td>104</td>
<td>Consumption of resource $k$ on flight $i$.</td>
</tr>
<tr>
<td>$u^k_s$</td>
<td>104</td>
<td>Consumption of resource $k$ at the beginning of the planning horizon.</td>
</tr>
<tr>
<td>$u^k_{sj}$</td>
<td>111</td>
<td>Consumption of resource $k$ before flight $j$.</td>
</tr>
<tr>
<td>$u^k_{ijg}$</td>
<td>103</td>
<td>Consumption of resource $k$ along arc $(i,j,g)$.</td>
</tr>
<tr>
<td>$v^p_i$</td>
<td>99</td>
<td>1 if flight $i$ is covered by pairing $p$, 0 ow.</td>
</tr>
<tr>
<td>$v^r_i$</td>
<td>98</td>
<td>1 if flight $i$ is covered by route $r$, 0 ow.</td>
</tr>
<tr>
<td>$\overrightarrow{w^k}_L$</td>
<td>178</td>
<td>Accumulation of resource $k$ since last repl. of $k$ along the path that label $L$ represents.</td>
</tr>
<tr>
<td>$\overrightarrow{w^k}_i$</td>
<td>179</td>
<td>Accumulation of resource $k$ since last repl. of $k$ along a path to node $i$.</td>
</tr>
<tr>
<td>$\overrightarrow{w^k}_{\kappa i}$</td>
<td>188</td>
<td>Resource accumulation of $k$ along the least cost or least weight path ($\kappa$) from $s$ to $i$.</td>
</tr>
<tr>
<td>$\overrightarrow{w^k}_{\kappa t}$</td>
<td>189</td>
<td>Resource accumulation of $k$ along the least cost or least weight path ($\kappa$) from $t$ to $i$.</td>
</tr>
<tr>
<td>$x^{ra}$</td>
<td>98</td>
<td>1 if route $r$ is chosen for aircraft $a$, 0 ow.</td>
</tr>
<tr>
<td>$y^{pb}$</td>
<td>99</td>
<td>1 if pairing $p$ is chosen for crew block $b$, 0 ow.</td>
</tr>
<tr>
<td>$z_{ij}$</td>
<td>100</td>
<td>1 if connection $(i,j) \in C_{Re}$ is used by a crew block but not by an aircraft, 0 ow.</td>
</tr>
<tr>
<td>$\alpha$</td>
<td>113</td>
<td>Dual variable associated with constraint 5.4.</td>
</tr>
<tr>
<td>$\beta$</td>
<td>115</td>
<td>Dual variable associated with constraint 5.5.</td>
</tr>
<tr>
<td>Symbol</td>
<td>Page</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>------</td>
<td>-------------</td>
</tr>
<tr>
<td>$\gamma$</td>
<td>113</td>
<td>Dual variable associated with constraint 5.1.</td>
</tr>
<tr>
<td>$\delta$</td>
<td>290</td>
<td>Index of CSUPP, $\delta \in \mathcal{B}$, where $\mathcal{B}$ is the set of CSUPPs.</td>
</tr>
<tr>
<td>$\epsilon$</td>
<td>157</td>
<td>If LP gap is smaller than $\epsilon$, we branch early.</td>
</tr>
<tr>
<td>$\zeta$</td>
<td>96</td>
<td>Minimum applicable credit for a duty.</td>
</tr>
<tr>
<td>$\eta$</td>
<td>178</td>
<td>A node in a path.</td>
</tr>
<tr>
<td>$\theta$</td>
<td>115</td>
<td>Dual variable associated with constraint 5.2.</td>
</tr>
<tr>
<td>$\Theta_o$</td>
<td>242</td>
<td>Score of pricing problem $o \in \mathcal{Q}$.</td>
</tr>
<tr>
<td>$\iota$</td>
<td>276</td>
<td>Index of ASUPP, $\iota \in \mathcal{A}$, where $\mathcal{A}$ is the set of ASUPPs.</td>
</tr>
<tr>
<td>$\kappa$</td>
<td>188</td>
<td>Least weight path is with respect to weight $\kappa \in K_a$.</td>
</tr>
<tr>
<td>$\lambda$</td>
<td>115</td>
<td>Dual variable associated with constraint 5.3.</td>
</tr>
<tr>
<td>$\mu^k$</td>
<td>112</td>
<td>A maintenance opportunity (MOPP) is required on average every $\mu^k$ days.</td>
</tr>
<tr>
<td>$\nu_{\kappa i}$</td>
<td>188</td>
<td>Indicator if $\kappa$-path from $s$ to $i$ is infeasible.</td>
</tr>
<tr>
<td>$\bar{\nu}_{\kappa i}$</td>
<td>189</td>
<td>Indicator if $\kappa$-path from $t$ to $i$ is infeasible.</td>
</tr>
<tr>
<td>$\xi^A$</td>
<td>166</td>
<td>Multiplier of fractional value for an aircraft connection.</td>
</tr>
<tr>
<td>$\xi^B$</td>
<td>166</td>
<td>Multiplier of fractional value for a crew connection.</td>
</tr>
<tr>
<td>$\pi_{ij}$</td>
<td>113</td>
<td>Dual variable associated with constraint 5.6.</td>
</tr>
<tr>
<td>$\rho_{ij}$</td>
<td>100</td>
<td>Penalty that is incurred if an aircraft change occurs on a restricted connection.</td>
</tr>
<tr>
<td>$\sigma$</td>
<td>113</td>
<td>Dual variable associated with constraint 5.7.</td>
</tr>
<tr>
<td>$\tau_i$</td>
<td>102</td>
<td>Departure time of flight $i$.</td>
</tr>
<tr>
<td>Symbol</td>
<td>Page</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>------</td>
<td>-------------</td>
</tr>
<tr>
<td>$\tau_s$</td>
<td>103</td>
<td>Time aircraft or crew block becomes available in the planning horizon.</td>
</tr>
<tr>
<td>$\tau_t$</td>
<td>103</td>
<td>Time aircraft or crew block becomes unavailable in the planning horizon.</td>
</tr>
<tr>
<td>$\tau_{ij}$</td>
<td>112</td>
<td>Start time of connection $(i,j)$.</td>
</tr>
<tr>
<td>$\nu$</td>
<td>19</td>
<td>Column generation iteration number.</td>
</tr>
<tr>
<td>$\phi$</td>
<td>92</td>
<td>On average a MOPP has to scheduled every $\phi$ days.</td>
</tr>
<tr>
<td>$\chi$</td>
<td>161</td>
<td>Multiplier to calculate suppressing depth.</td>
</tr>
<tr>
<td>$\psi$</td>
<td>148</td>
<td>Column age limit.</td>
</tr>
<tr>
<td>$\omega$</td>
<td>169</td>
<td>Multiplier to calculate the depth at which pricing is suppressed.</td>
</tr>
<tr>
<td>$\Upsilon$</td>
<td>242</td>
<td>Maximum number of pricing problems to solve per column generation iteration.</td>
</tr>
<tr>
<td>$\Omega$</td>
<td>242</td>
<td>Minimum number of pricing problems to solve per column generation iteration.</td>
</tr>
<tr>
<td>(INT)</td>
<td>100</td>
<td>Master problem.</td>
</tr>
</tbody>
</table>