Radboud University

SOLVING THE TRAIN CREW SCHEDULING PROBLEM

H.W.M. (Jeroen) de Jong

Supervised by Dan Roozemond and Wieb Bosma

Thesis submitted in Partial Fulfillment of the
Requirement for the Degree of
Master of Science

in the
Department of Mathematics
Faculty of Science

September 2017

1. ABSTRACT

In this thesis we study a real-world crew scheduling problem. We will solve the problem by using Column Generation, formulating the problem as a set covering, while modeling the labor rules in the Resource Constrained Shortest Path algorithm. We find the optimal solution by successfully executing the Branch and Price algorithm.
We speed up the whole process by using Interior Point Stabilization to enhance the column generation step that provides us with the information to find improving columns.
Finally we attempt to speed up the bottleneck, which is the generation phase of the column generation scheme, by using a combination of the resource constrained shortest path problem and constraint logic programming.

We show a new way of using the Resource Constrained Shortest Path, forbidding paths in the graph. We learn general lessons applicable to problems that can be solved with Branch and Price, while applying theory to a specific real-world problem. We gain valuable insights in the complex nature of the problem, by discovering what works well, for example branching decisions, and what does not, for example adding too many forbidden paths.

Key words: crew scheduling, column generation, linear programming, resource constrained shortest path, forbidden paths, branch and price, interior point stabilization, constraint logic programming

2. ACKNOWLEDGMENTS

This thesis would have never been possible without the support and expertise of those around me during the eight months it took to do research and write this thesis.

There is definitely one person without whom this thesis would never have gotten as far as it did. A thank you to Dan Roozemond, for allowing me to work with him in Quintiq, for guiding me through the theory and for providing ideas and inspiration. I have enjoyed working with him tremendously. I would also like to thank my second supervisor, Wieb Bosma, for bringing his wealth of experience and knowledge to improve my thesis wherever he could.

My thanks also go to the Fleet and Crew planner algorithm expert team at Quintiq, consisting of Edgar den Boef, Dirk-Jan Hoppenbrouwer, Ng Ee Ann and Koay Jun Ming. They showed me the ropes and helped me through the rough spots. I would like to thank Dirk-Jan in particular for playing so much ping-pong with me and listening to me rant about software bugs, often created by myself. A big thank you to the whole Products team at Quintiq for making my internship so very enjoyable.

I would like to thank my lovely girlfriend Birthe for listening to my boring stories about algorithms during dinner, proofreading my thesis and giving invaluable color advise. (I'm colorblind). I would thank my friends and family for helping me take my mind off my thesis every now and then, by having dinner, organizing weekend trips or discussing life until the wee hours.

Last but certainly not least I would thank you, the reader of this thesis, for your time and interest. I hope you enjoy my work and may you have as much fun solving problems as I did.

CONTENTS

1. Abstract . . . 2

2. Acknowledgments . . . 3

3. Introduction . . . 6
3.1 Problem overview . . . 6
3.1.1 Example . . . 6
3.1.2 Problem complexity . . . 8
3.2 Thesis outline . . . 8

4. Theory . . . 10
4.1 Literature Review . . . 10
4.1.1 Airline crew scheduling and vehicle routing . . . 10
4.1.2 Heuristics and meta-heuristics . . . 11
4.1.3 Set covering . . . 11
4.1.4 Additional techniques . . . 13
4.1.5 Conclusion literature review . . . 14
4.2 Column generation . . . 15
4.2.1 Dantzig-Wolfe decomposition . . . 15
4.2.2 The pricing problem . . . 16
4.3 Branch and Price . . . 17
4.3.1 Branching . . . 17
4.3.2 Bounding . . . 18
4.3.3 Pricing . . . 18
4.3.4 Tree traversal . . . 18
4.4 RCSP . . . 19
4.4.1 Mathematical formulation . . . 20
4.4.2 Dynamic programming algorithm for RCSP . . . 20
4.4.3 Dominance rules . . . 21
4.5 Stabilization techniques . . . 22
4.5.1 Factory example . . . 22
4.5.2 Dealing with degeneracy of dual variables . . . 23
4.5.3 du Merle stabilization . . . 24
4.5.4 Interior Point Stabilization . . . 24
4.6 Constraint Logic Programming . . . 27

5. Implementation . . . 28
5.1 Dataset . . . 28
5.1.1 Finding a sub puzzle . . . 28
5.2 Column Generation . . . 32
5.2.1 The master problem . . . 32
5.3 Branch and Price . . . 34
5.3.1 Bounding . . . 34

5.3.2 Branching strategy . . . 34
5.3.3 Choosing which diagram to fix . . . 36
5.3.4 Tree traversal . . . 37
5.4 RCSP . . . 38
5.4.1 RCSP solving the pricing problem . . . 38
5.4.2 Implementation . . . 38
5.5 Stabilization techniques . . . 45
5.5.1 Motivation . . . 45
5.5.2 Parameters and implementation . . . 45
5.5.3 du Merle problems . . . 46
5.6 RCSP + CLP . . . 47
5.6.1 RCSP graph reduction . . . 47
5.6.2 Modeling constraints . . . 48
5.6.3 Infeasibility and duplicates . . . 50
5.6.4 Forbidden paths . . . 50
5.6.5 Repricing paths . . . 51

6. Results . . . 52
6.1 Branch and Price . . . 52
6.1.1 Branch on a diagram only . . . 52
6.1.2 Branch on number of selected diagrams . . . 52
6.1.3 Choosing which diagram to fix . . . 54
6.1.4 Conclusion . . . 54
6.2 Stabilization techniques . . . 56
6.2.1 Experiments . . . 56
6.2.2 Results - full puzzles . . . 56
6.2.3 Results - Root Node . . . 58
6.2.4 Conclusion . . . 61
6.3 RCSP + CLP . . . 62
6.3.1 Graph reduction . . . 62
6.3.2 Forbidden paths . . . 62
6.3.3 Repricing paths . . . 65
6.3.4 LP convergence . . . 66
6.3.5 Conclusion . . . 66

7. Conclusion and discussion . . . 67
7.1 Column Generation . . . 67
7.2 Branch and Price . . . 67
7.3 Interior Point Stabilization . . . 67
7.4 Combination of techniques . . . 67

8. Future Research . . . 69
8.1 Tree structure . . . 69
8.2 Stabilization techniques . . . 69
8.3 RCSP + CLP . . . 69
8.4 Lagrange relaxation . . . 69

3. INTRODUCTION

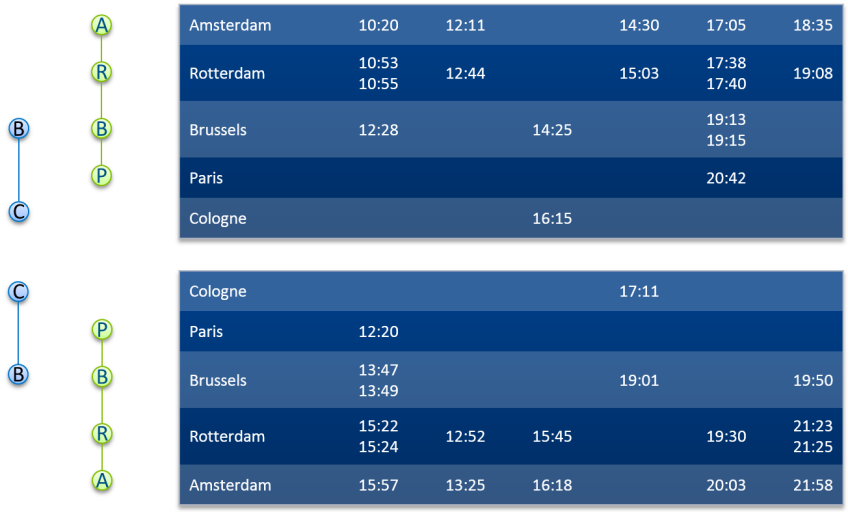
In this chapter we introduce the train crew scheduling problem as well as the structure in which the information in this thesis will be presented.

3.1 Problem overview

According to Jütte (2012) the railway sector keeps growing. To be competitive in this market it is very important to be efficient in your operations. One of the main cost factors for a rail service provider is labor expenses. It is thus essential to make sure available crew members are efficiently employed.
Each train operated by the company needs a driver. More specific, every *train activity* included in the days list of activities needs to be performed by an employee and therefore we call these activities *crew requirements*. Most often such an activity will be driving passengers from location A to location B, but it can also consist of parking the train in a shunting yard or delivering the train for maintenance. A sequence of these crew requirements in a single day is what we call a *shift* or *diagram*. Such a shift includes break moments and pass rides to the locations required to execute the crew requirements. The objective of the train crew scheduling problem is to find a selection of shifts of minimum cost that covers all crew requirements and fulfills the operational constraints (Verhave (2015)).

3.1.1 Example

Let us consider a small simplified example to familiarize ourselves with the problem. There are two steps that generally need to be performed before the crew requirements are determined. These steps are making a timetable and deciding what rolling stock will be used. Determining what rolling stock will be used consists of choosing the type and size of train the rail service provider will be using on the trips determined in the timetable. Let us consider an example timetable, visualized in Figure (3.1).



Amsterdam	10:20	12:11		14:30	17:05	18:35
Rotterdam	10:53 10:55	12:44		15:03	17:38 17:40	19:08
Brussels	12:28		14:25		19:13 19:15	
Paris					20:42	
Cologne			16:15			

Cologne				17:11		
Paris	12:20					
Brussels	13:47 13:49			19:01		19:50
Rotterdam	15:22 15:24	12:52	15:45		19:30	21:23 21:25
Amsterdam	15:57	13:25	16:18		20:03	21:58

Fig. 3.1: An example of a small timetable

This timetable tells us we ride our trains along two tracks. One from Amsterdam to Paris through Rotterdam and Brussels and another from Brussels to Cologne. We imagine ourselves tasked to solve the crew scheduling problem for this set of trains. A more intuitive representation of the crew requirements, hidden in the timetable, is givin in Figure (3.2).

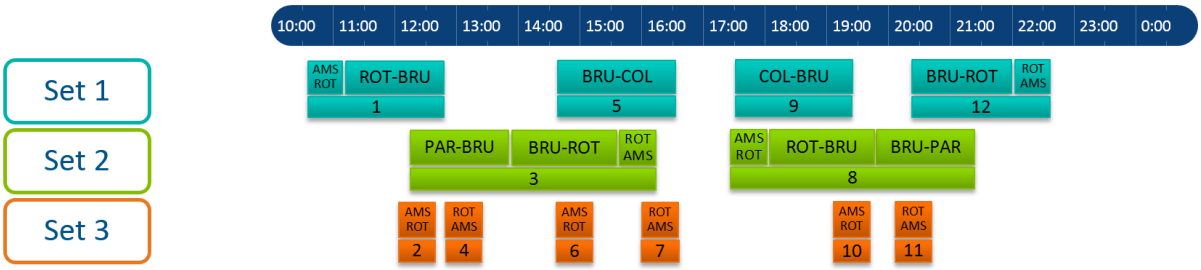


Fig. 3.2: Crew requirements visualized in a Gantt chart

We call this representation of tasks over time a Gantt chart and every block from location A to location B is a trip from the timetable, representing a crew requirement. Our objective is to find the smallest set of shifts consisting of these crew requirements in sequence such that all crew requirements are covered and we adhere to all labor rules. The labor rules we consider for this example are a minimum shift duration of 6 hours and a maximum shift duration of 10 hours. Additionally we require the start location and end location of a shift to be the same. This makes sure our driver starting at his home location will not end up a huge distance away from home after executing his shift. This is merely a subset of the labor rules that are considered in this thesis, which are described in Subsection (3.1.2).
We can create a plan by simply starting with a crew requirement, then taking the next departing train at the location we have arrived. This creates a planning solution as can be found in Figure (3.3).

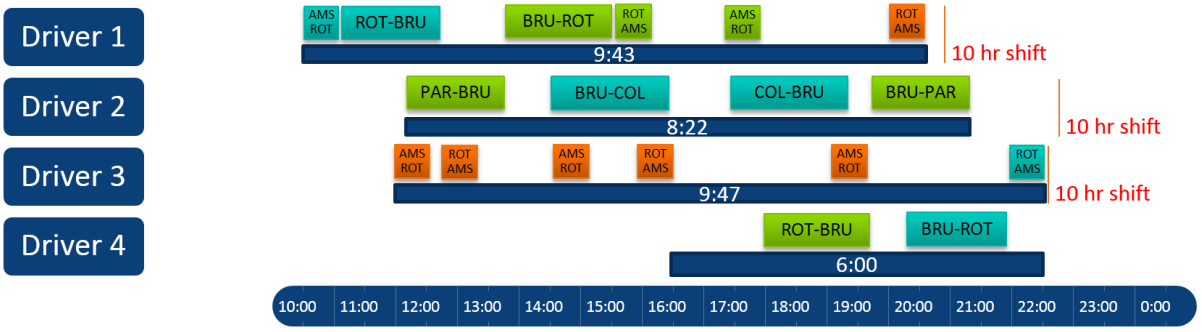


Fig. 3.3: A complete planning solution to the example problem

We can check the labor rules for every diagram and confirm that all shifts are within the shift duration bounds and start and end at the same location. Also every crew requirement is represented in the plan, so we can conclude the plan is valid. Thus we arrive at the next question. Can we solve the problem with three diagrams? For this small puzzle one could solve it by puzzling for a while, but generally we can only answer such questions with the help of algorithmic support. This thesis will describe the algorithms used to find optimal solutions like the one in Figure (3.4).

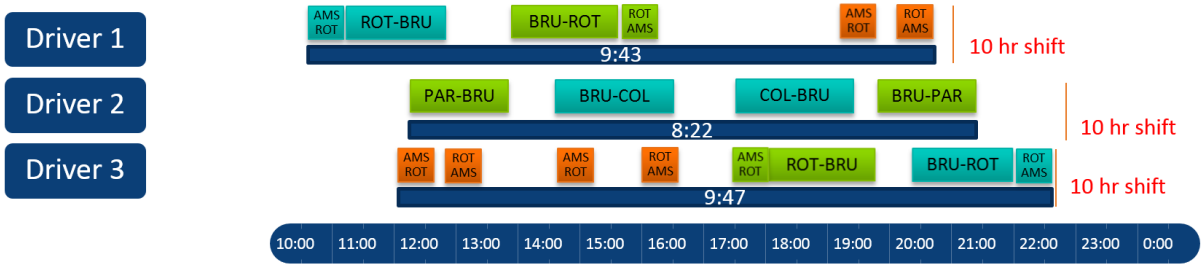


Fig. 3.4: The optimal solution to the example problem

3.1.2 Problem complexity

The problem has already been proven $\mathcal{NP}$-hard by Fischetti et al. (1989). This was done by solving the $\mathcal{NP}$-hard `BINPACKING` problem by solving the crew scheduling problem.
The `BINPACKING` problem requires you to pack a finite set of items $U = \{u_1, \ldots, u_n\}$ of size $s_j \in \mathbb{N}$ for each $u_j \in U$ into the minimum number of disjoint subsets $U_1, U_2, \ldots$, such that $\sum_{u_j \in U_i} s_j \leq B$, with B the integer bin capacity.
Now consider sequential crew requirements $R = \{cr_1, \ldots, cr_n\}$, each cr_j taking $s_j \in \mathbb{N}$ minutes. Now find the minimum set of shifts, with maximum shift length B minutes, covering all crew requirements. This solves the `BINPACKING` problem defined earlier.

Labor rules

We proved that adding one labor rule for the maximum shift length made it possible to prove NP-hardness. The crew scheduling problem solved in this thesis contains even more of these labor rules, some of which were also in the example problem.

- Minimum and maximum shift length
- Starting and ending at the same location
- Planning breaks during the shift
- Minimum and maximum duration between breaks
- Maximum duration of driving during a shift
- Maximum duration of continuous driving during a shift

Size blow-up

Due to its $\mathcal{NP}$-hard nature we find that the problem becomes exponentially harder whenever we try to plan more crew requirements, consider more labor rules or factor in more complex dependencies. The example problem consists of only 18 crew requirements, where real-world puzzles generally have somewhere from a few hundred to a few thousand crew requirements. In this thesis we do not consider the additional factor of route knowledge required by the drivers or specific skills to drive train types. Yet even without this extra complexity, solving the crew scheduling puzzle is no easy task and generally needs more than 50 drivers to execute a complete plan. To find out more about the specific puzzle that is solved in this thesis, you can read Section (5.1).

3.2 Thesis outline

This section describes the structure of the thesis. First we study the available theory about solving the crew scheduling problem. Then we show how we applied the theory to a real-world problem after which we present the results of our research. Finally we discuss topics for future research and present our conclusions.

Theory

First we describe the existing solution methods for solving the crew scheduling problem in Section (4.1). Then we dive into the general theory behind *Column Generation*, *Resource Constrained Shortest Path* (RCSP), *Branch and Price*, *Stabilization techniques* and *Constraint Logic Programming* in Sections (4.2), (4.4), (4.3), (4.5) and (4.6) respectively.

Implementation

In Section (5.1) we present the dataset of the real-world problem solved in this thesis. It also describes how we determined a smaller puzzle within that dataset. Then we describe how we implemented the theory of the earlier described subjects specifically for this problem set in the other sections of Chapter (5).

Experimental results

We show the experimental results for the practical application of the theory. First are the results we found when executing the combination of Column Generation and Branch and Price in Section (6.1), then the results for speeding up the process with Interior Point Stabilization in Section (6.2). Finally we discuss what we discovered when using the combination of RCSP and Constraint Logic Programming (CLP) in Section (6.3).

4. THEORY

4.1 Literature Review

Throughout the years many people have attempted to solve the crew diagramming puzzle. It is a relevant puzzle for any railway operator, as improvements of mere percentages can save millions of operational costs as described by Abbink (2008). This explains the continuous interest from not only scholars, but also research institutions and railway operators themselves. Some of the more important ones are the *Computer Aided Scheduling of Public Transport* (CASPT) (Snijders and Saldanha, 2015) and *Practice and Theory of Automated Timetabling* (PATAT). (Lau and Gunawan, 2012) Improvements to the solution methods are heavily sought after, as the practical application of the theory is common and impactful. These improvements are guided by the increase in computational power and the continuous improvement of mathematical program solvers. For these reasons there is a broad range of possible algorithms that can be used to solve the crew diagramming puzzle and this chapter attempts to give an overview.
First, we discuss the closely related problems of airline crew scheduling and vehicle routing problems, which share many properties and solution methods. Secondly, we touch upon heuristics and meta-heuristics, such as tabu search and genetic algorithms. Then, we dive into set covering, which is one of the most promising avenues of research. We discuss some additional steps that can be taken that are not solution methods on their own, but help in making the problem more tractable. Finally, we conclude the overview and present our choice of the subject of research in this thesis. Content of this literature review comes from cited articles, while it is also inspired by literature reviews of other articles like: (Jütte, 2012) (Jütte et al., 2011)(Verhave, 2015) (Nielsen, 2011) and (Laplagne, 2008). A recent survey on the different aspects of passenger railway optimization is provided by Caprara et al. (2007).

4.1.1 Airline crew scheduling and vehicle routing

Most of the techniques used to solve the crew diagramming puzzle in rail find their origin in the crew diagramming puzzle for airlines, as mentioned by Jütte (2012). These puzzles often contain less complexity, mostly due to the fact that there are not as many possibilities for travel combinations. When one flies from Amsterdam to New York, there is nothing a pilot can do in between. It is also clear that the pilot continues his shift in the region of New York. Flying from Sydney to Beijing is hardly plausible when one has just arrived in New York. Due to their lower complexity these problems were tractably solvable earlier than the rail crew diagramming problem. Problem instances in the railway industry often have shorter trips than the airline industry and thus more room for combining trips. In addition to that the size of the puzzles is often larger too. In contrast to a thousand trips in the airline industry, the railway industry typically has a few thousand trips to schedule.
Another problem that is often mentioned alongside crew diagramming is the vehicle routing problem, usually with time windows (VRPTW). The goal is to let trucks drive efficient routes in such a way that orders can be picked up and delivered in their respective time windows. This sequencing of picking up and delivering packages, with time window constraints, is quite similar to the sequencing of crew requirements, with labor rules. And thus also techniques to solve both problems are quite similar. Some lessons learned as described in Desrochers et al. (1992) are valuable for crew diagramming as well.

4.1.2 Heuristics and meta-heuristics

According to Laplagne (2008), no mathematical approach presented so far has been able to solve large-size, real-life crew scheduling problem instances without incorporating heuristics on at least one of the components of the system. In this thesis we also show heuristic measures implemented in certain steps. The use of heuristics stems from the history of the problem, where methods were heavily based on rules used by human schedulers. This was before the technology surrounding mathematical programming was capable of solving reasonably sized problems.
Next to heuristics also meta-heuristics can be relevant for solving this problem. Meta-heuristics are techniques that are not problem specific and can on their own improve the speed or quality of the solutions. One can use a wide range of these techniques, base a solution method on them or interweave other techniques with the concepts. In Section (4.3.2) we use the concept of tabu search to strengthen the original approach. Below are some examples of meta-heuristics successfully used to tackle the crew diagramming puzzle.

Tabu search

Tabu search is a meta-heuristic search method introduced by Glover (1989), employing local search methods used for mathematical optimization. Local searches take a potential solution to a problem and check its immediate neighbors in the hope of finding an improved solution. The downside of local search is that one can get stuck in suboptimal regions. Tabu search attempts to remedy this by also accepting worse neighboring solutions and discouraging the algorithm of picking previously-visited solutions. This technique was used by Cavique et al. (1999) to solve the crew diagramming puzzle. Their solution uses strategic oscillation techniques, by relaxing a number of problem constraints, to achieve an increase in the number of possibilities one can reach with a move.

Genetic algorithms

Genetic algorithms are governed by the idea of natural selection introduced by Charles Darwin. The concept can be used for problem solving by taking a large set of solutions and then using mutation and crossover techniques to improve the candidate solution. Survival of the fittest is executed by deleting the weakest elements of the solution, while the best elements are used more to create the next generation (offspring) of the solution. More on this can be found in Beasley and Chu (1996). The technique is also used by Souai and Teghem (2009) and makes use of the *roulette wheel* approach. This means the more suitable attributes the offspring has, the higher chance it has of being reproduced. Souai and Teghem (2009) also use an *elitist* technique, where the best parent and the best child get paired to create the next offspring. The created offspring sometimes violates constraints, thus facilitating the use of *legality repair* and *feasibility repair* heuristics.

4.1.3 Set covering

A common approach to solve the crew diagramming puzzle is to formulate a set covering mixed integer program (MIP), as described by Jütte and Thonemann (2012). The set covering tries to minimize the cost of the total plan, while covering all crew requirements, in order to end up with a minimal cost feasible plan. A total plan consists of a selection of columns in the MIP. Let R denote a set of crew requirements, D a set of feasible crew diagrams covering crew requirements. Let a_{rd} denote whether a diagram d covers a requirement r by being 1 and otherwise 0. Let $c_d > 0$ be the cost of diagram d. Decision variables are x_d, where $x_d = 1$ if the diagram d is part

of the solution schedule. The basic set covering formulation then becomes:

$$\text{minimize} \sum_{d \in D} c_d x_d \tag{4.1}$$

$$\text{subject to} \sum_{d \in D} a_{rd} x_d \geq 1 \quad \forall r \in R \tag{4.2}$$

$$x_d \in \{0, 1\} \quad \forall d \in D \tag{4.3}$$

The set covering problem is known to be $\mathcal{NP}-$hard as claimed by Karp (2010). The common problem for real-world crew diagramming puzzles is that the set of feasible diagrams D is intractable, such that generating all feasible diagrams is not an option. Other techniques are needed to solve that problem, such as column generation.

Column generation

Column generation seems to be the most used technique for solving the crew diagramming puzzle. It is successfully used in, among others, Desrosiers and Lübbecke (2005), Huisman (2007) and Verhave (2015). One of the most complete analyses on the subject can be found in Barnhart et al. (1998). The technique alternates between generating a small set of feasible diagrams and solving the set covering linear program on the increasing set of crew diagrams. The generation phase attempts to identify diagrams with a negative reduced cost, hence having the potential of improving the current solution. This generation phase is often called the *subproblem*. The subproblem needs certain information about the constraints of the set covering to determine these improving diagrams. This information can be obtained in multiple ways. For example with SIMPLEX the dual solution is obtained automatically, or with Lagrange relaxation the constraint information can be approximated. Due to the fact that only improving diagrams are generated, the intractable amount of diagrams can be avoided. A more detailed explanation of the technique can be found in Section 4.2.

Lagrange relaxation

The concept of Lagrangean relaxation is to relax some of the more difficult constraints that appear when using the set covering formulation. The violation of these constraints is then penalized in the objective function. This method can be used to find a lower bound on the optimal solution of the set covering. This method is often used in literature, for example by Fisher (1981), Barnhart et al. (1998) and Caprara et al. (1999), the latter even winning a competition by an Italian railway company for consistently finding near-optimal solutions in reasonable time. It is also used for related problems like timetabling and routing in Hassannayebi et al. (2016), Zhou and Teng (2016) and Juttner et al. (2001).

Column generation in combination with lagrange relaxation

Many articles report successful combination of the two main techniques of column generation and lagrange relaxation, for example Caprara et al. (1997), Huisman et al. (2005), Abbink (2008) and (Abbink et al., 2011). The penalty of the constraints is approximated with a subgradient technique. The technique starts with a set of feasible lagrange multipliers and step-wise moves into the direction of steepest descent regarding the objective function. These approximations are then used to determine the improving diagrams that are created during the generation phase of the column generation scheme. The best features of this approach are the speed and high quality of the approximation relative to this speed.

Solving the subproblem

Once information on the constraints of the set covering is determined, improving diagrams need to be constructed by solving the *subproblem*. Solving the *subproblem* can be done in multiple ways. One of the more straightforward approaches is to construct a graph G modeling all constraint information, as described by Jütte (2012). If G contains only non-negative edge costs, we can run *Dijkstra's algorithm* (Dijkstra, 1959) to find the shortest path in the graph. This path will denote a diagram with least reduced cost and thus solves the *subproblem*. In real-world applications we often deal with an extension of the shortest path problem, where we incorporate more constraints in the graph structure. The problem is then called the resource-constrained shortest path problem (RCSP), of which an extensive survey exists by Irnich and Desaulniers (2005). This survey mentions different algorithms to solve the RCSP with. The most common of which is based on *Dynamic Programming* and is a label pushing/pulling algorithm. Another option is to once again use lagrange relaxation to solve the problem. A linear program is defined with flow conservation constraints, such that the answer becomes a path. A Lagrangean relaxation is then obtained by relaxing resource consumption constraints. Yet another solution is to use a constraint programming technique, which relies on a model which is defined by a set of variables, each with an initial domain, and a set of constraints. A *domain reduction algorithm* is used in combination with a *propagation algorithm* to propagate the domain changes among the constraints. The merit of this approach is that many hard to capture constraints of the RCSP can be modeled more easily. For more information about RCSP read Section (4.4). For more about CLP read Section (4.6).

4.1.4 Additional techniques

As mentioned earlier, the railway crew diagramming puzzle is larger than the airline puzzle. Even between different railway operators the size of the puzzle differs quite substantially. In the datasets of Quintiq, even a factor 10 can be observed. Most, if not all, of the above described solution methods do not scale well with size. The following techniques are used to reduce the problem size.

Decomposition

A logical step to reduce problem size is to decompose the problem, preferably in parts that are independent of the other parts. One of the usual ways to do this for passenger railway puzzles is to divide the week plan into day plans, as described by Jütte (2012). Jütte herself is not able to split the week, since she is scheduling freight railway, cargo transport as opposed to passengers, which also operates during the night and thus the week can not be cut into independent parts. She, however, has discovered a way to decompose the puzzle in different regions, each of which can be optimized locally, while also implementing a mechanism such that the whole puzzle is optimized. These regions are smaller and thus have a smaller individual puzzle size. The regions have a certain amount of overlap and communicate information about these overlaps between each other. This connection ensures the optimization of the original puzzle.

Parallelization

Something that comes hand in hand with decomposition is the concept of parallelization. When algorithms scale exponentially in size, we win a lot of time by decomposing in independent puzzles and solving them sequentially. But we can win an extra factor by solving the puzzles in parallel. Jütte (2012) use parallelization to solve the regions described above. Also Alefragis et al. (2000) uses parallelization successfully and mentions the fact that it works well for systems of workstations instead of just using multiple cores of a processor.

4.1.5 Conclusion literature review

One of the clear hot topics of crew diagramming seems to be Lagrange relaxation. Yet in this thesis we chose not to use this particular technique. It seems that the strong qualities of Lagrange relaxation are the ability to quickly approximate dual variables for the column generation scheme and to optimize over complicated constraints. Both of these points did not seem to be relevant for the implementation by Quintiq, which used the CPLEX solver to obtain high quality dual variables in relatively negligible time, and most of the complicated constraints were modeled into the RCSP algorithm.
Instead we chose to analyze the method that was also prevalent in the literature and already successfully in use by Quintiq: the column generation scheme with the RCSP algorithm to solve the pricing puzzle. Since both Quintiq's experience and the literature point into this direction, we were quite certain this was state-of-the-art with respect to solving crew diagramming puzzles. Since there was also a constraint logic method for crew diagram generation in Quintiq and the literature mentioned it as a possible way to solve the pricing problem in the column generation scheme, we chose to further investigate that.

4.2 Column generation

Column generation is a technique to solve Linear Programs (LPs) that have a large number of variables. Instead of solving the intractable LP, we instead start with a small or empty set of variables and iteratively add new variables to the problem. These variables are chosen in such a way that every additional variable should improve the LP goal function. The strong aspect of this technique is that it can solve the LP to optimality.
We will follow the excellent introduction from Jütte (2012). Column generation is based on Dantzig-Wolfe decomposition (Dantzig and Wolfe, 1960), a technique that transforms an LP into an equivalent LP that has fewer constraints, but ranges over more variables.

4.2.1 Dantzig-Wolfe decomposition

Original linear program:

$$\begin{aligned} \min \quad & \boldsymbol{c}^T \boldsymbol{x} && (4.4)\\ \text{s.t.} & \\ A\boldsymbol{x} &\geq \boldsymbol{b} && (4.5)\\ B\boldsymbol{x} &\geq \boldsymbol{d} && (4.6)\\ \boldsymbol{x} &\geq 0 && (4.7) \end{aligned}$$

Let P denote the convex polyhedron of all nonnegative real tuples in n dimensions that satisfy the second subset of the above constraints, i.e.,

$$P := \{\boldsymbol{x} \in \mathbb{R}^n | B\boldsymbol{x} \geq \boldsymbol{d}, \boldsymbol{x} \geq 0\}. \tag{4.8}$$

If P is bounded, we know by polyhedral theory that each element $x \in P$ can be written as a convex combination of extreme points of P (Minkowski, 1896),

$$\boldsymbol{x} = \sum_{k=1}^{q} \lambda_k \boldsymbol{x}^k \tag{4.9}$$

$$\sum_{k=1}^{q} \lambda_k = 1 \tag{4.10}$$

$$\boldsymbol{\lambda} \geq 0 \tag{4.11}$$

where $\boldsymbol{x}^k$ $(k = 1, \ldots, q)$ are the extreme points of P and λ_k $(k = 1, \ldots, q)$ are nonnegative scalars. If we now substitute (4.9) in (4.4) and (4.5) and apply a linear transformation using

$$f_k := \boldsymbol{c}^T \boldsymbol{x}^k \quad \forall k = 1, \ldots, q \tag{4.12}$$

$$\boldsymbol{p}^k := A\boldsymbol{x}^k \quad \forall k = 1, \ldots, q \tag{4.13}$$

We can reformulate the original LP as follows (Dantzig-Wolfe master problem):

$$\begin{aligned} \min \quad & \sum_{k=1}^{q} f_k \lambda_k && (4.14)\\ \text{s.t.} & \\ \sum_{k=1}^{q} \boldsymbol{p}^k \lambda_k &\geq \boldsymbol{b} && (4.15)\\ \sum_{k=1}^{q} \lambda_k &= 1 && (4.16)\\ \boldsymbol{\lambda} &\geq 0 && (4.17) \end{aligned}$$

This formulation contains fewer constraints, but has a blowup of variables from dimension n to extreme points q. How do we solve a linear programs with a large number of variables? The answer is column generation.

4.2.2 The pricing problem

We consider the Dantzig-Wolfe master problem (4.14-4.17). For solving the LP, we will use the simplex method (Dantzig, 1963), which is based on the reduced costs of the LP variables. Let $\boldsymbol{\pi}$ be the dual values associated to the coupling constraints (4.15) and let π_0 be the dual value associated to the convexity constraint (4.16). Then, the reduced cost $\bar{f}_k$ of variable λ_k is calculated as

$$\begin{aligned} \bar{f}_k &= f_k - \boldsymbol{\pi}^T \boldsymbol{p}^k - \pi_0 && (4.18)\\ &= \boldsymbol{c}^T \boldsymbol{x}^k - \boldsymbol{\pi}^T A \boldsymbol{x}^k - \pi_0 && (4.19)\\ &= (\boldsymbol{c} - A^T \boldsymbol{\pi})^T \boldsymbol{x}^k - \pi_0 && (4.20) \end{aligned}$$

where the first reformulation is achieved by reapplying the linear transformations (4.12) and (4.13). In each iteration of the simplex method, we search for the variable with least reduced cost:

$$\begin{aligned} \bar{f}_s &= \min\{(\boldsymbol{c} - A^T \boldsymbol{\pi})^T \boldsymbol{x}^k - \pi_0 | k = 1, \ldots, q\} && (4.21)\\ &= \min\{(\boldsymbol{c} - A^T \boldsymbol{\pi})^T \boldsymbol{x}^k | k = 1, \ldots, q\} - \pi_0 && (4.22) \end{aligned}$$

If $\bar{f}_s$ is nonnegative, the LP is solved optimally. If $\bar{f}_s < 0$, the current objective function value of the LP can be improved by adding variable $x^s = \arg\min\{(\boldsymbol{c} - A^T \boldsymbol{\pi})^T \boldsymbol{x}^k | k = 1, \ldots, q\}$ to the basis. Since the number q of extreme points of the polygon P is typically very large, calculating the reduced cost for each variable does not make sense. However, we know from linear programming theory that, if the constraint set of a linear program is bounded, the optimal solution always occurs at an extreme point of this set. As a consequence, (4.22) is equivalent to the following linear program (Dantzig-Wolfe subproblem or pricing problem):

$$\begin{aligned} \min \quad & (\boldsymbol{c} - A^T \boldsymbol{\pi})^T \boldsymbol{x} - \pi_0 && (4.23)\\ \text{s.t.} \quad & && \\ B\boldsymbol{x} \quad & \geq \boldsymbol{d} && (4.24)\\ \boldsymbol{x} \quad & \geq 0 && (4.25) \end{aligned}$$

By enlarging the solution space of (4.22) to the set of all variables satisfying (4.24) and (4.25), we have in fact simplified the problem: instead of explicitly calculating the reduced costs for all variables, we can now enumerate the variables implicitly by solving the linear program (4.23 - 4.25) called the Dantzig-Wolfe subproblem. Solving the pricing problem many times, and thus extracting the positively contributing variables $\boldsymbol{x}^s$, is called column generation.

4.3 Branch and Price

The goal of Branch and Price is to find the optimal solution to the MIP formulation of the crew scheduling problem. From the Dantzig-Wolfe theory (Dantzig and Wolfe, 1960) we have learned that we are able to generate the optimal solution of the linear program, but now we have to close the gap between the fractional solution and the integer solution. To close the gap between the LP and the MIP we will restrict the LP more and more, until the optimal solution of the LP will also be integral. We are looking for an optimal solution, which means we are not allowed to ignore part of the MIP solution space unless we are absolutely certain there is no better solution to be found in that area. The technique of Branch and Price will guarantee to find the optimal solution and consists of three important steps:

- **Branching:** Restricting the LP in such a way that it will become more like a MIP, without ignoring feasible MIP solution space.
- **Bounding:** Ignoring MIP solution space, when we know there is not a better solution to be found there.
- **Pricing:** Generating additional diagrams such that we can solve the LP of the new node to optimality.

We will discuss the steps in more detail below, as described by Jütte (2012). This is a concise description following the argumentation of Wolsey (1998). The argumentation is based on a minimization problem, but the concepts could be easily transfered to a maximization problem.

4.3.1 Branching

Whenever we have solved the LP to optimality we know there are no variables that will further improve the LP goal value. By solving the LP we have solved a slightly different problem than the MIP. We can, however, add extra constraints to the LP to make its feasible search space looks more like the feasible search space of the MIP. To preserve all MIP solution space we often also have to consider the puzzle with the negation of the constraint we added. We call this procedure of adding constraints *branching*, as we split up our problem to consider multiple slightly different problems and in fact create a solution tree, as visualized in Figure (4.1). The branching decision, which determines the different constraints we add to create the new slightly different problems, can take many different shapes. The key points of a good branching decision are its ability to ignore a large part of the fractional search space, while not ignoring integer search space, and its ability to negatively impact the LP goal value. The effect of the last ability will be seen in the bounding step.

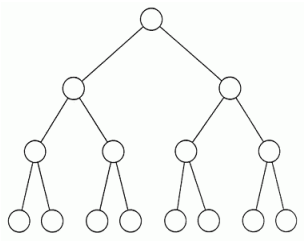


Fig. 4.1: A tree structure branching can create.

Examples

An example of a common branching step is fixing a selected variable in the solution, which is often fractionally selected in case of an LP. This creates two puzzles, where the first puzzle will completely select the variable, while the second puzzle will not select the variable at all.
Another example of a branching works on the integer level instead of the binary level. When the sum of all decision variables Σ is fractional in the form of $X + \frac{Y}{Z}$, we can decide to branch and create two puzzles. The first of which will have the constraint $\Sigma \geq X + 1$ and the second $\Sigma \leq X$. These branching decisions showcase how the freedom of an LP gets restricted in such a way that it will become more similar to a MIP, while not losing integer solution space.

4.3.2 Bounding

The second step in the Branch and Price method is the bounding step. Bounding means that we decide we do not continue looking at solutions further in the tree. There are three different reasons to bound the tree at a certain node.

- **Pruning by optimality:** Whenever the goal value of our MIP solution at a node is equal to the optimal LP goal value at a node, we know there is no room for improvement and can conclude that the MIP solution is optimal for that node. We will not gain anything by restricting the LP further towards integrality, since we have an integral solution to the problem with the same goal value already from the MIP.
- **Pruning by bound:** As we get deeper into the tree and keep restricting the LP further, the LP goal value will keep increasing, since we are solving a minimization problem. The LP will always be a lower bound for all MIPs that we will do from that point on in the tree. That is because every child node will be a problem with less freedom and will thus have a higher LP goal value than what we have found in the parent. The LP is a lower bound for the MIP for the same reason that the MIP has less freedom than the LP to solve the same problem. So when we have found an integer solution somewhere else in the Branch and Price tree that has a lower goal value than the optimal LP in the current node, we can stop. There is no possibility for us to find an integral solution in this part of the tree that is better than the one we already had.
- **Pruning by infeasibility:** When we find that the LP has no solution possible in the current node, we know there is nothing to gain further down the tree, where we would only further limit the LP. Feasible solutions do not exist in this branch of the tree.

Since every branching decision brings us closer to an integral solution, we will eventually reach a point where we will be able to prune. More detailed information on the pruning rules can be found in Wolsey (1998).

4.3.3 Pricing

The Branch and Price technique is executed to cross the gap between the LP master problem and the MIP master problem. By executing the branching step we create a child node with a more constrained linear program. This means that we are solving a different problem and that the LP solution from the parent is not guaranteed to be optimal for the child problem.
We can use column generation to solve the LP in the child node optimally, generating additional columns that improve the LP, but also the MIP. Executing column generation during the Branch and Price algorithm is what we call *Pricing*, which will eventually make us cross the gap between the LP and the MIP. Due to the constrained nature of the LP we are solving optimally, we are generating diagrams that improve the more integral version of the master problem. This means that the diagrams generated are not only improving the LP of the child node, but also the MIP master problem. So while constraining the LP will increase the goal value compared to the solution in the parent, generating new columns will decrease the goal value of the MIP.

4.3.4 Tree traversal

Another part of solving the Branch and Price tree efficiently is choosing the right strategy to traverse the tree. This strategy will determine which node will be solved next, after the LP of the current node has been solved to optimality. There is mainly one goal here. We want to maximize the amount of pruning we can do in the tree. The biggest influence the tree traversal strategy can have is by finding a good integral solution as fast as possible, so that the *prune by bound* rule can be applied earlier or more often.

4.4 RCSP

When using the column generation scheme an important part is generating the columns. This generation part is called the *pricing problem* and in this thesis we attempt to solve it by using the *Resource Constrained Shortest Path algorithm* (RCSP). The goal of the algorithm is to find a sequence of crew requirements, including pass rides, breaks and other scheduled elements to fulfill labor rules and other constraints. We evaluate the quality of the found sequence by means of the dual values generated by the LP solver. The algorithm can minimize this objective, while using resources to model the various labor rules and constraints. Thus it is extremely suitable for solving the pricing problem for the crew diagramming puzzle.
We follow the introduction from Jütte (2012). That introduction is based on Desrosiers et al. (1995) and Cormen et al. (2001). We consider a directed acyclic graph $G = (V, E)$ with vertices V and edges $E \subset V \times V$ and let $c_{ij} \in \mathbb{R}$ denote the cost of a directed edge $(i, j) \in E$. Let $(v_1, \ldots, v_p)$ be a path in G, i.e., a sequence of vertices such that each two consecutive vertices v_i, v_{i+1} are connected by an edge. The cost of a path is defined as the sum of the costs of its constituent edges. For an origin vertex $o \in V$ and a destination vertex $d \in V$ the (*single-pair*) *shortest path problem* (SPP) consists of finding, if existent, a path from o to d in G with minimum cost. If G is a directed graph and all edges have non-negative costs, the single-pair shortest path problem can be solved with *Dijkstra's algorithm* (Dijkstra, 1959). The algorithm successively constructs the shortest paths from the origin vertex o to all other vertices. The information on vertices with already known shortest paths is used for the shortest-path calculations of the next vertices.
In real-world applications, we often deal with an extension of the shortest path problem, the *Resource Constrained Shortest Path problem* (or *shortest path problem with resource constraints*), which we denote with RCSP. See Irnich and Desaulniers (2005) for a comprehensive survey on the subject. In addition to the above formulation we are given *resources* $r \in R$ and resource consumptions $t_{ij}^r \in \mathbb{R}_+$ for each edge (i, j) and each resource r. For a vertex i and a resource r, the accumulated resource consumptions on a path from the origin o to vertex i are required to lie in the interval $[a_i^r, b_i^r]$, which we call the *resource window* of vertex i for resource r. The RCSP is then defined as follows: find a feasible path from origin o to destination d at minimum cost, such that the resource limits a_i^r and b_i^r are met for each resource r and intermediate vertex i of the path. Or in other words: such that the path satisfies all resource windows along the path.

4.4.1 Mathematical formulation

Let x_{ij} denote the flow on an edge $(i,j) \in E$ and let T_i^r denote the accumulated consumption of resource r for a path from origin o to vertex i. The mathematical formulation of the RCSP is

$$\min \quad \sum_{(i,j)\in E} c_{ij}x_{ij} \tag{4.26}$$

$$s.t. \tag{4.27}$$

$$\sum_{j\in V} x_{oj} - \sum_{j\in V} x_{jo} = 1 \tag{4.28}$$

$$\sum_{j\in V} x_{dj} - \sum_{j\in V} x_{jd} = -1 \tag{4.29}$$

$$\sum_{j\in V} x_{ij} - \sum_{j\in V} x_{ji} = 0 \quad \forall i \in V\backslash\{o,d\} \tag{4.30}$$

$$x_{ij} \geq 0 \quad \forall (i,j) \in E \tag{4.31}$$

$$x_{ij}(T_i^r + t_{ij}^r - T_j^r) \leq 0 \quad \forall (i,j) \in E, r \in R \tag{4.32}$$

$$T_i^r \geq a_i^r \quad \forall i \in V, r \in R \tag{4.33}$$

$$T_i^r \leq b_i^r \quad \forall i \in V, r \in R \tag{4.34}$$

4.4.2 Dynamic programming algorithm for RCSP

If we deal with an acyclic graph, the RCSP can be solved by the algorithm of Desrosiers et al. (1995), which uses *dynamic programming* and a *label pulling process*, which can be found in Algorithm (1).

For each path from the origin o to a vertex i, we define a label $(\mathbf{T_i}, C_i)$, where $\mathbf{T_i} = (T_i^1, \ldots, T_i^{|R|})$ represents the consumption of the resources on the path and C_i is the cost value of the path which is calculated from $\mathbf{T_i}$. Let $(\mathbf{T_i^I}, C_i^I)$ and $(\mathbf{T_i^{II}}, C_i^{II})$ be two labels that are associated with two different paths from o to i. The first label is said to *dominate* the second one if its cost value and resource consumptions are no larger than those of the second label, i.e. $C_i^I \leq C_i^{II}$ and $(T_i^r)^I \leq (T_i^r)^{II}$ for all resources r. We call a label on a vertex (and the associated path) *efficient* if it is not dominated by any other label on the same vertex.

Domination, as defined above, works only in case of resources, which can often be freely increased by waiting, like time. When resources are considered that are both increasing and decreasing or resources that can not freely grow to the minimum of a window, one may wish to update constraint (4.32) to $x_{ij}(T_i^r + t_{ij} - T_j^r) = 0$ in general. The idea of domination then needs to be changed to the following: Let $(\mathbf{T_i^I}, C_i^I)$ and $(\mathbf{T_i}^{II}, C_i^{II})$ be two labels that are associated with two different paths from o to i. The first label is said to dominate the second label when all feasible paths from the second label are also feasible paths from the first label in addition to $C_i^I \leq C_i^{II}$ in case of our minimization problem. More on how this is achieved in Section 4.4.3.

Algorithm 1 Dynamic programming algorithm for RCSP - Label pushing

Input: graph $G = (V, E)$, resources $r \in R$, edge costs c_{ij}, resource consumption on edges t_{ij}^r, bounds (a_i^r, b_i^r) on resource consumption at paths from origin to vertices, stop after feasible paths k.

$C_o := 0$; $T_o^r := a_o^r \quad \forall r \in R$;
$L_o := \{(T_o^1, \ldots, T_o^{|R|}, C_o)\}$ ▷ Set of labels at origin vector
$U := V$; ▷ Set of unprocessed vertices
while $U \neq \{\}$ **and** $|L_d| < k$ **do**
    choose $i \in U$ such that all predecessors have been processed;
    for all successors j of i **do**
        for all labels in L_i **do**
            $feasible :=$ true;
            $C_j := C_i + c_{ij}$;
            for all resources r **do**
                $T_j^r := \max\{a_j^r, T_i^r + t_{ij}^r\}$; ▷ depending on resource r
                $T_j^r := T_i^r + t_{ij}^r$ ▷ depending on resource r
                if $T_j^r > b_j^r$ **then**
                    $feasible \leftarrow$ false;
                    break;
                end if
            end for
            if $feasible$ **then**
                if $(T_j^1, \ldots, T_j^{|R|}, C_j)$ efficient w.r.t. L_j **then**
                    $L_j \leftarrow L_j \cup \{(T_j^1, \ldots, T_j^{|R|}, C_j)\}$;
                    remove dominated labels in L_j;
                end if
            end if
        end for
    end for
    $U \leftarrow U \backslash \{i\}$;
end while
return L_d;

4.4.3 Dominance rules

Dominance rules are used during the RCSP algorithm to cut away paths that are no longer relevant w.r.t. the minimization objective of feasible paths. To solve the crew scheduling problem we have defined suitable heuristic dominance rules and strict dominance rules. The heuristic rules do not guarantee the optimal path will be found, but the strict rules do.

4.5 Stabilization techniques

The column generation scheme determines the dual variables to evaluate whether new columns are improving the LP solution. These dual variables are generally determined by the LP solver. However there are situations where the quality of the dual variables determined by the LP solver is inadequate. In this section we introduce three techniques to improve the quality of the dual variables, but first we discuss an example to showcase the interesting behavior of the dual values.

4.5.1 Factory example

We consider a factory that can produce two products x and y, but has only limited capacity on their sole machine A. Product x is by far more valuable than product y, selling for 5 units instead of 1. The two products take the same amount of time to be produced on machine A. We have only a certain amount of supplies and can make only D products of x. We will choose different values for D and observe the behavior of the dual variables.
First we model the problem in a linear program:

$$\begin{aligned} \max \quad & 5x + y && && (4.35)\\ \text{s.t.} \quad & && && \\ x + y \quad & \leq 8 && \text{(machine } A\text{)} && (4.36)\\ x \quad & \leq D && \text{(max amount } x\text{)} && (4.37)\\ x, y \quad & \geq 0 && && (4.38) \end{aligned}$$

We determine the dual variables by the following reasoning: The dual variables should capture the improvement of the goal value if we were to increase the right hand side of a constraint a small amount. We also determine the dual variables with CPLEX. Below we discuss several scenarios. We will refer to the dual variable of a constraint by $\pi_{\text{constraint}}$.

We consider $D = 7$. We have a maximum goal value of $7 \cdot 5 + 1 = 36$. Both constraint (4.36) and (4.37) are tight.

(4.36) If we increase the right hand side of (4.36) to 9, our maximum goal value increases to $7 \cdot 4 + 2 \cdot 1 = 37$ and $\pi_{4.36} = 1$.

(4.37) If we increase D by one, our maximum goal value increases to $8 \cdot 5 = 40$. So $\pi_{4.37} = 4$.

(4.38) If we increase the lower bound of y, nothing changes, so $\pi_{4.38} = 0$.

If we now check the dual solution we find: $4 \cdot 7 + 1 \cdot 8 = 36$. This is as we expect the same as the maximum goal value of the primal, due to strong duality. CPLEX also confirms the dual variables we have reasoned above.

We consider $D = 9$. Our maximum goal value is $8 \cdot 5 = 40$. Constraint (4.36) and (4.38) are tight in this case.

(4.36) If we increase the right hand side of (4.36) to 9, our maximum goal value increases to $9 \cdot 5 = 45$ and $\pi_{4.36} = 5$.

(4.37) If we increase D by one we get no movement, so $\pi_{4.37} = 0$.

(4.38) If we increase the lower bound of y, our maximum goal value changes to $7 \cdot 5 + 1 \cdot 1 = 36$. This means that $\pi_{4.38} = -4$.

If we now check the dual solution we find: $5 \cdot 8 + 0 \cdot 9 - 4 \cdot 0 = 40$. This is as we expect the same as the maximum goal value of the primal, due to strong duality. We can see that the dual variable $\pi_{4.38}$ does not influence the dual solution, but both our reasoning and CPLEX assign it the value -4.

We consider $D = 8$. We have again a maximum goal value of $8 \cdot 5 = 40$. The interesting thing is that all constraints are tight in this case.

(4.36) If we increase the right hand side of (4.36) to 9, our maximum goal value increases to $8 \cdot 5 + 1 = 41$, so $\pi_{4.36} = 1$ by our reasoning.

(4.37) If we increase D by one we get no movement, so $\pi_{4.37} = 0$.

(4.38) If we increase the lower bound of y, our maximum goal value changes to $7 \cdot 5 + 1 \cdot 1 = 36$. This means that $\pi_{4.38} = -4$.

If we now check the dual solution we find: $1 \cdot 8 + 0 \cdot 9 - 4 \cdot 0 = 8 \neq 40$, so we do not find a valid dual solution. CPLEX tells us that the dual variable $\pi_{4.36} = 5$, which would indeed result in a valid dual solution. However even if we increase the right hand side of constraint (4.36) we do not improve 5 times that amount in the goal value, since we are immediately bounded by constraint (4.37). Here are the reasons why:

- One reason is that we have many different dual solutions: $(5 - a) \cdot 8 + a \cdot 8 + b \cdot 0 = 40$, with $0 \leq a \leq 5$ and b arbitrary.
- Another reason is that an increase of the right hand side of the constraint for machine A would force us to produce a y to stay optimal, so we need to take into account the dual variable for y as well. We then find the true interpretation of the goal value changes when we take the dual 5 from machine A, but add the -4 from the lower bound of y and end up with 1, our earlier deduced value for the dual variable of A.

We can soften the effect of the first reason, which will be discussed below. The second reason is the downside of the column generation scheme. The dual variables give a reduced cost to the new columns, but there is no guarantee this reduced cost will be the true effect on the LP goal value. There might be some hidden dependencies, other columns that need to be chosen or other constraints that get violated, such that the reduced cost can not be fully realized.

D	**Reasoning**				**CPLEX**			
	(4.36)	(4.37)	(4.38)	dual	(4.36)	(4.37)	(4.38)	dual
7	1	4	0	36	1	4	0	36
9	5	0	-4	40	5	0	-4	40
8	1	0	-4	8	5	0	-4	40

Tab. 4.1: Summary of the factory example

4.5.2 Dealing with degeneracy of dual variables

We have seen in our example the effect of the existence of many different dual solutions and others encounter the same difficulty with the resulting erratic behavior of the dual variables. (Rousseau et al., 2007) (Zhouchun Huang, 2014). There are so called stabilization techniques, which deal with the degeneracy of the dual variables. We will investigate two techniques that are used to stabilize the dual variables. The 'du Merle' method and the Interior Point Stabilization (IPS) method.
Both methods will attempt to find higher quality dual variables by not selecting the dual variables from the extreme point of the LP solution. These can be too high or too small. Instead both use different reasoning to come up with better dual variables. Both methods also scale only a linear factor of the original construction and computation time of the original LP and are often negligible compared to the time to solve the pricing subproblem. We follow the introduction from Rousseau et al. (2007).

4.5.3 du Merle stabilization

To prevent dual variables from taking on extreme values, a valid strategy is to limit the deltas of the dual variables between iterations. There are two main techniques to control this behavior. A first technique attempts to limit the deltas by confining the dual variables to a predetermined box. The master problem is altered in such a way that the new dual variables can only obtain values contained in the box. A second technique is to incentivize the master problem to be reserved in changing the dual variables by a large amount. We do this by giving a linear penalty to the absolute size of the delta. The master problem will then try to limit the delta. The first technique was proposed by Marsten et al. (1975) and the second technique by Kim et al. (1995). Du Merle et al. (1999) have proposed a technique to stabilize the dual variables with a method that combines the above two concepts. The dual variables λ are again put in a box, but this time they are not limited to the box. They are able to leave the box, but will have to pay a linear penalty for the distance they move away from the box, while they can move inside the box freely. The box is defined by (d^-, d^+) and these are given as parameters to the method. The λ variables can leave the box (when $\omega^- > 0$ or $\omega^+ > 0$), but then the objective is penalized by $(\varepsilon^-, \varepsilon^+)$, which are also parameters to the problem.

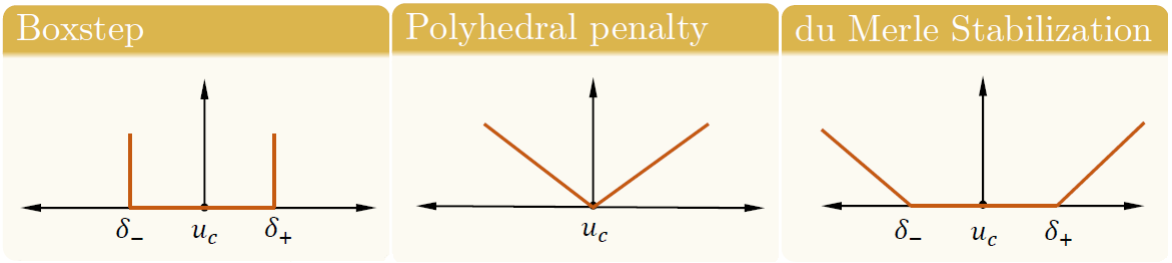


Fig. 4.2: left: first technique, *middle:* second technique, *right:* du Merle

The master problem is modified into (M') to implement these ideas and can be written as:

$$\min \sum_{d \in D} c_d x_d + \sum_{r \in R} -d_r^- y_r^- + d_r^+ y_r^+ \tag{4.39}$$

$$\text{s.t.} \quad \sum_{d \in D} a_{rd} x_d - y_r^- + y_r^+ \geq 1 \qquad \forall\, r \in R \tag{4.40}$$

$$y_r^- \leq \varepsilon^- \qquad \forall\, r \in R \tag{4.41}$$

$$y_r^+ \leq \varepsilon^+ \qquad \forall\, r \in R \tag{4.42}$$

$$x, y^-, y^+ \geq 0 \tag{4.43}$$

With associated dual:

$$\max \sum_{r \in R} \lambda_r - \varepsilon_r^- \omega_r^- - \varepsilon_r^+ \omega_r^+ \tag{4.44}$$

$$\text{s.t.} \quad \sum_{r \in R} \lambda_r a_{rd} \leq c_d \qquad \forall\, d \in D \tag{4.45}$$

$$\lambda_r + \omega_r^- \geq d_r^- \qquad \forall\, r \in R \tag{4.46}$$

$$\lambda_r - \omega_r^+ \leq d_i^+ \qquad \forall\, r \in R \tag{4.47}$$

$$\lambda, \omega^-, \omega^+ \geq 0 \tag{4.48}$$

4.5.4 Interior Point Stabilization

The idea behind Interior Point Stabilization (IPS) is to generate a dual solution that is an interior point of the optimal dual space rather than an extreme point. The proposed way to

achieve this goal is to generate several extreme points of the optimal dual polyhedron and to generate an interior point corresponding to a convex combination of all these extreme points.

Theory

When we formulate the relaxed set covering problem (M) in the following way:

$$\min \sum_{d \in D} c_d x_d \tag{4.49}$$

$$\text{s.t.} \quad \sum_{d \in D} a_{rd} x_d \geq 1 \qquad \forall\, r \in R \tag{4.50}$$

$$x \geq 0 \tag{4.51}$$

We find the dual, denoted by (D), in the form of:

$$\max \sum_{r \in R} \lambda_r \tag{4.52}$$

$$\text{s.t.} \quad \sum_{r \in R} \lambda_r a_{rd} \leq c_d \qquad \forall\, d \in D \tag{4.53}$$

$$\lambda \geq 0 \tag{4.54}$$

When we solve (M) we find a set D^* with selected diagrams, so $x_d > 0$. We call S the set containing all crew requirements that were not tightly covered. Using complementary slackness conditions, the optimal dual polyhedron $\mathcal{D}$ containing all optimal values of λ is defined by:

$$\sum_{r \in R} \lambda_r a_{rd} \leq c_d \qquad \forall\, d \in D \backslash D^* \tag{4.55}$$

$$\sum_{r \in R} \lambda_r a_{rd} = c_d \qquad \forall\, d \in D^* \tag{4.56}$$

$$\lambda_r = 0 \qquad \forall\, r \in S \tag{4.57}$$

$$\lambda_r \geq 0 \qquad \forall\, r \in R \backslash S \tag{4.58}$$

To obtain a point in this polyhedron one can add a random objective function $u\lambda$, with u a vector chosen uniformly over $[0, 1]$, and solve the following LP, denoted (D^u).

$$\max \sum_{r \in R} u_r \lambda_r \tag{4.59}$$

$$\text{s.t.} \quad \sum_{r \in R} \lambda_r a_{rd} \leq c_d \qquad \forall\, d \in D \backslash D^* \tag{4.60}$$

$$\sum_{r \in R} \lambda_r a_{rd} = c_d \qquad \forall\, d \in D^* \tag{4.61}$$

$$\lambda_r = 0 \qquad \forall\, r \in S \tag{4.62}$$

$$\lambda_r \geq 0 \qquad \forall\, r \in R \backslash S \tag{4.63}$$

Solving this LP with a simplex method gives us an extreme point of $\mathcal{D}$. Different instances of $\mathcal{D}^u$ can be generated by defining multiple objective functions (u vectors) and thus several extreme points of $\mathcal{D}$ can be obtained. Since $\mathcal{D}$ is a convex set, any convex combination of its extreme points will lie within it. We expect the convex combination of several extreme points to results in an interior point of $\mathcal{D}$. We take the average of all obtained extreme points, which will give us this interior point and we expect that solution of $\mathcal{D}$ to better represent the marginal costs in the master problem.

This procedure for obtaining an interior point is simple, but requires constructing the dual problem. This step can be troublesome when extra constraints are present in the master problem. To avoid it, we can consider the dual of $\mathcal{D}^u$, denoted by $\mathcal{P}^u$.

$$\min \sum_{d \in D} c_d x_d \tag{4.64}$$

$$\text{s.t.} \quad \sum_{d \in D} a_{rd} x_d \geq u_r \qquad \forall\, r \in R \backslash S \tag{4.65}$$

$$\sum_{d \in D} a_{rd} x_d \geq -\infty \qquad \forall\, r \in S \tag{4.66}$$

$$x_d \geq 0 \qquad \forall\, d \in D \backslash D^* \tag{4.67}$$

$$x_d \quad \textit{free} \qquad \forall d \in D^* \tag{4.68}$$

Interior point stabilization can thus be executed by making small changes to the original set cover problem. Extreme points of $\mathcal{D}$ can be obtained by taking the dual variables of $\mathcal{P}^u$. We can generate different instances of $\mathcal{P}^u$ by changing the u vector. The interior point is again found by averaging the extreme points from $\mathcal{D}$.
It is also possible to change only a subset of the $R \backslash S$ constraints to a random number, decreasing time needed to alter the master problem and the time needed for the LP solver to reach optimality.

LP solver setting

Another way to control the behavior of the dual variables was a setting for the LP solver, called the Simplex dual solve method, which reduces the dual variable size significantly. The setting makes the LP solver solve the dual problem, instead of automatically choosing how to solve the problem. We believe this makes the algorithm create a solution for the dual, instead of the primal. When the algorithm has its focus on the dual formulation, the dual variables do not grow out of control and obtain reasonable values.

4.6 Constraint Logic Programming

In this section we will describe the constraint logic programming (CLP) used to model constraints. CLP relies on a model which is defined by a set of variables, each with an initial domain, and a set of constraints, as described by Irnich and Desaulniers (2005). A CLP approach is composed of a *search mechanism* to explore the solution space, a *domain reduction algorithm* for each constraint that tries to remove inconsistent values from the domains of the variables involved in that constraint, and a *propagation algorithm* that propagates these domain changes among the constraints. It can model a wide spectrum of constraints, both algebraic and non-algebraic.
The Quintiq CLP solver is linked to the open source solver Gecode, which stands for Generic Constraint Development environment. For more information see Gecode (2017). Quintiq has developed their own schedule layer on top of Gecode which is capable of modeling resources and tasks on those resources. These resources originally modeled various machines in a factory on which different actions needed to be planned in sequence, with the particular goal to solve the flow shop scheduling problem. This layer gives the possibility to model dates, time and durations directly on the tasks instead of being limited to boolean and integer types.

5. IMPLEMENTATION

5.1 Dataset

In this thesis we solve a real-world problem from a high-speed rail company, stemming from 2012. The puzzle contains 340 crew requirements that need to be planned and therefore we will call it the 340 puzzle in this thesis. We decided to research this puzzle since it contains the complexity that comes with a real-world puzzle, but its size still allows for good solutions to be found within an hour. We know this, because Quintiq has been testing their algorithms on this problem for quite a while and thus has a very developed benchmark. There are many different crew types for which the dataset contains puzzles, but we focused mainly on the driver puzzle.

5.1.1 Finding a sub puzzle

Even though relatively good solutions can be found within an hour, this is quite a long time for quick testing of implementations and makes developing different algorithms problematic. We also wanted to be able to compute an optimal solution for a puzzle. For these reasons we set out to find a smaller puzzle. We decided that we wanted a puzzle with enough complexity such that a good solution would exist, containing a certain number of diagrams, but also a better solution with fewer diagrams. It needed to be small enough such that it would be computationally feasible to calculate the optimum, while also big enough to not be trivial. This led us to the benchmark we had for the puzzle, containing a good solution with 49 diagrams and a better solution with 44 diagrams. This indicated that from the good solution we were able to win 5 diagrams. We wanted to try and make the puzzle smaller by taking a subpuzzle, while still retaining this interesting distinction between a good and better solution. This would give us exactly the puzzle we needed for testing.

Mini MIP

First we formally describe the problem and then show the definition for the Mixed Integer Program (MIP) that solved the problem for us.
Lets call the set with 49 crew diagrams and corresponding crew requirements set A and the set with 44 crew diagrams and corresponding crew requirements set B. Now we want to select a puzzle (a set of crew requirements) A' defined by a selection of n crew diagrams in A, such that $A' \subseteq A$. Then we make a selection of m crew diagrams in B defining a sub puzzle B', such that $B' \subseteq B$, the difference between both solutions $|A' - B'|$ is minimal and $n \geq m$, but preferably $n > m$.
The solution from the problem above should result in two subpuzzle solutions consisting of crew diagrams that satisfy the properties that one is a good solution and another a better solution, while also being small enough to make testing algorithms take a reasonable amount of time. However when solving the specific problem above we found that there are almost no solutions that can satisfy the perfect balance of covering exactly the same set of crew requirements. In addition to that we also found that some crew requirements are covered by more than one crew diagram in the same set. We introduce the concept of unbalance in the coverage and also an auxiliary variable deciding coverage of a crew requirement so we are able to ignore double coverage of the same diagram. We define the following MIP to solve this question:

List of variables

A	set of crew diagrams forming the 49 solution
B	set of crew diagrams forming the 44 solution
a	a crew diagram from A
b	a crew diagram from B
x_a, x_b	decision variable for selecting a crew diagram
i	number of crew diagrams we want to select from A.
R	set of crew requirements
r	a crew requirement
c_{ar}	boolean variable: indicates whether a covers r or not
c_{br}	boolean variable: indicates whether b covers r or not
u_r	unbalance due to selected crew diagram a from A
o_r	unbalance due to selected crew diagram b from B
j	amount of unbalance we want

Goal function

$$\min \sum_{b \in B} x_b \tag{5.1}$$

$$\tag{5.2}$$

Constraints

$$\sum_{a \in A} c_{ar} x_a - \sum_{b \in B} c_{br} x_b - u_r + o_r = 0 \qquad \forall\, r \in R \tag{5.3}$$

$$\sum_{a \in A} x_a = i \tag{5.4}$$

$$\sum_{r \in R} u_r + o_r = j \tag{5.5}$$

$$x_a, x_b \in \{0, 1\} \tag{5.6}$$

We do not have clearly defined what solution to the above problem would instantiate the best subpuzzle, so we would like to discover many and choose the one we like the most. For this purpose we use the parameters i and j and loop over them $i \in \{1, \ldots, 49\}$ and $j \in \{0, \ldots, 10\}$. The puzzle did not have any perfect solutions and the results were also influenced by some small crew diagrams containing 2 or 3 crew requirements. So every 2 or 3 extra unbalance will create one more difference in the A,B diagram selection.

The 33 puzzle

We settled on a puzzle that is quite small (33 crew requirements from the 340 total), but has a nice solution A' in the A set and a slightly better solution B' in the B set. Figures 5.2 and 5.3 show the two solutions to the sub puzzle. The solution does not differ in the amount of diagrams, but does differ in the goal value. The solution from B has packed the crew requirements slightly more compact, thus having smaller total work duration and smaller cost. The pass rides used to make the diagrams valid are also smaller, which reduces work duration and cost.
To make sense of the solutions the following legend is helpful:

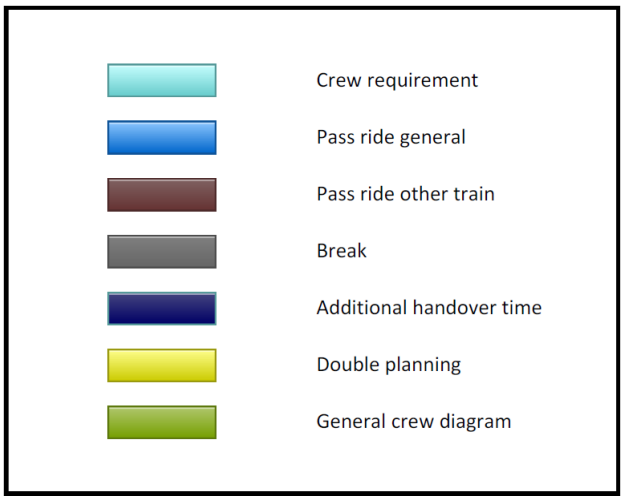


Fig. 5.1: Legend for the crew diagramming solutions

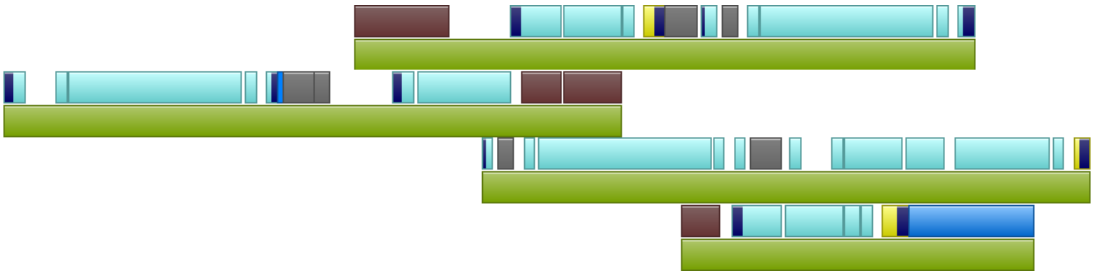


Fig. 5.2: The solution A' to the sub puzzle from original set A

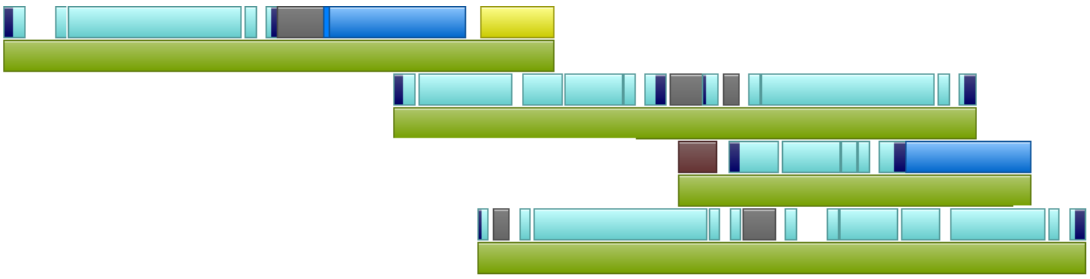


Fig. 5.3: The solution B' to the sub puzzle from original set B

We can see that they share the bottom two diagrams, but the upper two diagrams differ in length. This makes the solution from B a higher quality solution. However quite quickly we discovered the optimal solution with the technique described in this thesis. The concepts of the algorithm were working and we could move on to more complex and bigger puzzles. The optimal solution is visualized in Figure 5.4 and we can see that it is almost identical to the solution from B, except for the top diagram which is shorter in the optimal solution.

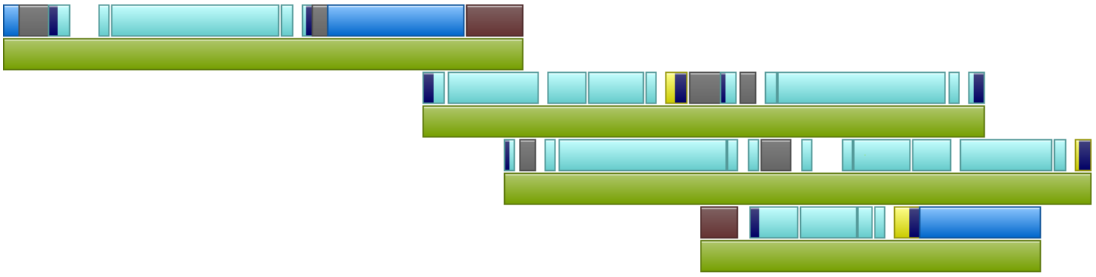


Fig. 5.4: The optimal solution to the 33 puzzle

The 33 puzzle was still useful to run small tests on, to make sure the algorithm was still running correctly. Eventually we also figured out a way to forbid the optimal solution of 4 diagrams. We made sure the algorithm would not generate any of those 4 diagrams and thus we artificially created a new problem. This problem had more complexity and was interesting enough to continue the research with. All further mention of the 33 puzzle refers to the same set of crew requirements defining the puzzle, but with the 4 diagrams in the above solution forbidden. In the end we were also able to find the optimal solution to this new 33 puzzle, which is visualized in figure 5.5. In this thesis we proof that it is now impossible to form a solution with only 4 diagrams. Crew requirements get shuffled around in such a way that we still have some diagrams containing a lot of planned crew requirements.

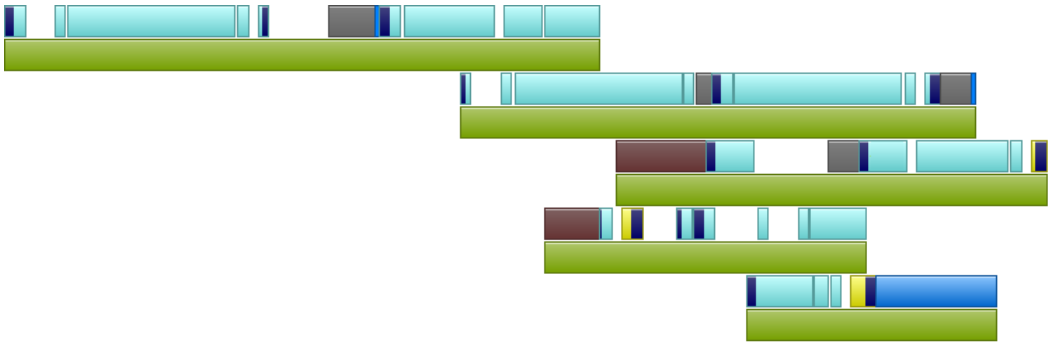


Fig. 5.5: The optimal solution to the 33 puzzle with forbidden diagrams

5.2 Column Generation

To solve the crew scheduling problem for our dataset we model a set covering Mixed Integer Program (MIP), where our variables are crew diagrams. We want to make a selection of crew diagrams in such a way that all crew requirements are covered. When we relax the integer decision variables of selecting a diagram, we find an LP. This LP is in fact a Dantzig-Wolfe decomposition of an LP, where all non-linear constraints, the labor rules, have disappeared into the pricing problem. On this LP we can execute the column generation scheme. Since we use the Dantzig-Wolfe decomposition concept to put the non-linear constraints in the pricing problem, we cannot use linear programming to solve the pricing problem. Instead we solve the pricing problem with the Resource-Constrained Shortest Path (RCSP) algorithm, which will be discussed in detail in Section (4.4) and (5.4). First we will outline the MIP/LP that models the set covering.

5.2.1 The master problem

List of variables

D	set of crew diagrams
d	a crew diagram
R	set of crew requirements
r	a crew requirement
$\bar{c_d}$	cost of a crew diagram
x_d	decision variable for selecting a crew diagram
a_{rd}	whether crew requirement r is covered by diagram d
u_r	under-covering of a crew requirement
t_r	duration in minutes to execute crew requirement
$\bar{u}$	under-covering cost

Goal function

$$\min \sum_{d \in D} \bar{c_d} \cdot x_d \tag{5.7}$$

$$+ \sum_{r \in R} \bar{u} \cdot t_r \cdot u_r \tag{5.8}$$

Constraints

$$\Big(\sum_{d \in D} a_{rd} x_d \Big) + u_r \geq 1 \qquad \forall r \in R \tag{5.9}$$

$$x_d \geq 0 \tag{5.10}$$

Explanation

The above is the formal definition of the mathematical program solving the master problem. We have two parts in the goal function and two constraints. The first thing to note is that we are trying to minimize our cost. Our crew diagrams have a fixed cost in addition to being more expensive the longer the shift takes. We also define a penalty on the under-coverage of a crew requirement. This way we give the program the incentive to try and cover all crew requirements. Exactly what we want in the eventual plan. The first part of the goal function:

$$\min \sum_{d \in D} \bar{c_d} \cdot x_d$$

We minimize the total cost of the plan, where every crew diagram has to pay its cost relative to the amount it is included in the plan. A fully selected diagram pays the full price, while a non-selected diagram contributes no cost to the plan. The crew diagram cost is defined as:

$$\bar{c_d} = \text{crew activity cost} + \text{working performance cost} + \text{fixed cost per diagram}$$

Crew activity cost is the execution cost of the crew requirements. Working performance cost is the duration of the shift multiplied by the hourly wage. We include a fixed cost per diagram, which should take into account the expenses of hiring an additional worker to execute the plan. Next up we have the penalty for under-coverage, which is determined in the following formula:

$$+ \sum_{r \in R} \bar{u} \cdot t_r \cdot u_r$$

Often $\bar{u}$ is chosen relatively high compared to $\bar{c_d}$, since we want to give the optimizer a strong incentive to cover all crew requirements. To distinguish between severity of not covering different crew requirements, we scale with the duration of the crew requirement. Thus under-covering a long crew requirement will be more costly than under-covering a short crew requirement. The variable u_r will be the amount we have not covered the crew requirement r, forced to its value by the constraints.
The constraints make sure that we have a feasible plan. We want to make sure that our plan covers all crew requirements or else knows we have to pay a high penalty.

$$\Big(\sum_{d \in D} a_{rd} x_d \Big) + u_r \geq 1 \qquad \forall r \in R$$

For every crew requirement r we count how much we have covered said crew requirement r. We do this by summing all selection variables of crew diagrams that cover the crew requirement. We want this sum to be equal or higher than 1, to indicate we have covered it fully. Whenever this sum is below 1, our auxiliary variable u_r, called a slack variable, will be forced to fill up the gap between the sum and 1. This forces the plan to take a penalty in the goal function, whenever a crew requirement is not fully covered.
In different phases of the process the program will be a normal linear program and otherwise a mixed integer linear program, depending on whether constraint (5.10) is of the form:

Constraint	**Variable type**	**Program type**
$x_d \geq 0$	continuous non-negative variable	linear program
$x_d \in 0, 1$	binary integer variable	mixed integer program

This distinction is needed because linear programs are solved more easily and quicker than the MIPs and have the added bonus of determining dual variables, which are the values of the selection variables in the dual corresponding to the constraints in the primal version of the problem. These will be needed to solve the pricing problem.
Now that we have defined the master problem, we can start solving the linear program variant to optimality by executing the column generation scheme. Next we describe how we implemented the Branch and Price method to solve the MIP of the master problem in Section (5.3).

5.3 Branch and Price

In this section we discuss in detail how we execute the Branch and Price concept described in Section (4.3). Where that section was more general, this section is more problem specific.

5.3.1 Bounding

As can be read in Section (5.2), we have modeled the LP to never be infeasible. Instead we suffer a heavy penalty when not satisfying the covering constraints. Infeasibility will thus be pruned by bound rather than by infeasibility.
Bounding is the reason we are able to make searching for the optimal solution tractable. Selecting the right way to branch is important to trigger one of the bounding steps. When a decision negatively impacts the LP goal value, the chance is higher that we can *prune by bound*. Decisions that greatly shift the LP towards an integral solution will be easier to *prune by optimality*. Since we will not encouter infeasibility in our model, we can observe the following three things after solving the LP to optimality at the current node of the Branch and Price tree:

- **Worse:** The LP goal value is worse than the best known integral solution. The decision that was used to branch towards this node was apparently not the best decision. We can apply the *prune by bound* rule here.
- **Equal:** The LP goal value is equal to the best known or local integral solution. We can use the *prune by optimality* rule here.
- **Better:** The LP goal value is better than the best known integral solution. We have room for improvement and will need to branch to get closer to the integral solution and perhaps generate new diagrams that can improve both the new LP as well as the integral solution. No pruning can be applied here.

5.3.2 Branching strategy

To solve the crew scheduling problem to optimality we need to determine our branching strategy. This determines the way we restrict the LP to become more integral. We went with the choice to fix variables and thus in this case to fix diagrams. Fixing a diagram $\mathbf{d}$ means executing a binary branching decision. We create two puzzles where we either fully choose $\mathbf{d}$, adding a constraint such that its decision variable $x_{\mathbf{d}} = 1$ in one puzzle, or not choose it all, adding a constraint such that $x_{\mathbf{d}} = 0$ in the other puzzle.

Fixing a diagram

There are a few steps needed to correctly branch on fixing a diagram, since we also need to account for the fact that we will need a puzzle where we are not allowed to pick the diagram:

- **Fix the diagram:** We tell the linear program that the decision variable for this diagram should have a lower bound of 1.
- **Do not generate the fixed diagram:** The fixed diagram is completely selected in the solution, which means that its reduced cost in the pricing problem is 0 and thus it will not be generated.
- **Forbid the diagram** We do not give the diagram a decision variable in the linear program, so it can not be chosen.
- **Do not generate the forbidden diagram:** This is the hardest part, since we now forbid a diagram that was originally in the solution of the LP. This means it will have a reduced

cost and the pricing problem will want to generate it. So we need a way to let the pricing problem know that this diagram is off-limits. We can use the RCSP resource system to make this happen.

Lemma 5.3.1 (**Forbidden Path**). *Given a diagram* $\mathbf{d}$ *that covers crew requirements* $R' = (r_1, \ldots, r_n)$*, we can define a new resource* $\mathcal{R}$ *that blocks the RCSP algorithm from generating that diagram* $\mathbf{d}$ *only. We define the resource* R*, with default edge value* $\delta_{edge} = 0$ *and default node window* $[-\infty, +\infty]$*. Additionally* $\forall r \in R$*, we assign the crew requirement edge in the RCSP value* $\delta_{edge} = 1$ *and* $\forall r \in R'$*, we assign the crew requirement edge in the RCSP value* $\delta_{edge} = -1$*. Finally we assign the destination a node window* $[-\infty, n - \frac{1}{2}]$.

Proof. Consider a path p containing exactly the crew requirements $R' = (r_1, \ldots, r_n)$ and a resource $\mathcal{R}$ defined in the above way to block R'. Then the resource accumulated value at the destination for path p will be $\sum_{r \in R'} \delta^r_{edge} = n \cdot 1 = n$, which will not be accepted by the node window $[-\infty, n - \frac{1}{2}]$.
Now consider a path p^* containing exactly the crew requirements $R^* = (r_1, \ldots, r_m)$, for which hold $R^* \neq R'$. This must mean $\exists r^* \in R$, for which holds either:

- $r^* \in R^*$ and $r^* \notin R'$: The resource accumulated value for p^* at the destination is bounded by $n + \delta^{r^*}_{edge} = n - 1$. Thus the path is not blocked by resource $\mathcal{R}$.
- $r^* \notin r^*$ and $R^* \in R'$: The resource accumulated value for p^* at the destination is bounded by $n - \delta^{r^*}_{edge} = n - 1$. Thus the path is not blocked by resource $\mathcal{R}$.

Concluding that the resource $\mathcal{R}$ as defined above blocks the RCSP algorithm from accepting the path p, and thus generating diagram, $\mathbf{d}$ only. □

Note that we use the node window $[-\infty, n - \frac{1}{2}]$ instead of $[-\infty, n - 1]$ to avoid rounding errors that might occur for a path with accumulated resource value $n-1$ at the destination. More on resources can be found in Section (5.4)

Branch on number of selected diagrams

A typical thing for LP set covering solutions is that not only the variables are selected fractionally, also the sum of all selection variables is fractional. For the optimal LP solution for the 340 puzzle, the number of diagrams selected is 41.10185. Of course every integral solution will also have an integral number of diagrams selected. This motivates the following branching strategy. Whenever we reach the optimal LP solution with the total number of diagrams $s = \sum_{cd \in D} x_{cd}$ fractional, we will branch the problem in P_l and P_r such that P_l should have $\sum_{cd \in D} x_{cd} \leq \lfloor s \rfloor$ and P_r should have $\sum_{cd \in D} x_{cd} \geq \lceil s \rceil$. This method is also used in the vehicle routing problem with time windows (VRP(TW)), which also uses column generation and Branch and Price to solve their instances, see Desrochers et al. (1992).
This step potentially cuts away a lot of fractional solution space, containing a lot of good fractional solutions. This means it will probably have a strong increasing effect on the LP goal value and is thus great for *pruning by bound*, but it is inherently also great for *pruning by optimality*, since it is also a big step towards integrality. A hidden effect is that after branching on the number of selected diagrams, the fixing of diagrams becomes more effective as well. Where earlier fixing or forbidding a diagram would result in a small shift around one of the selected diagrams, this is now impossible since the solution needs to conform to the number of selected diagrams bound.

Changes in the pricing problem

In contrast to the branching decision where we fix a diagram, the decision to branch on the number of selected diagrams really adds a constraint to the linear program. That means the constraint will have a dual variable that needs to be incorporated in the pricing problem.
In general to find a correct reduced cost for a diagram, we need to take into account the dual variables from the constraints in which the diagram appears. We handle the cover constraint of a crew requirement r by taking the dual value from the constraint and subtracting it from the original cost of r to determine the reduced cost of the crew requirement. Now if a new diagram would appear in the cover constraint of r, it would need to take into account that reduced cost. This happens in RCSP as described in Section (5.4), since a diagram covering r is a path through RCSP along the edge of r, which has as cost the reduced cost of r.
To correctly incorporate the dual value for the constraint on the number of selected diagrams, we need to consider which diagrams appear in the constraint. The answer to that is all of them. So to find the reduced cost of a diagram we need to always subtract the dual variable of this constraint, similar to how we would always start with the fixed cost of a diagram. Taking care of the dual value in this way is confirmed in Barnhart et al. (1998), where the convexity constraint ranges over all variables and the reduced cost of a variable is determined by always subtracting the corresponding dual.
In case of problem P_l the dual value will be negative, since the constraint is $\sum_{cd \in D} x_{cd} \leq \lfloor s \rfloor$. Increasing the right-hand side of the constraint would give the problem more freedom, thus the goal value of a minimization problem would decrease and therefore the dual value is negative. For problem P_r, this reasoning is the other way around and thus its dual value is positive. In both cases we subtract the dual variable, thus for P_l a diagram starts off with a higher reduced cost than normal and for P_l the diagram starts off with a lower reduced cost than normal. Since the dual solution still has the same goal value as the primal solution, we observe different behavior from the dual variables in both problems. In problem P_r the dual value for the constraint reduces the other dual variables, since the total needs to stay the same. In problem P_l the dual value for the constraint gives the other dual variables room to grow. Due to this fact we observe very large dual variables, that summed are still equal to the primal goal value. This erratic behavior is what led us to the stabilization techniques.

5.3.3 Choosing which diagram to fix

Goal

Whenever we branch to fix a diagram, we have to choose one of the diagrams in the pool. There are many different choices here and every choice can impact the eventual depth and size of the tree. The goal in branch and price is always to prune the tree as much as possible. Since we have only two pruning rules we use in practice, our decision should be such that we increase our chances of using those two rules. To use the *prune by optimality* rule, we should branch towards integrality as fast as possible. To use the *prune by bound* rule, we should either branch such that our solution can not be better than our best know integral solution or that we find a better integral solution.

Strategy

Because of the above motivation we decided to branch on the non-fixed diagrams that have the highest fractional selection value in the optimal LP. This diagram is apparently so good even the LP selects it a lot and is thus probably part of a good integer solution too. Since a good solution is often also one with fewer diagrams, and therefore reaching integrality sooner, we improve our chances on *pruning by optimality*. But we also improve our chances on *pruning by bound*, since the puzzle where we are not allowed to select the diagrams, misses an important diagram and

will probably have a worse optimal solution. The bigger the impact on the LP goal value, the higher chance we can *prune by bound*. At the same time the solution we are going towards by fixing the diagram will probably be quite good, since it was already selected by the LP. This solution can lead to improving our best known integral solution, also giving more opportunity to use the *prune by bound* rule.

5.3.4 Tree traversal

A tree traversal strategy is often best decided in combination with the branch strategy. As described above we decide to branch on highly selected diagrams in the optimal LP solution. We expect that when we follow the fixed diagram path towards integrality, we should find an excellent integral solution. This should provide us the information we need to *prune by bound* on the decisions we expect to strongly negatively effect the LP goal value, because we have forbidden good diagrams in those paths. We can formalize this strategy as a Depth First Search (DFS), where we will always follow the fixing step before following the forbidding step.

5.4 RCSP

5.4.1 RCSP solving the pricing problem

We have seen how the RCSP algorithm works in general in Section (4.4). We will discuss why it is so heavily used for solving the crew diagramming pricing puzzle. Afterwards in Section (5.4.2) we discuss how the constraints are implemented.

Reduced cost

One of the main reasons for using RCSP is the fact that it is designed to optimize over resources, by use of dominance rules. We will see that resources will often determine feasibility without the need to be optimized, but this is not true for the most important resource, modeling the reduced cost of a path. It is important to mention that paths in the RCSP correspond to crew diagrams, which we want to generate. From the column generation scheme we obtain the dual variables, which we will suitably allocate as resource consumptions for the edges in the RCSP graph. This resource will be able to calculate the reduced cost and will be the biggest factor in dominating other paths that have a worse reduced cost. Placing a $[-\infty, 0]$ node window on the target will ensure that we only return paths that have a chance of improving the LP solution.

Graph structure

Another important strength of the algorithm is that it is extremely suitable for modeling the crew diagramming labor rules and constraints. Some constraints can be captured by using the graph structure. When you want to start in location A and end in location A, you make sure that the source only connects to location A and the only way to reach the target is also from location A. Dependencies between crew requirements can also be modeled by connecting or disconnecting crew requirements.

Modeling with resources

Other constraints are suitably modeled by using the resource system. One can measure the duration of a shift by introducing a resource with resource consumption on the edges related to the duration of an edge. An edge modeling the journey from location A to location B by train will have a certain duration in the real world, let us say one hour. We then give that edge the resource consumption of 60 to indicate that shift duration has increased by 60 minutes. Then adding a node window of $[120, 240]$ to the target would model a constraint that shifts may not take less than two hours or more than four.

5.4.2 Implementation

In this section we describe how the RCSP is implemented exactly to model the various constraints and rules. The algorithm itself is run by the C++ Boost library, which is interfaced with the Quintiq software. Defining the RCSP graph requires the following steps:

1. Create the nodes and edges of the graph.
2. Determine edge filters, that activate or deactivate edges.
3. Create resources by defining node windows and resource consumption along edges.
4. Define the algorithm by choosing edge filters and activating resources.

We will focus mostly on steps (1) and (3). All these steps are done in the Quintiq software, which then sends the modeled graph to the C++ Boost library. Boost eventually returns a set of paths corresponding to crew diagrams, which is influenced by the chosen dominance rule.

Input

First we describe the information that is needed to construct the graph where we execute RCSP on.

Info	**Description**
Week day	The day that we are currently optimizing, like Sunday or Tuesday.
Crew type	The crew type that we are scheduling, like the drivers or the machinists.
Crew depots	The stations where the crew type is allowed to start and end their shift.
Crew requirements	The elements that need to be planned eventually.
Shift patterns	The different shift patterns that capture the rules for start time and shift duration allowed for the crew diagrams, including information about break duration and frequency.

Tab. 5.1: Input information for the RCSP algorithm

Graph structure

We describe step (1): the creation of the nodes and edges.

Node name	**Description**
Source	The start node for every path.
Sink	The end node for every path.
Start of depot	The node modeling that we enter a shift at the depot. Created for every depot.
End of depot	The node modeling that we leave a shift at the depot. Created for every depot.
Start of CR	The start of a crew requirement.
End of CR	The end of a crew requirement.
Start of break	The start of a break period
End of break	The end of a break period
Break internal	Determines the duration of a break
Wait	Created before the start of every crew requirement

Tab. 5.2: The nodes created for the RCSP graph

Next we describe the edges that are needed to connect these nodes.

Edge name	From node	To node	Description
Initial	source	start of depot	The edge every path will start with. Can be used by an edge filter to determine at which locations we can start.
Terminal	end of depot	sink	The edge every path will end with. Can be used by an edge filter to determine at which locations we can end.
Crew requirement	start of CR	end of CR	The edge containing the information about the crew requirement.
Wait to CR*	wait	start of CR	For every start of CR node, there will one wait node created. They are connected by this edge.
Waiting*	wait	wait	We want to be able to not execute crew requirements, but idle during the shift.
CR to wait*	end of CR	wait	A very important edge. Every end of CR node is connected to all reachable wait nodes at different locations. These edges can embody a passride between two locations by use of, for example, a train or taxi, but can also embody just waiting at the current station.
Depot to CR*	start of depot	start of CR	Contains sign on duration and can additionally embody a passride.
CR to depot*	end of CR	end of depot	Contains sign off duration and can additionally embody a passride.
CR to CR*	end of CR	start of CR	Contains the duration between two crew requirements.
Depot to break*	start of depot	start of break	This edge will always contain a passride, since it is not allowed to start a shift with a break.
Break to Depot*	end of break	end of depot	This edge will always contain a passride, since it is not allowed to end a shift with a break.
Break to CR*	end of break	start of CR	Can include a passride
CR to break*	end of CR	start of break	Can include a passride
Break to wait*	end of break	wait	
Start to inter	start of break	break internal	
Inter break	break internal	break internal	This edge will determine the break duration.
Inter to end	break internal	end of break	This node will always be at the same time as a break end.

* are considered *non work edges*

Tab. 5.3: The edges created for the RCSP graph

In Figure (5.6) we show how these nodes and edges are combined to form the graph.

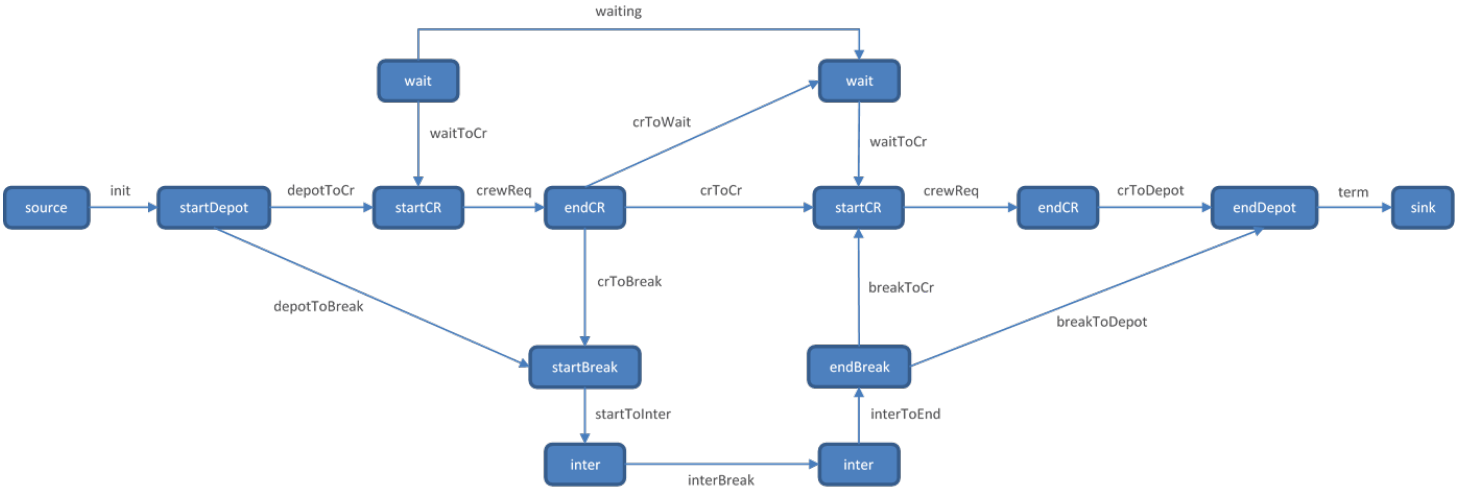


Fig. 5.6: The different nodes and edges used by RCSP

Resources

In this subsection we describe the different labor rules and constraints that are captured by the resource system of RCSP. We describe the rule or constraint and define the node windows and resource consumptions used to model the constraints. For every resource r we first describe the a_i^r for all node types i and then b_i^r for all node types i. Then we describe the different t_{ij}^r for all the different edge types $(i, j) \in E$.
We will often describe something for the set of *non work edges*, which are marked with a * in Table (5.3). When talking about *default*, we mean all edges. When *default* is not mentioned the following defaults are in use: the minimum of a node window is by default $-\infty$ and the maximum by default $+\infty$. The resource consumption is by default 0.0.

Reduced cost

We start with the most important resource, the only one that is being minimized or optimized the reduced cost resource. This will model the improvement the path, and thus the diagram it represents, will cause for the LP set covering we are solving during the column generation scheme. The column generation provides dual variables for all crew requirements and these need to be incorporated in this resource. As a reminder, we are looking for as much negative reduced cost as we can find and this is the reason we are minimizing this resource.

Type	**Name**	**Value**	**Description**
Minimum	All	–	The lower the better
Maximum	Sink	0.0	We only want to find improving diagrams
		Description	
Consumption	Initial	Fixed cost per diagram	
	Non work edges	Pass ride cost + work performance cost	
	CR to CR	Dual variable + work performance cost	

Tab. 5.4: The reduced cost resource

Shift length

A crew diagram has to adhere to certain shift lengths depending on its shift type. There are a few shift types, each with their own length, allowed start time and allowed ending time. The start and ending time is taken care of by edge filters. The shift length by this resource. Every shift type has it's own minimum shift length $s_{\min}$ and maximum shift length $s_{\max}$.

Type	**Name**	**Value**	**Description**
Minimum	Sink	$s_{\min}$	A shift may not take less time than the minimum shift time
	Default	0.0	A negative shift length is impossible
Maximum	Default	$s_{\max}$	A shift may not take longer than the maximum shift length
		Description	
Consumption	All	The duration of the edge	

Tab. 5.5: The shift length resource

Break rule

This break rule defines a duration required before a break, a duration required between two breaks, and also how many breaks are required in a diagram depending on the length of the diagram. As there are many break patterns, each with their own minimum shift duration $s^b_{\min}$ and maximum shift duration $s^b_{\max}$, but also minimum duration before breaks $b_{\min}$ and maximum duration before breaks $b_{\max}$. In some cases a break pattern consists of a primary and secondary break. In this case there is also a minimum duration between breaks $bb_{\min}$, while $b_{\max}$ is constraining for the duration between shift start and 2nd break end.

Type	Name	Value	Description
Minimum	Sink	$s^b_{\min}$	The minimum shift duration needed for this break pattern
	End of Break	$b_{\min}$ + break duration	The minimum amount of time before a break is allowed
	Default	0.0	Negative work amount is impossible
Maximum	Start of Break (1st)	$b_{\max} - bb_{\min}$ - break duration	For two breaks this becomes the maximal duration before the 2nd break
	Start of Break (2nd)	$b_{\max}$	The maximum duration before break
	Default	$s^b_{\max}$	The maximum shift duration allowed for this break pattern
		Description	
Consumption	All	The duration of the edge	

Tab. 5.6: The break rule resource

Driving time

There are labor laws per crew depot stating that a driver cannot drive for more than a certain number of hours within a day. The following resource will make sure that we will not go above the maximum driving duration $d_{\max}$. Since driving is only done when executing a crew requirement, we have a relatively simple set up:

Type	Name	Value	Description
Minimum	–	–	–
Maximum	Default	$d_{\max}$	Every node will make sure we don't go over the maximum
		Description	
Consumption	Crew requirement	The duration of the only edge that models driving	

Tab. 5.7: The driving time resource

Continuous driving time

Among the labor rules for driving time, there is also a rule stating that a driver may not drive more than $cd_{\max}$ minutes continuously during a shift. Driving time is counted only when executing a crew requirement. The continuity is broken during a break or longer continuous non-driving like waiting. This continuity breaking time period is defined as the minimum gap $g_{\min}$.
We will use a special feature called *snap to minimum*, that allows us to have a value below the a^r_i node window value. The algorithm will then increase the value to satisfy the window. We use this by letting the continuous driving time accumulate during the crew requirement edges and then whenever a break occurs we lower the accumulated time to the maximum negative number $-\infty$. *Snap to minimum* will then make sure we start the next continuous period at 0 accumulated time.

Type	Name	Value	Description
Minimum	Inter break	$-\infty$	To allow us to decrease the accumulated time
	Default	0.0	To make *snap to minimum* work and negative accumulated time should not be a thing
Maximum	Default	$cd_{\max}$	Every node will make sure we don't go over the maximum
Consumption	Crew requirement	The duration of the only edge that models driving	
	Start to inter	$-\infty$	Whenever we have a break, we reset the accumulated time
	CR to cr	$-\infty$	Only when the edge duration is greater than $g_{\min}$
	CR to wait	$-\infty$	Only when the edge duration is greater than $g_{\min}$

Tab. 5.8: The continuous driving time resource

Maximum duration between breaks

The break rule we have seen earlier only makes sure that the total shift duration matches the break pattern and the minimum and maximum duration before a break are enforced. In the case of a break pattern with two breaks, we also need to make sure that the maximum duration between breaks is respected. We will do this by resetting the resource to 0 at the end of the first break and accumulating time after that. Then at the second break we have a node window that only accepts a duration below the maximum duration between breaks $d_{\max}$.

Type	Name	Value	Description
Minimum	End break (1st)	0.0	To start with a clean slate with *snap to minimum*
Maximum	Start break (2nd)	$d_{\max}$	The second break will determine whether the duration was short enough
Consumption	Start to inter	$-\infty$	Whenever we have a break, we reset the accumulated time
	Other		Edge duration for all other edges

Tab. 5.9: The maximum duration between breaks resource

Minimum duration between breaks

Like we have the maximum duration between breaks we also have the minimum duration between breaks $d_{\min}$. We can not model this in the same resource as for the maximum, because we use *snap to minimum* there to reset. If we were to have a minimum at the start of the second break, the resource would be able to just grow to the minimum to satisfy it. That is not the desired behavior so we give the minimum duration its own resource. Since we want to use the resetting power of *snap to minimum*, we will model this resource slightly differently by starting from $d_{\min}$ and counting down. So the resource consumption will be negative.

Type	Name	Value	Description
Minimum	Start break (1st)	0.0	To start with a clean slate with *snap to minimum*
Maximum	End break (2nd)	- break duration	When counting down we want all $d_{\min}$ time to be gone, along with the break duration itself, so $d_{\min}$ was indeed reached before the break
Consumption	Start to inter (1st)	$d_{\min}$	Start the counter at the minimum duration between breaks
	Start to inter (2nd)	- break duration	Count down the time by using edge duration
	Other	- edge duration	Count down the time by using edge duration

Tab. 5.10: The minimum duration between breaks resource

Break count

As mentioned before, sometimes we need multiple breaks during a shift. We enforce that with the following resource.

Type	Name	Value	Description
Minimum	Start break (2nd)	1.0	We need to have had the first break before starting the second break
	Sink	1.0 or 0.0	Depending on whether the shift pattern contains the break type
	Default	0.0	
Maximum	Start break (2nd)	1.0	We need to have had the first break before starting the second break
	Sink	1.0 or 0.0	Depending on whether the shift pattern contains the break type
Consumption	Start to inter (1st)	1.0	Count when we have executed the first break

Tab. 5.11: The break count resource

5.5 Stabilization techniques

In this Section we describe the reason we investigated stabilization techniques. Afterwards we describe the parameters and implementation of the Stabilization techniques described in Section (4.5).

5.5.1 Motivation

When executing branch and price runs on the 340 puzzle we discovered that the dual variables behaved different from what we would expect. Whenever we generated diagrams with a certain reduced cost, the next LP would not improve the same amount in the goal value. Theory dictates that this should be the case. (Desrosiers and Lübbecke, 2005). We also encounter uncharacteristic behavior of the constraint from the branch decision on the number of selected diagrams. The dual variable of this constraint would blow up to incredible heights. Where ordinarily the dual values are in the hundreds, the dual values of all tight constraints would go into many millions.
There are several reasons for this erratic behavior. The main reason is that there are multiple optimal dual solutions, each with a different distribution of dual variables on the different crew requirements. When we branch on the total amount of selected crew diagrams, the added constraint has a negative dual variable in the case of $\leq \lfloor s \rfloor$, while the cover constraints have a positive dual variable. They can scale arbitrarily big and still cancel each other out to arrive at an optimal solution. Another reason is that the reduced cost expresses the improvement if the diagram was to be selected fully. This is almost never the case, since there is often a careful balance between the selected diagrams, which are often selected fractionally. The new diagram can thus not be selected fully and the improvement is only partially realized. Moreover, since there can be multiple optimal dual solutions and therefore multiple distributions of dual variables on the different crew requirements, every distribution will have different diagrams that have a negative reduced cost. Sometimes a new diagram is generated only to shift the dual solution to one of the different optimal solutions. Eventually the right dual solution with the right distribution will be found to initialize the pricing problem in such a way that crew diagrams will be generated that improve the goal value, instead of just shifting the dual solution.

5.5.2 Parameters and implementation

The du Merle method has three parameters: the size of the box d, the initial starting value for the linear penalty ε and the growth of the linear penalty $\varepsilon_{\text{growth}}$. Whenever we find diagrams with reduced cost, we increase the linear penalty $\varepsilon_{i+1} = \varepsilon_i \cdot \varepsilon_{\text{growth}}$. When no reduced cost diagrams are found we instantly set $\varepsilon = 0$. When $\varepsilon = 0$ we are solving the original master problem again and can thus guarantee optimality. We tested several different choices for these parameters.

The Interior Point Stabilization method has only one parameter. The amount of extreme points we generate, from which we calculate the average point to find an interior point. When we execute the interior point method we first solve the original master problem. This will give us the set S of non-tightly covered crew requirements. Every time we generate a new extreme point we generate a random value $u_r \in [0, 1]$ uniformly for every crew requirement in $R \backslash S$. All other constraints for crew requirements in S are deactivated. This will create the $\mathcal{P}^u$ problem and corresponding extreme point.

Both these techniques and the normal runs can be run with or without the simplex dual solve setting as it defines the way the LP is solved, while the described techniques modify the LP formulation itself.

5.5.3 du Merle problems

We discovered that the du Merle approach was not behaving as expected. We were unable to discover why this was the case. It might be possible we chose the wrong starting values for the box d. We do not present experimental results for the du Merle technique, as the results are not representative for the technique, but caused by human error in the implementation.

5.6 RCSP + CLP

When solving the problem, we noticed that the RCSP implementation was clearly the bottleneck of the algorithm. We decided to try a new approach for solving the pricing problem. As described in Irnich and Desaulniers (2005), the RCSP problem can be solved with constraint logic programming (CLP). We did not believe that CLP would be able to completely calculate through the problem faster, but perhaps it would be able to successfully take some of the constraints away from RCSP. We chose to let CLP do the break related constraints and the pass rides, while RCSP would take care of all other constraints, still including pass rides to be able to combine crew requirements at different locations. We believed reducing the size of the RCSP graph would speed that part up tremendously, while CLP would be able to quite quickly plan breaks in possible diagrams returned by RCSP. In this section we describe how we implemented this combination, how we solved issues arising along the way. In Section (6.3) we discuss the impact of the combination on solving the puzzle.

5.6.1 RCSP graph reduction

Reducing the size of the RCSP graph, and thus the execution time of the algorithm, was the main goal of using the combination. We let CLP take care of the break related constraints and therefore we can delete all break related nodes and edges from the RCSP graph.

Node name	**Description**
Start of break	The start of a break period
End of break	The end of a break period
Break internal	Determines the duration of a break

Tab. 5.12: The nodes for the RCSP graph related to breaks

Edge name	**From node**	**To node**	**Description**
Depot to break	start of depot	start of break	This edge will always contain a pass ride, since it is not allowed to start a shift with a break.
Break to Depot	end of break	end of depot	This edge will always contain a pass ride, since it is not allowed to end a shift with a break.
Break to CR	end of break	start of CR	Can include a pass ride
CR to break	end of CR	start of break	Can include a pass ride
Break to wait	end of break	wait	
Start to inter	start of break	break internal	
Inter break	break internal	break internal	This edge will determine the break duration.
Inter to end	break internal	end of break	This node will always be at the same time as a break end.

Tab. 5.13: The edges for the RCSP graph related to breaks

A big effect of deleting the break concept from the RCSP graph is that the break related resources are not necessary anymore, since they can not be modeled without the break nodes. The following resources may be removed.

- The break rule resource, taking care of the shift length for every break pattern.
- The maximum duration between breaks resource.
- The minimum duration between breaks resource.

- The break count resource, making sure we have the number of breaks in the break pattern.
- The continuous driving time resource, since we deleted the nodes and edges that would reset this resource.

Another effect of deleting the break concept from RCSP is the fact that there are less algorithms running during one RCSP invocation. Normally we would have an algorithm for every crew depot, shift pattern and break pattern combination. Now with the break pattern not being used, we lose a significant factor of algorithms. This results in an RCSP invocation taking less time, not only because the execution of one algorithm takes less time, but also because there are less algorithms to execute.

5.6.2 Modeling constraints

In this subsection we describe the various elements that will be planned by the CLP procedure.

Input

We will call the CLP procedure with a sequence of crew requirements, that were combined by the reduced RCSP. Since RCSP has given us this path, we know that every two crew requirements in sequence can be reached, either by passride or just because end and start location are the same. We also know that RCSP has evaluated a negative reduced cost, but has not taken into account breaks between crew requirements. We will discuss the impact of this fact in Section (5.6.4). In addition to the crew requirements we provide the start end end position we want for the eventual diagram.

Planning crew requirements

We start by defining the crew diagram, which means we initialize a resource with a setup task and teardown task by default, representing the beginning and end of the diagram respectively. On this diagram we can already start planning the static elements of our puzzle: the crew requirements. We calculate the starting time and ending time for every crew requirement, where we have to take into account possible handover time, and add them to the diagram.

Planning pass rides

The next thing we do is add possible pass rides. When the location differs between two tasks, for example between crew requirements or between the start of the diagram and the first crew requirement, we enforce at least one pass ride to be planned between those tasks. When the locations are the same between crew requirements, no pass ride is needed, but we allow optional pass rides at the beginning and end of the diagram.
Let us consider an example of pass rides at the beginning of the diagram. Suppose we have a set of viable pass rides $\mathcal{P}$ that connect the start location of the diagram with the start location of the first crew requirement on the diagram. Let us define the decision variable on whether we have planned the pass ride or not to be d_p. Then for the optional pass rides and respectively mandatory pass rides we have the statements:

$$\sum_{p \in P} d_p \leq 1 \qquad \sum_{p \in P} d_p = 1$$

We need to bind the end of the setup task, which is the start of the diagram, to the start of the first activity on the diagram, but this depends on the choice of pass ride. Therefore we

define the start of the diagram as s_{cd}, start of the pass rides as s_p and start of the first crew requirement as s_{cr}. We then need the following statements:

$$\big(d_p = 1\big) \Rightarrow \big(s_{cd} = s_p\big) \tag{5.11}$$

$$\big(\sum_{p \in P} d_p = 0\big) \Rightarrow \big(s_{cd} = s_{cr}\big) \tag{5.12}$$

This allows us to define shift length of a diagram as $e_{cd} - s_{cd}$, where e_{cd} is the end time of the diagram and defined in a similar way as above.

Planning breaks

Last but not least we need to plan the breaks on the diagram. As seen above we can have optional pass rides at the beginning and end of the diagram such that the shift length can change depending on the choice of pass rides. The shift length influences which break pattern we are allowed to use for the diagram, since a break pattern has a maximum shift length as a constraint. Shifts taking less than 6 hours are allowed to have no breaks at all, while shifts taking longer than 6 hours need a break. Since breaks are not allowed at the beginning or end of a diagram, choosing a pass ride can provide the new opportunity to plan a break between the pass ride and crew requirement it connects to. CLP can quite quickly navigate these choices and find the optimal configuration of pass rides and breaks for the diagram.
We define the end of the setup task, the begin of the diagram, as s_{end}, the start and end of the first and second break as $b^1_{start}, b^1_{end}, b^2_{start}, b^2_{end}$ and the start of the teardown task, the end of the diagram, as t_{start}. With these definitions, we can model the different constraints working on breaks in the following way:

- breaks cannot be planned at the beginning and end of the diagram. The Quintiq schedule layer provides the capability to define tasks that may not be planned before and after a specific task. We can add the beginning of the diagram modeled by the setup task as forbidden to be planned before a break and similarly for teardown task as forbidden after the break.

- Break rule constraint. We need the diagrams length to be between the minimum shift length $sh_{\min}$ and maximum shift length $sh_{\max}$ specific to the break pattern:

$$t_{start} - s_{end} \geq sh_{\min} \tag{5.13}$$

$$t_{start} - s_{end} \leq sh_{\max} \tag{5.14}$$

- Maximum duration between breaks $b_{\max}$. We can define the following rules:

$$b^1_{start} - s_{end} \leq b_{\max} \tag{5.15}$$

$$b^2_{start} - b^1_{end} \leq b_{\max} \tag{5.16}$$

$$t_{start} - b^2_{end} \leq b_{\max} \tag{5.17}$$

  These rules make sure that the time between the breaks is less than the maximum.

- Minimum duration between breaks $b_{\min}$. Similarly as for the maximum duration:

$$b^1_{start} - s_{end} \geq b_{\min} \tag{5.18}$$

$$b^2_{start} - b^1_{end} \geq b_{\min} \tag{5.19}$$

$$t_{start} - b^2_{end} \geq b_{\min} \tag{5.20}$$

- Break count is modeled by adding decision variables d^i_{break} for every needed break i and forcing them to be true.

5.6.3 Infeasibility and duplicates

Since RCSP gives CLP a path consisting of a sequence of crew requirements without taking into account the break constraints, it can happen that the sequence of crew requirements can not be combined into a feasible crew diagram. Let us call such a path an infeasible path. The existence of infeasible paths is not a problem in itself, but it becomes problematic when RCSP is only returning infeasible paths, which can happen during execution of the column generation scheme. The problem is that these infeasible paths are often infeasible because they are so time efficient that breaks can not be fit inside. This means that these diagrams often have a negative reduced cost, as most efficient diagrams do, and thus RCSP will generate them. But since those paths do not result in new diagrams we do not find any new solutions. This results in no new dual variables, therefore RCSP just keeps on returning the same infeasible paths.
Another problem we encounter is that when we have created a diagram, RCSP might still try to generate the same sequence of crew requirements. This happens when the CLP has to make choices for pass rides and breaks, such that the diagram becomes longer. RCSP does not realize this is necessary and will try to generate the shorter diagram, which has less cost and thus can have negative reduced cost, resulting in a duplicate diagram. To solve both problems of unwanted paths we can reuse a mechanism we have discovered with Lemma (5.3.1): the forbidden paths.

5.6.4 Forbidden paths

The problem and solution we encounter during the RCSP plus CLP combination is quite similar to executing a branching step during the Branch and Price algorithm, where we are not allowed to use a diagram anymore. In both cases the RCSP wants to return a path that we do not want it to return. In the Branch and Price case, we fixed this by adding an additional resource that makes sure that RCSP does not generate the path as described by Lemma (5.3.1).
Whenever CLP says a diagram is infeasible, we instantiate a forbidden path consisting of the crew requirements in the path found by RCSP. Also whenever we find a duplicate diagram we create a forbidden path with those crew requirements. In this way we make sure we always give RCSP the knowledge to ignore infeasible and duplicate diagrams, making sure it would eventually find the diagrams needed to solve the problem.

Depot specific forbidden paths

During development of the algorithm we encountered a situation that resulted in wrong behavior of the algorithm. It was not able to generate the optimal LP solution. To understand what is happening we need to remember that RCSP generates paths that start and end at a specific depot. CLP also takes a starting and ending depot as arguments to correctly combine pass rides to form a feasible diagram. However forbidden paths were not coupled with a depot and would block the path for any depot.
In that specific situation, we had two depots D_a and D_b and a sequence of crew requirements R. One of the diagrams in the optimal solution is the diagram that starts and ends at depot D_b and follows the crew requirements in R. However during execution we would first find the diagram starting at D_a and following the crew requirements in R. Afterwards we would find the same diagram through D_a again, resulting in a duplicate diagram. This in turn instantiated a forbidden path with the crew requirements R, which blocked the optimal diagram through D_b from ever being found.
We solved this in the first place by making sure every RCSP path was given to CLP with every starting depot as input, disregarding the depot RCSP generated the path for. We would create a forbidden path whenever one of the depots would receive an infeasible verdict by CLP, thus increasing the total amount of forbidden paths more than strictly necessary.
After concluding that this amount of forbidden paths was detrimental to the runtime of the

algorithm, we decided to implement a more efficient approach. We coupled forbidden paths optionally to crew depots, so we could have general forbidden paths and depot specific forbidden paths. An RCSP algorithm from depot D_a would only consider general pass rides and forbidden paths coupled to D_a.

5.6.5 Repricing paths

The following idea was inspired by an idea from Desrosiers and Lübbecke (2005). Their idea was to introduce dual variables for the diagrams in the pool. With these dual variables they could evaluate the importance of the diagrams to close the gap and could delete potential worthless diagrams. Similarly we should be able to reevaluate forbidden paths to discover their importance. RCSP will only generate paths that have negative reduced costs. When a forbidden path would have a positive reduced cost, there is no sense in blocking it explicitly with a forbidden paths, since the reduced cost resource would have ignored the path anyway. We decided to implement an additional step in the column generation scheme that would reprice the paths with the current dual variables of the crew requirements. Whenever a path would have too much reduced cost, we would simply delete it. We calculated the cost of a path in the following way.
Let us say we have a forbidden path p consisting of a sequence of crew requirements $R = \{cr_1, \ldots, cr_n\}$, for which we have dual variables π_{cr} $\forall cr \in R$. We consider a fixed cost per diagram dc and a working time cost of f per minute. Depending on whether a depot was specified for the forbidden path, we are able to determine whether we need pass rides pr_{start}, pr_{end} to reach the first crew requirement and to get home. We would estimate it's reduced cost r in two ways:

$$r = dc + (cr_n^{endtime} - cr_1^{startime}) * f + \sum_{cr \in R} \pi_{cr} \quad \text{(when no pass rides are needed)} \tag{5.21}$$

$$r = dc + (pr_{end}^{endtime} - pr_{start}^{starttime}) * f + \sum_{cr \in R} \pi_{cr} \quad \text{(when pass rides are needed)} \tag{5.22}$$

Finally we need to decide the cutoff value for r to determine which forbidden paths are not necessary anymore and can be deleted. We do this in Section (6.3.3).

6. RESULTS

6.1 Branch and Price

In this section we observe the results and performance of the Branch and Price algorithm. First we will see the performance when we utilize only the decision to branch on a diagram. Then we will see the result of branching on the number of diagrams. Finally we discuss the effects of choosing which diagram to fix.

6.1.1 Branch on a diagram only

After implementing this branching step we found there were some obstacles to overcome. There are a few downsides to using the fixed diagram strategy. One of the obvious downsides is that when we forbid a diagram we have to add an additional resource to RCSP. With RCSP already being a bottleneck in the column generation scheme, adding more complexity to the algorithm is not a good idea. This, however, seemed not to have such a large impact. In the smaller 33 puzzle the normal RCSP took around 3 seconds. Adding 15 additional resources to account for 15 forbidden diagrams pushed the running time to 4 seconds. The running time of RCSP is potentially exponential on the number of nodes and edges of the graph, but scales linearly with the number of resources. However when the number of forbidden paths rises due to a large branch and price tree, this could become a notable factor.
A more serious problem with the branching strategy was that the impact on the LP goal value was not strong enough. The algorithm had to go very deep into the tree, restricting the LP more and more, until the LP goal value became worse than the best known MIP solution. For the small 33 puzzle the algorithm had to solve more than 12000 nodes and was still not finished after 1 day and 11 hours, as seen in Table (6.1).

Total nodes	**Solved nodes**	**Unsolved nodes**	**Run duration**
12339	12205	134	1 day, 11:30:22

Tab. 6.1: Intractable Branch and Price tree

6.1.2 Branch on number of selected diagrams

The result of this branching strategy is rather significant. Where earlier we were unable to solve the intractable tree of the 33 puzzle with more than 12000 nodes, the tree becomes a lot smaller. This enabled us to find and proof the solution to the 33 puzzle optimal. The results can be found in Table (6.2).

When we force the number of diagrams to be smaller than $\lfloor s \rfloor$ the problem becomes infeasible and takes a hefty penalty, so we can *prune by bound*. When we force the number of diagrams to be bigger than $\lceil s \rceil$, we can instantly *prune by optimality*. The tree is only 3 nodes large and

Node name	Final LP goal value	Final MIP goal value	Solve duration
Root	641.44	709.10	0:02:57 (831)
Root child P_r	709.10	709.10	0:00:35 (804)
Root child P_l	3000620	3000620	0:00:13 (294)

Tab. 6.2: Tractable Branch and Price tree

is solved in less than 4 minutes.
With the confirmation that the 33 puzzle became tractable we set out to solve the 340 puzzle and reached the following result.

340 Puzzle optimality

We were able to traverse the complete branch and price tree in 11 hours and 7 minutes, to prove a solution with 41 diagrams and a goal value of 6218.9 optimal for the 340 puzzle.

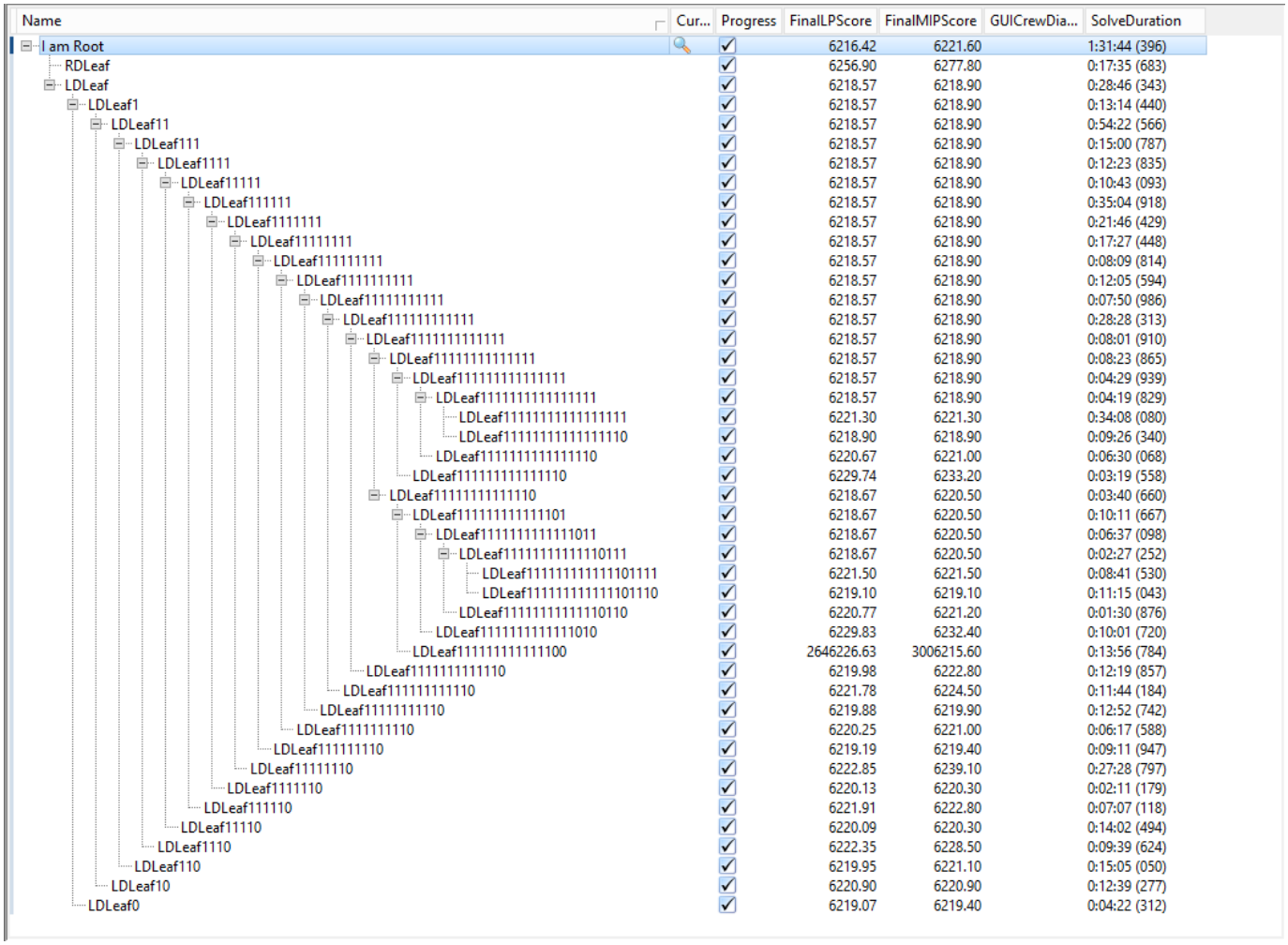


Fig. 6.1: Branch and Price tree for the 340 puzzle

Figure (6.1) is a screenshot from the developer screen in the Quintiq software.
In the first column we see a hierarchical list. Every step further in the hierarchy is a level further in the Branch and Price tree, and thus contains an extra branching constraint.
We first branch on the number of selected diagrams creating the nodes `RDLeaf` and `LDLeaf`,

respectively P_r and P_l. Then every 1 in a name signifies the fixing of a diagram and a 0 signifies forbidding a diagram.
The next two columns are mainly a visual aid during execution of the algorithm and show what parts of the tree have already been solved.
Column `FinalLPScore` and `FinalMIPScore` show the eventual optimal LP goal value for the problem node and MIP solution goal value we found after the pricing step finished. These values dictate which pruning rules we can execute.
The last interesting column is the `SolveDuration` column, showing the time it took to solve the particular node.

6.1.3 Choosing which diagram to fix

Implementation of this method showed that the size of the tree would vary a lot between runs. This is a result of the fact that there can be quite a lot of diagrams that are being selected with the same fractional value. For example there are quite a few diagrams selected fully, with a selection variable of 1.0. From these diagrams with the same highest fractional selection value, we selected the actual diagram we will branch on randomly. This randomness caused the tree to grow differently in different optimization runs of the puzzle. In an attempt to make the selection of the diagram more deterministic we devised a stricter selection method. From the pool of diagrams with the same highest fractional selection value we would select the one covering the most crew requirements. Motivated by the idea that covering more crew requirements for one diagram indicates higher quality and thus will strengthen the chances of pruning as described above. If we then still have multiple candidates, we will select the one with lowest diagram execution cost. Motivated by the idea that being able to execute many crew requirements efficiently with a low cost indicates high quality for a diagram. In theory this does not guarantee a unique diagram to be selected, but in practice this is sufficient.
However after testing we encountered still different branch and price trees. It became apparent our assumption that LP optimality in a node would give us the same highly fractionally selected diagrams was false. Due to symmetry in the problem, there were many different optimal solution for a node. These optimal solutions could be different, having a diagram covering few crew requirements fully selected in one solution and a diagram covering many crew requirements fully selected in the other. So again chance dictated the optimal solution we would find and thus on which diagram we branched.
To solve this problem we decided we should break the symmetry of the problem in some way. To do this we decided to incentivize the LP to want to select the diagrams covering more crew requirements slightly more. We reduced the cost of a diagram d by $\varepsilon \cdot |\{a_{rd} : a_{rd} = 1, r \in R\}|$, where we selected ε small enough such that it would not change the optimal solution. For the 340 puzzle we chose $\varepsilon = 0.0001$. This resulted in the same Branch and Price tree across different runs of the algorithm, as well as a slight increase in performance.

6.1.4 Conclusion

We observed that the power of the Branch and Price algorithm is very dependent on the branching decision. Where only fixing diagrams had nearly no effect, adding an additional branching decision for the number of diagrams selected reached impressive results. We were able to find and proof the solution to a real-world problem optimal. We discovered that choosing which diagram to fix can have a big impact on the size of the Branch and Price tree. We were content with how DFS performed to traverse the tree.
Although the $\mathcal{NP}$-hard nature of the crew scheduling problem became very apparent while using the Branch and Price algorithm and observing the run duration, we were satisfied with the performance. Although a rail service provider might not want to wait for the full Branch and Price tree to be solved, we can still take away the power of the branching decision on the

number of selected diagrams. We can see that often this branching step alone is enough to find the optimal solution, just not to prove it optimal.

6.2 Stabilization techniques

In this section we present the results of using stabilization techniques.

6.2.1 Experiments

We tested the stabilization methods as an additional subroutine in the Linear Program step to generate dual variables before executing the RCSP to solve the pricing problem. We tested the effect on the 33 puzzle and the 340 puzzle, where we solve the complete Branch and Price tree. In addition to that we tested the effect solely on the root node of the Branch and Price tree.

6.2.2 Results - full puzzles

First let us observe the difference between the IPS Branch and Price versus the normal Branch and Price approach for the 33 puzzle. The first column shows the parameter for the IPS method, the amount of D^u problems solved, which combined determine the dual values. Next are the amount of LP, MIP and RCSP steps executed during the run. The duration column shows the duration of the run. The poolsize column shows the amount of diagrams that were created during the run.

Param.	LP steps	MIP steps	RCSP steps	Duration	Poolsize
3	37	3	40	0:02:34	437
10	39	3	42	0:02:37	441
10	30	3	33	0:02:10	451
20	35	3	38	0:02:29	479
40	34	3	37	0:02:24	458
100	39	3	42	0:02:42	518
Average	**35.67**	**3**	**38.67**	**0:02:29**	**464**

Tab. 6.3: IPS results for 33 puzzle with dual solve setting activated

Param.	LP steps	MIP steps	RCSP steps	Duration	Poolsize
3	34	3	37	0:02:23	442
10	30	3	33	0:02:13	462
10	35	3	38	0:02:27	496
20	37	3	40	0:02:33	515
40	31	3	34	0:02:14	447
100	32	3	35	0:02:19	476
Average	**33.17**	**3**	**36.17**	**0:02:22**	**473**

Tab. 6.4: IPS results 33 puzzle without dual solve setting

When we analyze the results we can conclude there is quite a bit of variance between the individual runs, but we can look at the averages to counteract this effect.

We can see that the IPS method needs fewer step invocations and can solve the puzzle in less time than the normal counterpart. We can also see some influence of the dual setting with which the LP is solved. Even though the setting reduces the dual variables to normal sizes, it

	LP steps	MIP steps	RCSP steps	Duration	Poolsize
	38	3	41	0:02:37	493
	36	3	39	0:02:31	522
	44	3	47	0:02:57	550
	38	3	41	0:02:35	477
	38	3	41	0:02:36	495
	44	3	47	0:02:57	572
Average	**39.67**	**3**	**42.67**	**0:02:42**	**518.17**

Tab. 6.5: Normal results 33 puzzle with dual solve setting activated

	LP steps	MIP steps	RCSP steps	Duration	Poolsize
	29	3	32	0:02:11	497
	36	3	39	0:02:38	523
	31	3	34	0:02:21	515
	44	3	47	0:02:53	506
	36	3	39	0:02:33	552
	43	3	46	0:02:55	572
Average	**36.5**	**3**	**39.5**	**0:02:35**	**527.5**

Tab. 6.6: Normal results 33 puzzle without dual solve setting

Type	LP steps	MIP steps	RCSP steps	Duration	Poolsize
IPS*	35.67	3	38.67	0:02:29	464
Normal*	39.67	3	42.67	0:02:42	518.17
IPS	33.17	3	36.17	0:02:22	473
Normal	36.5	3	39.5	0:02:35	527.5

* is with simplex dual setting

Tab. 6.7: Average results for the 33 puzzle

has a negative impact on the amount of step invocations and solving duration. We can also see that IPS ends with fewer diagrams than the normal runs. This indicates it is capable of finding the diagrams that are part of the optimal LP solution faster than a normal run, which generates more useless diagrams. Since we generate diagrams during the pricing problem, which is completely determined by the dual variables, we can conclude that IPS is able to generate more useful dual variables. The quality of the diagrams that are generated is higher, which in turn can also lead to a better LP solution, which produces better dual variables. This becomes a positive spiral that explains the difference in poolsize.
We turn towards the bigger puzzle to see whether the same pattern holds. The type column shows whether the run was with or without IPS, normal being without. The nodes column contains the number of nodes that were created in the Branch and Price tree. The dual setting column shows whether we told the LP solver to solve the dual, instead of solving the primal.

Type	Nodes	LP steps	MIP steps	RCSP steps	Duration	dual setting
Normal	51	545	51	596	9:14:11	No
Normal+	53	517	53	570	9:04:07	No
Normal+	53	532	53	585	9:08:56	No
Normal+	53	589	53	642	9:43:22	No
Normal+	53	559	53	612	9:09:58	No
IPS	51	1002	51	1053	15:18:37	No
IPS	41	672	41	713	10:34:22	Yes
IPS+	53	692	53	745	10:38:51	No
IPS+	55	884	55	939	14:04:32	No

+ is with a more deterministic branching strategy

Tab. 6.8: Results for the 340 puzzle

We do not see the improvement here that we saw earlier. There seems to be some variance in the normal run and significant variance in the IPS run. This variance is what led us to develop the more deterministic branching strategy. We have to conclude that for the full 340 puzzle IPS improves neither the invocation amount nor the duration of the full branch and price procedure. But a lot is happening during the branch and price and we have seen quite some improvement with the 33 puzzle, so we aim our attention towards the root node to ignore most of the branch and price influence.

6.2.3 Results - Root Node

The largest amount of time spent in the branch and price tree is almost always at the root node. Here we generate the most diagrams of all nodes in the tree. We have gathered the same data as for the whole branch and price tree, but now only what was used to solve the root nodes. We ignore the amount of nodes, as we only consider one node. We can now also consider the MIP goal value, which need not be optimal at the end of the root node. The LP goal and MIP goal columns show the respective LP and MIP goals at the end of the run.

Param.	LP steps	MIP steps	RCSP steps	Duration	LP goal	MIP Goal
3	30	2	29	0:02:00	641.4375	727
10	32	2	31	0:02:02	641.4375	709.1
10	25	2	24	0:01:41	641.4375	709.1
20	30	2	29	0:02:00	641.4375	709.1
40	29	2	28	0:01:54	641.4375	716.1
100	32	2	31	0:02:07	641.4375	725.1
Avg	**29.67**	**2**	**28.67**	**0:01:57**	**641.4375**	**715.92**

Tab. 6.9: IPS root node runs 33 puzzle - Simplex dual setting

Param.	LP steps	MIP steps	RCSP steps	Duration	LP goal	MIP Goal
3	26	2	25	0:01:45	641.4375	727.6
10	25	2	24	0:01:43	641.4375	729
10	30	2	29	0:01:58	641.4375	729
20	33	2	32	0:02:08	641.4375	716.1
40	26	2	25	0:01:45	641.4375	709.1
100	27	2	26	0:01:49	641.4375	716.1
Avg	**27.83**	**2**	**26.83**	**0:01:51**	**641.4375**	**721.15**

Tab. 6.10: IPS root node runs 33 puzzle - No simplex dual setting

	LP steps	MIP steps	RCSP steps	Duration	LP goal	MIP Goal
	31	2	30	0:02:01	641.4375	709.1
	28	2	27	0:01:52	641.4375	709.1
	35	2	34	0:02:14	641.4375	709.1
	32	2	31	0:02:03	641.4375	709.1
	31	2	30	0:02:00	641.4375	709.1
	34	2	33	0:02:12	641.4375	709.1
Avg	**31.83**	**2**	**30.83**	**0:02:04**	**641.4375**	**709.1**

Tab. 6.11: Normal root node runs 33 puzzle - Simplex dual setting

	LP steps	MIP steps	RCSP steps	Duration	LP goal	MIP Goal
	25	2	24	0:01:45	641.4375	716.1
	28	2	27	0:01:58	641.4375	709.1
	25	2	24	0:01:47	641.4375	716.1
	38	2	37	0:02:21	641.4375	709.1
	29	2	28	0:01:57	641.4375	716.1
	33	2	32	0:02:10	641.4375	716.1
Avg	**29.67**	**2**	**28.67**	**0:02:00**	**641.4375**	**713.767**

Tab. 6.12: Normal root node runs 33 puzzle - No simplex dual setting

Type	LP steps	MIP steps	RCSP steps	Duration	LP goal	MIP Goal
IPS*	29.67	2	28.67	0:01:57	641.4375	715.917
Normal*	31.83	2	30.83	0:02:04	641.4375	709.1
IPS	27.83	2	26.83	0:01:51	641.4375	721.15
Normal	29.67	2	28.67	0:02:00	641.4375	713.767

* is with simplex dual setting

Tab. 6.13: Average results for the 33 root node puzzle

Once more we see that the interior point method outperforms the normal approach, both in

solve duration and step invocations. An interesting observation is that the MIP goal value of the IPS runs is worse than in the normal runs. This is probably due to the fact that IPS needs fewer iterations to converge to the optimal LP goal value and thus generates fewer diagrams. This reduces the quality of the MIP solution, which needs more diagrams to be able to combine the good ones into a solution. These combination diagrams are not generated for the LP, since it can select fractionally. We again observe how the big puzzle behaves, when restricting ourselves to only the root node.

Param.	LP steps	MIP steps	RCSP steps	Duration	LP goal	MIP Goal	Poolsize
10	66	0	67	1:00:43	6216.417	6225.3	5258
10*	65	0	66	0:56:05	6216.417	6233.3	5141
10*	72	0	73	1:05:42	6216.417	6225.3	5168
40	69	0	70	1:01:08	6216.417	6224.3	5288
10+	70	0	71	1:02:40	6216.382	6218.9	5260
10+	66	0	67	1:00:22	6216.382	6232.8	5167
Average	**68**	**0**	**69**	**1:01:07**	**6216.405**	**6226.6**	**5213.67**

* is with simplex dual setting, + is with deterministic branching and new diagram cost

Tab. 6.14: IPS root node runs 340 puzzle

Param.	LP steps	MIP steps	RCSP steps	Duration	LP goal	MIP Goal	Poolsize
Normal	81	0	82	1:12:41	6216.417	6244.4	6139
Normal*	93	0	94	1:33:17	6216.417	6218.9	6232
Normal+	81	0	82	1:10:47	6216.382	6230.6	6072
Normal+	79	0	80	1:08:55	6216.382	6243.9	6076
Normal+	82	0	83	1:09:44	6216.382	6218.9	6057
Average	**83.2**	**0**	**84.2**	**1:15:05**	**6216.396**	**6231.3**	**6115.2**

* is with simplex dual setting, + is with deterministic branching and new diagram cost

Tab. 6.15: Normal root node runs 340 puzzle

Param.	LP steps	MIP steps	RCSP steps	Duration	LP goal	MIP Goal	Poolsize
IPS	68	0	69	1:01:07	6216.405	6226.6	5213.67
Normal	83.2	0	84.2	1:15:05	6216.396	6231.3	6115.2

Tab. 6.16: Average results root node runs 340 puzzle

Now we can still see a clear improvement of IPS over normal runs. Both the invocation of the steps and the duration has improved. Unexpectedly we can see no big difference in the MIP goal value, even though there is a significant difference in the amount of diagrams in the pool. We have to conclude that somewhere in the child nodes of the Branch and Price tree this head start in step invocations and duration is lost. Exactly why this is the case is a topic for future research.

6.2.4 Conclusion

We investigated the parameters for the Interior Point Stabilization method and concluded that a combination of 10 different solutions was sufficient. When using IPS we observed an increase in performance for the 33 puzzle and for the 340 puzzle root node, both in RCSP invocation as well as run time. However, we were unable to conclude the same for the full 340 puzzle, where the technique resulted in more RCSP invocations and run time instead.
We recommend for further research to answer the question why the combination between Branch and Price and IPS did not result in a performance increase. The improvement while using IPS in solving the root node warrants the recommendation to use the technique at least for the root node in the Branch and Price algorithm, where no effect of branching can be observed. The LP convergence is positively influenced by IPS there.

6.3 RCSP + CLP

In this section we describe the results of using the RCSP and CLP combination. We will see the effects of the graph reduction, run time decline due to forbidden paths and the results of attempts to remedy that decline.

6.3.1 Graph reduction

Deletion of the nodes and edges described in Section (5.6) and Tables (5.12) and (5.13), resulted in a size reduction. Since nodes and edges are filtered out depending on crew depot, shift type and break pattern, a maximum size and average size are more descriptive. They can be found in Table (6.17) and visualized in Figure (6.2).

Puzzle size	**Type**	**RCSP**	**RCSP + CLP**
33	Maximum nodes	3612	121
	Average nodes	2180.82	121
	Maximum edges	6955	619
	Average edges	3988.69	471
340	Maximum nodes	19135	1036
	Average nodes	11878.20	1036
	Maximum edges	57691	8371
	Average edges	35087.20	7040.80

Tab. 6.17: The RCSP graph size for both puzzles

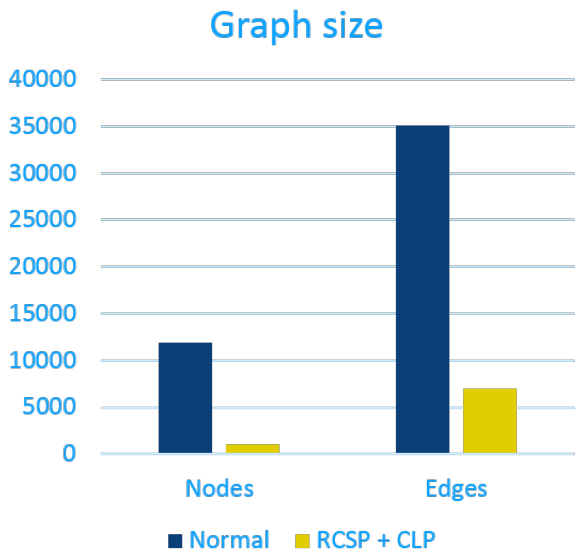


Fig. 6.2: Graph size 340 puzzle

6.3.2 Forbidden paths

In contrast to the branch and price algorithm, the RCSP plus CLP combination is bothered more by the downsides of using forbidden paths. The number of forbidden paths can grow very large and have a significant impact on the runtime of the RCSP algorithm.

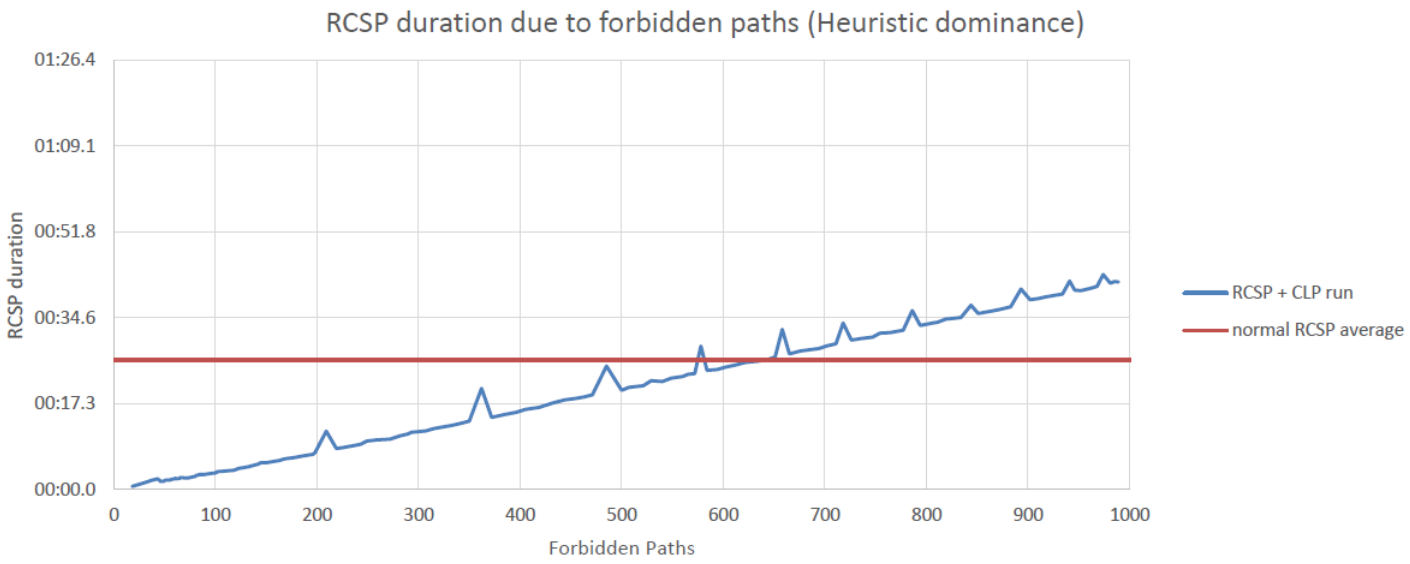


Fig. 6.3: RCSP duration with heuristic dominance average

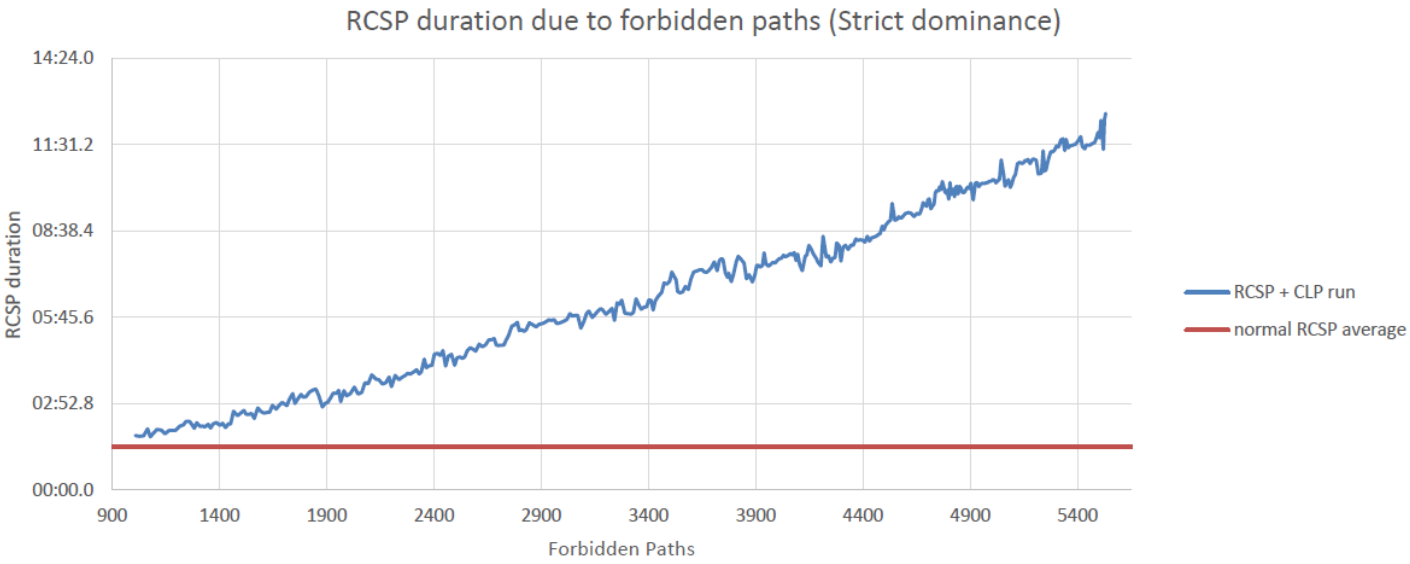
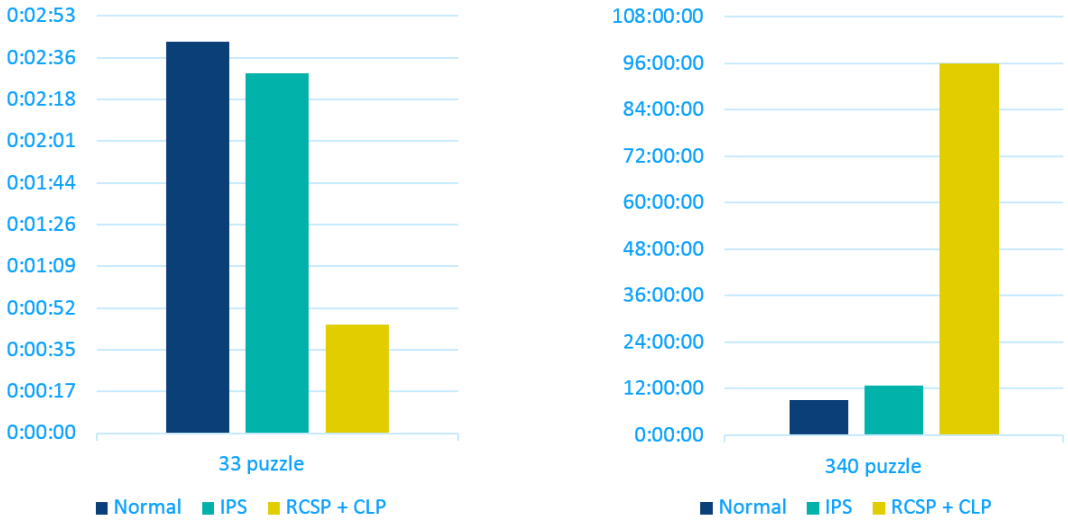


Fig. 6.4: RCSP duration with strict dominance average

From Figure (6.3) we can conclude that the combination between RCSP and CLP does start off faster than the average time it takes for the original RCSP with heuristic dominance to finish an invocation, but eventually the combination slows down to take about twice as much time. Figure (6.4) shows that when we switch over to strict dominance our time loss becomes even more severe. The amount of forbidden paths is already high enough for the combination to start off slower than the average duration of an original RCSP run with strict dominance. We conclude that this approach scales linearly with the amount of forbidden paths and thus eventually becomes worse than the roughly constant runtime of the original RCSP. An upside of using the RCSP and CLP combination is the fast execution time of the CLP component, taking only a fraction of the runtime of the RCSP. Where the RCSP runtime grows into the minutes, the CLP runtime stays below mere seconds and can be effectively disregarded.
Even though the RCSP takes very long to execute for the bigger puzzle, we do see a significant

increase in speed for smaller puzzles, which can be solved before the scalability takes over the average time of the normal RCSP. We solved the 33 puzzle with this same approach and found the following results as displayed in Figure (6.5).



(a) Solve duration for the 33 puzzle

(b) Solve duration for the 340 puzzle

Fig. 6.5: Results using normal, IPS and RCSP + CLP

To reduce the impact of the forbidden paths slowing the runtime of RCSP, we tried two different approaches:

Depot specific forbidden paths

After implementation of depot specific forbidden paths we observe an improvement in Figure (6.6).

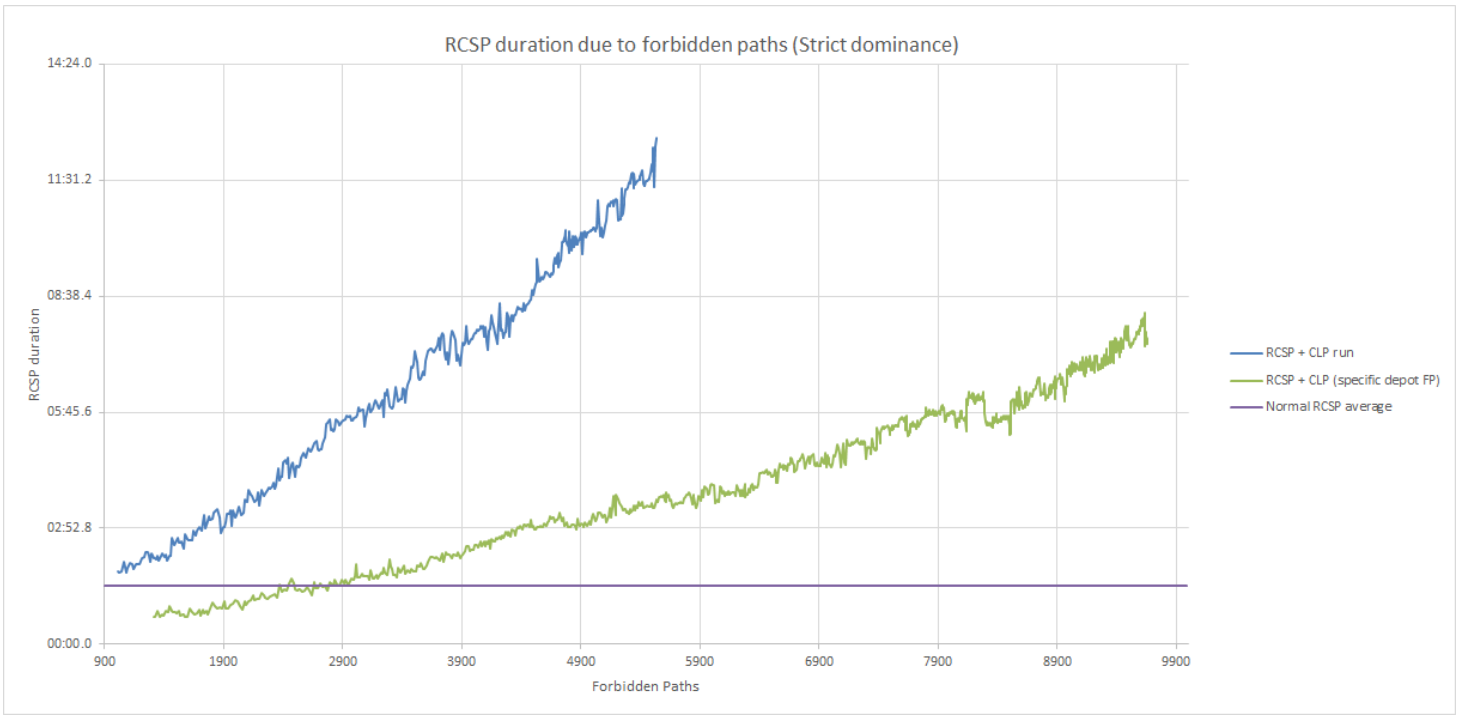


Fig. 6.6: RCSP duration with strict dominance average

Figure (6.6) shows the approach still has some merit when using the strict dominance rule, but we find ourselves a lot slower than the average original RCSP duration.
An interesting fact is the distribution of forbidden paths among the different depots in the problem, as shown in Figure (6.7).

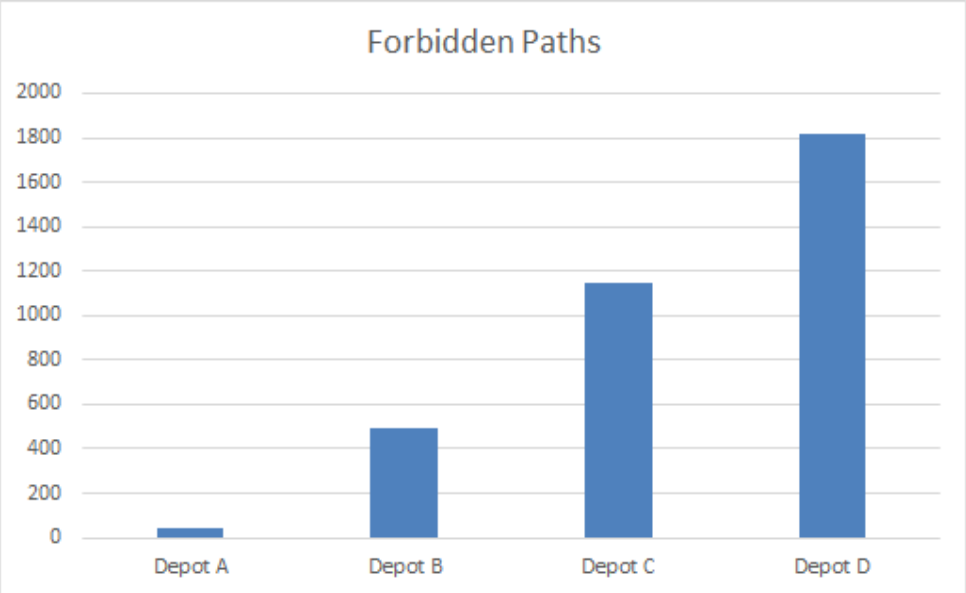


Fig. 6.7: Forbidden path distribution among the depots

The data is taken from another run than the one providing the RCSP duration in Figure (6.6), which explains why the forbidden paths do not add up to 9800. The result shown in Figure (6.7) indicates that there can be a significant difference between the depots in the problem. It might be that improving diagrams favor a specific starting depot or that optimal diagrams are more easily found for certain depots, resulting in RCSP visiting depots more often or less often respectively. When no paths are found, starting at a depot, no forbidden paths will be created for that depot.

6.3.3 Repricing paths

In Subsection (5.6.5) we determined that we could evaluate whether we still needed forbidden paths by repricing them, but first we need to decide the cutoff value for r, the repriced value of a path. First we decided to try out deleting paths when $r > -10.0$, with the reasoning that wrongly deleted paths would just be generated again, but during execution, the algorithm entered a loop. It generated the forbidden paths, deleted them after a few steps and would then generate those exact paths again, making no progress. This happened at the end of the run, when the negative reduced cost of diagrams becomes very small, so the $r > -10.0$ was too broad a definition to delete paths. We decided to only delete a path when $r > 0.0$, which would ensure that we would only delete paths that RCSP would never generate anyway, due to it having positive reduced cost.

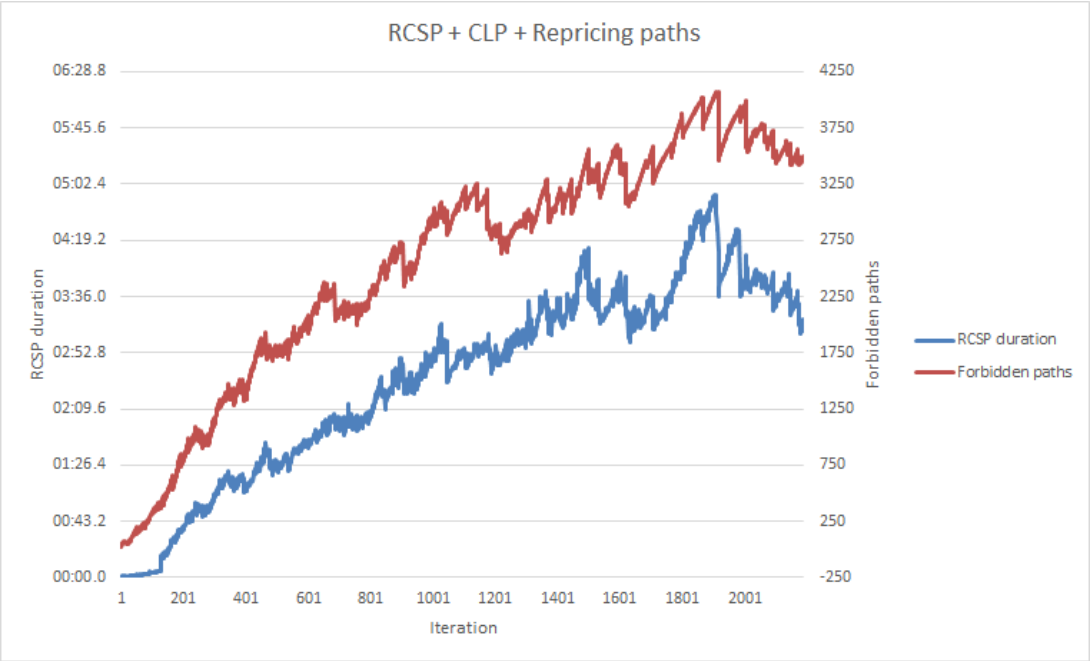


Fig. 6.8: Results of using repricing on paths

From Figure (6.8) we can conclude again a significant speed increase of the RCSP duration and can observe the fluctuations in the amount of forbidden paths that result from the deletion step, which is also visible in the RCSP duration.

6.3.4 LP convergence

Aside from the partial speed gain that is lost after accumulating too many forbidden paths, there is also the problem of the decreased speed in LP convergence. In the normal RCSP approach we execute more algorithms per RCSP invocation and a side effect of that is the fact that we generate less diagrams during the RCSP + CLP combination per invocation. Combine that with the concept of infeasible diagrams being generated and you lose a lot of space to generate the diagrams you need to converge towards the optimal LP solution. This means that even though we managed to speed up the RCSP + CLP combination, it is still not capable of finding the optimal LP solution in reasonable time, taking over 4 days instead of the 1 hour we achieve with Interior Point Stabilization.

6.3.5 Conclusion

Combining RCSP and CLP to solve the pricing problem is effective in reducing the graph size and performs great on small puzzles. The weakness of the combination becomes apparent in larger puzzles, where dealing with infeasible diagrams and duplicates with forbidden paths takes a big toll on the RCSP runtime as well as needing more generation steps to converge the solution to the optimal solution, since less quality diagrams are found per invocation. The relatively low CLP execution time inspires ideas of using a fast heuristic combination to create the initial diagram pool and kick start the column generation scheme.

7. CONCLUSION AND DISCUSSION

In this chapter we will revisit the most important topics researched in this thesis, which were used to solve the train crew scheduling problem.

7.1 Column Generation

We were very satisfied with the power of *Column Generation* to find its way towards a good solution. We believe every problem instance would benefit from using this method. The combination with the *Resource Constrained Shortest Path* (RCSP) algorithm allows the technique to adapt and be flexible, resulting in the ability to tackle many additional labor rule or constraint applicable to the crew scheduling problem. One of the challenges one can face is the exponential character of these algorithms, where you have to let go of optimality and resort to more heuristics to achieve feasible runtime.

7.2 Branch and Price

The *Branch and Price* algorithm is an essential step to bridge the gap between the LP solution and the real-world MIP solution we are really interested in. We believe every problem being solved with column generation benefits tremendously from using at least some insights from the Branch and Price algorithm. The main differences between similar problems and problem instances are the constraints we add in branching steps and the way we traverse the tree. Although these steps can vary per problem, we believe universally good branching strategies are those that heavily impact the LP goal value, often by disregarding a large part of the fractional solution space. Additionally we advise to follow the branch with the lowest optimal LP value, which we believe to have the highest chance of resulting in a good integral solution, with which the tree size can be further reduced by bounding.

7.3 Interior Point Stabilization

To improve the quality of the dual variables we used *Interior Point Stabilization* (IPS). It is worth noting the trade-off between the additional time to solve IPS and the results of the quality gain. In our examples and what we believe generally happens, is a decrease in generation steps and thus often a decrease in runtime of the algorithm. The behavior of the algorithm in the Branch and Price algorithm is not yet fully understood. Currently we can only advise to use it during the root node of the Branch and Price tree, where concepts from Branch and Price are not yet used. Future research should answer whether the technique can be adapted to Branch and Price or that its decline is particular to this problem.

7.4 Combination of techniques

What we experienced to be the bottleneck in the algorithm is the generation phase. The exponential nature of the RCSP algorithm, combined with the increase in graph size when modeling constraints, makes this a time consuming step. Attempts to speed this up by using a combination between RCSP and *Constraint Logic Programming* (CLP) were successful only for small

subpuzzles. The way we combined the two algorithms is highly problem specific, but we believe the concept of combining algorithms can be universally applied. The strongest advise we can extrapolate from our experiments is to use the combination of RCSP and CLP in the beginning of the problem, when its merits outweigh the downsides, and continue with the original RCSP approach afterwards. To determine the ideal switching point is hard and problem specific, so with the eye on simplicity combining algorithms is a risk in general.

Finally we can heartily recommend you to try solving this or similar problems with the described techniques, as it is a lot of fun.

8. FUTURE RESEARCH

We briefly discuss topics we encountered during research that we did not have time to investigate and incorporate in the thesis.

8.1 Tree structure

We have seen quite a bit of difference in the tree size caused by non-deterministic selection of the fixed diagram. It would be advised to quantify the effect of the decisions that can be made there and extrapolate a strategy to minimize the amount of step invocations and the duration of solving the puzzle. In this thesis we use DFS successfully to traverse the tree. It would be a good idea to find out whether this technique is generally useful for different instances and to investigate whether other tree traversal strategies might prove more generally applicable.

8.2 Stabilization techniques

We have seen IPS improve the small puzzle runs and the root node runs for the 340 instance. We have not yet discovered exactly why the technique does not improve the statistics for the full 340 puzzle. Questions remain whether this effect comes from the branching strategy, which performs well for the normal run, but has a bad synergy with the IPS method. On the other hand the technique might not be suitable for dealing with the small differences between problems to generate the needed diagrams, while it performs great for solving the initial LP part of the puzzle. We would like to discover if we can make Branch and Price and IPS work together.

8.3 RCSP + CLP

We have seen how the RCSP and CLP combination has trouble converging to the optimal LP solution. We wonder whether we could adjust the algorithm to find improving diagrams more easily or ignore the diagrams that are infeasible.
The combination opens up possibilities to research a kickstart procedure for the column generation scheme. We would like to find out what we could gain by starting the column generation with a pool of diagrams of reasonable quality.
We observed that the graph size reduction resulted in substantially lower run times. Would it be possible to gain a graph size reduction by modeling the break resource differently in RCSP, so we would gain time while not losing convergence speed?

8.4 Lagrange relaxation

Lastly we have seen that Lagrange relaxation is quite common in the literature. However we did not use it, since we saw no immediate benefits to using it. It would be great to discover exactly why this technique is so commonly used and whether it can outperform or even work in tandem with RCSP and linear programming.

BIBLIOGRAPHY

Abbink, Erwin (2008), "Solving large scale crew scheduling problems by using iterative partitioning." Technical report.

Abbink, Erwin JW, Luis Albino, Twan Dollevoet, Dennis Huisman, Jorge Roussado, and Ricardo L Saldanha (2011), "Solving large scale crew scheduling problems in practice." *Public Transport*, 3, 149–164.

Alefragis, Panayiotis, Peter Sanders, Tuomo Takkula, and Dag Wedelin (2000), "Parallel integer optimization for crew scheduling." *Annals of Operations Research*, 99, 141–166.

Azadeh, Ali, M Hosseinabadi Farahani, H Eivazy, S Nazari-Shirkouhi, and G Asadipour (2013), "A hybrid meta-heuristic algorithm for optimization of crew scheduling." *Applied Soft Computing*, 13, 158–164.

Barnhart, Cynthia, Ellis L Johnson, George L Nemhauser, Martin WP Savelsbergh, and Pamela H Vance (1998), "Branch-and-price: Column generation for solving huge integer programs." *Operations research*, 46, 316–329.

Beasley, John E and Paul C Chu (1996), "A genetic algorithm for the set covering problem." *European journal of operational research*, 94, 392–404.

Bradley, Stephen, Arnoldo Hax, and Thomas Magnanti (1977), "Applied mathematical programming." URL http://web.mit.edu/15.053/www/AMP-Chapter-04.pdf.

Caprara, Alberto, Matteo Fischetti, and Paolo Toth (1999), "A heuristic method for the set covering problem." *Operations research*, 47, 730–743.

Caprara, Alberto, Matteo Fischetti, Paolo Toth, Daniele Vigo, and Pier Luigi Guida (1997), "Algorithms for railway crew management." *Mathematical programming*, 79, 125–141.

Caprara, Alberto, Leo Kroon, Michele Monaci, Marc Peeters, and Paolo Toth (2007), "Passenger railway optimization." *Handbooks in operations research and management science*, 14, 129–187.

Cavique, L, C Rego, and I Themido (1999), "Subgraph ejection chains and tabu search for the crew scheduling problem." *Journal of the Operational Research Society*, 608–616.

Cormen, Thomas H, Charles E Leiserson, Ronald L Rivest, and Clifford Stein (2001), "Introduction to algorithms second edition."

Dantzig, George B (1963), "Linear programming and extensions."

Dantzig, George B and Philip Wolfe (1960), "Decomposition principle for linear programs." *Operations research*, 8, 101–111.

Desrochers, Martin, Jacques Desrosiers, and Marius Solomon (1992), "A new optimization algorithm for the vehicle routing problem with time windows." *Operations research*, 40, 342–354.

Desrosiers, Jacques (2000), "Column generation.", URL https://www.iro.umontreal.ca/~gendron/PLU6000/Column%20Generation.ppt. Visited on 2017-03-20.

Desrosiers, Jacques, Yvan Dumas, Marius M Solomon, and François Soumis (1995), "Time constrained routing and scheduling." *Handbooks in operations research and management science*, 8, 35–139.

Desrosiers, Jacques and Marco E Lübbecke (2005), "A primer in column generation." In *Column generation*, 1–32, Springer.

Dijkstra, Edsger W (1959), "A note on two problems in connexion with graphs." *Numerische mathematik*, 1, 269–271.

Du Merle, Olivier, Daniel Villeneuve, Jacques Desrosiers, and Pierre Hansen (1999), "Stabilized column generation." *Discrete Mathematics*, 194, 229–237.

Ferris, Michael C. (2002), "Sensitivity analysis and parametric linear programming.", URL http://pages.cs.wisc.edu/~ferris/cs525/sec92.pdf. Visited on 2017-05-17.

Fischetti, Matteo, Silvano Martello, and Paolo Toth (1989), "The fixed job schedule problem with working-time constraints." *Operations Research*, 37, 395–403.

Fisher, Marshall L (1981), "The lagrangian relaxation method for solving integer programming problems." *Management science*, 27, 1–18.

Freling, Richard, Ramon M Lentink, and Albert PM Wagelmans (2004), "A decision support system for crew planning in passenger transportation using a flexible branch-and-price algorithm." *Annals of Operations Research*, 127, 203–222.

Gavanelli, Marco and Francesca Rossi (2010), "Constraint logic programming." In *A 25-year perspective on logic programming*, 64–86, Springer.

Gecode (2017), "Generic constraint development environment.", URL http://www.gecode.org/index.html. Visited on 2017-09-15.

Glover, Fred (1989), "Tabu search—part i." *ORSA Journal on computing*, 1, 190–206.

Gualandi, Stefano (2013), "Crew scheduling: Models and algorithms.", URL http://www.imada.sdu.dk/~marco/DM204/Slides/lecture-2.pdf. Visited on 2017-03-20.

Hassannayebi, Erfan, Seyed Hessameddin Zegordi, and Masoud Yaghini (2016), "Train timetabling for an urban rail transit line using a lagrangian relaxation approach." *Applied Mathematical Modelling*, 40, 9892–9913.

Hoogeveen, Han (2016), "Lagrangean relaxation.", URL http://www.cs.uu.nl/docs/vakken/stt/Slides-Lagrange.pdf. Visited on 2017-03-20.

Huisman, Dennis (2007), "A column generation approach for the rail crew re-scheduling problem." *European Journal of Operational Research*, 180, 163–173.

Huisman, Dennis, Raf Jans, Marc Peeters, and Albert PM Wagelmans (2005), "Combining column generation and lagrangian relaxation." *Column generation*, 247–270.

Irnich, Stefan and Guy Desaulniers (2005), "Shortest path problems with resource constraints." *Column generation*, 33–65.

Jiang, Bo, Jin Tang, Xiaochun Cao, and Bin Luo (2017), "Lagrangian relaxation graph matching." *Pattern Recognition*, 61, 255–265.

Jütte, Silke (2012), *Large-Scale Crew Scheduling-Models, Methods, and Applications in the Railway Industry*. Kölner Wissenschaftsverlag.

Jütte, Silke, Marc Albers, Ulrich W Thonemann, and Knut Haase (2011), "Optimizing railway crew scheduling at db schenker." *Interfaces*, 41, 109–122.

Jütte, Silke, Daniel Müller, and Ulrich W Thonemann (2017), "Optimizing railway crew schedules with fairness preferences." *Journal of Scheduling*, 20, 43–55.

Jütte, Silke and Ulrich W Thonemann (2012), "Divide-and-price: A decomposition algorithm for solving large railway crew scheduling problems." *European Journal of Operational Research*, 219, 214–223.

Jütte, Silke and Ulrich W Thonemann (2015), "A graph partitioning strategy for solving large-scale crew scheduling problems." *OR spectrum*, 37, 137–170.

Juttner, Alpar, Balazs Szviatovski, Ildikó Mécs, and Zsolt Rajkó (2001), "Lagrange relaxation based method for the qos routing problem." In *INFOCOM 2001. Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 2, 859–868, IEEE.

Karp, Richard M (2010), "Reducibility among combinatorial problems." In *50 Years of Integer Programming 1958-2008*, 219–241, Springer.

Kim, Sehun, Kun-Nyeong Chang, and Jun-Yeon Lee (1995), "A descent method with linear programming subproblems for nondifferentiable convex optimization." *Mathematical programming*, 71, 17–28.

Kwan, Raymond SK (2011), "Case studies of successful train crew scheduling optimisation." *Journal of Scheduling*, 14, 423–434.

Laplagne, Ignacio Eduardo (2008), *Train driver scheduling with windows of relief opportunities*. Ph.D. thesis, University of Leeds.

Lau, Hoong Chuin and Aldy Gunawan (2012), "The patrol scheduling problem." PATAT.

Lehtihet, H.E. (2012), URL https://www.researchgate.net/post/What_are_the_differences_between_heuristics_and_metaheuristics. Visited on 2017-09-08.

Marsten, Roy E, WW Hogan, and Jacob Watson Blankenship (1975), "The boxstep method for large-scale optimization." *Operations Research*, 23, 389–405.

Mohan Akella, Avijit Sarkar, Sharad Gupta (2004), "Branch and price: Column generation for solving huge integer programs.", URL https://www.acsu.buffalo.edu/~nagi/courses/684/price.pdf. Visited on 2017-03-13.

Nielsen, Lars Kjaer (2011), *Rolling Stock Rescheduling in Passenger Railways: Applications in short-term planning and in disruption management*. EPS-2011-224-LIS.

Papavasiliou, Anthony (2013), "Lagrange relaxation: Operations research.", URL https://perso.uclouvain.be/anthony.papavasiliou/public_html/LagrangeRelaxation.pdf. Visited on 2017-03-20.

Pugliese, Luigi Di Puglia (2012), "Models and methods for the constrained shortest path problem and its variants." *4OR*, 10, 395–396.

Qiao, Wenxin, Masoud Hamedi, and Ali Haghani (2010), "Algorithm for crew-scheduling problem with bin-packing features." *Transportation Research Record: Journal of the Transportation Research Board*, 80–88.

Rousseau, Louis-Martin, Michel Gendreau, and Dominique Feillet (2007), "Interior point stabilization for column generation." *Operations Research Letters*, 35, 660–668.

Şahin, Güvenç and Birol Yüceoğlu (2011), "Tactical crew planning in railways." *Transportation Research Part E: Logistics and Transportation Review*, 47, 1221–1243.

Snijders, Hilbert and Ricardo L Saldanha (2015), "Security crew scheduling at netherlands railways."

Sobel, Joel (2012), "Lecture notes on linear programming for course economics 172a.", URL http://econweb.ucsd.edu/~jsobel/172f12/172f12home.htm. Visited on 2017-05-16.

Souai, Nadia and Jacques Teghem (2009), "Genetic algorithm based approach for the integrated airline crew-pairing and rostering problem." *European Journal of Operational Research*, 199, 674–683.

Talbi, El-Ghazali (2016), "Combining metaheuristics with mathematical programming, constraint programming and machine learning." *Annals of Operations Research*, 240, 171–215.

Vance, Pamela H, Alper Atamturk, Cynthia Barnhart, Eric Gelman, Ellis L Johnson, Alamuru Krishna, Deepa Mahidhara, George L Nemhauser, and Ranjit Rebello (1997), "A heuristic branch-and-price approach for the airline crew pairing problem." *preprint*.

Verhave, Martine MC (2015), "Column generation with a resource constrained shortest path algorithm applied to train crew scheduling."

Villeneuve, Daniel and Guy Desaulniers (2005), "The shortest path problem with forbidden paths." *European Journal of Operational Research*, 165, 97–107.

Wolsey, Laurence A (1998), *Integer programming*. Wiley.

Yaghini, Masoud, Mohammad Karimi, and Mohadeseh Rahbar (2015), "A set covering approach for multi-depot train driver scheduling." *Journal of Combinatorial Optimization*, 29, 636–654.

Zhou, Wenliang and Hualiang Teng (2016), "Simultaneous passenger train routing and timetabling using an efficient train-based lagrangian relaxation decomposition." *Transportation Research Part B: Methodological*, 94, 409–439.

Zhouchun Huang, Qipeng Phil Zheng (2014), "Acceleration and stabilization techniques for column generation.", URL http://www.iems.ucf.edu/qzheng/grpmbr/seminar/HuangZhouchun_Acceleration_of_Column_generation.pdf. Visited on 2017-06-12.