Optimisation-Based Solution Methods for Set Partitioning Models

Rasmussen, Matias Sevel

Publication date:
2011

Optimisation-Based Solution Methods for Set Partitioning Models

Applications in Crew Scheduling

Matias Sevel Rasmussen

Kgs. Lyngby, 2011

Matias Sevel Rasmussen
Optimisation-Based Solution Methods for Set Partitioning Models: Applications in Crew Scheduling
(Optimeringsbaserede løsningsmetoder for set partitioning modeller: Anvendelser i arbejdsplanlægning)
Ph.D. thesis, 2011

ISBN [ISBN]

Department of Management Engineering
Technical University of Denmark
Produktionstorvet, Building 424
DK-2800 Kgs. Lyngby, Denmark
Phone: +45 45 25 48 00, Fax: +45 45 25 48 05
phd@man.dtu.dk
www.man.dtu.dk

Print: Vester Kopi, Virum, Denmark

To Samantha

Summary

The scheduling of crew, i.e. the construction of work schedules for crew members, is often not a trivial task, but a complex puzzle. The task is complicated by rules, restrictions, and preferences. Therefore, manual solutions as well as solutions from standard software packages are not always sufficient with respect to solution quality and solution time. Enhancement of the overall solution quality as well as the solution time can be of vital importance to many organisations. The fields of operations research and mathematical optimisation deal with mathematical modelling of difficult scheduling problems (among other topics). The fields also deal with the development of sophisticated solution methods for these mathematical models.

This thesis describes the set partitioning model which has been widely used for modelling crew scheduling problems. Integer properties for the set partitioning model are shown, and exact and optimisation-based heuristic solution methods for the model are described. All these methods are centered around the well-known column generation technique. Different practical applications of crew scheduling are presented, and some of these applications are considered in detail in four included scientific papers. It is shown how these applications all fit into a generalisation of the set partitioning model. Each of the four papers contribute a novel solution method for the specific application treated in the paper.

Resumé (Danish summary)

Arbejdsplanlægning for medarbejdere, dvs. udarbejdelse af arbejdsplaner for medarbejdere, er ikke altid en let opgave, men snarere et komplekst puslespil. Opgaven kompliceres af regler, restriktioner og præferencer. Manuelle løsninger, såvel som løsninger fundet ved hjælp af standard software, er derfor ikke altid tilstrækkelige med hensyn til løsningskvalitet og løsningstid. For mange organisationer kan forbedringer af løsningskvaliteten og løsningstiden være af afgørende betydning. Fagområderne operationsanalyse og matematisk optimering omhandler matematisk modellering af vanskelige planlægningsproblemer (blandt andre emner). Fagområderne omhandler også udvikling af sofistikerede løsningsmetoder for disse matematiske modeller.

Denne afhandling beskriver set partitioning modellen, som er blevet brugt i vid udstrækning til at modellere arbejdsplanlægningsproblemer. Heltalsegenskaber for set partitioning modellen vises, og eksakte samt optimeringsbaserede heuristiske løsningsmetoder til modellen beskrives. Metoderne er alle centrerede omkring den velkendte søjlegenereringsmetode. Forskellige praktiske anvendelser inden for arbejdsplanlægning bliver præsenteret, og nogle af disse bliver behandlet i detaljer i fire inkluderede videnskabelige artikler. Det bliver vist hvordan disse applikationer alle passer ind i en generaliseret udgave af set partitioning modellen. Hver af de fire artikler bidrager med en ny løsningsmetode for den specifikke applikation som behandles i artiklen.

Preface

This thesis has been submitted to the Department of Management Engineering, Technical University of Denmark in partial fulfilment of the requirements for acquiring the Ph.D. degree. The work has been supervised by Associate Professor Jesper Larsen, Department of Management Engineering, Technical University of Denmark, and co-supervised by Professor David Ryan, Department of Engineering Science, The University of Auckland. David Ryan is also an Adjunct Professor at Department of Management Engineering and Department of Transport, Technical University of Denmark.

The thesis is made up of two parts. The first part forms an introduction to the topics that have been investigated throughout the Ph.D. project. The second part is a compilation of four scientific papers that have been written during the Ph.D. project. The Ph.D. project has been carried out from May 2008 to June 2011.

Kgs. Lyngby, June 2011

Matias Sevel Rasmussen

Papers and Other Work

The papers listed in the following have been produced during the Ph.D. project. The first list contains the most significant papers. These papers are also found as appendices. The second list contains other papers such as technical reports, conference papers, and conference abstracts. The papers on the two lists have overlapping content, as they describe the same projects, though at different stages of the projects. The last list shows theses that have been supervised during the Ph.D. project.

Included papers

- **Paper A:** M. S. Rasmussen, T. Justesen, A. Dohn, and J. Larsen (2011f). "The Home Care Crew Scheduling Problem: Preference-Based Visit Clustering and Temporal Dependencies". *European Journal of Operational Research*. (Accepted for publication)
- **Paper B:** A. Dohn, M. S. Rasmussen, and J. Larsen (2011). "The Vehicle Routing Problem with Time Windows and Temporal Dependencies". *Networks*. (Accepted for publication)
- **Paper C:** M. S. Rasmussen, R. M. Lusby, D. M. Ryan, and J. Larsen (2011d). "Subsequence Generation for the Airline Crew Pairing Problem". *Transportation Research Part E*. (Submitted)
- **Paper D:** M. S. Rasmussen, R. M. Lusby, D. M. Ryan, and J. Larsen (2011b). "An IP Framework for the Crew Pairing Problem using Subsequence Generation". *Journal of the Operational Research Society*. (Submitted)

Other papers, technical reports, and conference abstracts

- **Journal paper:** A. Dohn, M. S. Rasmussen, T. Justesen, and J. Larsen (2008a). "The Home Care Crew Scheduling Problem". *Orbit* 13, pp. 19–23
- **Technical report:** M. S. Rasmussen, T. Justesen, A. Dohn, and J. Larsen (2010c). *The Home Care Crew Scheduling Problem: Preference-Based Visit Clustering and Temporal Dependencies*. Tech. rep. Department of Management Engineering, Technical University of Denmark
- **Technical report:** A. Dohn, M. S. Rasmussen, and J. Larsen (2009c). *The Vehicle Routing Problem with Time Windows and Temporal Dependencies*. Tech. rep. Department of Management Engineering, Technical University of Denmark
- **Technical report:** M. S. Rasmussen, R. M. Lusby, D. M. Ryan, and J. Larsen (2011e). *Subsequence Generation for the Airline Crew Pairing Problem*. Tech. rep. Department of Management Engineering, Technical University of Denmark
- **Technical report:** M. S. Rasmussen, R. M. Lusby, D. M. Ryan, and J. Larsen (2011c). *An IP Framework for the Crew Pairing Problem using Subsequence Generation*. Tech. rep. Department of Management Engineering, Technical University of Denmark
- **Competition paper:** M. S. Rasmussen, R. M. Lusby, D. M. Ryan, and J. Larsen (2011a). *A Subsequence Generation Approach for the Airline Crew Pairing Problem*. AGIFORS Anna Valicek Award 2011. (Submitted)
- **Conference paper:** A. Dohn, M. S. Rasmussen, T. Justesen, and J. Larsen (2008b). "The Home Care Crew Scheduling Problem". In: *ICAOR '08 – Proceedings, 1st International Conference on Applied Operational Research*. Ed. by K. Sheibani. Tadbir Institute for Operational Research, pp. 1–8
- **Conference paper:** M. S. Rasmussen, D. M. Ryan, R. M. Lusby, and J. Larsen (2009a). "Solving the Airline Crew Pairing Problem using Subsequence Generation". In: *Proceedings of the 44th Annual conference of the Operational Research Society of New Zealand*
- **Conference paper:** M. S. Rasmussen, D. M. Ryan, R. M. Lusby, and J. Larsen (2010a). "Solving the Airline Crew Pairing Problem using Subsequence Generation". In: *Proceedings of the 8th International Conference on the Practice and Theory of Automated Timetabling (PATAT)*. Ed. by B. McCollum, E. Burke, and G. White

- **Conference paper:** J. Larsen, A. Dohn, M. S. Rasmussen, and T. Justesen (2010). "The Home Care Crew Scheduling Problem". In: *Proceedings of the 8th International Conference on the Practice and Theory of Automated Timetabling (PATAT)*. Ed. by B. McCollum, E. Burke, and G. White

- **Conference abstract:** M. S. Rasmussen, T. Justesen, A. Dohn, and J. Larsen (2009b). *The Home Care Crew Scheduling Problem*. 3rd Nordic Optimization Symposium (NOS)

- **Conference abstract:** M. S. Rasmussen, A. Dohn, and J. Larsen (2009c). *The Vehicle Routing Problem with Time Windows and Temporal Dependencies*. International workshop in vehicle routing, intermodal transport and related areas (ROUTE2009)

- **Conference abstract:** M. S. Rasmussen, R. M. Lusby, D. M. Ryan, and J. Larsen (2010b). *Solving the Airline Crew Pairing Problem using Subsequence Generation*. 4th Nordic Optimization Symposium (NOS)

Projects supervised

- **Master's thesis:** P. D. Kostka (2009). "Planning of Air Crew Licenses". Master's thesis. Department of Management Engineering, Technical University of Denmark
- **Master's thesis:** C. Schmidt (2009). "Solving the Crew Pairing Problem for Cimber Air". Master's thesis. Department of Management Engineering, Technical University of Denmark
- **Master's thesis:** J. Ahmt and J. S. Sigtenbjerggaard (2010). "A New Approach to the Container Positioning Problem". Master's thesis. Department of Management Engineering, Technical University of Denmark
- **Bachelor's thesis:** K. R. Poulsen (2009). "Optimering af flyoptankning [Optimisation of Aircraft Fueling]". Bachelor's thesis. Department of Management Engineering, Technical University of Denmark
- **Bachelor's thesis:** N.-C. Pedersen (2011). "A Heuristic Approach to the Shortest Path Problem with Resource Constraints". Bachelor's thesis. Department of Management Engineering, Technical University of Denmark

Acknowledgements

Firstly, I would like to thank my main supervisor, Associate Professor Jesper Larsen, for support and guidance through the project period. With his broad knowledge Jesper has often suggested the right directions for my research, when I was in doubt, and he has participated in an uncountable number of ad hoc discussions which have moved me forward in research questions. Also, I thank him for jokingly firing me only three or four times (despite numerous verbal warnings), showing his very tolerant nature. Now that my time at DTU has come to an end, I will miss being fired from time to time.

During my one-year research stay in Auckland I had the chance to work with Professor David Ryan. I thank David for letting me take part in his long and successful campaign on promoting optimisation-based methods. His and Ruth Ryan's hospitality is unparallelled, and I especially enjoyed the research discussions that were forced to take place on Waiheke Island and on Mount Ruapehu. Another Kiwi family to whom I owe great thanks is the Lusby family. Not only did they provide us with excellent wheels during our stay, we were also invited to participate in an unforgettable Kiwi style Christmas with poolside barbecue brunch and all other similar kinds of fantastic, relaxed traditions.

Two persons fit into the noble category of friend-and-colleague: Tor Justesen and Richard Lusby. I am grateful for the good work and the good fun that we have shared. In the future I hope that we will do a minimum of work and have a maximum of fun together.

Tor and Richard plus Charlotte Vilhelmsen have kindly proofread and commented on this thesis for which I am deeply thankful. Also, I would like to

thank Anders Dohn for being a fantastic co-author, and of course everyone at Operations Research at DTU and at Engineering Science in Auckland (hereunder the German settlement in Western Springs) for providing a great work atmosphere. I thank DTU Management Engineering for the Ph.D. grant, and I thank Tuborgfondet for supporting my stay in Auckland.

Furthermore, I would like to thank my family and friends for support, dinners, and general good times. Especially, I thank my younger brother, Anders, for being as close to me (location-wise and figuratively speaking) as when he was four and had the very important position as fire dog in my fire brigade.

Lastly, Samantha, my companion in life. Some time ago, I asked you to write these lines in order to get them right. However, I have not received the text yet, and with the deadline approaching, I have to give it a try myself: Every morning I wake up beside you, I feel fortunate. Truth be told, I dare not seem as unrealistically in love with you, as I am.

Contents

Part I

Theory and Applications

Chapter 1

Introduction

Crew scheduling problems arise in many areas, and sometimes the scheduling of crew is a trivial and uncomplicated task. However, for many crew scheduling problems, there is a large number of rules, regulations, and restrictions that must be taken into account. These complicating factors originate from laws, union agreements, and local agreements (often only existing as tradition). Furthermore, individual preferences of the crew and the clients are important to respect as much as possible in order to maximise crew satisfaction. Within the fields of operations research and mathematical optimisation, crew scheduling problems are described mathematically. When finding solutions to specific problem instances, it can indeed be very difficult to respect all complicating factors, while still maintaining a high solution quality, if a structured, mathematical, and computer-aided approach is not taken.

In many areas of work, employees are both a scarce and an expensive resource. Therefore, good utilisation of crew is of vital importance to many organisations. The potential savings from using sophisticated decision support systems, that can deliver optimal or high quality solutions to crew scheduling problems, are large. One example is from Air New Zealand, where Butchers et al. (2001) report annual savings of NZD 15.7 million since the introduction of a new crew scheduling system. In addition, the schedules produced by the system, were better at respecting crew preferences. Another example is shown by Abbink et al. (2005). They describe how the major Dutch railway operator, Netherlands

Railways, have saved USD 4.8 million per year after the introduction of a crew scheduling system. These examples should motivate the need for continued research within solution methods for crew scheduling problems.

This thesis deals with modelling and finding solutions for selected practical problems arising in crew scheduling. Sometimes it is sufficient to model the given crew scheduling problem and then let a standard solver find solutions to the problems. In many cases, however, this is not viable, simply because the problem instances are too large, and therefore tailored solution methods must be crafted. The tailored solution methods fall in five categories: Heuristics, metaheuristics, approximation algorithms, exact methods, and optimisation-based methods. The focus of this thesis will be on the latter two.

1.1 Thesis organisation

The thesis is divided into two parts. Part I describes the problem setting plus relevant theory and highlights the findings from the collection of papers presented in Part II.

Part I is made up of six chapters in addition to this introductory chapter. Chapter 2 describes the well-known set packing, set covering, and set partitioning models. Integer properties for the models are discussed, and finally a general core model that captures the applications presented Part II. Chapter 3 covers solution methods that have been widely used in the literature for solving problems based on the aforementioned models. The covered solution methods are a foundation for the solution methods presented in the papers of Part II. Chapter 4 introduces crew scheduling in general terms and a selection of important crew scheduling applications. Some of these applications have been investigated in detail in the papers of Part II. In Chapter 5 we give brief introductions to the papers of Part II. Additional computational experiments, that have not been described in the papers, are reported in Chapter 6. Finally, in Chapter 7, we will give a conclusion, highlight our contributions, and point out particularly interesting directions for future research.

Part II is a collection of four scientific papers. These papers are the outcomes of the work done during the Ph.D. project. All of the papers have been submitted to international journals, and two of the papers have been accepted for publication at the time of writing. Changes are made to the layout of the papers in order to give this thesis a uniform look, but the content is exactly the same as the content submitted to the journals.

Chapter 2

Set Partitioning Extensions and Properties

We begin this chapter with an introduction to three important and well-known models: set packing, set covering, and set partitioning. The three models are related in the sense that they are concerned with finding an optimal family of subsets of elements from a set, and they are used as the basis for the applications in this thesis. A comprehensive survey on especially the theory of the set packing, set covering, and set partitioning models can be found in Balas and Padberg (1976), and in Vemuganti (1998) a survey with emphasis on applications of the models can be found.

We state a model that unifies the three problems. We present four classes of matrices that can appear as input to the problems. All four classes have the property, that one can automatically solve the problem by solving a relaxed and computationally easier version of the problem. At last we extend the unified model to also cover additional types of constraints.

2.1 Set packing

The first model we will present is the *set packing problem* in which a pairwise disjoint family of subsets from a given set must be found, such that the added profit of the subsets is maximised. The set packing problem is well-studied in the literature and has been used to model many practical applications. For instance: Rönnqvist (1995) develops a set packing model for a cutting stock problem and solves it with Lagrangian relaxation combined with subgradient optimisation. Lusby et al. (2009) give a survey of models and methods for railway track allocation, including formulations that rely on the set packing model. Avella et al. (2006) model the Intelligent Tourist Problem (scheduling of a tourist's visits with maximum satisfaction) with a set packing model and solve it with a linear programming based heuristic. Rossi and Smriglio (2001) model the Ground Holding Problem from airports and solve it with a branch-and-cut algorithm.

The set packing problem is formally defined below.

Definition 2.1 *[Set packing problem] Let $S = \{1, \ldots, m\}$ be a set of m elements, and let $\{S_1, \ldots, S_n\}$ be a family of n subsets of S, that is, $S_j \subseteq S$ for all $j \in \{1, \ldots, n\}$. A* packing *$Q = \{S_{j_1}, \ldots, S_{j_k}\}$ is a subfamily of $\{S_1, \ldots, S_n\}$, where the elements in Q are pairwise disjoint, i.e. $S_{j_s} \cap S_{j_t} = \emptyset$ for all $s, t \in \{1, \ldots, k\}$ with $s \neq t$. Let c_j be the profit associated with subset S_j for all $j \in \{1, \ldots, n\}$. The* set packing problem *is then to find a packing with maximum profit.*

The set packing problem with unit profits is $\mathcal{NP}$-hard (Garey and Johnson, 1979; Crescenzi and Kann, 1997), and therefore the more general set packing problem defined above is also $\mathcal{NP}$-hard. It should be noted that there always exists at least one feasible solution, namely $Q = \emptyset$.

Let $\boldsymbol{A}$ be an $m \times n$ zero-one matrix with entries a_{ij} for all $i \in \{1, \ldots, m\}$ and $j \in \{1, \ldots, n\}$ defined by

$$a_{ij} = \begin{cases} 1 & \text{if } i \in S_j \\ 0 & \text{otherwise} \end{cases}, \tag{2.1}$$

and let x_j for all $j \in \{1, \ldots, n\}$ be a binary decision variable defined by

$$x_j = \begin{cases} 1 & \text{if } S_j \in Q \\ 0 & \text{otherwise} \end{cases}, \tag{2.2}$$

and let $\boldsymbol{c}$ be the vector of profits c_j for all $j \in \{1, \ldots, n\}$. Now we can write the

set packing problem as the following binary integer programme:

$$\begin{aligned} &\text{maximise} && \boldsymbol{c}^\top \boldsymbol{x} && (2.3)\\ &\text{subject to} && \boldsymbol{A}\boldsymbol{x} \leq \mathbf{1} && (2.4)\\ & && \boldsymbol{x} \in \{0,1\}^n \ . && (2.5) \end{aligned}$$

The jth column $\boldsymbol{A}_j = (a_{1j}, \ldots, a_{mj})^\top$ of the $\boldsymbol{A}$ matrix is the column representation of the subset S_j. Constraints (2.4) enforce that the subsets are pairwise disjoint.

2.2 Set covering

A problem related to the set packing problem is the *set covering problem*. In this problem a minimum cost family of subsets from a given set, where the subsets cover the whole set, must be found. The literature has many practical applications of set covering models. Huisman (2007) models rail crew re-scheduling with a set covering model and devises a column generation-based solution algorithm. In some periods of time, rail tracks are out of service due to maintenance, and this affects the timetable, the rolling stock schedules and hence also the crew schedules. The crew re-scheduling is due to planned maintenance and not a recovery due to an unforeseen disruption. Real-world instances from Netherlands Railways are used for testing. Rubin (1973) uses a set covering formulation to solve the airline crew pairing problem.

The set covering problem is formally defined below.

Definition 2.2 *[Set covering problem] Let $S = \{1, \ldots, m\}$ be a set of m elements, and let $\{S_1, \ldots, S_n\}$ be a family of n subsets of S, that is, $S_j \subseteq S$ for all $j \in \{1, \ldots, n\}$. A* cover *$Q = \{S_{j_1}, \ldots, S_{j_k}\}$ is a subfamily of $\{S_1, \ldots, S_n\}$, where Q covers all elements of S, that is $\bigcup_{s=1}^{k} S_{j_s} = S$. Let c_j be the cost associated with subset S_j for all $j \in \{1, \ldots, n\}$. The* set covering problem *is then to find a cover with minimum cost.*

The set covering problem with unit costs is $\mathcal{NP}$-hard (Garey and Johnson, 1979; Crescenzi and Kann, 1997), and therefore the more general set covering problem defined above is also $\mathcal{NP}$-hard. It should be noted that, if $\bigcup_{j=1}^{n} S_j = S$, there always exists at least one feasible solution, namely $Q = \{S_1, \ldots, S_n\}$.

Let again $\boldsymbol{A}$ be an $m \times n$ zero-one matrix with entries a_{ij} for all $i \in \{1, \ldots, m\}$ and $j \in \{1, \ldots, n\}$ defined by Equation (2.1), and let x_j for all $j \in \{1, \ldots, n\}$ be a binary decision variable defined by Equation (2.2). Let $\boldsymbol{c}$ be the vector of

costs c_j for all $j \in \{1, \ldots, n\}$. The set covering problem can now be written as the following binary integer programme:

$$\begin{aligned} &\text{minimise} && \boldsymbol{c}^\top \boldsymbol{x} && (2.6)\\ &\text{subject to} && \boldsymbol{A}\boldsymbol{x} \geq \boldsymbol{1} && (2.7)\\ & && \boldsymbol{x} \in \{0,1\}^n \ . && (2.8) \end{aligned}$$

Still, the jth column $\boldsymbol{A}_j = (a_{1j}, \ldots, a_{mj})^\top$ of the $\boldsymbol{A}$ matrix is the column representation of the subset S_j. Constraints (2.7) enforce that the subsets cover all elements of S.

2.3 Set partitioning

The basic version of the core problem in this thesis can now be presented: The *set partitioning problem*. In this problem a minimum cost family of pairwise disjoint subsets from a given set, where the subsets cover the whole set, must be found. A lot of attention has been given to the set partitioning problem in the literature. Balinski and Quandt (1964) formulate a truck delivery problem as a set partitioning model and solve the model by a cutting plane algorithm where the columns are generated a priori. Garfinkel and Nemhauser (1969) introduce the set partitioning problem as a set covering problem with equality constraints and give a solution algorithm. Desaulniers et al. (1997) use a set partitioning model solved with column generation to do crew scheduling for Air France. In Rezanova and Ryan (2010) a recovery problem for train driver duties is modelled by a set partitioning model and solved in a column generation-based framework. Brønmo et al. (2010) use a set partitioning model and column generation to solve a short-term ship scheduling problem where the cargo sizes are flexible. Cargoes have a given profit rate, and some cargoes must be carried out while others can be negotiated on the spot market. A schedule, i.e. a sequence of ports, for each ship must be build, and the objective is to maximise profit.

The set partitioning problem is formally defined below.

Definition 2.3 *[Set partitioning problem] Let $S = \{1, \ldots, m\}$ be a set of m elements, and let $\{S_1, \ldots, S_n\}$ be a family of n subsets of S, that is, $S_j \subseteq S$ for all $j \in \{1, \ldots, n\}$. A* partitioning *$Q = \{S_{j_1}, \ldots, S_{j_k}\}$ is a subfamily of $\{S_1, \ldots, S_n\}$, where the elements in Q are pairwise disjoint, i.e. $S_{j_s} \cap S_{j_t} = \emptyset$ for all $s, t \in \{1, \ldots, k\}$ with $s \neq t$, and also Q covers all elements of S, that is $\bigcup_{s=1}^{k} S_{j_s} = S$. Let c_j be the cost associated with subset S_j for all $j \in \{1, \ldots, n\}$. The* set partitioning problem *is then to find a partitioning with minimum cost.*

As before, let $\boldsymbol{A}$ be a $m \times n$ zero-one matrix with entries a_{ij} for all $i \in \{1, \ldots, m\}$ and $j \in \{1, \ldots, n\}$ defined by Equation (2.1), and let x_j for all $j \in \{1, \ldots, n\}$ be a binary decision variable defined by Equation (2.2). Let $\boldsymbol{c}$ be the vector of costs c_j for all $j \in \{1, \ldots, n\}$. The set partitioning problem can now be written as the following binary integer programme:

$$\begin{aligned} &\text{minimise} && \boldsymbol{c}^\top \boldsymbol{x} && (2.9)\\ &\text{subject to} && \boldsymbol{A}\boldsymbol{x} = \boldsymbol{1} && (2.10)\\ & && \boldsymbol{x} \in \{0,1\}^n \ . && (2.11) \end{aligned}$$

Again, the jth column $\boldsymbol{A}_j = (a_{1j}, \ldots, a_{mj})^\top$ of the $\boldsymbol{A}$ matrix is the column representation of the subset S_j. Constraints (2.10) enforce that the subsets are pairwise disjoint and cover all elements of S, i.e. every element of S is in exactly one of the subsets in the solution.

The set partitioning problem is well-known to be $\mathcal{NP}$-hard, and we will here very briefly sketch the proof. The core of the proof is to reduce set packing to set partitioning. For an introduction to $\mathcal{NP}$-completeness proofs we refer to Garey and Johnson (1979), Cormen et al. (2001), and Cook et al. (1998). Consider a general set packing problem instance. Now, the original family of subsets is augmented with zero-cost singleton subsets for each of the elements. If we multiply all costs by minus one, we can then solve the set packing instance as a set partitioning instance. As the set packing problem as described earlier is $\mathcal{NP}$-hard, we can conclude that also the set partitioning problem is $\mathcal{NP}$-hard.

2.4 A unified model

The set partitioning problem can be transformed to both the set packing problem and the set covering problem. These transformations are well-known, see for instance Balas and Padberg (1976), Darby-Dowman and Mitra (1985), and Vemuganti (1998).

As we have shown previously, the set packing problem can be transformed to the set partitioning problem (see Section 2.3), we can now classify the two problems as equivalent problems. This makes sense intuitively: In the set packing problem and the set partitioning problem each of the constraints imposes that there must be at most one or exactly one of the many variables, that covers the constraint, at value one. In the set covering problem each of the constraints enforce that there must be at least one of the many variables, that covers the constraint, at value one. So, set packing and set partitioning can be said to be more restricted than set covering.

The set packing problem, the set covering problem, and the set partitioning problem can be combined in a unified model that covers all three problems. The idea behind the unified model (see also Darby-Dowman and Mitra (1985)) is to allow undercovering (thereby yielding set packing) or overcovering (thereby yielding set covering) of the set partitioning constraints.

Let $\boldsymbol{A}$ be an $m \times n$ zero-one matrix and let $\boldsymbol{c} \in \mathbb{R}^n$ be an n-vector of costs. Let $\boldsymbol{y}^{\text{uc}} \in \{0,1\}^m$ be a vector of decision variables controlling whether or not the ith constraint is undercovered. If the ith constraint is undercovered, then $y_i^{\text{uc}} = 1$, otherwise $y_i^{\text{uc}} = 0$. Let $\boldsymbol{y}^{\text{oc}} \in \mathbb{Z}_+^m$ be a vector of decision variables controlling how much the ith constraint is overcovered. If the ith constraint is overcovered, then $y_i^{\text{oc}} = o_i$, where o_i is the number of times it is overcovered, otherwise $y_i^{\text{oc}} = 0$. Let $\boldsymbol{c}^{\text{uc}} \in \mathbb{R}_+^m$ be an m-vector of undercoverage costs, and let $\boldsymbol{c}^{\text{oc}} \in \mathbb{R}_+^m$ be an m-vector of overcoverage costs. Let $\boldsymbol{x} \in \{0,1\}^n$ be a vector of decision variables controlling whether or not the column j is selected in the solution. Now we can write the *unified set partitioning model* as:

$$\begin{aligned}
&\text{minimise} && \boldsymbol{c}^\top \boldsymbol{x} + {\boldsymbol{c}^{\text{uc}}}^\top \boldsymbol{y}^{\text{uc}} + {\boldsymbol{c}^{\text{oc}}}^\top \boldsymbol{y}^{\text{oc}} && (2.12)\\
&\text{subject to} && \boldsymbol{A}\boldsymbol{x} + \boldsymbol{I}\boldsymbol{y}^{\text{uc}} - \boldsymbol{I}\boldsymbol{y}^{\text{oc}} = \boldsymbol{1} && (2.13)\\
& && \boldsymbol{x} \in \{0,1\}^n && (2.14)\\
& && \boldsymbol{y}^{\text{uc}} \in \{0,1\}^m && (2.15)\\
& && \boldsymbol{y}^{\text{oc}} \in \mathbb{Z}_+^m\ . && (2.16)
\end{aligned}$$

Here $\boldsymbol{I}$ denotes the $m \times m$ identity matrix. We now show how to capture the set packing, the set covering, and the set partitioning problem in the unified model:

- **Set packing:** Let us call the cost vector from the set packing problem (2.3)–(2.5) for $\boldsymbol{p}$. Then, in order to solve the set packing problem in the unified set partitioning model, define the costs by $c_j = -p_j$ for all $j \in \{1,\ldots,n\}$, and set $\boldsymbol{c}^{\text{uc}} = \boldsymbol{0}$ to allow undercoverage. Furthermore, set $c_i^{\text{oc}} = M$ for all $i \in \{1,\ldots,m\}$, where M is a sufficiently large positive number, to disallow overcoverage. Sufficiently large would be fulfilled by $M > \sum_{j=1}^n \max\{|c_j|\}$. Then the unified model will have the same optimal solutions as the set packing model.
- **Set covering:** In order to solve the set covering problem (2.6)–(2.8) in the unified set partitioning model set $\boldsymbol{c}^{\text{oc}} = \boldsymbol{0}$ to allow overcoverage. Furthermore, set $c_i^{\text{uc}} = M$ for all $i \in \{1,\ldots,m\}$, where M again is a sufficiently large positive number, to disallow undercoverage. Sufficiently large would be fulfilled as before. Then the unified model will have the same optimal solutions as the set covering model, in the cases where the set covering problem has feasible solutions.

- **Set partitioning:** In order to solve the set partitioning problem (2.9)–(2.11) in the unified set partitioning model set $c_i^{\text{uc}} = c_i^{\text{oc}} = M$ for all $i \in \{1, \ldots, m\}$, where M is defined as before, to disallow both under- and overcoverage. Then the unified model will have the same optimal solutions as the set partitioning model, in the cases where the set partitioning problem has feasible solutions.

In many practical applications where a set partitioning model is used, there might not exist feasible solutions to the model. In such cases, a solution that is just close to set partitioning feasible could still be interesting. Such a solution would overcover or undercover some constraints in the problem. This situation could, however, not be captured by the pure set partitioning model, but it can be captured in the unified model. In the unified model, one can put a specific price on over- or undercoverage of each of the constraint. The unified model also captures situations where over- or undercoverage of a few constraints is more preferable than a costly exact partition. Examples of the use of over- and undercoverage are Desaulniers et al. (1997) and Darby-Dowman and Mitra (1985) as well as Paper A, Paper C, and Paper D.

Since the set partitioning problem as described earlier is $\mathcal{NP}$-hard and contained in the unified set partitioning problem, we can conclude that also the unified set partitioning problem is $\mathcal{NP}$-hard.

2.5 The integer property

Let $\boldsymbol{A}$ be an $m \times n$ integer matrix, and let $\boldsymbol{b}$ be an integer n-vector. Let $\boldsymbol{c} \in \mathbb{R}_+^n$ be an n-vector of costs, and let $\boldsymbol{x}$ be an n-vector of decision variables. Consider the integer programme given by

$$\begin{aligned} &\text{minimise} && \boldsymbol{c}^\top \boldsymbol{x} && (2.17)\\ &\text{subject to} && \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} && (2.18)\\ & && \boldsymbol{x} \in \mathbb{Z}_+^n \ . && (2.19) \end{aligned}$$

We will now investigate in which situations, we can get an optimal solution to (2.17)–(2.19) by solving the LP relaxation, i.e. the linear programme

$$\begin{aligned} &\text{minimise} && \boldsymbol{c}^\top \boldsymbol{x} && (2.20)\\ &\text{subject to} && \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} && (2.21)\\ & && \boldsymbol{x} \geq \boldsymbol{0} \ . && (2.22) \end{aligned}$$

where (2.19) is substituted with $\boldsymbol{x} \geq \boldsymbol{0}$. Being able to get an optimal IP solution by solving an LP relaxation is very attractive, as finding LP solutions is computationally much faster than finding IP solutions.

Definition 2.4 *[Integer property] The polyhedron*

$$\{\boldsymbol{x} \in \mathbb{R}^n : \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}, \boldsymbol{x} \geq \boldsymbol{0}\}$$

is said to have the integer property *if it has only integral extreme points.*

In the following we will present four classes of matrices that all ensure that the accompanying polyhedrons have the integer property.

2.5.1 Totally unimodular matrices and unique subsequence

Let us first establish a sufficient condition that enables the LP relaxation to solve also the IP. A basic feasible solution when solving an LP (2.20)–(2.22) looks like (see for instance Hillier and Lieberman (2005) and Andersen (2006)):

$$\boldsymbol{x} = \begin{pmatrix} \boldsymbol{x}_B \\ \boldsymbol{x}_N \end{pmatrix} = \begin{pmatrix} \boldsymbol{B}^{-1}\boldsymbol{b} \\ \boldsymbol{0} \end{pmatrix}$$

where $\boldsymbol{x}_B$ are the basic variables, $\boldsymbol{x}_N$ are the non-basic variables, and $\boldsymbol{B}$ is an $m \times m$ nonsingular submatrix of $\boldsymbol{A}$.

From Messer (1994), we have that $\boldsymbol{B}^{-1} = \text{adj}(\boldsymbol{B})/\det(\boldsymbol{B})$, where $\text{adj}(\boldsymbol{B})$ and $\det(\boldsymbol{B})$ is the adjoint matrix respectively the determinant of $\boldsymbol{B}$. As $\boldsymbol{B}$ is a submatrix of the integer matrix $\boldsymbol{A}$, and as the entries of $\text{adj}(\boldsymbol{B})$ are all products of the entries in $\boldsymbol{B}$, then $\text{adj}(\boldsymbol{B})$ is integral. Now, $\boldsymbol{b}$ is also integral, so if $\det(\boldsymbol{B}) = \pm 1$, then $\boldsymbol{B}^{-1}\boldsymbol{b}$ is integral. Hence, we have proved a sufficient condition for the integer property, see also Wolsey (1998):

Proposition 2.1 *[LP solves IP] If the determinant of the optimal basis $\boldsymbol{B}$ is* 1 *or* -1*, then the LP relaxation solves the IP.*

Based on this, we can now define a class of matrices that impose the integer property. The class is called *totally unimodular* matrices and was first introduced in Hoffman and Kruskal (1956) and is described in, for instance, Wolsey (1998) and Nemhauser and Wolsey (1988).

Definition 2.5 *[Totally unimodular matrices] A matrix $\boldsymbol{A}$ is said to be* totally unimodular *if every square submatrix of $\boldsymbol{A}$ has determinant* 1*,* -1*, or* 0*.*

$$\begin{pmatrix} \boxed{1} & \boxed{1} & 1 & 1 & \boxed{1} & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \boxed{1} & 1 & 0 \\ \boxed{1} & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & \boxed{1} & 1 & 0 & 0 & 1 & 0 \end{pmatrix}$$

Figure 2.1: *Subsequence set for row 1.*

It follows immediately from this definition that if the constraint matrix $\boldsymbol{A}$ from (2.20)–(2.22) is totally unimodular, then any optimal basis $\boldsymbol{B}$ fulfills the sufficient condition (Proposition 2.1), because the optimal basis $\boldsymbol{B}$ is non-singular (i.e. $\det(\boldsymbol{B}) \neq 0$).

It also follows immediately from the definition that the entries a_{ij} of a totally unimodular matrix $\boldsymbol{A}$, must have $a_{ij} \in \{-1, 0, 1\}$, since also all 1×1-submatrices must have determinant 1, -1, or 0.

Since the set partitioning, set packing and set covering problems are all defined for a zero-one constraint matrix $\boldsymbol{A}$, we will in the following restrict ourselves to only consider zero-one matrices, i.e. matrices where the entries $a_{ij} \in \{0, 1\}$.

Another means to characterise matrices is the notion of *subsequence set* or *subsequences* for a given row s, that is the pairs of rows (s, t) where a one in s is followed (meaning that there are only zeros between them in a column) by a one in t.

Definition 2.6 *[Subsequence set] For an $m \times n$ zero-one matrix $\boldsymbol{A}$ with entries a_{ij}, the* subsequence set *or the* subsequences*, $\mathcal{S}(s)$, for any row s is given by*

$$\mathcal{S}(s) = \{(s, t) : [\exists j \in \{1, \ldots, n\} : a_{sj} = 1, a_{ij} = 0 \text{ for } s < i < t, a_{tj} = 1]\} \ ,$$

and the subsequence count*, $SC(s)$, for any row s is given by*

$$SC(s) = |\mathcal{S}(s)| \ .$$

Figure 2.1 shows an example where the subsequence set for row 1 is $\mathcal{S}(1) = \{(1, 3), (1, 4), (1, 6)\}$, yielding the subsequence count $\mathrm{SC}(1) = 3$. We can now define *unique subsequence* matrices:

Definition 2.7 *[Unique subsequence matrices] Zero-one $m \times n$ matrices where the subsequence count $SC(s) \leq 1$ for all $s \in \{1, \ldots, m\}$ are said to have* unique subsequence*.*

Ryan and Falkner (1988) prove an important property for unique subsequence matrices:

Proposition 2.2 *[Unique subsequence and totally unimodularity] Let* $\boldsymbol{A}$ *be an* $m \times n$ *zero-one matrix. If* $\boldsymbol{A}$ *has unique subsequence, then* $\boldsymbol{A}$ *is totally unimodular.*

Therefore, when $\boldsymbol{A}$ has unique subsequence, then the polyhedron $\{\boldsymbol{x} \in \mathbb{R}^n : \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}, \boldsymbol{x} \geq \boldsymbol{0}\}$ has the integer property.

2.5.2 Balanced and perfect matrices

An odd order two-cycle square matrix is a square matrix of odd dimension, where there are exactly two ones in each row and each column, i.e. every row and column sum is two. The smallest example of such a matrix is

$$\begin{pmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix} .$$

Introduced by Berge (1972) is the class of *balanced* matrices:

Definition 2.8 *[Balanced matrices] A zero-one matrix is said to be* balanced*, if it does not contain any odd order two-cycle square submatrices.*

Balanced matrices impose the integer property on the set partitioning polyhedron when $\boldsymbol{b} = \boldsymbol{1}$, see Ryan and Falkner (1988).

Another important class of matrices with respect to the integer property, is the class of *perfect* matrices, introduced by Padberg (1974). This class contains the class of balanced matrices, but allows the odd order two-cycle square submatrices, that cannot exist in a balanced matrix. The odd order two-cycles are then "tamed" by a row outside the submatrix. The definition from Padberg (1974) is:

Definition 2.9 *[Perfect matrices] A zero-one matrix is said to be* perfect*, if the set packing polytope*

$$\{\boldsymbol{x} \in \mathbb{R}^n : \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{1}, \boldsymbol{x} \geq \boldsymbol{0}\} \tag{2.23}$$

has only integral extreme points.

Hence, if $\boldsymbol{A}$ is perfect then also the set partitioning polytope

$$\{\boldsymbol{x} \in \mathbb{R}^n : \boldsymbol{A}\boldsymbol{x} = \mathbf{1}, \boldsymbol{x} \geq \mathbf{0}\}$$

will have only integral extreme points (assuming there are other extreme points than $\mathbf{0}$), and therefore, per definition, perfect matrices impose the integer property for $\boldsymbol{b} = \mathbf{1}$.

Consider the matrix

$$\boldsymbol{A}_1 = \begin{pmatrix} 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{pmatrix} .$$

The matrix has an odd order two-cycle square submatrix, so it is not balanced. Let us use $\boldsymbol{A}_1$ in the polytope (2.23). Then the extreme points are $(0,0,0,0)^\top$, $(0, \frac{1}{2}, \frac{1}{2}, \frac{1}{2})^\top$, and $(1,0,1,0)^\top$. Now, if the constraint $x_2 + x_3 + x_4 \leq 1$ is added, we get

$$\boldsymbol{A}_2 = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{pmatrix} ,$$

and the extreme points are then restricted to $(0,0,0,0)^\top$ and $(1,0,1,0)^\top$. The matrix $\boldsymbol{A}_2$ is therefore perfect.

The example tells us something about how perfect matrices can be recognised. We will need the following definition from Padberg (1974):

Definition 2.10 *[Property $\Pi_{\beta,k}$] An $m \times k$ zero-one matrix $\boldsymbol{A}$ with $k \leq m$ is said to have the* property $\Pi_{\beta,k}$ *if*

(i) *$\boldsymbol{A}$ contains a $k \times k$ nonsingular submatrix $\boldsymbol{B}_k$ with all row and column sums equal to β.*

(ii) *Each row of $\boldsymbol{A}$ which is not in $\boldsymbol{B}_k$ is either componentwise equal to a row of $\boldsymbol{B}_k$ or has row sum strictly less than β.*

Now perfect matrices can be recognised by looking for matrices with that property. This is formalised in the following proposition by Padberg (1974).

Proposition 2.3 *[Perfect matrix recognition] Let $\boldsymbol{A}$ be an $m \times n$ zero-one matrix. The following conditions are equivalent:*

(i) *$\boldsymbol{A}$ is perfect.*

(ii) For $\beta \geq 2$ and $3 \leq k \leq m$, $\boldsymbol{A}$ does not contain any $m \times k$ submatrix $\boldsymbol{A}_k$ with the property $\Pi_{\beta,k}$.

We can put this proposition to immediate use and derive a result about matrices with a special type of constraint, namely the so-called *generalised upper bound* constraint.

Definition 2.11 *[Generalised upper bound constraint] Let $\boldsymbol{A}$ be an $m \times n$ zero-one matrix. A* generalised upper bound (GUB) *set packing constraint for $\boldsymbol{A}$ is a constraint of the form $\mathbf{1}^\top \boldsymbol{x} \leq 1$, where $\mathbf{1}$ is an n-vector. A GUB set partitioning constraint is a constraint of the form $\mathbf{1}^\top \boldsymbol{x} = 1$.*

Now, a matrix with a GUB constraint will prevent the existence of submatrices with property $\Pi_{\beta,k}$. The GUB row will always have a row sum of at least β, and hence it must be a part of $\boldsymbol{B}_k$ (or componentwise equal to a row of $\boldsymbol{B}_k$) for Definition 2.10 (ii) to be respected. In that case $\beta = k$ and then $\boldsymbol{B}_k$ is a unit square matrix and thus singular, why Definition 2.10 (i) is not respected. Hence, by Proposition 2.3, we have showed the following corollary.

Corollary 2.1 *[GUB constraint and perfect matrix] Let $\boldsymbol{A}$ be a zero-one matrix with a GUB set packing or set partitioning constraint. Then $\boldsymbol{A}$ is a perfect matrix.*

The corollary is a fast means to identify perfect matrices, which in turn gives the integer property.

The last class of matrices, that are interesting to mention with respect to the integer property, is that of *ideal* matrices, introduced by Lehman (1979). We will not go into details about this class of matrices, but just state its definition for completeness.

Definition 2.12 *[Ideal matrices] A zero-one matrix is said to be* ideal*, if the set covering polyhedron*

$$\{\boldsymbol{x} \in \mathbb{R}^n : \boldsymbol{A}\boldsymbol{x} \geq \mathbf{1}, \boldsymbol{x} \geq \mathbf{0}\}$$

has only integral extreme points.

From the definition it can be seen that the class of ideal matrices is the set covering counterpart to perfect matrices.

In order to give an overview, we illustrate the set relations between the presented classes of matrices which impose the integer property on Figure 2.2. The relations are shown in Berge (1972), Padberg (1974), Ryan and Falkner (1988), and Conforti et al. (2001).

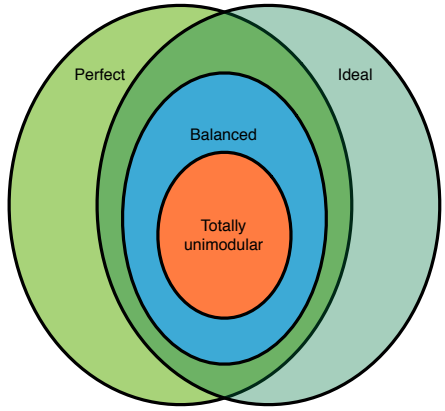


Figure 2.2: *Set relations between the different classes of matrices which impose the integer property.*

2.6 Generalised set partitioning

The pure set partitioning model presented in Section 2.3 as well as the unified set partitioning model presented in Section 2.4 are sometimes not general enough to capture the requirements from a practical application. In such cases, a *generalised set partitioning problem* is used. The generalised set partitioning problem allows the entries of the constraint matrix as well as the right hand side to be different from zero and one. The problem also allows the constraints to be packing, covering and/or partitioning constraints. The generalised set partitioning problem is also widely used in the literature. For instance: Nissen and Haase (2006) have a generalised set partitioning model for airline crew re-scheduling and solve it using column generation in a branch-and-price framework. The goal is to create new crew schedules fast and with minimal deviation from the original schedule. Test instances are generated based on the flight schedule from a major European carrier. Avella et al. (2004) use a generalised set partitioning model for solving a fuel delivery problem with column generation. A company with one warehouse and a fleet of trucks with different capacities delivers different types of fuel to fuel pumps located at clients in an urban area. The trucks are routed and scheduled to their client visits in a daily planning horizon.

We will define the generalised set partitioning problem below with offset in the unified set partitioning integer programme (2.12)–(2.16).

Let $\boldsymbol{A}$ be an $m \times n$-matrix where the entries $a_{ij} \in \mathbb{Z}$ for all $i \in \{1, \ldots, m\}$ and $j \in \{1, \ldots, n\}$. Let $\boldsymbol{c} \in \mathbb{R}^n$ be an n-vector of costs. Let $\boldsymbol{b} \in \mathbb{Z}^m$ be an

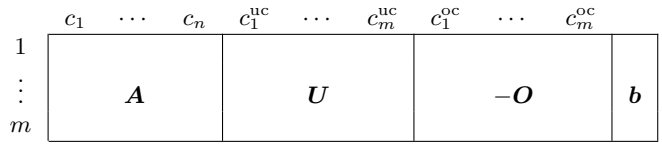


Figure 2.3: *Constraint matrix with right hand side for the generalised set partitioning model.*

m-vector of right hand sides to the constraints. Let $\boldsymbol{y}^{\text{uc}} \in \mathbb{R}_+^m$ be a vector of decision variables controlling how much the ith constraint is undercovered. Let $\boldsymbol{y}^{\text{oc}} \in \mathbb{R}_+^m$ be a vector of decision variables controlling how much the ith constraint is overcovered. Let $\boldsymbol{c}^{\text{uc}} \in \mathbb{R}_+^m$ be an m-vector of undercoverage costs, and let $\boldsymbol{c}^{\text{oc}} \in \mathbb{R}_+^m$ be an m-vector of overcoverage costs. Let $\boldsymbol{x} \in \{0,1\}^n$ be a vector of decision variables controlling whether or not the column j is selected in the solution. Now we can write the *generalised set partitioning model* as:

$$\begin{aligned}
&\text{minimise} && \boldsymbol{c}^\top \boldsymbol{x} + \boldsymbol{c}^{\text{uc}^\top} \boldsymbol{y}^{\text{uc}} + \boldsymbol{c}^{\text{oc}^\top} \boldsymbol{y}^{\text{oc}} && (2.24)\\
&\text{subject to} && \boldsymbol{A}\boldsymbol{x} + \boldsymbol{U}\boldsymbol{y}^{\text{uc}} - \boldsymbol{O}\boldsymbol{y}^{\text{oc}} = \boldsymbol{b} && (2.25)\\
& && \boldsymbol{x} \in \{0,1\}^n && (2.26)\\
& && \boldsymbol{y}^{\text{uc}}, \boldsymbol{y}^{\text{oc}} \in \mathbb{R}_+^m \ . && (2.27)
\end{aligned}$$

Here, $\boldsymbol{U}$ and $\boldsymbol{O}$ are the $m \times m$ coefficient matrices for undercoverage respectively overcoverage, where the entries $u_{ij}, o_{ij} \in \mathbb{Z}$ for all $i, j \in \{1, \ldots, m\}$. Note that $\boldsymbol{U}$ and $\boldsymbol{O}$ are not necessarily identity matrices, because that would give a less general model unable of capturing some important applications, see Chapter 4.

As the set partitioning problem is $\mathcal{NP}$-hard (see Section 2.3) and contained in the generalised set partitioning problem, we can conclude that also the generalised set partitioning problem is $\mathcal{NP}$-hard.

Constraints (2.25) can be rewritten to

$$(\boldsymbol{A} \mid \boldsymbol{U} \mid -\boldsymbol{O}) \cdot \begin{pmatrix} \boldsymbol{x} \\ \boldsymbol{y}^{\text{uc}} \\ \boldsymbol{y}^{\text{oc}} \end{pmatrix} = \boldsymbol{b}$$

and the constraint matrix $(\boldsymbol{A} \mid \boldsymbol{U} \mid -\boldsymbol{O})$ with its right hand side can then be illustrated as on Figure 2.3. If the matrix multiplications in (2.24)–(2.27) are

written out we get the following integer programme:

$$
\begin{aligned}
\text{minimise} \quad & \sum_{j=1}^{n} c_j x_j + \sum_{i=1}^{m} c_i^{\text{uc}} y_i^{\text{uc}} + \sum_{i=1}^{m} c_i^{\text{oc}} y_i^{\text{oc}} & \\
\text{subject to} \quad & \sum_{j=1}^{n} a_{ij} x_j + \sum_{j=1}^{m} u_{ij} y_j^{\text{uc}} - \sum_{j=1}^{m} o_{ij} y_j^{\text{oc}} = b_i & \forall i \in \{1, \ldots, m\} \\
& x_j \in \{0, 1\} & \forall j \in \{1, \ldots, n\} \\
& y_i^{\text{uc}}, y_i^{\text{oc}} \in \mathbb{Z}_+ \ . & \forall i \in \{1, \ldots, m\}
\end{aligned}
$$

When we are considering a generalised set partitioning problem instead of a pure set partitioning problem, the theoretical results about integer properties from Section 2.5 do not hold any more. However, intuitively, the fewer non-pure set partitioning constraints the constraint matrix has, the more likely it is to maintain integer properties. Considering unique subsequence matrices, the results from Ryan and Falkner (1988) indicate that the closer the matrix is to unique subsequence, the more integer it is likely to be.

Chapter 3

Solution Methods

In this chapter we will go through exact solution methods for optimisation problems. We will present the column generation technique with its master and pricing problem setup for linear programmes, as well as the Dantzig-Wolfe decomposition method. For integer programmes we will describe the branch-and-bound approach and also branch-and-bound with embedded column generation. Finally, we will present optimisation-based heuristic approaches.

3.1 Column generation

Let $\boldsymbol{A}$ be an $m \times n$ matrix, and let $\boldsymbol{b}$ be an m-vector. Let $\boldsymbol{c}$ be the vector of costs, and let $\boldsymbol{x}$ be the solution vector. We assume $n > m$ and $\text{rank}(\boldsymbol{A}) = m$. Consider the linear programme (LP) given below.

$$\begin{aligned} &\text{minimise} && \boldsymbol{c}^\top \boldsymbol{x} && (3.1)\\ &\text{subject to} && \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} && (3.2)\\ & && \boldsymbol{x} \geq \boldsymbol{0}\ . && (3.3) \end{aligned}$$

A linear programme is typically solved by the simplex method, see for instance Hillier and Lieberman (2005), Andersen (2006), and Lasdon (2002).

Suppose that the LP (3.1)–(3.3) has a huge number of columns n, say millions, but only a relatively few number of constraints m, say hundreds. From simplex theory, we know that every basis has m columns, so most of the columns will not be in the basis and very likely never be part of a basis in any of the iterations. Moreover, the calculations and bookkeeping for a huge number of columns is computationally undesirable, and even just the generation of all possible columns can be unachievable in reasonable time. All in all, this suggests that columns should only be generated when needed. The majority of the columns will be non-basic in an optimal solution anyway.

The *column generation* technique for linear programmes is a two-phase method, divided in a *master problem* and a *pricing problem*, which we will explain in the following sections. Column generation was introduced by Gilmore and Gomory (1961) and Gilmore and Gomory (1963). For an introduction to column generation, see, for instance, Desrosiers and Lübbecke (2005), Lasdon (2002), and Dirickx and Jennergren (1979).

3.1.1 Master problem

The master problem is Model (3.1)–(3.3), but as mentioned, explicit pricing over all possible columns could be impossible or, at least, extremely time-consuming. Therefore, we use the *restricted master problem*, where only a small subset of the columns are present:

$$\begin{aligned} &\text{minimise} && \boldsymbol{c'}^{\top}\boldsymbol{x'} && (3.4)\\ &\text{subject to} && \boldsymbol{A'x'} = \boldsymbol{b} && (3.5)\\ & && \boldsymbol{x'} \geq \boldsymbol{0}\ . && (3.6) \end{aligned}$$

The restricted master problem must be feasible for the method to work. This is ensured by adding dummy columns. In each iteration of the column generation algorithm the restricted master problem (3.4)–(3.6) is solved, and the vector of dual variables is passed on to the pricing problem. The pricing problem returns a column with negative reduced cost, or proves that no such column exists. The column $\boldsymbol{A}_j$ from the pricing problem with cost c_j is then appended to the restricted master problem, yielding the new restricted master problem for the

next iteration of the column generation algorithm:

$$\begin{aligned} \text{minimise} \quad & \begin{pmatrix} \boldsymbol{c}' \\ c_j \end{pmatrix}^\top \begin{pmatrix} \boldsymbol{x}' \\ x_j \end{pmatrix} \\ \text{subject to} \quad & (\boldsymbol{A}' \mid \boldsymbol{A}_j) \begin{pmatrix} \boldsymbol{x}' \\ x_j \end{pmatrix} = \boldsymbol{b} \\ & \begin{pmatrix} \boldsymbol{x}' \\ x_j \end{pmatrix} \geq \boldsymbol{0} \ . \end{aligned}$$

3.1.2 Pricing problem

Consider the LP (3.1)–(3.3). Let $\boldsymbol{B}$ denote the $m \times m$ matrix of basic variables, and let $\boldsymbol{N}$ be the $m \times (n-m)$ matrix of non-basic variables. We split the cost vector in $\boldsymbol{c} = (\boldsymbol{c}_B, \boldsymbol{c}_N)^\top$, where $\boldsymbol{c}_B$ are the costs corresponding to $\boldsymbol{B}$, and $\boldsymbol{c}_N$ are the costs corresponding to $\boldsymbol{N}$. In every iteration of the simplex method a non-basic variable that can enter the basis (i.e. the set of basic variables) is found. The m-vector of dual prices $\boldsymbol{\pi}$ for the basis of the current iteration is given by

$$\boldsymbol{\pi}^\top = \boldsymbol{c}_B^\top \boldsymbol{B}^{-1}$$

and the reduced cost vector $\bar{\boldsymbol{c}}_N$ for the variables not in basis is given by

$$\bar{\boldsymbol{c}}_N^\top = \boldsymbol{c}_N^\top - \boldsymbol{c}_B^\top \boldsymbol{B}^{-1} \boldsymbol{N} = \boldsymbol{c}_N^\top - \boldsymbol{\pi}^\top \boldsymbol{N} \ ,$$

so the individual reduced cost for non-basic variable j is then given by

$$\bar{c}_j = c_j - \boldsymbol{\pi}^\top \boldsymbol{A}_j = c_j - \sum_{i=1}^{m} \pi_i a_{ij} \ .$$

Here $\boldsymbol{A}_j = (a_{1j}, \ldots, a_{mj})^\top$ is the jth column of the $\boldsymbol{A}$ matrix.

The pricing problem now searches among all columns for a column j where $\bar{c}_j < 0$ or proves that no such column exists. In the latter case it is then proof that the current optimal solution to the restricted master problem is also an optimal solution to the master problem, and the column generation algorithm can terminate.

Often the pricing problem will return the column with the most negative reduced cost, but this is not necessary, although it could speed up convergence. Any column with negative reduced cost can be added. Therefore, the pricing problem could be solved heuristically up to the point, where it needs to be proved that no more negative reduced cost columns exist (Irnich and Desaulniers, 2005). It is also possible for the pricing problem to return more than one column with negative reduced cost, which again could speed up convergence (Kohl, 1995).

3.1.3 Dantzig-Wolfe decomposition

If the LP (3.1)–(3.3) has the special *block-angular form* (Lasdon, 2002; Tebboth, 2001) the constraints (3.2) can be rewritten to

$$\begin{pmatrix} \boldsymbol{A}^1 & \boldsymbol{A}^2 & \cdots & \boldsymbol{A}^K \\ \boldsymbol{D}^1 & & & \\ & \boldsymbol{D}^2 & & \\ & & \ddots & \\ & & & \boldsymbol{D}^K \end{pmatrix} \begin{pmatrix} \boldsymbol{x}^1 \\ \boldsymbol{x}^2 \\ \vdots \\ \boldsymbol{x}^K \end{pmatrix} = \begin{pmatrix} \boldsymbol{b}^0 \\ \boldsymbol{b}^1 \\ \boldsymbol{b}^2 \\ \vdots \\ \boldsymbol{b}^K \end{pmatrix}, \tag{3.7}$$

and then the LP (3.1)–(3.3) can be written as

$$\text{minimise} \quad \sum_{k=1}^{K} \boldsymbol{c}^{k\top} \boldsymbol{x}^k \tag{3.8}$$

$$\text{subject to} \quad \sum_{k=1}^{K} \boldsymbol{A}^k \boldsymbol{x}^k = \boldsymbol{b}^0 \tag{3.9}$$

$$\boldsymbol{D}^k \boldsymbol{x}^k = \boldsymbol{b}^k \qquad \forall k \in \{1, \ldots, K\} \tag{3.10}$$

$$\boldsymbol{x}^k \geq \boldsymbol{0} \qquad \forall k \in \{1, \ldots, K\} \tag{3.11}$$

where $\boldsymbol{c}^k$ is the part of the cost vector $\boldsymbol{c}$ that corresponds to $\boldsymbol{x}^k$. Given an LP with this block-angular form, the LP can be reformulated by *Dantzig-Wolfe decomposition* (Dantzig and Wolfe, 1960b) into a master problem/subproblem setup suitable for column generation, see Martin (1999), Barnhart et al. (1998), Tebboth (2001), Kalvelagen (2003), and Desrosiers et al. (1995). The master problem will then have fewer constraints than the original LP, but the other constraints are then enforced by the pricing problem. The master problem, though, will have many more columns than the original LP.

The basic idea in Dantzig-Wolfe decomposition is to enforce the constraints (3.10) in K subproblems, while keeping the so-called *coupling constraints* (3.9) in a master problem.

For all $k \in \{1, \ldots, K\}$ the polyhedron

$$X^k = \left\{ \boldsymbol{x}^k \in \mathbb{R}^{n_k} : \boldsymbol{D}^k \boldsymbol{x}^k = \boldsymbol{b}^k, \boldsymbol{x}^k \geq \boldsymbol{0} \right\}$$

is the feasible region for the kth subproblem. Here n_k is the column dimension of $\boldsymbol{D}^k$ and $\boldsymbol{A}^k$. Let $\text{conv}(X^k)$ denote the convex hull of X^k and let

$\{\boldsymbol{x}^{k,1}, \ldots, \boldsymbol{x}^{k,P_k}\}$ be the extreme points of $\text{conv}(X^k)$ with

$$\begin{aligned} \boldsymbol{x}^{k,1} &= \left(x_1^{k,1}, \ldots, x_{n_k}^{k,1}\right)^\top \\ &\vdots \\ \boldsymbol{x}^{k,P_k} &= \left(x_1^{k,P_k}, \ldots, x_{n_k}^{k,P_k}\right)^\top . \end{aligned}$$

Let $\{\boldsymbol{y}^{k,1}, \ldots, \boldsymbol{y}^{k,R_k}\}$ be the extreme rays of $\text{conv}(X^k)$. Now, by Minkowski's Theorem (Nemhauser and Wolsey, 1988), we can describe any point $\boldsymbol{x}^k \in X^k$ as a convex combination of the extreme points plus a non-negative combination of the extreme rays of $\text{conv}(X^k)$:

$$\boldsymbol{x}^k = \sum_{p=1}^{P_k} \lambda_k^p \boldsymbol{x}^{kp} + \sum_{r=1}^{R_k} \mu_k^r \boldsymbol{y}^{kr} \tag{3.12}$$

$$\sum_{p=1}^{P_k} \lambda_k^p = 1 \tag{3.13}$$

$$\lambda_k^p \geq 0 \quad ,\forall p \in \{1, \ldots, P_k\} \tag{3.14}$$

$$\mu_k^r \geq 0 \quad ,\forall r \in \{1, \ldots, R_k\} \tag{3.15}$$

If the set X^k is bounded, then we do not need extreme rays, so (3.12) becomes $\boldsymbol{x}^k = \sum_{p=1}^{P_k} \lambda_k^p \boldsymbol{x}^{kp}$ and (3.15) is not necessary.

If we substitute with (3.12)–(3.15) in (3.8)–(3.11) and remove the constraints (3.10), we get the following master problem:

$$\text{minimise} \quad \sum_{k=1}^{K} \boldsymbol{c}^{k^\top} \left(\sum_{p=1}^{P_k} \lambda_k^p \boldsymbol{x}^{kp} + \sum_{r=1}^{R_k} \mu_k^r \boldsymbol{y}^{kr} \right) \tag{3.16}$$

$$\text{subject to} \quad \sum_{k=1}^{K} \boldsymbol{A}^k \left(\sum_{p=1}^{P_k} \lambda_k^p \boldsymbol{x}^{kp} + \sum_{r=1}^{R_k} \mu_k^r \boldsymbol{y}^{kr} \right) = \boldsymbol{b}^0 \tag{3.17}$$

$$\sum_{p=1}^{P_k} \lambda_k^p = 1 \quad ,\forall k \in \{1, \ldots, K\} \tag{3.18}$$

$$\lambda_k^p \geq 0 \quad ,\forall k \in \{1, \ldots, K\}, \forall p \in \{1, \ldots, P_k\} \tag{3.19}$$

$$\mu_k^r \geq 0 \quad ,\forall k \in \{1, \ldots, K\}, \forall r \in \{1, \ldots, R_k\} \tag{3.20}$$

The constraints (3.10) are enforced implicitly through the subproblems, so the master problem does not need to enforce these explicitly, yielding a simpler problem. However, the number of columns is potentially enormous due to the description by extreme points and rays. Most of these columns are not needed, so the column generation technique could obviously be applied.

3.2 Finding integer solutions

Let $\boldsymbol{A}$ be an $m \times n$ matrix and let $\boldsymbol{b}$ be an m-vector. Let $\boldsymbol{c}$ be the vector of costs, and let $\boldsymbol{x}$ be the solution vector. Consider the integer programme given below.

$$\begin{aligned} &\text{minimise} && \boldsymbol{c}^\top \boldsymbol{x} && (3.21)\\ &\text{subject to} && \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} && (3.22)\\ & && \boldsymbol{x} \in \mathbb{Z}_+^n \ . && (3.23) \end{aligned}$$

The *branch-and-bound* framework for solving integer programmes as (3.21)–(3.23) was introduced by Land and Doig (1960) and is widely used and described, see e.g. Wolsey (1998). The backbone of the branch-and-bound method is a decision tree. In the root node of the tree a relaxed version of (3.21)–(3.23) is defined. Often the integrality requirements are relaxed, and this is known as the LP relaxation. Relaxing the integrality requirements $\boldsymbol{x} \in \mathbb{Z}_+^n$, we can write the relaxed problem as

$$\begin{aligned} &\text{minimise} && \boldsymbol{c}^\top \boldsymbol{x} && (3.24)\\ &\text{subject to} && \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} && (3.25)\\ & && \boldsymbol{x} \geq \boldsymbol{0} \ . && (3.26) \end{aligned}$$

The method starts by finding an optimal solution to the relaxed problem (3.24)–(3.26) in the root node. If the solution is integral, the solution is also optimal for (3.21)–(3.23), and the algorithm terminates. If at least one of the relaxed constraints is violated, then the solution space is divided into two or more subsets, each corresponding to a *branch* in the decision tree, see Section 3.2.1, and the new nodes corresponding to the branches are added to the pool of unprocessed nodes. The union of the subsets contains all feasible solutions of the original solution space of (3.21)–(3.23), and the solution from the relaxed problem (3.24)–(3.26) (which is infeasible in the original problem) is made infeasible in each of the branches. Branching can be done for all nodes in the decision tree.

The full decision tree corresponds to a total enumeration of all possible solutions. A total enumeration is of course not desirable, and therefore subtrees are cut off whenever possible. This is called *pruning* of the tree. For each unexplored node in the tree, a lower bound is calculated. If the lower bound of a node is larger than or equal to the value of the current best feasible solution for the original problem—the so-called *incumbent*—the node is *pruned by bound*. A node is *pruned by feasibility* if the optimal solution to the relaxed problem in the node is also feasible with respect to the original problem. If the solution value is lower than the incumbent, the incumbent is updated. A node is *pruned*

by infeasibility if there are no feasible solutions to the relaxed problem in the node. Branching and pruning continue until the pool of unprocessed nodes is empty.

Given a pool of unprocessed nodes in the branch-and-bound tree, one has to decide on a strategy for picking the next node to process. Classically, *depth-first search*, where the node at the highest level in the tree is picked, or *best-first search*, where the node with the most promising lower bound is picked, have been used. Depth-first search find a feasible solution fastest, whereas best-first might find an optimal solution fastest. The two can be used in combination, where depth-first is used until a feasible solution is found, and then best-first is used to prove optimality. More sophisticated search strategies are described in Achterberg et al. (2005b).

3.2.1 Branching

As mentioned, branching is the means that is used to force an infeasible solution to a feasible one. Consider the relaxed problem (3.24)–(3.26) and let $S = \{\boldsymbol{x} \in \mathbb{R}^n : \boldsymbol{A}_1\boldsymbol{x} = \boldsymbol{b}_1, \boldsymbol{x} \geq 0\}$ denote the solution space. Now, if a variable x_j is fractional at value $\hat{x}_j$ in the solution to the relaxed problem, then it is clearly infeasible in the original problem (3.21)–(3.23).

Traditional *variable branching*, as described by Wolsey (1998), suggests to branch into two branches, where the solutions spaces are given by

$$\begin{aligned} S_1 &= \{\boldsymbol{x} \in S : x_j \leq \lfloor \hat{x}_j \rfloor\} \\ S_2 &= \{\boldsymbol{x} \in S : x_j \geq \lceil \hat{x}_j \rceil\} \ . \end{aligned}$$

Thereby the violating solution is forbidden in both branches. Let us look at the set partitioning problem which is a special case of (3.21)–(3.23). Now $\boldsymbol{A}$ is an $m \times n$ zero-one matrix and the decision vector $\boldsymbol{x}$ becomes binary:

$$\begin{aligned} &\text{minimise} && \boldsymbol{c}^\top \boldsymbol{x} && (3.27) \\ &\text{subject to} && \boldsymbol{A}\boldsymbol{x} = \boldsymbol{1} && (3.28) \\ & && \boldsymbol{x} \in \{0,1\}^n \ . && (3.29) \end{aligned}$$

If the integrality requirements are relaxed for (3.27)–(3.29) the solution space becomes $S = \{\boldsymbol{x} \in \mathbb{R}^n : \boldsymbol{A}\boldsymbol{x} = \boldsymbol{1}, \boldsymbol{x} \geq \boldsymbol{0}, \boldsymbol{x} \leq \boldsymbol{1}\}$. For a variable x_j which is fractional at value $\hat{x}_j$ with $0 < \hat{x}_j < 1$, variable branching divides S into

$$\begin{aligned} S_1 &= \{\boldsymbol{x} \in S : x_j = 0\} \\ S_2 &= \{\boldsymbol{x} \in S : x_j = 1\} \ . \end{aligned}$$

The branch for S_1 (called the 0-branch) forces the jth column not to be used. In most cases there are a lot of other variables that can enter the basis and yield a new fractional solution at the same objective function value. The branch for S_2 (called the 1-branch) forces the jth column to be selected and is thus likely to give rise to an increase in the objective function value. The result of this is a very unbalanced branch-and-bound tree, because pruning by bound is mostly not possible for the 0-branches, see also Ryan (1992b).

Another problem with variable branching is that it does not work well with column generation performed in every branch-and-bound node, as we will describe in Section 3.3. It is difficult to enforce the 0-branch in the pricing problems, i.e. it is not simple to prevent the banned column from being re-generated by the pricing problem.

Constraint branching

An alternative to variable branching is *constraint branching*, which has shown great efficiency in resolving fractions for set partitioning problems. Constraint branching was introduced by Ryan and Foster (1981) and is also described in Barnhart et al. (1998) and Vanderbeck and Wolsey (1996).

Let $J(s,t)$ denote the subset of columns that cover rows s and t in the constraint matrix from (3.27)–(3.29). Formally, we can write

$$J(s,t) = \{j \in \{1,\ldots,n\} : a_{sj} = 1 \text{ and } a_{tj} = 1\} \ .$$

In any optimal fractional solution to the LP relaxation of (3.27)–(3.29), at least one (s,t)-pair exists such that

$$0 < \sum_{j \in J(s,t)} x_j < 1 \ .$$

This is formally proved in Barnhart et al. (1998) and Vanderbeck and Wolsey (1996). Now, for the 0-branch we enforce

$$\sum_{j \in J(s,t)} x_j = 0 \ , \tag{3.30}$$

and for the 1-branch we enforce

$$\sum_{j \in J(s,t)} x_j = 1 \ . \tag{3.31}$$

In the 0-branch (3.30) all columns from $J(s,t)$ are removed from the problem, either explicitly or implicitly by setting their upper bound to zero. The 1-branch (3.31) can be imposed by enforcing

$$\sum_{j \in \bar{J}(s,t)} x_j = 0 \ ,$$

where $\bar{J}(s,t) = \{j \in \{1,\ldots,n\} : (a_{sj} = 1 \text{ and } a_{tj} = 0) \text{ or } (a_{sj} = 0 \text{ and } a_{tj} = 1)\}$, so all columns from $\bar{J}(s,t)$ are removed from the problem.

In both the 0-branch and the 1-branch potentially many columns are removed, so this branching scheme leads to a more balanced branch-and-bound tree than variable branching. In a balanced branch-and-bound tree, the lower bounds of the nodes increase evenly, instead of one bound increases significantly and the other bound is almost constant.

When one has to choose an (s,t)-pair with $0 < \sum_{j \in J(s,t)} x_j < 1$ to branch on, often the pair (s^*, t^*) where the sum of fractions is closest to one, i.e. where

$$(s^*, t^*) = \arg\max_{(s,t)} \left\{ \sum_{j \in J(s,t)} x_j : 0 < \sum_{j \in J(s,t)} x_j < 1 \right\}$$

is chosen. This strategy follows directly the preference of the LP for covering two constraints in the same column. Together with depth-first search this strategy can lead to a feasible and often high-quality integer solution fast.

Another strategy that tries to aim for a more balanced branch-and-bound tree by selecting the candidate closest to one-half, i.e.

$$(s^*, t^*) = \arg\min_{(s,t)} \left\{ \left| \sum_{j \in J(s,t)} x_j - \frac{1}{2} \right| : 0 < \sum_{j \in J(s,t)} x_j < 1 \right\} \ .$$

3.3 Branch-and-price

Let $\boldsymbol{A}$ be an $m \times n$ matrix, and let $\boldsymbol{b}$ be an m-vector. Let $\boldsymbol{c}$ be the vector of costs, and let $\boldsymbol{x}$ be the solution vector. Consider the integer programme (IP) given below.

$$\begin{aligned} &\text{minimise} && \boldsymbol{c}^\top \boldsymbol{x} && (3.32)\\ &\text{subject to} && \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} && (3.33)\\ & && \boldsymbol{x} \in \mathbb{Z}_+^n \ . && (3.34) \end{aligned}$$

Suppose as in Section 3.1 that (3.33)–(3.34) have an enormous number of feasible columns. We cannot use column generation directly, as (3.33)–(3.34) is not a linear programme, but fortunately column generation can be embedded in the branch-and-bound framework. This technique is known as *branch-and-price* and is described by for instance Desrosiers et al. (1984), Desrochers and Soumis (1989), Barnhart et al. (1998), and Ralphs et al. (2003).

In the branch-and-price method, column generation is performed in every node of the branch-and-bound tree. Let us denote (3.33)–(3.34) the *master problem* (MP). We restrict the enormous number of columns and then consider the *restricted master problem* (RMP). For column generation we need the LP relaxation of the restricted master problem, that is, we work with the *relaxed restricted master problem* (RRMP). Other constraints than the integrality requirements can be relaxed in RRMP.

Every time a node is selected from the node pool, the relaxed MP optimum is computed using column generation, taking into account the restrictions imposed in the current node. Therefore, the branch-and-price method is an exact approach that will yield the optimal solution to the MP integer programme, if such a solution exists. A flowchart for a branch-and-price algorithm can be seen on Figure 3.1. Whenever a branching decision is enforced in a node, it is necessary not only to remove columns that violate the branching restrictions, but also to prevent the pricing problem from generating such violating columns.

An alternative (or a companion) to column generation is *cut generation*. A so-called *separation algorithm* detects violated cuts from a collection of cuts, and the violated cuts are then added to the LP, thereby cutting away the solution that is infeasible in the original LP. Cut generation is described in detail in, for instance, Wolsey (1998). Instead of using a separation algorithm where the cuts are implicitly represented, a *cut pool* can be used. In a cut pool the cuts are explicitly represented. One can also use *constraint reintroduction*. In constraint reintroduction all the constraints or some of the constraints in the LP (3.1)–(3.3) are left out when the relaxed LP is solved. Whenever a removed constraint is found to be violated, it is reintroduced in the LP, thereby removing the infeasible solution.

Also cut generation and constraint reintroduction can be embedded in the branch-and-bound framework, giving rise to a *branch-and-cut* algorithm, see Ralphs et al. (2003). Here, cuts are generated or constraints reintroduced in every node of the branch-and-bound tree. Branch-and-price and branch-and-cut can be combined to a *branch-and-cut-and-price* algorithm where both column generation as well as cut generation (and constraint reintroduction) is carried out in every branch-and-bound node. Again we refer to Ralphs et al. (2003). A

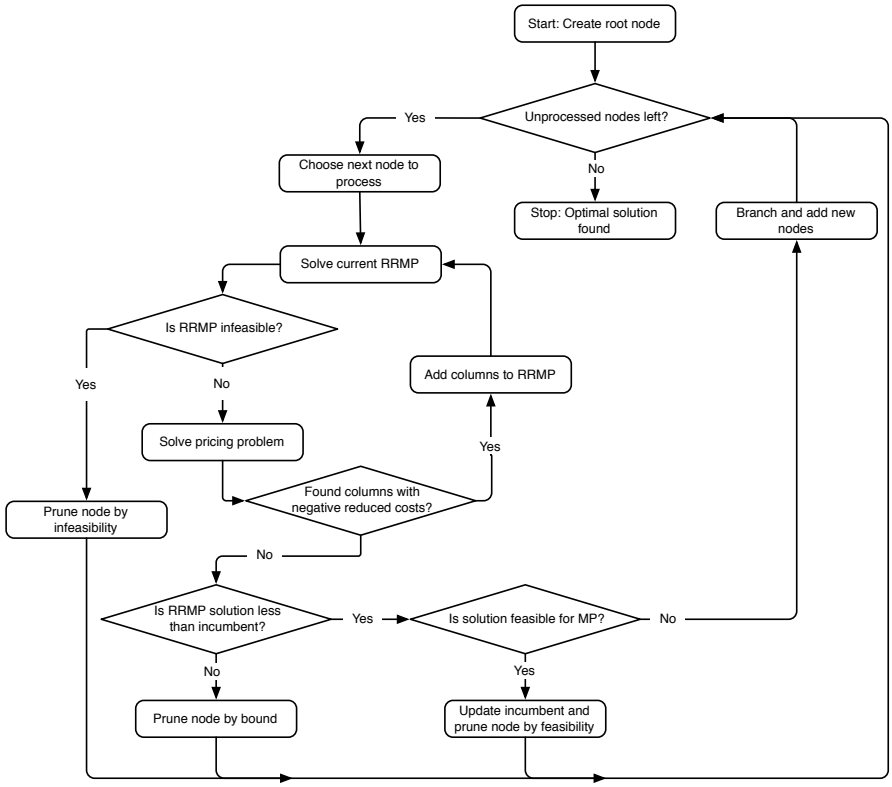


Figure 3.1: *Flowchart for the branch-and-price algorithm.*

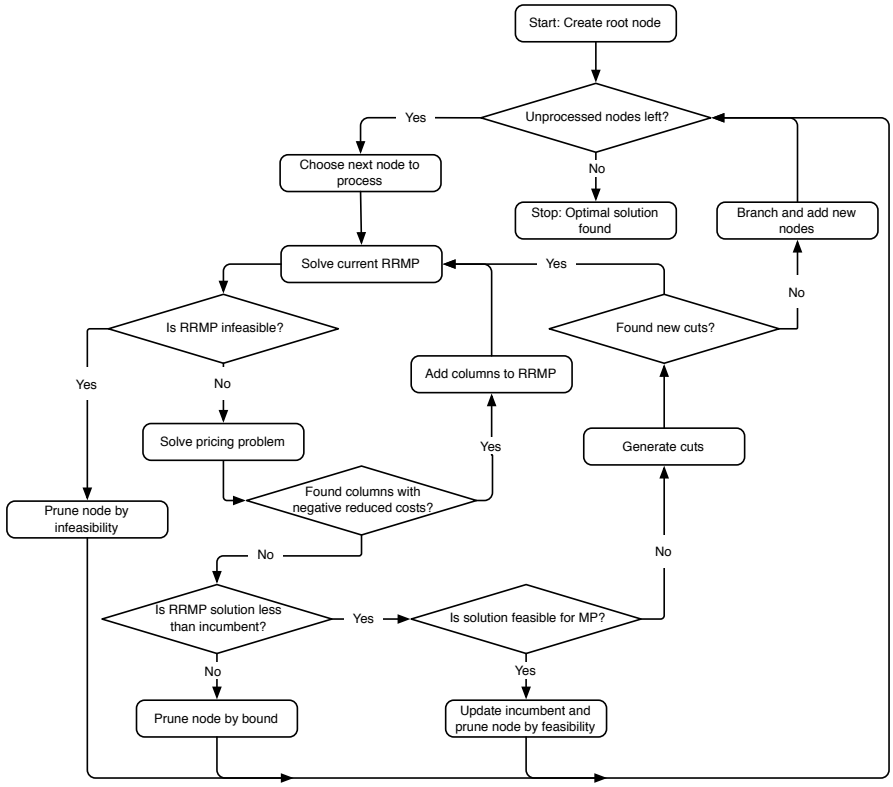


Figure 3.2: *Flowchart for the branch-and-cut-and-price algorithm.*

flowchart for a branch-and-cut-and-price algorithm can be seen on Figure 3.2.

3.4 Heuristic solution approaches

The methods presented thus far have all been exact solution approaches. Sometimes, however, computation time restrictions, implementation time restrictions, or computer hardware restrictions do not allow for an exact solution approach. Therefore, many—if not every—mathematical problem presented in the literature has at least one *heuristic* solution approach tailored for the specific problem. Some of these heuristic approaches have been generalised to so-called *metaheuristics*, i.e. generic heuristic frameworks for a solving a wide range of

problems. See for instance Burke and Kendall (2005) and Glover and Kochenberger (2003) for more on metaheuristics.

When a heuristic can guarantee that the objective value of any solution found by the heuristic is within a specified gap to the optimal objective value, it is called an *approximation algorithm*. This class of heuristics have been well-studied, see e.g. Hochbaum (1997) and Vazirani (2001). Approximation algorithms are tailored for the specific problem and thus not generic in any way as metaheuristics.

3.4.1 Optimisation-based heuristics

The last class of heuristics that we will present, is the class of *optimisation-based heuristics*. The methods in this class are based on the solution approaches described in the preceding sections of this chapter, but some components are made heuristic in order to speed up computation time.

One idea is to use heuristic, i.e. not necessarily true, lower bounds in the branch-and-bound tree. When the lower bound used for pruning by bound is not necessarily a true lower bound, then parts of the tree, that would be kept by an exact approach, can be cut off. This will speed up the search, but could also cut off a subtree containing the optimal solution.

The solution space S for the problem in question can be restricted to $S' \subset S$. The removed part $\bar{S} = S \backslash S'$ of the solution space must be wisely selected such that solutions from $\bar{S}$ are predicted to be unattractive in terms of the objective function. A smaller solution space will give faster pricing problem computation times and also a smaller branch-and-bound tree. Furthermore, a smaller solution space means fewer possible columns in the master problem, and thus the subsequence counts are likely to decrease. If the size of the solution space is decreased due to structural restrictions, then we would—as seen in Paper A, Paper C, and Paper D—directly decrease the subsequence counts. As mentioned in Section 2.6, low subsequence counts should lead to a less fractional LP relaxation, which will require less branching to obtain an IP solution.

The drawback is that the optimal solution could be in $\bar{S}$. In order to remedy this, parts of $\bar{S}$ can be reintroduced. If the reintroduction happens only in the root node, then lower bounds in the tree behaves as usually, but if the reintroduction can happen in every branch-and-bound node, then the LP objective value for a child node can be lower than the LP objective value of its parent node. I.e. the LP objective value of the parent node is not a true lower bound. Therefore the branch-and-bound tree can develop somewhat unpredictably, yet still be deterministic. Examples of this is in Paper A and Paper D.

In the search of the branch-and-bound tree, depth-first searching can be used and the algorithm can be stopped, when the first feasible solution is found. Alternatively, one can continue exploring the tree (for instance using best-first search) until some upper limit on computation time is hit, see Cordeau et al. (2002). Both suggestions are likely to give a heuristic solution, but might also produce optimal solutions.

As mentioned in Section 3.1.2, the pricing problem can be solved heuristically until it needs to be proved that no more negative reduced cost columns exist. This is done in Paper A and Paper B. The last step where the proof is needed, can often be very time-consuming, so a heuristic approach would be to skip the step. This would of course save the time spent to compute the proof, and would also decrease the number of iterations in the solve pricing problem/solve LP loop (the loop can be seen on Figure 3.1), but would sacrifice the proof of optimality.

Chapter 4

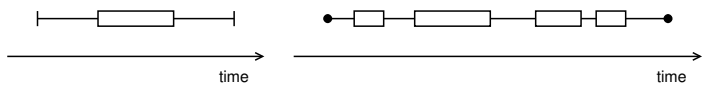
Crew Scheduling Applications

In this chapter we present practical applications of the set partitioning models introduced in Chapter 2. Some of the applications that we present here are described in detail in the papers in Part II. The applications that we show deal with crew scheduling, i.e. the assignment of task sequences to crew.

4.1 Scheduling of crew

The papers in Part II all deal with finding crew schedules. Therefore, we will here begin by introducing the concepts of tasks and schedules.

First, we define a *task*. In this thesis a task is a generic term that can denote a single piece of work, a shift, or collection of pieces of work. A task is carried out by a crew member. A task i has a *duration* δ_i and a *time window* $[\alpha_i, \beta_i]$ within which it must commence. The task cannot be started before the beginning of the time window even if a crew member is available. Note, that the special case $\alpha_i = \beta_i$ gives a fixed start time. Also, the task i has a *travelling time* s_{ij} to another task j. It can be convenient to include the duration in the travelling time. Again note the special case where the tasks are located at the same geographical place. In this case all travelling times are zero. Figure 4.1(a) illustrates a task.



(a) A task. (b) A crew schedule with four tasks.

Figure 4.1: *Tasks and crew schedules. The duration of a task is shown as a rectangle. The time window of a task is shown as the horizontal line starting and ending with a smaller vertical line. In the crew schedule, the start times of the tasks are decided. The start and the end of the schedule are shown as dots. The lines between the tasks in the schedule indicate travelling or idle time.*

A *crew schedule* is then a feasible sequence of tasks with start times. If the start times are fixed, then they are only given implicitly. The crew schedule is either anonymous or for a specific crew member. Feasible schedules must respect durations, time windows, and travelling times as well as other regulations and rules. Every schedule has an associated cost, which is a function of the task sequence. Examples of cost measures are the total travelling time or the total idle time for a schedule. Figure 4.1(b) illustrates a crew schedule. The *crew scheduling problem* can now be defined as finding a minimum cost combination of schedules that cover all tasks.

As mentioned, the notion of a task should be understood generally. In some applications tasks are aggregated, so one task can be a sequence of tasks, i.e. a partial schedule. Therefore, the meaning, and in particular the length, of a crew schedule can also vary from one application to another. Dohn (2010) elaborates on the concepts of tasks and schedules.

Set partitioning models can be used to assign tasks to crew. In such a case the jth column in the constraint matrix from the set partitioning model corresponds to schedule j, and the ith row corresponds to task i. Let $\boldsymbol{A}$ be an $m \times n$ zero-one matrix with entries a_{ij} for all $i \in \{1, \ldots, m\}$ and $j \in \{1, \ldots, n\}$ defined by

$$a_{ij} = \begin{cases} 1 & \text{if task } i \text{ is in schedule } j \\ 0 & \text{otherwise} \end{cases},$$

and let x_j for all $j \in \{1, \ldots, n\}$ be a binary decision variable defined by

$$x_j = \begin{cases} 1 & \text{if schedule } j \text{ is selected} \\ 0 & \text{otherwise} \end{cases}.$$

The crew scheduling problem can now be written as the following binary integer

programme (see also Section 2.3):

$$\begin{aligned} &\text{minimise} && \boldsymbol{c}^\top \boldsymbol{x} && (4.1)\\ &\text{subject to} && \boldsymbol{A}\boldsymbol{x} = \boldsymbol{1} && (4.2)\\ & && \boldsymbol{x} \in \{0,1\}^n \ . && (4.3) \end{aligned}$$

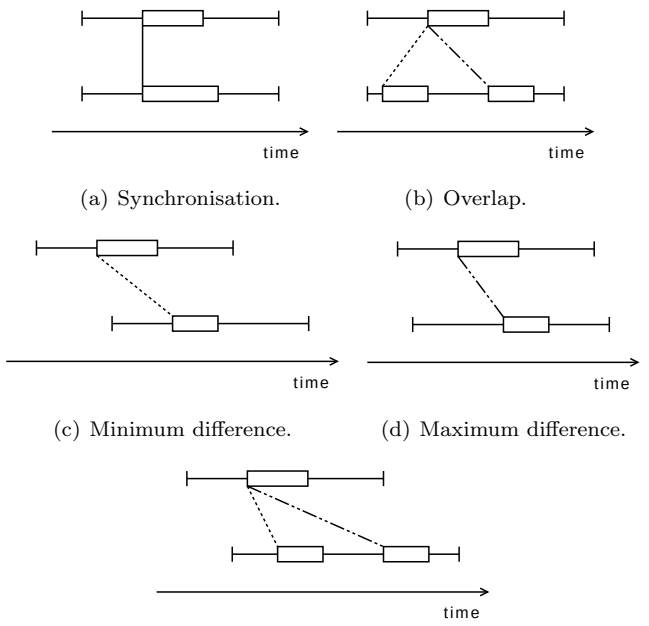
Constraints (4.2) ensures that tasks are covered exactly once. Each work schedule has an associated cost c_j, and the model then picks a minimum cost combination of work schedules that together cover all tasks exactly once.

Sometimes crew schedules has to be found for individual crew members, a process also known as *rostering*. This is accomplished by adding a row to $\boldsymbol{A}$ for each of the crew members. Every column is then given a one in a crew member row, if the column represents a crew schedule belonging to that crew member, and a zero otherwise. An example of this can be seen in Paper A.

In many cases this pure set partitioning model does not capture the specifics of the application. Thus, a variant of the generalised set partitioning model (2.24)–(2.27) must be used. As mentioned in Section 2.6, the theoretical results about integer properties from Section 2.5 do not hold any more, when the set partitioning problem is non-pure. However, if the non-pure constraints are few, it is still expected that certain constraint matrices can have an integerising effect. In most practical cases the constraint matrix does not belong to one of the matrix classes presented in Section 2.5. Therefore, in some of the solution approaches in the papers of Part II, the idea is to impose restrictions on the constraint matrix, so that the constraint matrix will almost be a member of one of the matrix classes from Section 2.5. This is done by exploiting structural properties of the specific crew scheduling problem.

4.1.1 Temporal dependencies

In Paper A and Paper B, tasks are interconnected by *temporal dependencies*, that restrict the starting times of tasks between work schedules. Consider two tasks i and j. *Synchronisation* (see Figure 4.2(a)) requires i and j to start at the exact same time, and is used in Ioachim et al. (1999) and Dohn et al. (2009a). *Minimum difference* (see Figure 4.2(c)) and *maximum difference* (see Figure 4.2(d)) require task j to start with a minimum respectively maximum time difference to the starting time of task i. *Min+max difference* (see Figure 4.2(e)) requires both a minimum and a maximum time difference. Minimum, maximum and min+max difference are described in Brucker and Knust (2006). The last example of a temporal dependency we will show, is *overlap* (see



(a) Synchronisation. (b) Overlap.

(c) Minimum difference. (d) Maximum difference.

(e) Min+max difference.

Figure 4.2: *Examples of temporal dependencies. Each of the five subfigures shows the time windows of two tasks i (top) and j (bottom) with a temporal dependency between them. Assuming some start time for task i, the dotted line shows the earliest feasible start time for task j, and the dashed dotted line shows the latest feasible start time. For synchronisation (a) the two lines coincide, and are drawn as one full line.*

Figure 4.2(b)), which require the execution of task i and task j to overlap in time. This is a special case of min+max difference.

These temporal dependencies can be modelled by introducing *generalised precedence constraints* (Dorndorf, 2002; Brucker and Knust, 2006; Brucker and Knust, 2010) of the form

$$\sigma_i + p_{ij} \leq \sigma_j \ ,$$

where σ_i denotes the start time of task i, and $p_{ij} \in \mathbb{Z}$ quantifies the required gap. The set of pairs of tasks $(i, j) \in \{1, \ldots, m\}^2$ for which a generalised precedence constraint exists is denoted $\mathcal{P}$.

The generalised precedence constraint simply implies that j starts minimum p_{ij} time units after i. The temporal dependency that is probably most used in practice is that of synchronisation, where two visits are required to start

	Temporal dependency	p_{ij}	p_{ji}
(a)	Synchronisation	0	0
(b)	Overlap	$-\delta_j$	$-\delta_i$
(c)	Minimum difference	$\text{diff}_{\min}$	N/A
(d)	Maximum difference	N/A	$-\text{diff}_{\max}$
(e)	Minimum+maximum difference	$\text{diff}_{\min}$	$-\text{diff}_{\max}$

Table 4.1: *Values for p_{ij} for the five temporal dependencies of Figure 4.2. δ_i is the duration of task i, $\text{diff}_{\min}$ is the minimum difference required and $\text{diff}_{\max}$ is the maximum difference required.*

at the same time. Synchronisation is modelled by two generalised precedence constraints (i, j) and (j, i) with $p_{ij} = p_{ji} = 0$. Table 4.1 shows how to model all the temporal dependencies of Figure 4.2 with generalised precedence constraints. It can be seen that (a), (b) and (e) each requires two generalised precedence constraints, whereas (c) and (d) only need one each.

Given these additional constraints, the work schedules must also give a start time σ_i for each task i in the schedule, and the start times must respect these additional constraints.

4.1.2 Time window branching

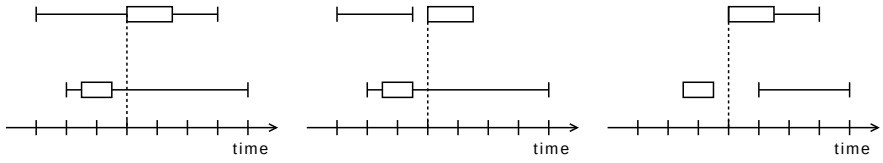
In a branch-and-price setup, generalised precedence constraints can either be enforced directly in the master problem or they can be enforced through the branching. We will describe the latter approach here.

Let again σ_i denote the start time for task i in a solution to the master problem. If a generalised precedence constraint $(i, j) \in \mathcal{P}$ is violated, then

$$\sigma_i + p_{ij} \nleq \sigma_j \ .$$

The suggested *time window branching* scheme now branches in two and alters the time window of i in both branches. In Ioachim et al. (1999) and Dohn et al. (2009a) time window branching have been used to enforce synchronisation temporal dependencies. In Gélinas et al. (1995) time window branching is used to impose integrality.

A split time σ_{split} is selected where $\sigma_j - p_{ij} + 1 \leq \sigma_{\text{split}} \leq \sigma_i$. We here assume that time is discretised. In the left branch the time window of task i is changed to $[\alpha_i, \sigma_{\text{split}} - 1]$, see Figure 4.3(b). In the right branch the time window of task i is changed to $[\sigma_{\text{split}}, \beta_i]$. In the right branch we can also use the



(a) Parent node. (b) Left child node. (c) Right child node.

Figure 4.3: *Time window branching example. Each of the subfigures shows the time windows of two tasks i (top) and j (bottom), and the start times of the tasks in a solution to the master problem. The violated generalised precedence constraint for $(i, j) \in \mathcal{P}$ has $p_{ij} = 2$. The dotted line shows the chosen split time σ_{split}, and the distance between the ticks on the time line is two time units.*

generalised precedence constraint (i, j) to change the time window of task j to $[\sigma_{\text{split}} + p_{ij}, \beta_j]$, see Figure 4.3(c). The time window branching scheme divides the solution space of the parent node, see Figure 4.3(a) into two subspaces where the violating solution is infeasible in both.

When time window branching is employed, we pick the time window as well the accompanying split time that has most impact when branched upon. That is, we rank time windows with split times according to how many generalised precedence constraints violations that can be "repaired". As mentioned, time window branching can also be used to impose integrality. Therefore, we also rank according to how much fractionality that can potentially be removed. Details on this ranking can be found in Paper B.

4.2 Selected applications

In this section we will present a number of different crew scheduling problems. The ambition is to show the diversity of the crew scheduling definition from the previous section. We do certainly not cover every application; instead we will refer to the comprehensive surveys Ernst et al. (2004b) and Ernst et al. (2004a).

4.2.1 Train driver scheduling

Scheduling of train drivers is most often divided in two problems, see Rezanova (2009), Caprara et al. (2007), and Caprara et al. (1997). The decomposition of

the problem in two phases is done in order to make the problem computationally tractable. In the first phase, train trips (tasks) are combined into duties (partial schedules). Each duty must respect driver regulations, such as for instance labor union rules, and each duty has an associated cost, that reflects e.g. how much idle time is spent in the duty. A minimum cost set of duties, that covers all train trips is found, see Abbink et al. (2005). This first phase is often modelled by a set covering formulation, and the cost measure seeks to—among other objectives—minimise idle time for the drivers.

The second phase combines duties into schedules, and is sometimes referred to as the rostering phase. Caprara et al. (2007) suggest that a network flow formulation works better than a set partitioning formulation. The cost measure in this step incorporates an even distribution of attractive and unattractive work.

4.2.2 Nurse scheduling

In the *nurse scheduling problem* (also called the nurse rostering problem), shifts, i.e. blocks of work hours, must be assigned to nurses. In between the shifts days off must be scheduled, according to rules. The shifts have different start times, so that the shifts cover full 24-hour periods. Normally, crew schedules for nurses have a length of one month. By considering shifts as tasks, nurse scheduling fits into our definition of crew scheduling, where the time windows are fixed and there are no travelling times.

The problem is complicated by different qualification of the nurses and correspondingly different qualification requirements of the shifts. Also several soft constraints, such as personal preferences for specific shift must be taken into account. In nurse scheduling the number of crew is fixed, so the aim is not to minimise the number of necessary crew members. The cost measure instead seeks to maximise crew preferences and satisfaction. The literature on nurse scheduling is surveyed in Burke et al. (2004). A recent optimisation-based solution approach is given by Dohn et al. (2010), where a generalised set partitioning model is used.

4.2.3 Home care crew scheduling

Paper A describes a crew scheduling problem from home care. In home care, a staff of home carers have to carry out services (tasks) in the homes of elderly citizens. The idea behind home care is to allow elderly or disabled citizens to stay

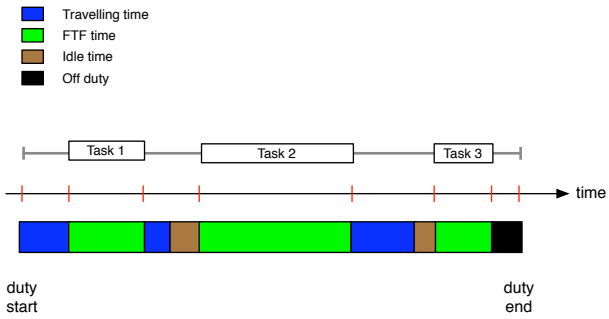


Figure 4.4: *Duty for a home carer. FTF time is face-to-face time, i.e. the time the home carer spends on actual visits.*

in their own homes. The services include cleaning, laundry assistance, preparing food, and support for other everyday tasks. They may also include assistance with respect to more personal needs, e.g. getting out of bed, bathing, dressing, and dosing medicine. As a consequence of the variety of services offered, people with many different qualifications are employed as home carers.

The *home care crew scheduling problem* (HCCSP) is a daily planning problem that assigns visits (tasks) to home carers, thereby giving the individual home carer a schedule and a route, see Figure 4.4. The most important goal is to cover as many visits as possible. If it is not possible to cover all tasks, the different priorities of the visits are taken into account when deciding which tasks to cancel. It is also important that a visit is handled by a so-called *preferred* home carer, so that, for instance, the citizen will be serviced by a familiar home carer if possible. Lastly, we prefer to minimise the total travel time.

HCCSP has temporal dependencies between some of the visits. For instance, a temporal dependency can be used to allow a washing machine to finish before the machine is emptied. Also shared visits are a part of the HCCSP. A shared visit is e.g. needed when an elderly person needs to be lifted out of the bed. Shared visits are modelled as two tasks at the same location having a synchronisation temporal dependency.

In HCCSP, temporal dependencies must still be respected even if a visit is cancelled, because cancelled visits are covered by substitutes by a manual planner later on.

Thomsen (2006) and Lessel (2007) describe HCCSP based on instances from home care in Denmark. Eveborn et al. (2006), Bredström and Rönnqvist (2007),

and Bredström and Rönnqvist (2008) base their work on home care in Sweden, and the two first of these use a generalised set partitioning formulation. Work on home care in other countries are described by Begur et al. (1997), Cheng and Rich (1998), and Bertels and Fahle (2006).

4.2.4 Scheduling of ground crew

In the *ground crew rostering problem*, work schedules for airport ground crew must be produced. Schedules are made by combining shifts, and it is ensured that a forecasted demand is met. The problem is somewhat similar to the nurse scheduling problem, but with fewer soft constraints. Also the literature on this problem is sparser than for nurse scheduling. We refer to Clausen (2010) and Lusby et al. (2011). The problem is modelled as a generalised set partitioning problem.

Another scheduling problem for ground crew is the daily assignment of tasks to available crew. Tasks have different skill requirements and the crew have different qualifications to match that. The problem is similar to home care crew scheduling, but only synchronisation temporal dependencies are used, and, in most cases, travelling times are smaller. The problem is called the *manpower allocation problem with time windows and job-teaming constraints* and is described by Dohn et al. (2009a), and is here modelled as a generalised set partitioning problem. Li et al. (2005) describe the same problem, but in a port instead of an airport.

4.2.5 Vehicle routing with time windows and temporal dependencies

Paper B describes the *vehicle routing problem with time windows and temporal dependencies* (VRPTWTD), which is a new variant of the well-known *vehicle routing problem with time windows* (VRPTW). VRPTW deals with routing (i.e. scheduling) of customers (tasks) to vehicles (crew). All tasks must be scheduled within their associated time window and the capacity of the individual vehicles cannot be exceeded. The literature on VRPTW is vast, see for instance Cordeau et al. (2002), Kallehauge et al. (2005), and Cordeau et al. (2007). VRPTW is often modelled as a set partitioning problem to allow for column generation solution algorithms.

VRPTWTD then adds temporal dependencies (see Section 4.1.1) to VRPTW,

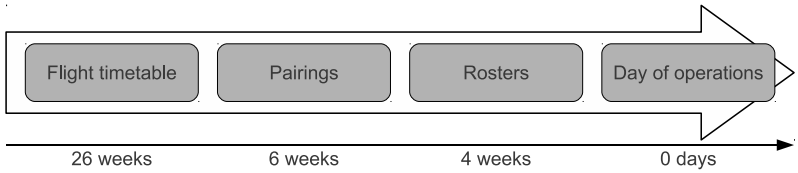


Figure 4.5: *The steps for airline crew scheduling. The times are for Air New Zealand's domestic scheduling.*

thereby complicating the problem notably, as now a scheduled time of one customer may restrain the scheduling options of other customers.

4.2.6 Airline crew scheduling

Scheduling of airline flight deck and cabin crew is the assignment of crew members to the flights in the airline's timetable. The aim is to provide monthly work schedules (rosters) for the crew members. Airline crew scheduling is traditionally carried out in a fashion similar to that of scheduling of train drivers, i.e. the optimisation problem is decomposed to two phases. The two phases are: *Crew pairing* and *crew rostering*. These two phases follow the publication of the flight timetable, and, naturally, they precede the day where the flights are flown. Figure 4.5 shows the scheduling steps. Airline crew scheduling is described in Gopalakrishnan and Johnson (2005), Barnhart et al. (2003), and Desaulniers et al. (1998). Desaulniers et al. (1997) solve crew pairing instances from Air France. The instances have between 150 and 1 100 flights and solve in between 20 and 14 000 seconds.

As mentioned, the first phase of airline crew scheduling is the *airline crew pairing problem*, which is the topic of Paper C and Paper D. Here, partial schedules are constructed. The partial schedules are sequences of flights that can be either operated or passengered (deadheaded) in a feasible way, separated by rest periods. These sequences of flights are called *pairings* and are anonymous, that is they are not associated with a specific crew member. An airline has one or more home bases where its crew is located, and a pairing must start and end at a home base. So-called *base constraints* then govern the minimum and the maximum number of pairings that can be flown out of each home base. An illustration of a pairing is shown on Figure 4.6.

The crew pairing problem can be solved separately for cockpit crew and cabin crew, and it can also be solved separately per aircraft type qualification of

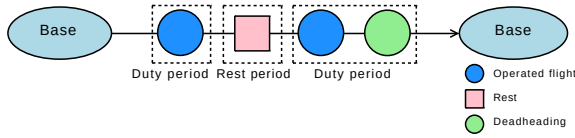


Figure 4.6: *An illustration of a pairing.*

the crew. The cost of a pairing are calculated in different and quite complex ways by different airlines, for examples of this see Gopalakrishnan and Johnson (2005). The crew pairing problem then finds a minimum cost combination of pairings that covers all flights. This is most often modelled as a generalised set partitioning problem.

The second phase of airline crew scheduling is crew rostering where pairings are combined to form schedules for individual crew member. This phase is called the *airline crew rostering problem* (Kohl and Karisch, 2004), and in this phase a schedule needs to be found for each crew member. Individual preferences and requests are often taken into account in this step. Again, a generalised set partitioning model is used in the majority of the literature on airline crew rostering.

Chapter 5

Overview of Papers

In this chapter we will introduce the papers from Part II. We show how to model the problems presented in the papers with the generalised set partitioning model from Section 2.6, and we describe which solution methods from Chapter 3, that have been used.

5.1 Paper A: The Home Care Crew Scheduling Problem: Preference-Based Visit Clustering and Temporal Dependencies

The home care crew scheduling problem (HCCSP) is the topic of Paper A: *The Home Care Crew Scheduling Problem: Preference-Based Visit Clustering and Temporal Dependencies* (Rasmussen et al., 2011f). We have presented similar results in Rasmussen et al. (2010c), and preliminary results in Larsen et al. (2010) and Dohn et al. (2008b), where the conference paper won the Best Paper Award. HCCSP has also been described in the journal of the Danish Operations Research Society (Dohn et al., 2008a). Additional computational experiments are described in Section 6.1.

The HCCSP is modelled by a compact formulation and then reformulated by

Dantzig-Wolfe decomposition (see Section 3.1.3) to a generalised set partitioning model, see Section 2.6. Figure 5.1 illustrates how HCCSP fits into Model (2.24)–(2.27). Temporal dependencies must still be respected even if a visit is cancelled, so the generalised precedence constraints then take the form

$$\beta_j y_j^{\text{uc}} + \sigma_j x_j - \sigma_i x_i - \alpha_i y_i^{\text{uc}} \geq p_{ij} \quad \text{, for all } (i,j) \in \mathcal{P} \ .$$

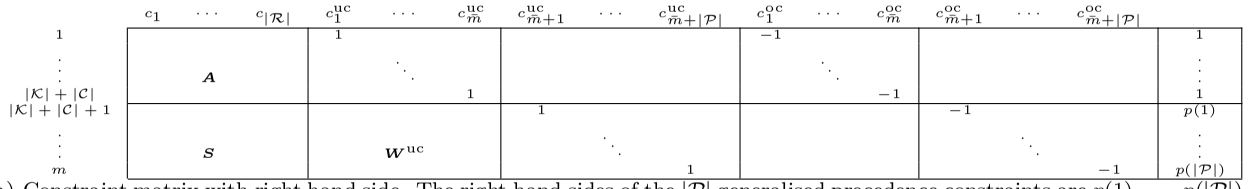
The solution approach of Paper A is an optimisation-based heuristics, see Section 3.4.1. The foundation is a branch-and-price algorithm, see Section 3.3, but we restrict the solution space, by using preference-based clustering of the visits. The original solution space is restricted by only allowing visits to be handled by preferred home carers. This makes it possible to tackle larger problem instances, but in some situations is it necessary to cover visits with non-preferred home carers. Therefore, uncovered visits are iteratively added to also non-preferred home carers, hence reintroducing parts of the removed solution space.

The restriction of the solution space by clustering of the visits makes the pricing problem faster to solve. Moreover, it is very likely to decrease the subsequence counts for all rows, so that fewer fractions in the LP solutions are expected.

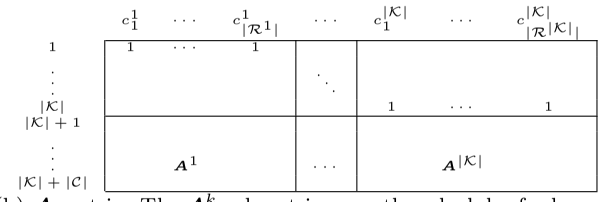
Exactly one individual work schedule needs to be found for each of the home carers. Hence, our model, see Figure 5.1(b), will include GUB constraints (see Section 2.5.2), and as a consequence the constraint submatrices corresponding to a home carer are perfect matrices. Therefore, fractions in the LP solution only occur when different home carers compete for a visit. This is exploited in the developed branching scheme, where we use constraint branching (see Section 3.2.1) on a home carer-visit pair. In our approach we also relax the temporal dependencies, so the generalised precedence constraints have to be enforced through a second branching scheme, namely time window branching, see Section 4.1.2.

The pricing problem is solved first heuristically and then refines to an exact solve using a resource constrained elementary shortest path solver. Our shortest path solver is implemented based on ideas from Feillet et al. (2004), Chabrier (2006), and Irnich and Desaulniers (2005).

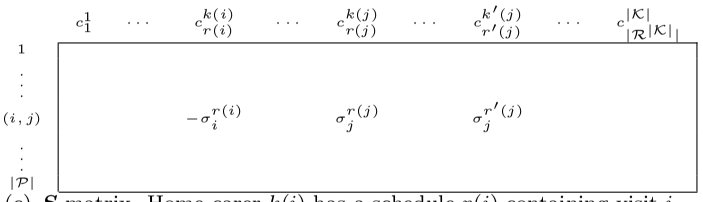
We perform computational tests on 94 realistic test instances. Four of these instances are real-life instances from Denmark, and 60 other instances are generated based on the real-life instances. Lastly, 30 instances are made available to us by Bredström and Rönnqvist (2007). Parsers for all instance classes have been developed. We compare different clustering schemes to an exact solution approach, and show that computation times can be lowered by 50–70%, where the loss in solution quality is insignificant apart from a few instances.



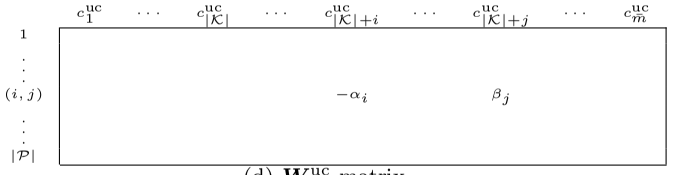
(a) Constraint matrix with right hand side. The right hand sides of the $|\mathcal{P}|$ generalised precedence constraints are $p(1), \ldots, p(|\mathcal{P}|)$.



(b) $\boldsymbol{A}$ matrix. The $\boldsymbol{A}^k$ submatrices are the schedules for home carers $k \in \mathcal{K}$.



(c) $\boldsymbol{S}$ matrix. Home carer $k(i)$ has a schedule $r(i)$ containing visit i.



(d) $\boldsymbol{W}^{\text{uc}}$ matrix.

Figure 5.1: *Constraint matrix with right hand side. Only non-zero entries are shown. $\mathcal{K}$ is the set of home carers and $\mathcal{C}$ is the set of visits. Set $\bar{m} = |\mathcal{K}| + |\mathcal{C}|$, so $m = \bar{m} + |\mathcal{P}|$ is the number of constraints. $\mathcal{R}$ is the set of schedules, and $\mathcal{R}^k$ is the schedules for home carer $k \in \mathcal{K}$. Let M' and M be large constants with $M' < M$. Set $c_i^{uc} = M'$ for all $i \in \{|\mathcal{K}| + 1, \ldots, |\mathcal{K}| + |\mathcal{C}|\}$ and set $c_i^{uc} = M$ otherwise. This allows for cancellation of visits. Set $c_i^{oc} = 0$ if $i \in \{\bar{m} + 1, \ldots, \bar{m} + |\mathcal{P}|\}$ and set $c_i^{oc} = M$ otherwise in order to get generalised precedence constraints.*

The main contribution of Paper A lies in the developed optimisation-based heuristic that utilises clustering of the visits. The paper shows a solution method to a real-life problem with promising results, and also branching on time windows to enforce generalised precedence constraints is novel for real-life problems.

5.2 Paper B: The Vehicle Routing Problem with Time Windows and Temporal Dependencies

The vehicle routing problem with time windows and temporal dependencies (VRPTWTD) is presented in Paper B: *The Vehicle Routing Problem with Time Windows and Temporal Dependencies* (Dohn et al., 2011). An earlier presentation is Dohn et al. (2009c).

As mentioned in Section 4.2.5, VRPTWTD is a new extension of the vehicle routing problem with time windows. Two compact formulations for the problem are presented. One of the compact formulations is similar to the HCCSP model, see Figure 5.1. However, there are no GUB rows in the $\boldsymbol{A}$ matrix and $\boldsymbol{W}^{\text{uc}} = \boldsymbol{0}$, as visits cannot be cancelled in VRPTWTD. The two formulations are then reformulated using Dantzig-Wolfe decomposition (Section 3.1.3), so that a setup suitable for column generation is achieved.

Four different master problem formulations are proposed, and they are ranked according to the tightness with which they describe the solution space. It is proved that one formulation is tighter than another. The solution approaches used are exact branch-and-price or branch-and-cut-and-price, see Section 3.3. Time window branching (Section 4.1.2) is used to enforce feasibility when generalised precedence constraints are relaxed. We show that time window branching on generalised precedence constraints is as strong as the special case of time window branching on synchronisation constraints introduced by Ioachim et al. (1999) and also used in Dohn et al. (2009a).

Tests are carried out in order to compare the different formulations. Three of the formulations are tested: A time-indexed formulation, a time-indexed formulation with only a limited number of constraints, and a direct formulation where the generalised precedence constraints are relaxed. The test instances are extensions of the well-known VRPTW-instances by Solomon (1987b). The results show, that the best performance is achieved either by relaxing the generalised precedence constraints in the master problem, or by using the time-indexed model with only a limited number of constraints, where generalised precedence constraints are added as cuts when they become severely violated.

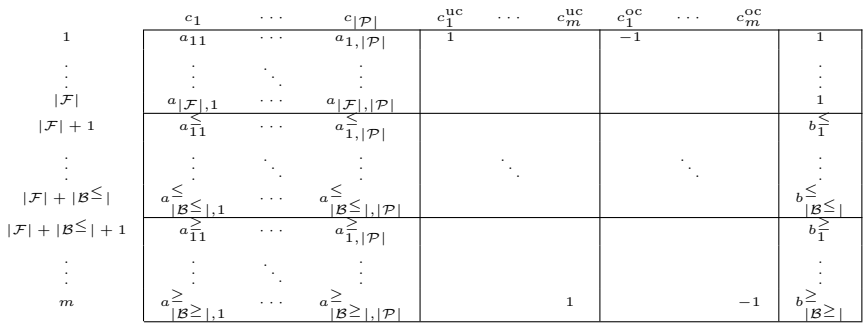


Figure 5.2: *Constraint matrix with right hand side. Only non-zero entries are shown. $\mathcal{F}$ is the set of flights, $\mathcal{P}$ is the set of pairings, and $\mathcal{B}^{\leq}$ ($\mathcal{B}^{\geq}$) is the set of base constraints that puts an upper (lower) bound on the number of pairings that are flown out of the base. The number of constraints is $m = |\mathcal{F}| + |\mathcal{B}^{\leq}| + |\mathcal{B}^{\geq}|$. Let M be a large constant. Set $c_i^{uc} = 0$ if $i \in \{|\mathcal{F}|+1, \ldots, |\mathcal{F}|+|\mathcal{B}^{\leq}|\}$ and set $c_i^{uc} = M$ otherwise. Set $c_i^{oc} = 0$ if $i \in \{|\mathcal{F}|+|\mathcal{B}^{\leq}|+1, \ldots, m\}$ and set $c_i^{oc} = M$ otherwise. By these cost settings, we get set packing and set covering base constraints.*

The main contribution of Paper B is the formulation and comparison of models for VRPTWTD along with an efficient solution approach, that handles temporal dependencies in a generic way. Moreover, it is a contribution that time window branching on generalised precedence constraints is as strong as the special case of time window branching on synchronisation constraints. Finally, we have introduced a new set of benchmark instances which we hope will motivate future research in this area.

5.3 Paper C: Subsequence Generation for the Airline Crew Pairing Problem

The airline crew pairing problem (ACPP) is the topic of Paper C: *Subsequence Generation for the Airline Crew Pairing Problem* (Rasmussen et al., 2011d). It has also been presented in Rasmussen et al. (2011e) as well as in the conference papers Rasmussen et al. (2009a) and Rasmussen et al. (2010a).

The ACPP is modelled directly as a generalised set partitioning model, see Section 2.6. The columns correspond to pairing and the rows correspond to flights that have to be covered. Base constraints are modelled as additional constraints in the model. Figure 5.2 illustrates how ACPP fits into Model (2.24)–(2.27).

To solve ACPP we propose a new optimisation-based heuristic: *Subsequence generation*. Initially the solution space is restricted severely (see also Section 3.4.1), by forcing the subsequence counts very low. When we severely limit the number of allowed subsequent flights, i.e. the subsequences, we significantly decrease the number of possible columns. Set partitioning problems with limited subsequence counts are very likely (Ryan and Falkner, 1988) to be easier to solve, resulting in a decrease in solution time, and it is therefore possible to enumerate all pairings. The problem though, is that some of the omitted subsequences might be needed for an optimal or near-optimal solution. Therefore, we try to identify or generate such subsequences that potentially can improve the solution value. Such attractive subsequences are found by collecting statistics based on the dual values.

The pricing problem is solved by a resource constrained shortest path solver on an acyclic network. The solver has been implemented based on the ideas from Desrochers et al. (1992b), Chabrier (2006), and Irnich and Desaulniers (2005).

Computational tests are performed on 19 real-life test instances from Air New Zealand, and data parsers have been implemented. The subsequence generation approach is benchmarked on the real-life test instances against a classical column generation approach. The LP relaxation is considered and comparison is done on quality and integrality of the solutions. The LP solutions from the subsequence generation approach are less fractional, but it comes at the cost of a worse solution quality.

The contribution of Paper C lies in the description and implementation of the novel subsequence generation approach and the comparison to a standard column generation algorithm.

5.4 Paper D: An IP Framework for the Crew Pairing Problem using Subsequence Generation

In Paper D: *An IP Framework for the Crew Pairing Problem using Subsequence Generation* (Rasmussen et al., 2011b) the airline crew pairing problem (ACPP) is again the topic. It has also been presented in Rasmussen et al. (2011c) and in Rasmussen et al. (2011a). Additional unpublished computational experiments are described in Section 6.2.

The ACPP is modelled in the same way as in Paper C, see Section 5.3. Fig-

ure 5.2 illustrates how the problem looks in the generalised set partitioning model (2.24)–(2.27).

The solution method used in this paper builds on top of the subsequence generation approach from Paper C. However, we here develop a full IP framework for the ACPP. The framework has similarities to branch-and-price (see Section 3.3), but instead of performing column generation in every node of the branch-and-bound tree, subsequence generation is performed.

The chosen branching scheme is follow-on branching, where a subsequence of two flights is either required or banned. Follow-on branching is a special case of constraint branching, see Section 3.2.1.

We carry out computational tests with the 19 real-life instances from Air New Zealand also used in Paper C. We benchmark against a standard branch-and-price approach, and the results show the developed framework is a viable alternative to the standard method.

The main contribution of Paper D is the development of a novel branch-and-price-like IP framework, where subsequence generation is embedded in branch-and-bound. Another contribution is the comparison to classic column generation.

Chapter 6

Additional Computational Experiments

In this chapter we will present computational experiments that have been carried out in connection with the projects from the papers in Part II, but which have not been described in any of the papers.

6.1 Comparison to current practice in home care

We have compared solutions from the approach described in Paper A with solutions from current practice. The current practice solutions are provided by a Danish provider of home care software. The same company has also provided the test instances, which are from two Danish municipalities. The current practice is based partly on an automated heuristic and partly on manual planning. We compare on three quality parameters: Uncovered visits, constraint adjustments, and total travel time. The results presented here have also been shown in Dohn et al. (2008b).

An uncovered visit is a visit, where a home carer has not been assigned. In

Instance	Home carers	Visits	Current			B&P			
			Uncovered visits	Const. adjust.	Total travel time	Uncovered visits	Const. adjust.	Total travel time	Run time (s)
`hh`	15	150	9	0	427	5	0	448	2044
`ll1`	8	99	11	26	256	6	0	280	69
`ll2`	7	60	1	10	155	0	0	141	3
`ll3`	6	61	0	25	311	1	0	128	39

Table 6.1: *Comparison of solution quality. 'Current' is the current practice and 'B&P' is the developed branch-and-price algorithm.*

practical situations, this will lead to that, either the visit is cancelled, or a substitute is assigned to the visit. An uncovered visit can also be dealt with by adjusting the original constraints, so that the visit can be fitted into the schedule. Constraints are adjusted by either reducing the duration of the visit, extending the time window of the visit, or extending the duty hours of one or more of the home carers. This happens frequently in practice. However, any of these adjustments decreases the overall quality of the schedule. In the solution method that we have developed, we have chosen not to adjust constraints, thereby letting the constraint adjustment be manual post-processing work. This is supported by that it is difficult to put a quantitative penalty on all possible adjustments before solving. The number of constraint adjustments for our solutions will hence always be equal to zero.

The total travelling time is measured in minutes for all home carers for the whole daily schedule. Minimising the total travelling time is not as important as minimising the two other measurements. Still, keeping the total travelling time to a minimum is preferred. The total travelling time may be slightly larger in our developed approach than in current practice, as more visits are included in the schedule. The results from the computational experiments are shown in Table 6.1.

The results clearly suggest that solution quality can be improved by the developed branch-and-price algorithm. There is a significant decrease in the number of uncovered visits (from 5% of the visits to 3% on average) and a large drop in the number of constraint adjustments (from 15 on average to zero). Table 6.1 also reveals that the improved solution quality can be obtained in reasonable time.

The constraint adjustments in the solution from current practice cover both small (minor reductions of the travelling times) and severe modifications (significant enlargements of the time windows). They are consequences of the complexity found in the manual scheduling process. The manual planner cannot overlook all possible solutions of the whole daily schedule and thus has to modify the constraints in order to create a good schedule. The most important job for the manual planner is to avoid uncovered visits, so the planner is often willing to accept a rather large number of constraint adjustments.

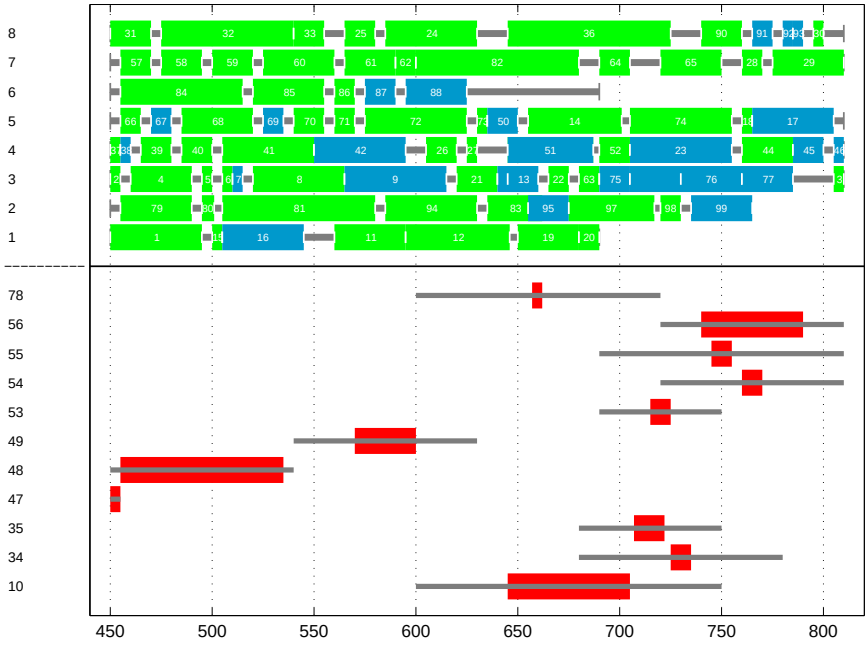
Figure 6.1(a) visualises the solution for the instance `ll1` from current practice. Time is running on the horizontal axis. Each feasible scheduled visit is represented by a green box, and the home carers are represented by underlying bars, depicting the duty hours of that home carer. The red boxes below the line are uncovered visits with the bars showing the time window of the visits. The blue visits have had their constraints adjusted. From Table 6.1, we can see that this sums to 26 constraint adjustments. As also visible in Figure 6.1(a), there are eleven uncovered visits.

Figure 6.1(b) visualises the solution to `ll1` for the developed branch-and-price algorithm. The uncovered visits are subject to skill requirements and therefore some home carer/visit combinations are not possible.

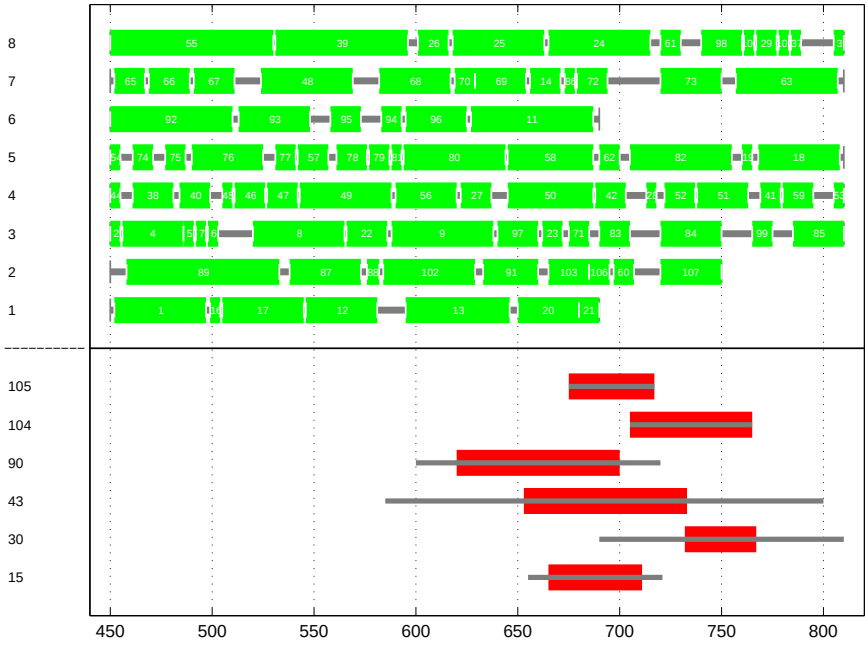
6.2 Subsequence removal

The subsequence generation approach used in Paper C and Paper D works with a limited set of subsequences, thereby maintaining a beneficial severe restriction of the solution space. The limited set of subsequences is then expanded whenever the algorithm identifies a subsequence as attractive. However, it is possible that the subsequence identification procedure guesses wrong and at one point identifies a subsequence as attractive, which then later turns out to be not attractive. This would mean that the LP relaxation would contain (and be slowed down by) a lot of unnecessary columns—all the columns containing the subsequence in question.

Therefore, we have developed a *subsequence removal* procedure, that can remove subsequences from the limited set of subsequences. Subsequences that have not appeared (i.e. columns containing the subsequence have not been selected even fractionally) in the LP solution for a given span of iterations are removed. When a subsequence is removed, all columns containing the subsequence are removed from the LP relaxation.



(a) Solution from current practice for `lll`.



(b) Solution from the developed solution algorithm for `lll`.

Figure 6.1: *Comparison of solutions for* `lll`.

Table 6.2 shows test runs for the 19 data instances. The data instances are described in Paper C and Paper D. The table should be compared to Table D.2.

For each instance we show the objective value of the best integer solution and the time it took to reach this solution. We show an indication of the quality of this solution through a comparison with objective function value of the relaxed master problem (root LP) as well as the objective function value of the relaxed master problem obtained using standard column generation. Standard column generation does not limit the possible subsequences for a flight. On all instances the column generation procedure timed out. Table 6.2 shows the objective function value of the root node at time out (cg LP). We also provide the gap between the objective value of our best integer solution and the column generation root LP value at time out. Furthermore, we present the number of uncovered flights in our solution (UF), the number of branch-and-bound nodes, and the total run time for the algorithm.

In Table 6.3 we show the accumulated numbers over all instances. The integer objective values and the LP objective values have been divided by 10^8. The integer objective value, the root LP objective value, the gap, and the number of uncovered flights, are all lower when subsequence removal is not used. Furthermore, the run times are significantly lower when subsequence removal is not used. Only on the number of branch-and-bound nodes, subsequence removal seems to have a positive effect. The overall immediate conclusion to be drawn, must be, that subsequence removal is not worth the overhead associated with the procedure. At least more investigations and more work on the procedure must be carried out.

instance	Best Integer		Statistics					
	obj	time (s)	root LP	cg LP	gap (%)	UF	nodes	time (s)
w08r01a	6.13041e+08	1027	6.13542e+08	6.02042e+08	1.83	6	77	1027.01
w08r01b	3.07036e+08	70.54	3.07677e+08	3.00034e+08	2.33	3	5	70.54
w08r01c	5.06036e+08	52.89	5.10038e+08	3.00031e+08	68.66	5	3	52.89
w08r01d	2.11033e+08	1010.45	2.12434e+08	2.03031e+08	3.94	2	43	1091.72
w08r01e	8.23035e+08	1147.06	8.22035e+08	6.05033e+08	36.03	8	55	1147.1
w08r02a	5.00023e+08	19.99	5.00023e+08	5.00021e+08	0.00	5	3	19.99
w08r02b	3.00023e+08	220.44	3.00023e+08	3.00023e+08	0.00	3	33	220.44
w08r02c	8.00019e+08	84.84	8.00521e+08	4.00019e+08	100.0	8	13	84.85
w08r02d	3.00029e+08	693.66	3.0003e+08	3.00025e+08	0.00	3	57	693.67
w08r02e	5.20032e+08	821.31	5.20031e+08	5.12029e+08	1.56	5	53	821.34
w08r03a	8.01023e+08	55.19	8.03024e+08	4.0002e+08	100.25	8	9	55.2
w08r03b	2.11029e+08	277.76	2.13027e+08	2.00034e+08	5.5	2	29	277.77
w08r03c	2.12033e+08	401.54	2.1403e+08	2.02035e+08	4.95	2	39	401.55
w08r03d	3.12031e+08	249.98	3.12529e+08	3.00038e+08	4.0	3	29	249.98
w08r03e	8.2102e+08	8.01	8.2102e+08	4.11028e+08	99.75	8	1	8.01
w08r04a	4.10024e+08	12.77	4.10024e+08	4.01031e+08	2.24	4	1	12.77
w08r04b	4.17026e+08	126.67	4.18025e+08	4.07032e+08	2.46	4	15	126.67
w08r04c	3.16025e+08	29.38	3.16025e+08	3.05033e+08	3.6	3	5	29.38
w08r04d	8.09021e+08	24.23	8.10105e+08	4.00023e+08	102.24	8	5	24.23

Table 6.2: *Test runs with subsequence removal.*

	Best Integer		Statistics					
setting	obj	time (s)	root LP	cg LP	gap (%)	UF	nodes	time (s)
No removal	89.80	4531.57	91.87	70.49	470.23	88	867	4531.75
Removal	91.90	6333.71	92.04	70.49	539.34	90	475	6415.11

Table 6.3: *Statistics with accumulated numbers for test runs with and without subsequence removal.*

Chapter 7

Conclusions

In this chapter we will summarise the content of the thesis. We will list the main contributions found in the included papers, and we will discuss ideas for future research.

In the first part of the thesis we have formulated a generalised set partitioning model that has been able to capture all applications from the second part of the thesis. We have presented solution methods from the literature. The solution methods have mostly been centered around the column generation technique. In the last chapters of the first part, we have presented crew scheduling problems from the literature, and we have introduced the application that we have worked on in detail in the papers. The second part of the thesis is containing four scientific articles, which we will describe in the following. All papers have been submitted to international journals, however only two of the papers have made it to the end of the reviewing process at the time of writing. For three out of four papers the computational experiments have been performed on realistic test instances. For the last paper computational experiments have been carried out on extensions of well-known academic instances.

We have described the home care crew scheduling problem, where a staff of home carers has to be assigned a number of visits to patients' homes. This have been done in Paper A. The problem is a generalisation of the vehicle routing problem with time windows. The challenge when assigning visits to home carers

lies in the existence of soft preference constraints and in temporal dependency constraints. We have introduced a novel visit clustering approach based on the soft preference constraints. The visit clusters are expanded when needed. The visit clustering decreases run times significantly, and only gives a loss of quality for few instances. Furthermore, the visit clustering allows us to find solutions to larger problem instances, which cannot be solved to optimality.

In Paper B we have formulated the vehicle routing problem with time windows and temporal dependencies—an extension of the well-known vehicle routing problem with time windows. Test instances are generated in order to study the impact of the temporal dependencies. Different variants of a branch-and-price solution approach have been compared. The result of the comparison is that it is best from a performance point of view to either relax the temporal dependencies from the master problem, or to have only the most violated constraints present in a time-indexed master problem.

We have proposed a novel solution method for the airline crew pairing problem in Paper C. In the solution method the number of subsequences for a flight is severely limited, thereby significantly decreasing the number of possible columns. Some optimal subsequences might have been omitted by this, and therefore we have sought to generate such subsequences based on the information in the dual vector. We have benchmarked the subsequence generation approach against a standard column generation approach on real-life test instances. The LP solutions from the subsequence generation approach are less fractional, but it comes at the cost of a worse solution quality.

In Paper D we have continued to work on the airline crew pairing problem. We have developed a new integer programming framework based on the concept of subsequence generation. Benchmarking on real-life test instances against standard column generation have showed that the developed framework is reasonable and an interesting alternative to standard column generation.

7.1 Main contributions

We will here list the main contributions of the thesis. These main contributions are extracted from the individual papers from Part II.

- Solution of real-life instances from home care with results superior to current practice.
- A novel preference-based clustering approach for scheduling of home care

workers. The clustering approach allows for solving larger instances and decreases run times significantly with a loss of quality on only a few instances.

- A time window branching scheme that enforces generalised precedence constraints. The branching scheme is used both for academic data sets and for real-life test instances.
- It has been shown that when considering the synchronisation temporal dependency, the general branching scheme for generalised precedence constraint is as tight as a branching scheme tailored for synchronisation.
- A new approach for the airline crew pairing problem has been developed. The core of the approach is subsequence generation as opposed to the standard column generation.
- The subsequence generation approach has been embedded in a novel branch-and-price-like framework in order to find integer solutions to the airline crew pairing problem.
- Subsequence generation as well as the subsequence generation integer framework have been tested on real-life crew pairing instances and compared to standard column generation.

7.2 Future research

Future research directions for the specific projects are described in the individual papers of Part II. Here, we will mention the most important ones, and also point out interesting future research on a more general level. For further ideas for future research we refer to the papers.

An area of research that would be exciting to explore is the applicability of the developed methods on other problems. We think specifically on the optimisation-based heuristics: Preference-based clustering and subsequence generation. A necessity for preference-based clustering to be successful for a given application, is that the objective function of the problem has terms with preferences or something that can be converted to a preference-like measure. For subsequence generation to work for a given problem, the problem must hold some of the same structural properties that the airline crew pairing problem holds, namely that subsequences can be ordered on their objective function contribution. For the airline crew pairing problem, the ordering is, that a subsequence with a short idle time is preferred to a subsequence with a longer idle time.

Disruption management and recovery problems have normally been the domain of heuristics as solutions are often needed within seconds or minutes. The optimisation-based heuristics proposed in this thesis have run times that are able to compete with the run times of pure heuristics. Hence, it would be very interesting to try to apply these optimisation-based heuristics for disruption management problems. A successful attempt in this area is Rezanova and Ryan (2010).

Considering the subsequence generation approach, post-result analysis showed that a potential gain was in the subsequence identification. This is a core part of the method and improvements in this part are predicted to have a great positive effect on the overall method.

For the preference-based task clustering method, new clustering schemes could be invented accompanied by cluster expansion schemes. For fixed-size clusters (see Paper A), it could be interesting to look into what determines a good cluster size. Perhaps it is possible to find a good cluster size as a function of number of tasks and number of available crew.

Bibliography

Abbink, E., M. Fischetti, L. Kroon, G. Timmer, and M. Vromans (2005). "Reinventing Crew Scheduling at Netherlands Railways". *Interfaces* 35.5, pp. 393–401.

Achterberg, T., T. Koch, and A. Martin (2005b). "Branching rules revisited". *Operations Research Letters* 33.1, pp. 42–54.

Andersen, E. D. (2006). *Linear optimization: Theory, methods, and extensions.* MOSEK.

Avella, P., M. Boccia, and A. Sforza (2004). "Solving a fuel delivery problem by heuristic and exact approaches". *European Journal of Operational Research* 152.1, pp. 170–179.

Avella, P., B. D'Auria, and S. Salerno (2006). "A LP-based heuristic for a time-constrained routing problem". *European Journal of Operational Research* 173.1, pp. 120–124.

Balas, E. and M. W. Padberg (1976). "Set Partitioning: A Survey". *SIAM Review* 18.4, pp. 710–760.

Balinski, M. L. and R. E. Quandt (1964). "On an Integer Program for a Delivery Problem". *Operations Research* 12.2, pp. 300–304.

Barnhart, C., E. Johnson, G. Nemhauser, M. Savelsbergh, and P. Vance (1998). "Branch-and-Price: Column Generation For Solving Huge Integer Programs". *Operations Research* 46.3, pp. 316–329.

Barnhart, C., A. M. Cohn, E. J. Johnson, D. Klabjan, G. L. Nemhauser, and P. H. Vance (2003). "Airline Crew Scheduling". In: *Handbook of Transportation Science.* Ed. by R. W. Hall. 2nd ed. Norwell: Kluwer Academic Publishers. Chap. 14, pp. 517–560.

Begur, S. V., D. M. Miller, and J. R. Weaver (1997). "An integrated spatial DSS for scheduling and routing home-health-care nurses". *Interfaces* 27.4, pp. 35–48.

Berge, C. (1972). "Balanced matrices". *Mathematical Programming* 2.1, pp. 19–31.

Bertels, S. and T. Fahle (2006). "A hybrid setup for a hybrid scenario: combining heuristics for the home health care problem". *Computers and Operations Research* 33.10, pp. 2866–2890.

Bredström, D. and M. Rönnqvist (Feb. 2007). *A branch and price algorithm for the combined vehicle routing and scheduling problem with synchronization constraints*. Discussion Papers 2007/7. Department of Finance, Management Science, Norwegian School of Economics, and Business Administration.

Bredström, D. and M. Rönnqvist (2008). "Combined vehicle routing and scheduling with temporal precedence and synchronization constraints". *European Journal of Operational Research* 191.1, pp. 19–31.

Brønmo, G., B. Nygreen, and J. Lysgaard (2010). "Column generation approaches to ship scheduling with flexible cargo sizes". *European Journal of Operational Research* 200.1, pp. 139–150.

Brucker, P. and S. Knust (2006). *Complex Scheduling*. Berlin: Springer.

Brucker, P. and S. Knust (2010). "On the complexity of scheduling". In: *Introduction to Scheduling*. Ed. by Y. Robert and F. Vivien. Boca Raton: CRC Press. Chap. 1, pp. 1–22.

Burke, E. K. and G. Kendall, eds. (2005). *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*. New York: Springer.

Burke, E. K., P. De Causmaecker, G. V. Berghe, and H. Van Landeghem (2004). "The State of the Art of Nurse Rostering". *Journal of Scheduling* 7.6, pp. 441–499.

Butchers, E. R., P. R. Day, A. P. Goldie, S. Miller, J. A. Meyer, D. M. Ryan, A. C. Scott, and C. A. Wallace (2001). "Optimized crew scheduling at Air New Zealand". *Interfaces* 31.1, pp. 30–56.

Caprara, A., M. Fischetti, P. Toth, D. Vigo, and P. L. Guida (1997). "Algorithms for railway crew management". *Mathematical Programming* 79.1–3, pp. 125–141.

Caprara, A., L. Kroon, M. Monaci, M. Peeters, and P. Toth (2007). "Passenger Railway Optimization". In: *Transportation*. Ed. by C. Barnhart and G. Laporte. Vol. 14. Handbooks in Operations Research and Management Science. Elsevier. Chap. 3, pp. 129–187.

Chabrier, A. (2006). "Vehicle Routing Problem with elementary shortest path based column generation". *Computers and Operations Research* 33.10, pp. 2972–2990.

Cheng, E. and J. L. Rich (1998). *A Home Health Care Routing and Scheduling Problem*. Tech. rep. Department of CAAM, Rice University.

Clausen, T. (2010). "Airport Ground Staff Scheduling". PhD thesis. Technical University of Denmark.

Conforti, M., G. Cornuéjols, A. Kapoor, and K. Vuskovic (2001). "Perfect, ideal and balanced matrices". *European Journal of Operational Research* 133.3, pp. 455–461.

Cook, W. J., W. H. Cunningham, W. R. Pulleyblank, and A. Schrijver (1998). *Compinatorial Optimization*. New York: John Wiley & Sons, Inc.

Cordeau, J.-F., G. Desaulniers, J. Desrosiers, M. M. Solomon, and F. Soumis (2002). "VRP with Time Windows". In: *The Vehicle Routing Problem*. Ed. by P. Toth and D. Vigo. Society for Industrial and Applied Mathematics. Chap. 7, pp. 176–213.

Cordeau, J.-F., G. Laporte, M. W. Savelsbergh, and D. Vigo (2007). "Vehicle Routing". In: *Transportation*. Ed. by C. Barnhart and G. Laporte. Vol. 14. Handbooks in Operations Research and Management Science. Elsevier. Chap. 6, pp. 367 –428.

Cormen, T. H., C. Stein, R. L. Rivest, and C. E. Leiserson (2001). *Introduction to Algorithms*. 2nd ed. Cambridge: The MIT Press.

Crescenzi, P. and V. Kann (1997). "Approximation on the web: A compendium of NP optimization problems". In: *Randomization and Approximation Techniques in Computer Science*. Ed. by J. Rolim. Vol. 1269. Lecture Notes in Computer Science. Springer, Berlin, pp. 111–118.

Dantzig, G. B. and P. Wolfe (1960b). "Decomposition Principle for Linear Programs". *Operations Research* 8.1, pp. 101–111.

Darby-Dowman, K. and G. Mitra (1985). "An extension of set partitioning with application to scheduling problems". *European Journal of Operational Research* 21.2, pp. 200 –205.

Desaulniers, G., J. Desrosiers, Y. Dumas, S. Marc, B. Rioux, M. Solomon, and F. Soumis (1997). "Crew pairing at Air France". *European Journal of Operational Research* 97.2, pp. 245–259.

Desaulniers, G., J. Desrosiers, M. Gamache, and F. Soumis (1998). "Crew Scheduling in Air Transportation". In: *Fleet Management and Logistics*. Ed. by T. G. Crainic and G. Laporte. Boston: Kluwer Academic Publishers, pp. 169–185.

Desrochers, M. and F. Soumis (1989). "A Column Generation Approach to the Urban Transit Crew Scheduling Problem". *Transportation Science* 23.1, pp. 1–13.

Desrochers, M., J. Desrosiers, and M. Solomon (1992b). "A new optimization algorithm for the vehicle routing problem with time windows". *Operations Research* 40.2, pp. 342–354.

Desrosiers, J. and M. E. Lübbecke (2005). "A Primer in Column Generation". In: *Column Generation*. Ed. by G. Desaulniers, J. Desrosiers, and M. Solomon. GERAD 25th Anniversary Series. Springer, pp. 1–32.

Desrosiers, J., F. Soumis, and M. Desrochers (1984). "Routing with Time Windows by Column Generation". *Networks* 14.4, pp. 545–565.

Desrosiers, J., Y. Dumas, M. M. Solomon, and F. Soumis (1995). "Time constrained routing and scheduling". In: *Network Routing*. Ed. by M. Ball, T. Magnanti, C. Monma, and G. Nemhauser. Vol. 8. Handbooks in Operations Research and Management Science. Elsevier. Chap. 2, pp. 35–139.

Dirickx, Y. M. I. and L. P. Jennergren (1979). *Systems Analysis by Multilevel Methods: With Applications to Economics and Management*. Chichester: John Wiley & Sons.

Dohn, A. (2010). "Rostering and Task Scheduling: Applications in Manpower Planning". PhD thesis. Department of Management Engineering, Technical University of Denmark.

Dohn, A., M. S. Rasmussen, T. Justesen, and J. Larsen (2008b). "The Home Care Crew Scheduling Problem". In: *ICAOR '08 – Proceedings, 1st International Conference on Applied Operational Research*. Ed. by K. Sheibani. Tadbir Institute for Operational Research, pp. 1–8.

Dohn, A., M. S. Rasmussen, T. Justesen, and J. Larsen (2008a). "The Home Care Crew Scheduling Problem". *Orbit* 13, pp. 19–23.

Dohn, A., E. Kolind, and J. Clausen (2009a). "The manpower allocation problem with time windows and job-teaming constraints: A branch-and-price approach". *Computers and Operations Research* 36.4, pp. 1145–1157.

Dohn, A., M. S. Rasmussen, and J. Larsen (2009c). *The Vehicle Routing Problem with Time Windows and Temporal Dependencies*. Tech. rep. Department of Management Engineering, Technical University of Denmark.

Dohn, A., A. Mason, and D. Ryan (2010). *A Generic Solution Approach to Nurse Rostering*. Tech. rep. Department of Management Engineering, Technical University of Denmark.

Dohn, A., M. S. Rasmussen, and J. Larsen (2011). "The Vehicle Routing Problem with Time Windows and Temporal Dependencies". *Networks*. (Accepted for publication).

Dorndorf, U. (2002). *Project Scheduling with Time Windows: From Theory to Applications*. Heidelberg: Physica-Verlag.

Ernst, A. T., H. Jiang, M. Krishnamoorthy, and D. Sier (2004a). "Staff scheduling and rostering: A review of applications, methods and models". *European Journal of Operational Research* 153.1, pp. 3–27.

Ernst, A., H. Jiang, M. Krishnamoorthy, B. Owens, and D. Sier (2004b). "An Annotated Bibliography of Personnel Scheduling and Rostering". *Annals of Operations Research* 127.1–4, pp. 21–144.

Eveborn, P., P. Flisberg, and M. Rönnqvist (2006). "Laps Care—an operational system for staff planning of home care". *European Journal of Operational Research* 171.3, pp. 962–976.

Feillet, D., P. Dejax, M. Gendreau, and C. Gueguen (2004). "An exact algorithm for the elementary shortest path problem with resource constraints: application to some vehicle routing problems". *Networks* 44.3, pp. 216–29.

Garey, M. and D. Johnson (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York: W.H. Freeman and Co.

Garfinkel, R. and G. Nemhauser (1969). "The Set-Partitioning Problem: Set Covering with Equality Constraints". *Operations Research* 17.5, pp. 848 – 856.

Gélinas, S., M. Desrochers, J. Desrosiers, and M. Solomon (1995). "A new branching strategy for time constrained routing problems with application to backhauling". *Annals of Operations Research* 61, pp. 91–109.

Gilmore, P. C. and R. E. Gomory (1961). "A Linear Programming Approach to the Cutting-Stock Problem". *Operations Research* 9.6, pp. 849–859.

Gilmore, P. C. and R. E. Gomory (1963). "A Linear Programming Approach to the Cutting Stock Problem – Part II". *Operations Research* 11.6, pp. 863–888.

Glover, F. and G. A. Kochenberger, eds. (2003). *Handbook of Metaheuristics*. Norwell: Kluwer Academic Publishers.

Gopalakrishnan, B. and E. L. Johnson (2005). "Airline Crew Scheduling: State-of-the-Art". *Annals of Operations Research* 140.1, pp. 305–337.

Hillier, F. S. and G. J. Lieberman (2005). *Introduction to Operations Research*. 8th ed. McGraw-Hill.

Hochbaum, D. S., ed. (1997). *Approximation Algorithms for NP-hard Problems*. Boston: PWS Publishing Company.

Hoffman, A. and J. Kruskal (1956). "Integral Boundary Points of Convex Polyhedra". In: *Linear Inequalities and Related Systems*. Ed. by H. Kuhn and A. Tucker. Princeton University Press, Princeton, pp. 223–246.

Huisman, D. (2007). "A column generation approach for the rail crew re-scheduling problem". *European Journal of Operational Research* 180.1, pp. 163–173.

Ioachim, I., J. Desrosiers, F. Soumis, and N. Bélanger (1999). "Fleet assignment and routing with schedule synchronization constraints". *European Journal of Operational Research* 119.1, pp. 75–90.

Irnich, S. and G. Desaulniers (2005). "Shortest Path Problems with Resource Constraints". In: *Column Generation*. Ed. by G. Desaulniers, J. Desrosiers, and M. M. Solomon. GERAD 25th Anniversary Series. Springer. Chap. 2, pp. 33–65.

Kallehauge, B., J. Larsen, O. B. Madsen, and M. Solomon (2005). "The Vehicle Routing Problem with Time Windows". English. In: *Column Generation*. Ed. by G. Desaulniers, J. Desrosiers, and M. M. Solomon. GERAD 25th anniversary series. New York: Springer. Chap. 3, pp. 67–98.

Kalvelagen, E. (2003). *Dantzig-Wolfe Decomposition with Gams*. Tech. rep. Amsterdam Optimization Modeling Group LLC.

Kohl, N. (1995). "Exact Methods for Time Constrained Routing and Related Scheduling Problems". PhD thesis. Institute of Mathematical Modelling, Technical University of Denmark.

Kohl, N. and S. E. Karisch (2004). "Airline Crew Rostering: Problem Types, Modeling, and Optimization". *Annals of Operations Research* 127 (1), pp. 223–257.

Land, A. H. and A. G. Doig (1960). "An Automatic Method of Solving Discrete Programming Problems". *Econometrica* 28.3, pp. 497–520.

Larsen, J., A. Dohn, M. S. Rasmussen, and T. Justesen (2010). "The Home Care Crew Scheduling Problem". In: *Proceedings of the 8th International Conference on the Practice and Theory of Automated Timetabling (PATAT)*. Ed. by B. McCollum, E. Burke, and G. White.

Lasdon, L. S. (2002). *Optimization Theory for Large Systems*. 1970. Mineola: Dover Publications, Inc.

Lehman, A. (1979). "On the width-length inequality". *Mathematical Programming* 17.1, pp. 403–417.

Lessel, C. R. (2007). "Ruteplanlægning i hjemmeplejen [Routing in Home Care]". Master's thesis. Department of Informatics and Mathematical Modelling, Technical University of Denmark.

Li, Y., A. Lim, and B. Rodrigues (2005). "Manpower allocation with time windows and job-teaming constraints". *Naval Research Logistics* 52.4, pp. 302–311.

Lusby, R., A. Dohn, T. M. Range, and J. Larsen (2011). "A column generation-based heuristic for rostering with work patterns". *Journal of the Operational Research Society*, pp. 1–17.

Lusby, R. M., J. Larsen, M. Ehrgott, and D. Ryan (2009). "Railway track allocation: models and methods". *OR Spectrum*, pp. 1–41.

Martin, R. K. (1999). *Large Scale Linear and Integer Optimization: A Unified Approach*. Boston: Kluwer Academic Publishers.

Messer, R. (1994). *Linear Algebra: Gateway to Mathematics*. New York: Harper-Collins College Publishers.

Nemhauser, G. L. and L. A. Wolsey (1988). *Integer and Combinatorial Optimization*. New York: John Wiley & Sons, Inc.

Nissen, R. and K. Haase (2006). "Duty-period-based network model for crew rescheduling in European airlines". *Journal of Scheduling* 9.3, pp. 255–278.

Padberg, M. W. (1974). "Perfect zero-one matrices". *Mathematical Programming* 6.1, pp. 180–196.

Ralphs, T., L. Ladányi, and M. Saltzman (2003). "Parallel branch, cut, and price for large-scale discrete optimization". *Mathematical Programming* 98.1–3, pp. 253–280.

Rasmussen, M. S., D. M. Ryan, R. M. Lusby, and J. Larsen (2009a). "Solving the Airline Crew Pairing Problem using Subsequence Generation". In: *Proceedings of the 44th Annual conference of the Operational Research Society of New Zealand*.

Rasmussen, M. S., D. M. Ryan, R. M. Lusby, and J. Larsen (2010a). "Solving the Airline Crew Pairing Problem using Subsequence Generation". In: *Proceedings of the 8th International Conference on the Practice and Theory of Automated Timetabling (PATAT)*. Ed. by B. McCollum, E. Burke, and G. White.

Rasmussen, M. S., T. Justesen, A. Dohn, and J. Larsen (2010c). *The Home Care Crew Scheduling Problem: Preference-Based Visit Clustering and Temporal Dependencies*. Tech. rep. Department of Management Engineering, Technical University of Denmark.

Rasmussen, M. S., R. M. Lusby, D. M. Ryan, and J. Larsen (2011a). *A Subsequence Generation Approach for the Airline Crew Pairing Problem*. AGIFORS Anna Valicek Award 2011. (Submitted).

Rasmussen, M. S., R. M. Lusby, D. M. Ryan, and J. Larsen (2011b). "An IP Framework for the Crew Pairing Problem using Subsequence Generation". *Journal of the Operational Research Society*. (Submitted).

Rasmussen, M. S., R. M. Lusby, D. M. Ryan, and J. Larsen (2011c). *An IP Framework for the Crew Pairing Problem using Subsequence Generation*. Tech. rep. Department of Management Engineering, Technical University of Denmark.

Rasmussen, M. S., R. M. Lusby, D. M. Ryan, and J. Larsen (2011d). "Subsequence Generation for the Airline Crew Pairing Problem". *Transportation Research Part E*. (Submitted).

Rasmussen, M. S., R. M. Lusby, D. M. Ryan, and J. Larsen (2011e). *Subsequence Generation for the Airline Crew Pairing Problem*. Tech. rep. Department of Management Engineering, Technical University of Denmark.

Rasmussen, M. S., T. Justesen, A. Dohn, and J. Larsen (2011f). "The Home Care Crew Scheduling Problem: Preference-Based Visit Clustering and Temporal Dependencies". *European Journal of Operational Research*. (Accepted for publication).

Rezanova, N. J. (2009). "The Train Driver Recovery Problem: Solution Method and Decision Support System Framework". PhD thesis. Department of Management Engineering, Technical University of Denmark.

Rezanova, N. J. and D. M. Ryan (2010). "The train driver recovery problem-A set partitioning based model and solution method". *Computers and Operations Research* 37.5, pp. 845–856.

Rönnqvist, M. (1995). "A method for the cutting stock problem with different qualities". *European Journal of Operational Research* 83.1, pp. 57–68.

Rossi, F. and S. Smriglio (2001). "A set packing model for the ground holding problem in congested networks". *European Journal of Operational Research* 131.2, pp. 400–416.

Rubin, J. (1973). "A Technique for the Solution of Massive Set Covering Problems, with Application to Airline Crew Scheduling." *Transportation Science* 7.1, pp. 34–48.

Ryan, D. M. and J. C. Falkner (1988). "On the integer properties of scheduling set partitioning models". *European Journal of Operational Research* 35.3, pp. 442 –456.

Ryan, D. M. (1992b). "The solution of massive generalized set partitioning problems in aircrew rostering". *Journal of the Operational Research Society* 43.5, pp. 459–467.

Ryan, D. M. and B. Foster (1981). "An integer programming approach to scheduling". *Computer Scheduling of Public Transport. Urban Passenger Vehicle and Crew Scheduling. Proceedings of an International Workshop*, pp. 269–280.

Solomon, M. M. (1987b). "Algorithms for the Vehicle Routing and Scheduling Problems with Time Window Constraints". *Operations Research* 35.2, pp. 254–265.

Tebboth, J. R. (2001). "A Computational Study of Dantzig-Wolfe Decomposition". PhD thesis. University of Buckingham.

Thomsen, K. (2006). "Optimization on Home Care". Master's thesis. Department of Informatics and Mathematical Modelling, Technical University of Denmark.

Vanderbeck, F. and L. A. Wolsey (1996). "An exact algorithm for IP column generation". *Operations Research Letters* 19.4, pp. 151–159.

Vazirani, V. V. (2001). *Approximation algorithms*. Berlin: Springer.

Vemuganti, R. (1998). "Applications of Set Covering, Set Packing and Set Partitioning Models: A Survey". In: *Handbook of Combinatorial Optimization*. Ed. by D.-Z. Du and P. M. Pardalos. Vol. 1. Kluwer Academic Publishers, pp. 573–746.

Wolsey, L. A. (1998). *Integer Programming*. New York: John Wiley & Sons, Inc.

Part II

Scientific Papers

Appendix A

The Home Care Crew Scheduling Problem: Preference-Based Visit Clustering and Temporal Dependencies

---

Matias Sevel Rasmussen, Tor Justesen, Anders Dohn, and Jesper Larsen

Accepted for publication in: *European Journal of Operational Research* (2011).

The Home Care Crew Scheduling Problem: Preference-Based Visit Clustering and Temporal Dependencies*

Matias Sevel Rasmussen[1], Tor Justesen[1], Anders Dohn[1], and Jesper Larsen[1]

In the Home Care Crew Scheduling Problem a staff of home carers has to be assigned a number of visits to patients' homes, such that the overall service level is maximised. The problem is a generalisation of the vehicle routing problem with time windows. Required travel time between visits and time windows of the visits must be respected. The challenge when assigning visits to home carers lies in the existence of soft preference constraints and in temporal dependencies between the start times of visits.

We model the problem as a set partitioning problem with side constraints and develop an exact branch-and-price solution algorithm, as this method has previously given solid results for classical vehicle routing problems. Temporal dependencies are modelled as generalised precedence constraints and enforced through the branching. We introduce a novel visit clustering approach based on the soft preference constraints. The algorithm is tested both on real-life problem instances and on generated test instances inspired by realistic settings. The use of the specialised branching scheme on real-life problems is novel. The visit clustering decreases run times significantly, and only gives a loss of quality for few instances. Furthermore, the visit clustering allows us to find solutions to larger problem instances, which cannot be solved to optimality.

Keywords: Home care, Health care, Home health care, Crew scheduling, Vehicle routing, Vehicle routing with time windows, Visit clustering, Clustering, Preferences, Generalised precedence constraints, Temporal dependency, Temporal dependencies, Synchronisation, Real-life application, Branch-and-price, Column generation, Dantzig-Wolfe decomposition, Set partitioning, Scheduling, Routing, Integer programming

*Accepted for publication in: *European Journal of Operational Research* (2011).

[1]Department of Management Engineering, Technical University of Denmark, Produktionstorvet, Building 424, DK-2800 Kgs. Lyngby, Denmark.

A.1 Introduction

The Home Care Crew Scheduling Problem (HCCSP) described in this paper has its origin in the Danish health care system. The home care service was introduced in 1958 and since then, there has been a constant increase in the number of services offered. The primary purpose is to give senior citizens and disabled citizens the opportunity to stay in their own home for as long as possible. The HCCSP is the problem of scheduling home carers in a way that maximises the service level, possibly even at a reduced cost. In 2005 in Denmark a little more than DKK 28 billion were spent in the whole area of elderly care, and around 200 000 senior citizens received home care, totalling between 1.0 and 1.1 million service hours per week at the senior citizens, see Sekretariatet for ministerudvalget vedrørende kvalitet i den offentlige sektor (2006). Therefore, the HCCSP is a very important optimisation problem.

When a citizen applies for home care service, a preadmission assessment is initiated. The result of the assessment is a list of granted services. The services may include cleaning, laundry assistance, preparing food, and support for other everyday tasks. They may also include assistance with respect to more personal needs, e.g. getting out of bed, bathing, dressing, and dosing medicine. As a consequence of the variety of services offered, people with many different competences are employed as home carers.

Given a list of services for each of the implicated citizens, a long-term plan is prepared. In the long-term plan, each service is assigned to specific time windows, which are repeated as frequently as the preadmission assessment prescribes. The citizens are informed of the long-term plan, and hence they know approximately when they can expect visits from home carers. From the long-term plan, a specific schedule is created on a daily basis. In the daily problem, home carers are assigned to visits. A route is built for each home carer, respecting the competence requirements and time window of each visit and working hours of the home carer.

In the following, we restrict ourselves to looking at the daily scheduling problem only. The problem is a crew scheduling problem with strong ties to vehicle routing with time windows. However, we have a number of complicating issues that differentiates the problem from a traditional vehicle routing problem. One complication is the multi-criteria nature of the objective function. It is, naturally, important to minimise the overall operation costs. However, the operation costs are not very flexible in the daily scheduling problem. More important is it to maximise the level of service that can be provided. The service level depends on a number of different factors. Often, it is impossible to fit all visits into the schedule in their designated time windows. Hence, some visits may have to

be rescheduled or cancelled. In our solutions, a visit is either scheduled within the given restrictions or marked as uncovered. The manual planner will deal with uncovered visits appropriately. The main priority is to leave as few visits uncovered as possible. Also, all visits are associated with a priority and it is important to only reschedule and cancel less significant visits. Furthermore, it is important to service each citizen from a small subgroup of the whole workforce (the so-called preferred home carers), as this establishes confidence with the citizen. As mentioned, home carers have different working hours, so that both early morning and evening can be covered, and home carers have different compentences in order to cover the different competence requirements of the visits. Also home carers can have different means of transportation, that is a home carer either drives, bicycles or walks. Another complication, compared to traditional vehicle routing, is that we have temporal dependencies between visits. The temporal dependencies constrain and interconnect the routes of the home carers. One temporal dependency is *synchronisation*. For instance, synchronisation of two visits is used when a citizen needs help to get in or out of bed. Here two home carers are required at the same time. The *overlap* temporal dependency is, for example, seen when a home carer has to pass on a key to the next home carer at a citizen. The temporal dependencies *minimum difference* and *maximum difference* are for example used when a home carer starts the washing machine at a citizen and a following home carer (perhaps the same home carer) empties the washing machine at the same citizen. The visits need to be separated by, say, two hours, but not more than four hours. This can of course also be accomplished by time windows, but that would limit the flexibility.

HCCSP generalises the Vehicle Routing Problem with Time Windows (VRPTW) for which column generation solution algorithms have proven successful, see Kallehauge et al. (2005). Therefore, we model HCCSP as a Set Partitioning Problem (SPP) with side constraints and develop a branch-and-price solution algorithm. Temporal dependencies are modelled by a single type of constraints: generalised precedence constraints. These constraints are enforced through the branching. Different visit clustering schemes are devised for the problem. The schemes are based on the existence of soft preference constraints. The visit clustering schemes for the exact branch-and-price framework are novel. The visit clustering will naturally compromise optimality, but will allow us to solve larger instances. We will compare the different visit clustering schemes by testing them both on real-life problem instances and on generated test instances inspired by realistic settings. To our knowledge, we are the first to enforce generalised precedence constraints in the branching for real-life problems. The contribution of this paper is hence twofold. Firstly, we devise visit clustering schemes for the problem, and secondly, we enforce generalised precedence constraints in the branching for the first time for real-life problems.

Optimisation methods for crew scheduling are widely used and described in the literature, especially regarding air crew scheduling. However, when it comes to scheduling of home care workers the literature is sparse. This work builds on top of two recent Master's theses, Lessel (2007) and Thomsen (2006). The most recent of these, Lessel (2007), uses a two-phase approach which first groups the visits based on geographical position, competences, and preferences. A home carer is associated to each group and the second phase considers each group as a Travelling Salesman Problem with Time Windows (TSPTW).

The other thesis, Thomsen (2006), treats the problem as a Vehicle Routing Problem with Time Windows and Shared Visits (VRPTWSV) and uses an insertion heuristic to feed a tabu search with initial solutions. The models and solution methods in Lessel (2007) and Thomsen (2006) can only handle connected visits where two home carers are at the same time at the citizen.

With offset in the Swedish home care system, Eveborn et al. (2006) describe a system in operation. They use a Set Partitioning Problem model and solve the problem heuristically by using a repeated matching approach. The matching combines home carers with visits. Eveborn et al. (2006) report that the travelling time savings in a moderate guess are 20% and that the time savings on the planning correspond to 7% of the total working time. Bredström and Rönnqvist (2008) show a mathematical model that can handle synchronisation constraints and precedence constraints between pairs of visits. The model is a VRPTW with the additional synchronisation and precedence constraints that tie the routes together. They solve the model using a heuristic. Bredström and Rönnqvist (2007) develop a branch-and-price algorithm to solve the model of Bredström and Rönnqvist (2008), but without the precedence constraints. The model is decomposed to an SPP and the integrality requirement on the binary decision variables is relaxed. Also, the synchronisation constraints are removed from the SPP. Instead, integrality and synchronisation are handled by the branching, and to our knowledge they are the first to use a non-heuristic solution approach to home care problems. Their subproblem is an Elementary Shortest Path with Time Windows (ESPPTW).

Bertels and Fahle (2006) use a combination of linear programming, constraint programming and metaheuristics for solving what they call the Home Health Care Problem. However, they do not incorporate connected visits, which makes their approach less interesting for our situation. Begur et al. (1997) describe a decision support system (DSS) in use in the United States. The DSS provides routes for home carers by using GIS systems. Their model is a Vehicle Routing Problem (VRP) without time windows and without shared visits, which again is not suitable for our needs. Cheng and Rich (1998) describe the Home Health Care Routing and Scheduling Problem which they model as a Vehicle Routing Problem with Time Windows (VRPTW). They distinguish between full-time

and part-time home carers. They use a two-phase heuristic approach, in which they first find an initial solution using a greedy heuristic. Next, the solution is improved using local search. The model does not include temporal connections between visits.

Related to the HCCSP is the Manpower Allocation Problem with Time Windows (MAPTW). A demanded number of servicemen must be allocated to each location within the time windows. Primarily the number of used servicemen must be minimised, and secondarily the used travel time. The jobs have different locations, skill requirements, and time windows. This problem is dealt with by Lim et al. (2004). More closely related to HCCSP is the Manpower Allocation Problem with Time Windows and job-Teaming Constraints (MAPTWTC). Li et al. (2005) present a construction heuristic combined with simulated annealing for solving MAPTWTC instances. Their model adds synchronisation constraints to the model of Lim et al. (2004), but does not include precedence constraints. MAPTWTC is also solved in Dohn et al. (2009a), again with multiple teams cooperating on tasks. An exact solution approach is introduced. They decompose to a set partitioning problem and develop a branch-and-price algorithm. The subproblem in the column generation is an ESPPTW.

The HCCSP can be seen as a VRPTW, but with the addition of the complicating connections between visits, and with another objective than the regular minimisation of total distance. When only synchronised visits are considered as connection type, the problem can be referred to as shared visits, yielding the VRPTWSV. The literature on VRPTW is huge. We refer to Kallehauge et al. (2005) and Cordeau et al. (2002). Recently, a variant of VRPTW very similar to the HCCSP has been described in Dohn et al. (2011). The authors formalise the concept of temporal dependencies in the Vehicle Routing Problem with Time Windows and Temporal Dependencies (VRPTWTD) and investigate the effectiveness of different formulations and solution approaches. HCCSP complicates VRPTWTD by adding visit preferences, visit priorities, visit cancellations, enforcement of temporal dependencies to cancelled visits, different working hours, different competences, and different means of transportation for the home carers.

The remainder of the paper is organised as follows. In Section A.2, we present an IP formulation of HCCSP. In Section A.3, we introduce a decomposed version of this formulation. In Section A.4, we present the specialised branching scheme used. Section A.5 introduces clustering of visits and other methods to decrease the solve time of the pricing problem. Real-life and generated test instances are described in Section A.6. In Section A.7, we present results from test runs on these instances. Finally, in Section A.8 we conclude on our work and suggest topics for future research.

A.2 Problem formulation

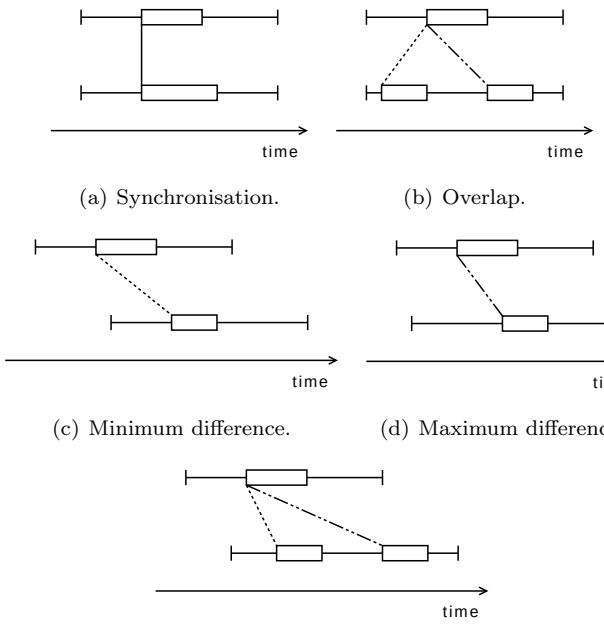
The set of home carers is denoted $\mathcal{K}$, and the set of visits at the citizens is denoted $\mathcal{C} = \{1, \ldots, n-1\}$. For each visit $i \in \mathcal{C}$ a time window is defined as $[\alpha_i, \beta_i]$, where $\alpha_i \geq 0$ and $\beta_i \geq 0$ specify the earliest respectively latest possible start time of the visit. For algorithmic reasons, we introduce artificial visits 0^k and n^k as the *start visit* respectively *end visit* for home carer $k \in \mathcal{K}$, and we define $\mathcal{N}^k = \mathcal{C} \cup \{0^k, n^k\}$ as the set of all potential visits for home carer k. The duty length for each home carer $k \in \mathcal{K}$ is given by the time window $[\alpha_{0^k}, \beta_{0^k}] = [\alpha_{n^k}, \beta_{n^k}]$, i.e. home carer $k \in \mathcal{K}$ cannot start his or her duty before time $\alpha_{0^k} \geq 0$ and must have finished his or her last visit before time $\beta_{0^k} \geq 0$. The travel distance between a pair of visits (i, j) is given by the parameter s_{ij}^k. The parameter depends on $k \in \mathcal{K}$ as the home carers use different means of transportation. If it is not possible to travel directly between visits i and j for home carer k, then $s_{ij}^k = \infty$. We define $s_{ii}^k = \infty$, $\forall k \in \mathcal{K}, \forall i \in \mathcal{N}^k$. The parameter s_{ij}^k includes the duration (service time) of visit i. Travelling between any two visits i and j gives rise to the costs c_{ij}^k dependent on the home carer $k \in \mathcal{K}$. For any combination of $i \in \mathcal{C}$ and $k \in \mathcal{K}$ the parameter $\rho_i^k = 1$ if k can be assigned to visit i, $\rho_i^k = 0$ otherwise. Also, for any combination of $i \in \mathcal{C}$ and $k \in \mathcal{K}$ the preference parameter $\delta_i^k \in \mathbb{R}$ gives the cost for letting home carer k handle visit i. A negative cost means that we would like home carer k to handle visit i, whereas a positive cost means that we would prefer not to let home carer k handle visit i. The parameter γ_i is the priority of visit $i \in \mathcal{C}$, the higher, the more important.

As described in Section A.1, visits may be temporally dependent due to different home care needs. In order to make it easier for the manual planner to assign substitutes to the uncovered visits, it is required that visits, which have a temporal dependency to an uncovered visit, still respect the temporal dependency. In other words, a temporal dependency is still respected even if one of the visits is uncovered. Five types of temporal dependencies are often seen in practice. The five types can be seen in Figure A.1. These temporal dependencies can be modelled by introducing generalised precedence constraints of the form

$$\sigma_i + p_{ij} \leq \sigma_j \ ,$$

where σ_i denotes the start time of visit i, and $p_{ij} \in \mathbb{R}$ quantifies the required gap. The set of pairs of visits $(i, j) \in \mathcal{C} \times \mathcal{C}$ for which a generalised precedence constraint exists is denoted $\mathcal{P}$.

As can be seen, this constraint simply implies that j starts minimum p_{ij} time units after i. An often encountered example of a temporal dependency is that of synchronisation, see Figure A.1(a), where two visits are required to start at



(a) Synchronisation.

(b) Overlap.

(c) Minimum difference.

(d) Maximum difference.

(e) Min+max difference.

Figure A.1: *Five types of temporal dependencies. Each of the five subfigures shows the time windows of two visits i (top) and j (bottom) with a temporal dependency between them. Assuming some start time for visit i, the dotted line shows the earliest feasible start time for visit j, and the dashed dotted line shows the latest feasible start time. For synchronisation (a) the two lines coincide, and are drawn as one full line.*

the same time. The way to handle this is to add both (i, j) and (j, i) to $\mathcal{P}$ with $p_{ij} = p_{ji} = 0$. As also described by Dohn et al. (2011), Table A.1 shows how to model all the temporal dependencies of Figure A.1 with generalised precedence constraints. It can be seen that (a), (b) and (e) each requires two generalised precedence constraints, whereas (c) and (d) only need one each.

	Temporal dependency	p_{ij}	p_{ji}
(a)	Synchronisation	0	0
(b)	Overlap	$-\text{dur}_j$	$-\text{dur}_i$
(c)	Minimum difference	$\text{diff}_{\min}$	N/A
(d)	Maximum difference	N/A	$-\text{diff}_{\max}$
(e)	Minimum+maximum difference	$\text{diff}_{\min}$	$-\text{diff}_{\max}$

Table A.1: *Values for p_{ij} for the five temporal dependencies of Figure A.1. dur_i is the service time of visit i, $diff_{\min}$ is the minimum difference required and $diff_{\max}$ is the maximum difference required.*

A.2.1 Integer programme

To model the problem, three sets of decision variables are defined: the binary routing variables x_{ij}^k, the scheduling variables $t_i^k \in \mathbb{R}_+$ and the binary coverage variables y_i. $x_{ij}^k = 1$ if home carer $k \in \mathcal{K}$ goes directly from visit $i \in \mathcal{N}^k$ to $j \in \mathcal{N}^k$, and $x_{ij}^k = 0$ otherwise. The scheduling variable t_i^k is the time the home carer $k \in \mathcal{K}$ starts handling visit $i \in \mathcal{N}^k$. $t_i^k = 0$ if home carer k is not assigned to visit i. $y_i = 1$ if visit $i \in \mathcal{C}$ is not covered by any home carer and $y_i = 0$ otherwise. The weights w_1, w_2 and w_3 are used to control the objective function.

HCCSP can now be formulated as the integer programme given below. The formulation is very similar to the formulation in Bredström and Rönnqvist (2007).

$$\min w_1 \sum_{k\in\mathcal{K}} \sum_{i\in\mathcal{N}^k} \sum_{j\in\mathcal{N}^k} c_{ij}^k x_{ij}^k + w_2 \sum_{k\in\mathcal{K}} \sum_{i\in\mathcal{C}} \sum_{j\in\mathcal{N}^k} \delta_i^k x_{ij}^k + \quad w_3 \sum_{i\in\mathcal{C}} \gamma_i y_i \tag{A.1}$$

$$\text{s.t.} \sum_{k\in\mathcal{K}} \sum_{j\in\mathcal{N}^k} x_{ij}^k + y_i = 1 \qquad \forall i \in \mathcal{C} \tag{A.2}$$

$$\sum_{j\in\mathcal{N}^k} x_{ij}^k \le \rho_i^k \qquad \forall k \in \mathcal{K}, \forall i \in \mathcal{C} \tag{A.3}$$

$$\sum_{j\in\mathcal{N}^k} x_{0^k,j}^k = 1 \qquad \forall k \in \mathcal{K} \tag{A.4}$$

$$\sum_{i\in\mathcal{N}^k} x_{i,n^k}^k = 1 \qquad \forall k \in \mathcal{K} \tag{A.5}$$

$$\sum_{i\in\mathcal{N}^k} x_{ih}^k - \sum_{j\in\mathcal{N}^k} x_{hj}^k = 0 \qquad \forall k \in \mathcal{K}, \forall h \in \mathcal{C} \tag{A.6}$$

$$\alpha_i \sum_{j\in\mathcal{N}^k} x_{ij}^k \le t_i^k \le \beta_i \sum_{j\in\mathcal{N}^k} x_{ij}^k \qquad \forall k \in \mathcal{K}, \forall i \in \mathcal{C} \cup \{0^k\} \tag{A.7}$$

$$\alpha_{n^k} \le t_{n^k}^k \le \beta_{n^k} \qquad \forall k \in \mathcal{K} \tag{A.8}$$

$$t_i^k + s_{ij}^k x_{ij}^k \le t_j^k + \beta_i (1 - x_{ij}^k) \qquad \forall k \in \mathcal{K}, \forall i,j \in \mathcal{N}^k \tag{A.9}$$

$$\alpha_i y_i + \sum_{k\in\mathcal{K}} t_i^k + p_{ij} \le \sum_{k\in\mathcal{K}} t_j^k + \beta_j y_j \qquad \forall (i,j) \in \mathcal{P} \tag{A.10}$$

$$x_{ij}^k \in \{0,1\} \qquad \forall k \in \mathcal{K}, \forall i,j \in \mathcal{N}^k \tag{A.11}$$

$$t_i^k \in \mathbb{R}_+ \qquad \forall k \in \mathcal{K}, \forall i \in \mathcal{N}^k \tag{A.12}$$

$$y_i \in \{0,1\} \qquad \forall i \in \mathcal{C} \tag{A.13}$$

The objective (A.1) is multi-criteria. Often, minimising uncovered visits (the third term) is prioritised over maximising home carer-visit preferences (the second term), which again is prioritised over minimising the total travelling costs (the first term). This can be accomplished by setting $w_1 = 1$, $w_2 = \sum_{k\in\mathcal{K}} \sum_{i\in\mathcal{N}^k} \sum_{j\in\mathcal{N}^k} c_{ij}^k$ and $w_3 = w_2 |\mathcal{C}| \max_{k\in\mathcal{K}, i\in\mathcal{C}} \delta_i^k$. Constraints (A.2) ensure that each visit is covered exactly once or left uncovered, and home carers can only handle allowed visits (A.3). Constraints (A.4)–(A.6) ensure that the home carers begin at the start visit, end at the end visit, and that routes are not segmented. Constraints (A.7) and (A.8) control that time windows are respected. Furthermore, Constraints (A.7) set $t_i^k = 0$ when k does not visit i, because in this case $\sum_{j\in\mathcal{N}^k} x_{ij}^k = 0$. Travelling distances are respected due to Constraints (A.9). Constraints (A.10) are the generalised precedence constraints. The sums $\sum_{k\in\mathcal{K}} t_i^k$ give the starting time of visit i, using $t_i^k = 0$ if k does not visit i. Hence, the only non-zero contribution to the sum will be the starting time of visit i for that single home carer that actually visits i (assuming i is not cancelled). The y-variable terms ensure that generalised precedence

constraints are respected even when visits are cancelled. If for example visit j is cancelled and i is serviced, then $\sum_{k\in\mathcal{K}} t_j^k = 0$ and the constraint reads $\sum_{k\in\mathcal{K}} t_i^k + p_{ij} \leq \beta_j$. This forces visit i to be scheduled p_{ij} time units before β_j, and therefore a manual planner can assign a substitute to visit j at least at time β_j, and hence respect the generalised precedence constraint. Finally, Constraints (A.11)–(A.13) set the domains of the decision variables.

The HCCSP formulation can be seen as a generalisation of an uncapacitated, multiple-depot VRPTW. The aim is to push flow for each home carer from start visit to end visit through as many profitable nodes as possible while respecting time windows and minimising costs. Also, it is only allowed for one home carer to go through each node.

The HCCSP generalises the Travelling Salesman Problem (TSP). TSP is well-known to be $\mathcal{NP}$-hard as its decision problem version is $\mathcal{NP}$-complete, see Problem ND22 in Garey and Johnson (1979). Therefore, also HCCSP is $\mathcal{NP}$-hard, and we can therefore not expect to solve the problem efficiently, i.e. in polynomial time. The $\mathcal{NP}$-hardness of the HCCSP is the reason why we develop a branch-and-price solution algorithm.

A.3 Decomposition

We will Dantzig-Wolfe decompose the HCCSP described in the previous section and model it as a set partitioning problem with side constraints. Then we will solve the model using dynamic column generation in a branch-and-price framework. This approach has presented superior results on VRPTW and the similarities to HCCSP are strong enough to suggest the same approach here. There is a vast amount of literature on column generation based solution methods for VRPTW, see e.g. Kallehauge et al. (2005) for a recent literature review and an introduction to the method. In a branch-and-price framework the problem is split into two problems, a master problem and a subproblem. The subproblem generates feasible home carer schedules, which are then subsequently combined in a feasible way in the master problem. In the master problem, given a large set of feasible schedules to choose from, one schedule is chosen for each home carer. Given a set of home carers $\mathcal{K}$, each home carer must choose a schedule from the set $\mathcal{R}^k$, which is the set of potential schedules for home carer k. Together, the schedules must cover as many visits as possible from the set $\mathcal{C}$.

A feasible schedule r for a home carer $k \in \mathcal{K}$ is defined as a route starting at 0^k and ending at n^k and respecting all constraints in the IP formulation from Section A.2.1 which do not link multiple routes. The schedule includes the

starting times of the visits. The parameter c_r^k gives the cost for home carer $k \in \mathcal{K}$ for schedule $r \in \mathcal{R}^k$, and $c_i = w_3\gamma_i$ gives the cost for leaving visit $i \in \mathcal{C}$ uncovered. The binary parameter $a_{ir}^k = 1$ if visit i is included in schedule r for home carer k and $a_{ir}^k = 0$ otherwise. Moreover, t_{ir}^k is the start time of visit i in schedule r for home carer k, if i is included in r for k. If i is not included in r for k, $t_{ir}^k = 0$.

A.3.1 Master problem

We introduce the binary decision variable λ_r^k where $\lambda_r^k = 1$ if schedule $r \in \mathcal{R}^k$ is chosen for home carer $k \in \mathcal{K}$, and $\lambda_r^k = 0$ otherwise. Furthermore, we introduce the binary decision variable Λ_i where $\Lambda_i = 1$ if visit $i \in \mathcal{C}$ is uncovered, and $\Lambda_i = 0$ otherwise. HCCSP can now be solved by finding a minimum cost combination of schedules such that all constraints are fulfilled. The master problem of the Dantzig-Wolfe decomposition of HCCSP is given by the mathematical programme shown below.

$$\min \sum_{k\in\mathcal{K}} \sum_{r\in\mathcal{R}^k} c_r^k \lambda_r^k + \sum_{i\in\mathcal{C}} c_i \Lambda_i \tag{A.14}$$

$$\text{s.t.} \sum_{k\in\mathcal{K}} \sum_{r\in\mathcal{R}^k} a_{ir}^k \lambda_r^k + \Lambda_i = 1 \qquad \forall i \in \mathcal{C} \tag{A.15}$$

$$\sum_{r\in\mathcal{R}^k} \lambda_r^k = 1 \qquad \forall k \in \mathcal{K} \tag{A.16}$$

$$\alpha_i \Lambda_i + \sum_{k\in\mathcal{K}} \sum_{r\in\mathcal{R}^k} t_{ir}^k \lambda_r^k + p_{ij} \leq \sum_{k\in\mathcal{K}} \sum_{r\in\mathcal{R}^k} t_{jr}^k \lambda_r^k + \beta_j \Lambda_j \qquad \forall (i,j) \in \mathcal{P} \tag{A.17}$$

$$\lambda_r^k \in \{0,1\} \qquad \forall k \in \mathcal{K}, \forall r \in \mathcal{R}^k \tag{A.18}$$

$$\Lambda_i \in \{0,1\} \qquad \forall i \in \mathcal{C} \tag{A.19}$$

The total costs of the schedules plus the costs of leaving visits uncovered are minimised (A.14). The cost of a schedule contains the remaining components of the original objective and is therefore determined by the travel costs and by the service level of the visits in the schedule. Constraints (A.15) ensure that all visits are either included in exactly one schedule or considered uncovered. One schedule must be assigned to each home carer (A.16), and the generalised precedence constraints must be respected (A.17). Again, the Λ-variable terms in Constraints (A.17) ensure that precedence constraints are respected even for uncovered visits. Integrality of the decision variables is ensured by Constraints (A.18) and (A.19). Any feasible solution to the decomposed problem is a feasible solution to the original problem, and any optimal solution to the decomposed problem is an optimal solution to the original problem.

To be able to solve the master problem in an LP-based branch-and-price framework, the integrality constraints on λ_r^k and Λ_i are relaxed. Also, the precedence constraints (A.17) are relaxed, as we thereby have no constraints interconnecting the starting times in the schedules of different home carers. This gives a simpler pricing problem, which will be explained further in Section A.3.2. The two relaxed constraints will be handled in the branching.

As the number of feasible schedules for each home carer is enormous, the set $\mathcal{R}^k$ of schedules for home carer $k \in \mathcal{K}$ is restricted to only contain a small subset $\mathcal{R'}^k$ of promising schedules, which will be generated by the column generating pricing problem. We abbreviate the relaxed and restricted master problem as RMP. For each primal solution to the RMP, we obtain a dual solution $[\pi, \omega]$, where π_i, $i \in \mathcal{C}$, and ω_k, $k \in \mathcal{K}$, are the dual variables of Constraints (A.15) and (A.16), respectively. The dual variables are used in the column generation technique to generate new schedules that lead to an improvement of the solution to the RMP.

A.3.2 Pricing problem

The pricing problem is used to find the feasible schedule with the most negative reduced cost (if any exists). As the home carers have different working hours and competences, the pricing problem is split into $|\mathcal{K}|$ independent pricing problems. The pricing problem is an Elementary Shortest Path Problem with Time Windows (ESPPTW), which has been proved $\mathcal{NP}$-hard in Dror (1994b). The pricing problem for a home carer $k \in \mathcal{K}$ is formulated as an integer programme below. Any feasible solution to the pricing problem with negative cost represents a column with negative reduced cost in the RMP.

$$\min \sum_{i \in \tilde{\mathcal{N}}^k} \sum_{j \in \tilde{\mathcal{N}}^k} \left(w_1 c_{ij}^k + w_2 \delta_i^k - \pi_i \right) x_{ij} - \omega_k \tag{A.20}$$

$$\text{s.t.} \sum_{j \in \tilde{\mathcal{N}}^k} x_{0^k,j} = 1 \tag{A.21}$$

$$\sum_{i \in \tilde{\mathcal{N}}^k} x_{i,n^k} = 1 \tag{A.22}$$

$$\sum_{i \in \tilde{\mathcal{N}}^k} x_{ih} - \sum_{j \in \tilde{\mathcal{N}}^k} x_{hj} = 0 \qquad \forall h \in \mathcal{C}^k \tag{A.23}$$

$$\alpha_i \sum_{j \in \tilde{\mathcal{N}}^k} x_{ij} \leq t_i \leq \beta_i \sum_{j \in \tilde{\mathcal{N}}^k} x_{ij} \qquad \forall i \in \mathcal{C}^k \cup \{0^k\} \tag{A.24}$$

$$\alpha_{n^k} \leq t_{n^k} \leq \beta_{n^k} \tag{A.25}$$

$$t_i + s_{ij}^k x_{ij} \leq t_j + \beta_i (1 - x_{ij}) \qquad \forall i,j \in \tilde{\mathcal{N}}^k \tag{A.26}$$

$$x_{ij} \in \{0,1\} \qquad \forall i,j \in \tilde{\mathcal{N}}^k \tag{A.27}$$

$$t_i \in \mathbb{R}_+ \qquad \forall i \in \tilde{\mathcal{N}}^k \tag{A.28}$$

Here, we have introduced the decision variables x_{ij} and t_i which are the same as in (A.1)–(A.13), without the k index. For a given $k \in \mathcal{K}$, the subset of visits $\mathcal{C}^k = \{i \in \mathcal{C} : \rho_i^k = 1\}$ is the set of visits allowed for k. Moreover, $\tilde{\mathcal{N}}^k = \mathcal{C}^k \cup \{0^k, n^k\}$, and we define $\delta_{0^k}^k = \delta_{n^k}^k = \pi_{0^k} = \pi_{n^k} = 0$.

The relatively simple expression (A.20) for the reduced costs of a column is one of the reasons why the generalised precedence constraints are relaxed. One could, as done in Akker et al. (2006) and in Dohn et al. (2011) have kept the generalised precedence constraints in the RMP. This would have implied a more complicated pricing problem, as the pricing problem then incorporates a means of adjusting the starting times in a schedule based on the dual variables. In Akker et al. (2006) they do not solve their pricing problem by an exact method, but use a heuristic method. Benchmark results from Dohn et al. (2011) show that in many cases it is as good to relax the generalised precedence constraints, as to keep them in the master problem.

We solve the pricing problem with a labelling algorithm built on ideas from Chabrier (2006) and Feillet et al. (2004).

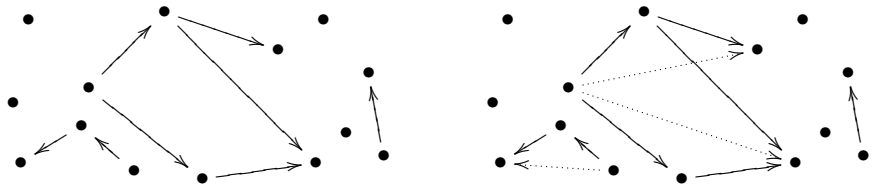
A.4 Branching

The generalised precedence constraints and the integrality constraints that were relaxed from the master problem are handled in the branching part of the

branch-and-price algorithm. To handle both types of constraints, we need to present two branching methods. One to handle the violation of an integrality constraint and another to handle the violation of a precedence constraint. The branching scheme used to handle precedence constraint violations is a time window branching scheme. This also enforces integrality to a certain point as shown in Gélinas et al. (1995). Nonetheless, one cannot solely rely on time window branching to enforce integrality, so we use an additional branching scheme to force the solution to integrality. First, we will present a preprocessing technique.

A.4.1 Preprocessing of time windows

The visits $\mathcal{C}$ can be grouped according to how they are connected by generalised precedence constraints. Define the directed *temporal dependency graph* $G = (V, A)$ by $V = \mathcal{C}$ and $A = \mathcal{P}$. The graph G consists of one or more sub-graphs, which corresponds to the connected components in the undirected version of G. An example of such a graph is shown in Figure A.2(a). From the



(a) Example of a temporal dependency graph.

(b) The same graph expanded with derived arcs.

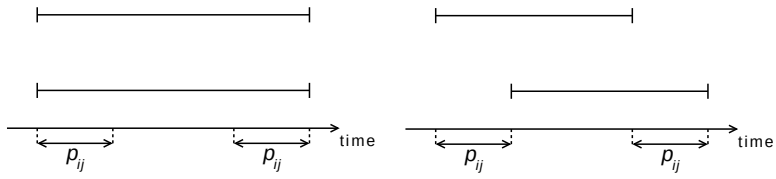
Figure A.2: *A temporal dependency graph.*

existing generalised precedence constraints, additional *derived generalised precedence constraints* can be found. In every subgraph with three or more nodes, we look for triples $i, j, k \in \mathcal{C}$ where $(i, j), (j, k) \in \mathcal{P}$ with $i \neq j, j \neq k, k \neq i$. If $(i, k) \notin \mathcal{P}$, then the pair is added to $\mathcal{P}$ with $p_{ik} = p_{ij} + p_{jk}$. If $(i, k) \in \mathcal{P}$ the offset is updated to $p_{ik} := \max\{p_{ik}, p_{ij} + p_{jk}\}$ in order to get the tightest constraint. The process is repeated until no further constraints can be derived or tightened. The example will now look as in Figure A.2(b). This derivation of generalised precedence constraints will make it possible to reduce more time windows, as there will be a greater number of precedence constraints on which to perform the following pair-wise reduction technique.

If two visits $i, j \in \mathcal{C}$ are connected via a (possibly derived) generalised precedence

constraint $(i, j) \in \mathcal{P}$, it might be possible to tighten the time windows of i and j, such that $[\alpha'_i, \beta'_i] = [\alpha_i, \min\{\beta_i, \beta_j - p_{ij}\}]$ and $[\alpha'_j, \beta'_j] = [\max\{\alpha_j, \alpha_i + p_{ij}\}, \beta_j]$ are the new time windows as illustrated in Figure A.3.

This preprocessing is repeated until no time windows are tightened. The preprocessing technique can be used in every node of the branch-and-bound tree. It should be noted that this time window reduction can only be carried out, because it is required that also temporal dependencies with cancelled visits must be respected. If this was not the case, then the cancellation of a visit i with $(i, j) \in \mathcal{P}$ would lead to the time window of j being "reset" (assuming it was previously reduced by preprocessing).



(a) Before preprocessing. (b) After preprocessing.

Figure A.3: *Adjustment of time windows in accordance to a generalised precedence constraint. Each of the subfigures shows the time windows of two visits i (top) and j (bottom).*

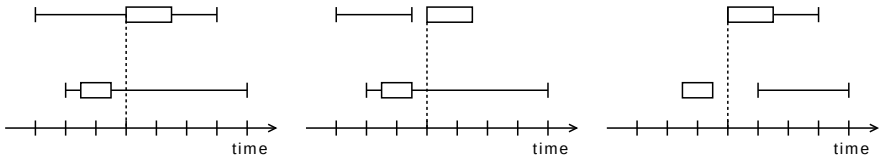
A.4.2 Branching on generalised precedence constraints

A generalised precedence constraint $(i, j) \in \mathcal{P}$ is violated if there exists positive variables $\lambda_{r_1}^{k_1} > 0$ and $\lambda_{r_2}^{k_2} > 0$ (the relaxation allows for $k_1 = k_2$ and $r_1 = r_2$, but we will prevent that in the subproblem) in the solution to the RMP for which

$$t_{i,r_1}^{k_1} + p_{ij} \not\leq t_{j,r_2}^{k_2} \ .$$

Therefore, to remedy this, we will alter the time windows in the branches. In the left branch the time window of visit i is set to $[\alpha_i, t_{\text{split}} - \epsilon]$, where t_{split} is the split time, and $\epsilon > 0$ is a very small fraction. However, almost always $\epsilon = 1$ can be used in practice as travelling times are integer (or easily convertible to integers), as each p_{ij} is integer, and as the way the labelling algorithm generates schedules sets the scheduling variables by summing up travelling times. In the right branch the time window of visit i is set to $[t_{\text{split}}, \beta_i]$. The preprocessing technique described in the previous section is used again, which will result in the

time window of visit j in the right branch being changed to $[t_{\text{split}} + p_{ij}, \beta_j]$. All previously generated schedules violating these new time windows are removed. The split time is selected such that $t^{k_2}_{j,r_2} - p_{ij} + \epsilon \leq t_{\text{split}} \leq t^{k_1}_{i,r_1}$. Hence, the branching scheme divides the solution space into two sets, where the solution that violates the precedence constraint for (i, j) becomes infeasible in each of them. Synchronisation constraints are often seen in home care instances. The branching scheme suggested here combined with preprocessing of time windows is as strong as the scheme tailored for synchronisation described in Ioachim et al. (1999). This is elaborated in Dohn et al. (2011), where it is also described how to pick the best split time in the given interval. An illustration of the generalised precedence constraint branching scheme can be found in Figure A.4.



(a) Parent node. (b) Left child node. (c) Right child node.

Figure A.4: *Example of the branching applied when a generalised precedence constraint is violated. Each of the subfigures shows the time windows of two visits i (top) and j (bottom), and the start times of the visits in an RMP solution. The violated constraint for (i, j) has $p_{ij} = 2$. The dotted line shows the chosen split time, and the distance between the ticks on the time line is two time units.*

A.4.3 Integer branching

In the following, we will let A^k denote the $|\mathcal{C}| \times |\mathcal{R}'^k|$-matrix where the elements are given by the parameter a^k_{ir} for a given home carer $k \in \mathcal{K}$, i.e. each column in A^k represents a schedule $r \in \mathcal{R}'^k$. Now, consider the structure of the constraint matrix of the RMP which is shown in Figure A.5. For clarity, we only show ones of the constraint matrix and introduce $m = |\mathcal{K}|$ and $\bar{n} = n - 1$.

We observe that the RMP has strong integer properties due to the generalised upper bound constraints (A.16) for each home carer, see e.g. Rezanova and Ryan (2010) for further details and references. That is, for all home carers $k \in \mathcal{K}$, their submatrix of the constraint matrix is perfect. Consequently, fractionality in the LP solutions will never appear within one home carer's block of schedules. Any fractions in the RMP can therefore only occur between blocks of columns,

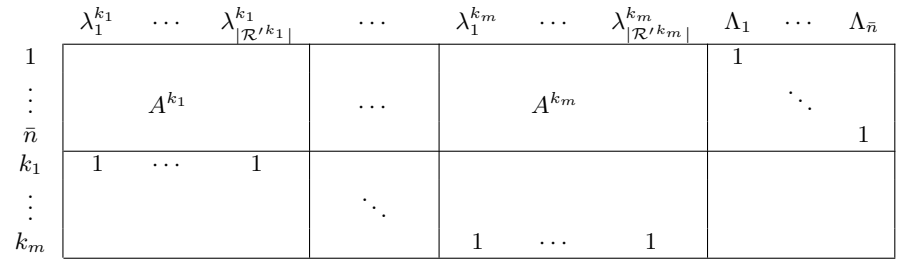


Figure A.5: *Constraint matrix for the RMP.*

belonging to different home carers. Hence, if the LP solution is fractional, it is because two or more home carers compete for one or more visits in their schedules. Let $i \in \mathcal{C}$ denote a visit for which home carer $k \in \mathcal{K}$ is competing with one or more other home carers. Since the visit can only be handled by one home carer, then in an integer solution either k handles i or k does *not* handle i.

We will exploit the strong integer properties of the constraint matrix of the RMP to apply a so-called constraint branching strategy, see Ryan and Foster (1981). We introduce the sum $S_i^k = \sum_{r \in \mathcal{R}'^k} a_{ir}^k \lambda_r^k$. If a fractional solution occurs, the constraint branching strategy is now to find a visit-home carer pair (i, k) of a visit $i \in \mathcal{C}$ and a home carer $k \in \mathcal{K}$ for which $0 < S_i^k < 1$. In the 1-branch visit i is forced to be handled by k and in the 0-branch prohibit visit i from being handled by k. Notice that since at least one of the unique λ_r^k is fractional then at least one sum S_i^k is also fractional.

Whenever the sum S_i^k is fractional for two or more visit-home carer pairs (i, k), we have to select one of these as the candidate for branching in the node. If S_i^k is close to 1, forcing $S_i^k = 1$ will probably not change the solution drastically, so only a small increase in the lower bound can be expected in this branch. On the other hand, as the LP solution suggests that home carer k should handle i in an optimal solution, ruling out this option ($S_i^k = 0$) is likely to cause a major increase in the lower bound. If S_i^k is close to 0, a similar line of reasoning also shows a skewed branching. In order to keep the branch-and-bound tree balanced, we select the "most fractional" candidate, i.e. the candidate closest to one half. More formally, we select $(i^*, k^*) = \arg\min_{(i,k) \in \mathcal{C} \times \mathcal{K} : 0 < S_i^k < 1} \left| S_i^k - \frac{1}{2} \right|$.

A.5 Clustering of visits and arc removal

The HCCSP exhibits a structural feature that can be used to group or *cluster* visits. HCCSP has, as opposed to VRPTW, a preference parameter for each home carer-visit combination. Moreover, test runs have suggested that the ESPPTW solver is a bottleneck in the branch-and-price algorithm. Therefore, we have developed schemes that reduce the sizes of the ESPPTW networks, which will in turn decrease the running time of the algorithm. For some larger instances, visit clustering is even needed to find feasible solutions.

When no reduction of the ESPPTW networks is used, every home carer $k \in \mathcal{K}$ can visit every $i \in \mathcal{C}$ where $\rho_i^k = 1$. However, in a good solution (assuming the objective function weights are set as suggested in Section A.2.1), a home carer k will only handle visits i where δ_i^k is favourable. Therefore, we have devised three ways of clustering visits for a home carer according to the preference parameter δ_i^k, thereby effectively reducing the network sizes for each home carer. Again, let $\mathcal{C}^k = \{i \in \mathcal{C} : \rho_i^k = 1\}$. All schemes operate with a cluster of visits $\bar{\mathcal{C}}^k \subseteq \mathcal{C}^k$ for a home carer. The first scheme only puts visits in the cluster, when it is directly profitable, i.e. $\bar{\mathcal{C}}^k = \{i \in \mathcal{C}^k : \delta_i^k < 0\}$.

In the second scheme preference parameters for home carer k are ordered non-decreasingly as $\delta_{i_1}^k \leq \cdots \leq \delta_{i_\xi}^k \leq \cdots \leq \delta_{i_{|\mathcal{C}^k|}}^k$ with ties broken arbitrarily. Define the set $\Delta_\xi^k = \{\delta_{i_1}^k, \ldots, \delta_{i_\xi}^k\}$ given the parameter ξ. The second scheme then defines the cluster as $\bar{\mathcal{C}}^k = \{i \in \mathcal{C}^k : \delta_i^k \in \Delta_\xi^k\}$. The cluster contains the ξ most profitable visits.

The two first clustering schemes do not guarantee that all visits are in a cluster. Therefore, all remaining visits $i \in \mathcal{C} \setminus \bigcup_{k \in \mathcal{K}} \bar{\mathcal{C}}^k$ are added to all clusters.

The third scheme seeks to exploit the integer properties of the problem described in Section A.4.3. If the home carers cannot compete for visits, the LP solutions will be naturally integer, and hence the run times will decrease significantly. Therefore, we make the visit clusters pairwise disjoint, i.e. $\forall k_1, k_2 \in \mathcal{K}, k_1 \neq k_2 : \bar{\mathcal{C}}^{k_1} \cap \bar{\mathcal{C}}^{k_2} = \emptyset$. Again, the preference parameters for each home carer are ordered non-decreasingly. Hereafter, the scheme iterates over the home carers in a round-robin fashion and puts the most profitable visit in the home carer's cluster (if it is not already put in another home carer's cluster). Suppose visit j is already in $\bar{\mathcal{C}}^k$ for home carer k, then there are two conditions under which another visit i is not permitted in the cluster. If i cannot be carried out before j, and also j cannot be carried out before i, then the visits can never be scheduled in the same route. This is detected whenever $\alpha_i + s_{ij}^k > \beta_j \wedge \alpha_j + s_{ji}^k > \beta_i$. The second condition is when there is a temporal dependency, which disallows any route with both

visits. This is the case when $(i,j),(j,i) \in \mathcal{P}$ and $-s^k_{ji} < p_{ij} \wedge -s^k_{ij} < p_{ji}$.

The use of clustering will sacrifice optimality, and later we will look into how big the gap to optimality is, and compare it against the benefit of improved run time. The closest to this idea we have seen in the VRP literature is the *petal method*, see e.g. Foster and Ryan (1976), which clusters the visits based on geographical position.

A.5.1 Expansion of visit clusters

The clustering of visits can lead to visits being uncovered not because it is optimal, but due to the clustering. Hopefully, these are only a very few visits. In order to remedy this, the clusters are made dynamic, in the sense that expansion of the clusters is allowed. For any branch-and-bound node, uncovered visits can be detected, by looking at the LP optimal solution. If there are uncovered visits, they are added to all clusters, and the LP problem is solved again. We suggest two versions of the cluster expansion. Either cluster expansion can happen only in the root node, or it can happen in any node of the branch-and-bound tree. Especially the latter adds a twist of unpredictability (though still deterministic) to the problem, because the problem basically can be changed anywhere in the branch-and-bound tree. It can happen that the lower bound for a child is lower than the lower bound for its parent, which is avoided when expansion is only allowed in the root node.

A.5.2 Removal of idle arcs

We will here present another method to reduce the network sizes. The time where the home carer is neither visiting a citizen nor travelling is called *idle time*. This is time where the home carer is basically just waiting for the time window of the next visit to open. Therefore, another means to reduce the sizes of the ESPPTW networks, is to remove arcs where the minimum idle time $\phi^k_{ij} = \alpha_j - (\beta_i + s^k_{ij})$ between two visits is high. Again, proof of optimality is sacrificed, but in a good solution, we probably would not see the use of many arcs with large idle time.

A.6 Test instances

We have had access to four authentic test instances from two Danish municipalities. These are named hh, ll1, ll2 and ll3. Based on the authentic instances we have generated 60 extra instances. These instances are constructed by generating new sets of generalised precedence constraints for each of the four authentic instances, while still keeping the original sets of home carers and visits and original travelling times. We found it necessary to generate new sets of generalised precedence constraints in order to test our algorithm thoroughly. Most original data sets from home care are very incomplete when it comes to registering the temporal dependencies. These are either dealt with manually or by changing the time windows. The new generalised precedence constraint sets are based on the five types of temporal dependencies from Figure A.1, and we have created five sets named td0–4. The generalised precedence constraints in the set td0 are of the temporal dependency type synchronisation (a). The set td1 is of the type overlap (b). The set td2 consists of the types minimum difference (c) and maximum difference (d). When values are drawn randomly, these categories collapse to one. The set td3 is of type minimum+maximum difference (e). Finally, td4 is a random combination of the other four types. Three sets of generalised precedence constraints were generated for each of these fives sets: Sets A, B, and C, where the number of generalised precedence constraints approximately is, respectively, 10%, 20%, and 30% of the number of visits. The sets of generalised precedence constraints were generated as in Dohn et al. (2011). Characteristics for these test instances can be seen in Table A.2. The notation td[0-4] expands to td0, td1, td2, td3, and td4. It is compacted in the table, because the instances share the same characteristics.

Furthermore, Bredström and Rönnqvist (2007) have very kindly given us access to their 30 data instances. These data instances are generated based on realistic settings and only contain synchronisation constraints. All visits have the same priority, and no visits are excluded for any of the home carers, i.e. $\rho_i^k = 1, \forall i \in \mathcal{C}, \forall k \in \mathcal{K}$. The preference parameter δ_i^k is drawn randomly between -10 and 10. For all of the instances, all home carers in that instance have the same duty hours. Characteristics for these test instances can also be seen in Table A.2. Again, br-[06-08][S,M,L]-td0 means that for each of br-06, br-07, and br-08, there are three instances: One with small (S) time windows, one with medium (M) time windows, and one with large (L) time windows.

	hh	ll1	ll2	ll3	hh-td[0-4]-A	hh-td[0-4]-B	hh-td[0-4]-C	ll1-td[0-4]-A	ll1-td[0-4]-B	ll1-td[0-4]-C	ll2-td[0-4]-A	ll2-td[0-4]-B	ll2-td[0-4]-C	ll3-td[0-4]-A	ll3-td[0-4]-B	ll3-td[0-4]-C	br-[01-05][S,M,L]-td0	br-[06-08][S,M,L]-td0	br-[09-10][S,M,L]-td0
$\|\mathcal{K}\|$	15	8	7	6	15	15	15	8	8	8	7	7	7	6	6	6	4	10	16
$\|\mathcal{C}\|$	150	107	60	61	150	150	150	107	107	107	60	60	60	61	61	61	20	50	80
$\|\mathcal{P}\|$	6	0	0	0	16	30	46	10	22	32	6	12	18	6	12	18	4	10	16

Table A.2: *Characteristics for the test instances.* $|\mathcal{K}|$ *is number of home carers,* $|\mathcal{C}|$ *is number of visits and* $|\mathcal{P}|$ *is number of generalised precedence constraints.*

A.7 Computational results

The aim in this section is to compare the different visit clustering techniques presented in Section A.5. We will also try to measure the effects of removal of idle time arcs and cluster expansion. Using clustering will sacrifice optimality, and we will here investigate how big the gap to optimality is, and compare it against the benefit of improved run time.

We measure three quality parameters, which are also the terms of the objective function: uncovered visits, home carer-visit preferences, and total travel costs. The weights of the objective function are set as suggested in Section A.2.1, so a hierarchical ordering is obtained. We seek to minimise the number of uncovered visits and maximise the preference level of the solution. The total travel costs are measured in minutes for all home carers for the whole daily schedule. We subtract the durations of the visits in the total travel time, hence giving preference to longer visits, and thereby maximising the so-called face-to-face time. More formally, we define the travel cost as $c_{ij}^k = s_{ij}^k - 2 \cdot \text{dur}_i$. Hence, if it were only possible to cover either visit i or the two visits j and h, coverage of visit i is preferred, whenever $\gamma_i \geq \gamma_j + \gamma_h$ and $\text{dur}_i > \text{dur}_j + \text{dur}_h$, assuming the travel time for both options is the same. Minimising the total travel costs are not as important as minimising the two other measurements, but low travel costs are naturally preferred. In order to be able to make comparisons this third measure is ignored, when we are performing tests on the instances from Bredström and Rönnqvist (2007).

The algorithm is implemented in the branch-and-cut-and-price framework from COIN-OR, see Lougee-Heimer (2003), using the COIN-OR open-source LP solver CLP. All tests are run on 2.2 GHz processors. As an outcome of preliminary tests, we return up to five negative reduced cost columns per home

carer per iteration. For all of the test runs we have set a time out limit of one hour. The implementation of the ESPPTW solver ensures that generalised precedence constraints, that make two visits mutually exclusive, are respected within the individual routes. This tightens the lower bounds and reduces the number of branch-and-bound nodes.

We have grouped the instances into 35 test groups based on their size, the type of temporal dependency included, and the number of temporal dependencies. The test groups can be seen in Tables A.3-A.4. For each of these groups, 13 different settings for the algorithm are compared. The settings are written as CS-RA-ER, abbreviating *clustering scheme*, *removal of arcs*, and *expansion in root only*, respectively. CS = 0 corresponds to no use of visit clustering. CS = 1 gives all-preferred clusters, i.e. clusters for home carer k where $\delta_i^k < 0$ for all visits i in the cluster, as described in Section A.5. CS = 2 gives fixed-size clusters of a fixed size ξ. CS = 3 gives pair-wise disjoint clusters. Before the preference parameters are sorted they are shuffled randomly in order to make the tie-breaking arbitrary. The setting RA is a binary parameter, which is 1, if we remove arcs based on idle time, and 0 otherwise. The setting ER is also a binary parameter, which is 1, if we only allow cluster expansion in the root node of the branch-and-bound tree, and 0 if cluster expansion is allowed in every node of the branch-and-bound tree. Thus, the settings 0-0-0 (as well as the redundant settings 0-0-1) give the optimal solution. However, some of the instances are not possible to solve to optimality within the time limit, they can only be solved using clustering. For CS = 2, we use a fixed cluster size $\xi = 12$. With a fixed cluster size of 12, the pricing problems (at least initially, before cluster expansion) are solved very fast. When arcs are removed, the largest allowed minimum idle time ϕ_{ij}^k is set to 10 minutes, both based on preliminary tests. The abbreviation BR is used when we show results from Bredström and Rönnqvist (2007).

In Tables A.3–A.4 the comparison is shown. The numbers are averages over all instances in the test group. In total, we have 1190 test runs, which gives a good statistical foundation. Let T and Z denote the run time and the objective value, respectively, of a given test run, and let T_{ref} and Z_{ref} denote the run time and the objective value of a reference solution, respectively. The time difference is then calculated as T/T_{ref} in percent, and the objective gap is calculated as $|(Z - Z_{\text{ref}})/Z_{\text{ref}}|$ in percent. When the objective is better than the reference objective a minus sign is added. The reference settings are the leftmost, in most cases it will be the settings 0-0-0, which is a very intuitive reference. All the instances in the test groups [hh,ll1] and [hh,ll1]-td[0-4]-[A,B,C] have not been possible to solve to optimality. Any attempt has, when the one hour time out limit is reached, ended up with around 70% or more of the visits uncovered in the best solution in the branch-and-bound tree. Therefore, the reference settings for these test groups will be 1-0-0. When using relative gaps for comparison, one

Group	Settings												
	0-0-0	1-0-0	1-0-1	1-1-0	1-1-1	2-0-0	2-0-1	2-1-0	2-1-1	3-0-0	3-0-1	3-1-0	3-1-1	BR
	Time difference (%)													
br-[01-05]	100	39	96	48	52	47	54	60	57	6	4	6	5	192
br-[06-08]	100	60	58	48	55	11	20	13	57	1	1	1	1	157
br-[09-10]	100	81	81	87	87	21	50	11	27	1	1	1	0	144
[hh,ll]	N/A	100	87	70	106	105	104	48	45	8	5	5	4	N/A
[hh,ll]-td0-A	N/A	100	246	99	322	1395	1384	1872	1770	77	35	66	36	N/A
[hh,ll]-td0-B	N/A	100	44	102	43	540	1362	160	1235	30	23	30	18	N/A
[hh,ll]-td0-C	N/A	100	45	72	88	266	645	168	971	63	41	56	42	N/A
[hh,ll]-td1-A	N/A	100	100	74	74	138	155	110	235	47	45	43	38	N/A
[hh,ll]-td1-B	N/A	100	78	89	75	161	126	123	133	40	34	38	33	N/A
[hh,ll]-td1-C	N/A	100	89	97	109	101	123	89	104	56	37	48	29	N/A
[hh,ll]-td2-A	N/A	100	97	89	80	29	29	39	36	5	5	5	6	N/A
[hh,ll]-td2-B	N/A	100	2788	216	253	121	124	100	91	15	15	15	13	N/A
[hh,ll]-td2-C	N/A	100	115	107	848	211	173	140	133	81	52	75	58	N/A
[hh,ll]-td3-A	N/A	100	43	91	90	510	440	258	247	5	4	6	5	N/A
[hh,ll]-td3-B	N/A	100	58	201	177	15	17	17	16	3	2	2	1	N/A
[hh,ll]-td3-C	N/A	100	104	104	104	11	8	8	5	1	1	2	1	N/A
[hh,ll]-td4-A	N/A	100	61	50	176	103	106	71	64	18	11	19	12	N/A
[hh,ll]-td4-B	N/A	100	207	67	119	351	351	106	90	8	5	11	7	N/A
[hh,ll]-td4-C	N/A	100	99	16	60	102	102	15	24	4	3	4	2	N/A
[ll2,ll3]	100	15	15	17	19	8	29	6	29	0	0	0	0	N/A
[ll2,ll3]-td0-A	100	14	16	37	71	2014	2014	1310	2014	1	1	1	1	N/A
[ll2,ll3]-td0-B	100	40	38	42	42	1	1	1	1	0	0	0	0	N/A
[ll2,ll3]-td0-C	100	31	29	30	29	3	3	3	3	2	1	2	2	N/A
[ll2,ll3]-td1-A	100	98	98	20	24	19	24	11	16	0	0	0	0	N/A
[ll2,ll3]-td1-B	100	65	63	21	34	20	4	5	5	0	0	0	0	N/A
[ll2,ll3]-td1-C	100	94	94	91	91	1	1	1	0	0	0	0	0	N/A
[ll2,ll3]-td2-A	100	12	12	11	11	1	17	1	11	0	0	0	0	N/A
[ll2,ll3]-td2-B	100	12	12	83	77	98	98	98	94	0	0	0	0	N/A
[ll2,ll3]-td2-C	100	7	8	11	10	66	62	79	76	1	1	1	1	N/A
[ll2,ll3]-td3-A	100	97	97	89	97	1	97	1	96	0	0	0	0	N/A
[ll2,ll3]-td3-B	100	21	22	12	12	940	1971	463	1970	0	0	1	1	N/A
[ll2,ll3]-td3-C	100	97	97	97	97	1	1	0	0	0	0	0	0	N/A
[ll2,ll3]-td4-A	100	99	98	102	102	79	98	45	11	0	0	0	0	N/A
[ll2,ll3]-td4-B	100	99	99	82	79	0	0	0	0	0	0	0	0	N/A
[ll2,ll3]-td4-C	100	93	93	50	92	0	0	0	0	0	0	0	0	N/A
Avg. (0-0-0)	100	56	59	52	57	175	239	111	235	1	1	1	1	N/A
Avg. (1-0-0)	N/A	100	180	115	157	704	884	480	884	14	10	13	10	N/A

Table A.3: *Comparison of time difference for different settings for the algorithm for the test groups. Settings are written as CS-RA-ER. Test groups br-[x-x]/[S,M,L]-td0 are shortened to br-[x-x]. The average 'Avg. (0-0-0)' uses the settings 0-0-0 as reference settings and does not include the test groups [hh,ll1] and [hh,ll1]-td[0-4]-[A,B,C], as test runs are not carried out in these groups with the settings 0-0-0. The average 'Avg. (1-0-0)' uses the settings 1-0-0 as reference settings and includes all test groups, but not the settings 0-0-0.*

	Settings													
Group	0-0-0	1-0-0	1-0-1	1-1-0	1-1-1	2-0-0	2-0-1	2-1-0	2-1-1	3-0-0	3-0-1	3-1-0	3-1-1	BR
	Objective gap (%)													
br-[01-05]	0	204	203	205	205	3	3	5	4	806	937	1070	1201	0
br-[06-08]	0	0	0	0	0	121	147	93	147	524	747	526	776	0
br-[09-10]	0	0	0	-1	-1	155	179	158	179	289	914	292	914	6
[hh,ll1]	N/A	0	26	26	26	0	0	3	3	75	83	83	90	N/A
[hh,ll1]-td0-A	N/A	0	4	-6	4	-20	-13	-19	-11	7	32	7	32	N/A
[hh,ll1]-td0-B	N/A	0	18	7	18	-7	0	-5	2	23	44	23	54	N/A
[hh,ll1]-td0-C	N/A	0	21	10	24	3	7	-1	6	15	40	32	53	N/A
[hh,ll1]-td1-A	N/A	0	0	-2	-2	-29	-2	-27	0	-4	4	6	13	N/A
[hh,ll1]-td1-B	N/A	0	19	0	28	-9	2	-9	6	15	34	17	36	N/A
[hh,ll1]-td1-C	N/A	0	20	0	20	-14	16	-10	12	20	35	20	35	N/A
[hh,ll1]-td2-A	N/A	0	2	4	4	15	15	-8	-8	14	14	46	46	N/A
[hh,ll1]-td2-B	N/A	0	2	6	6	-4	-2	-2	0	31	33	39	39	N/A
[hh,ll1]-td2-C	N/A	0	2	6	6	-6	2	-4	4	11	36	15	42	N/A
[hh,ll1]-td3-A	N/A	0	9	2	9	0	0	-2	-2	59	61	61	63	N/A
[hh,ll1]-td3-B	N/A	0	0	2	2	0	0	2	2	58	63	60	72	N/A
[hh,ll1]-td3-C	N/A	0	18	2	20	2	5	2	5	11	16	20	27	N/A
[hh,ll1]-td4-A	N/A	0	5	-5	5	12	5	-2	5	34	43	34	46	N/A
[hh,ll1]-td4-B	N/A	0	13	0	11	2	11	2	11	37	59	39	63	N/A
[hh,ll1]-td4-C	N/A	0	16	-2	10	2	8	0	2	18	54	18	66	N/A
[ll2,ll3]	0	0	0	0	0	456	570	456	570	1710	1710	3990	3990	N/A
[ll2,ll3]-td0-A	0	0	0	0	0	174	174	0	35	626	626	764	764	N/A
[ll2,ll3]-td0-B	0	0	0	0	0	246	246	246	246	369	390	635	717	N/A
[ll2,ll3]-td0-C	0	0	0	0	0	153	153	153	153	170	204	255	408	N/A
[ll2,ll3]-td1-A	0	0	0	0	0	456	570	456	570	1597	1597	2052	2052	N/A
[ll2,ll3]-td1-B	0	0	0	0	0	343	570	115	570	1711	2508	1597	3078	N/A
[ll2,ll3]-td1-C	0	0	0	0	0	229	570	570	798	1483	2053	1711	2167	N/A
[ll2,ll3]-td2-A	0	0	0	0	0	456	570	0	114	2622	2622	5015	5015	N/A
[ll2,ll3]-td2-B	0	0	0	0	0	103	103	21	21	185	185	451	451	N/A
[ll2,ll3]-td2-C	0	0	0	0	0	68	68	0	0	442	442	664	664	N/A
[ll2,ll3]-td3-A	0	0	0	0	0	0	114	456	570	2622	2622	3534	3534	N/A
[ll2,ll3]-td3-B	0	0	0	0	0	1	114	1	114	2508	2508	2964	2964	N/A
[ll2,ll3]-td3-C	0	0	0	0	0	1	1	1	1	1711	1711	2395	2395	N/A
[ll2,ll3]-td4-A	0	0	0	0	0	266	266	266	266	1012	1012	2077	2077	N/A
[ll2,ll3]-td4-B	0	0	0	0	0	266	266	266	266	1119	1119	1970	1970	N/A
[ll2,ll3]-td4-C	0	0	0	0	0	266	266	266	266	959	959	2343	2343	N/A
Avg. (0-0-0)	0	11	11	11	11	198	261	186	257	1182	1309	1806	1973	N/A
Avg. (1-0-0)	N/A	0	5	1	5	100	137	93	135	647	722	988	1086	N/A

Table A.4: *Comparison of objective gap for different settings for the algorithm for the test groups. Settings are written as CS-RA-ER. Test groups br-[x-x][S,M,L]-td0 are shortened to br-[x-x]. The average 'Avg. (0-0-0)' uses the settings 0-0-0 as reference settings and does not include the test groups [hh,ll1] and [hh,ll1]-td[0-4]-[A,B,C], as test runs are not carried out in these groups with the settings 0-0-0. The average 'Avg. (1-0-0)' uses the settings 1-0-0 as reference settings and includes all test groups, but not the settings 0-0-0.*

should be careful, because relative gaps are highly dependent on the objective measure. In our case we have a very high penalty on uncovered visits, so a single uncovered visit as opposed to no uncovered visits would lead to a large gap. Also, for all instances based on ll1, there will be eight visits that are impossible to cover, as they cannot be completed within the working hours of any of the home carers. This fixed cost for all generated routes makes the gaps smaller.

As mentioned earlier, the dynamic expansion of visit clusters makes the algorithm behave somewhat unpredictable. In the cases where we see the time difference being close to 100% and the objective gap at the same time being close to 0%, it is very likely that the clusters are expanded to nearly the entire set $\mathcal{C}^k$, thereby getting close to CS = 0. On the other hand, when the gap is small and the time difference significantly below 100%, then a good initial clustering is used. With regard to the time-quality trade-off, the all-preferred (CS = 1) and the fixed-size (CS = 2) clustering schemes both have instance groups where they are performing best. If dynamic cluster expansion was not used, then it would be expected that the fixed-size clustering scheme would be the fastest on larger instances (e.g. instances based on hh or ll1), as the cluster size is kept small. This does not happen, though, due to the clusters being expanded aggressively. The aggressive expansion happens when a lot of visits are uncovered and therefore added to every home carer's cluster. For the hh and ll1 instance groups, we therefore see that the fixed-size clustering is slower, but better than the all-preferred clustering. In some test runs with the fixed-size clustering scheme in the test groups [ll2,ll3]-td0-A and [ll2,ll3]-td3-B, the initial clustering has lead to very large branch-and-bound trees. This is visible in the averages. For the pair-wise disjoint clustering scheme the picture is more clear. As expected it is very fast, but it does not come without a price, as the solution qualities for this scheme generally are the worst.

Focusing on the impact of removal of idle time arcs (RA), Tables A.3–A.4 do not disclose much. It it very hard to find a pattern in the impact of this setting. Removal of idle time arcs may reduce the ESPPTW networks, but the removal could also lead to more visits being uncovered in the LP solution and therefore added to all home carer's clusters. This would increase the network sizes.

The table shows that when cluster expansion is allowed in every node in the branch-and-bound tree (ER = 0), the solution quality tends to be just better than when expansion is only allowed in the root node (ER = 1). This is expected, but still, if many visits are uncovered in the root LP solution, this could lead to large clusters, and thereby better solution quality.

Looking at the numbers for the test groups ending with A, B, and C, there does not seem to be a correlation between the performance of the different

clustering schemes and the number of generalised precedence constraints for an instance. None of the clustering schemes stand out with a consequently good or bad performance in either size A, B, or C. Likewise, there does not seem to be a correlation between the type of temporal dependency and the performance of the schemes. This is also sensible, as the clustering is preference-based and as such independent of types and numbers of temporal dependencies.

If we compare our results against the results from Bredström and Rönnqvist (2007), we are significantly faster in all test groups. We are able to verify their optimal solution values for the groups br-[01-05][S,M,L]-td0 and br-[06-08][S,M,L]-td0, and we are able to improve the best known solution values for the group br-[09-10][S,M,L]-td0 by 6% on average. For some instances we can prove optimality of the improved solutions. The settings 1-1-0 and 1-1-1 give better solution quality on average for the group br-[09-10][S,M,L]-td0 than the setting 0-0-0. This is possible, because we reach the time out limit on some test runs, and therefore the returned solution is not necessarily optimal, but only the best solution in the branch-and-bound tree at time out.

Table A.5 shows detailed statistics for individual test runs. The test runs shown here are chosen, because they are representative for the numbers from Tables A.3-A.4. It should be noted, that we have integerised the preference parameter in the instances from Bredström and Rönnqvist (2007), by scaling it with a factor of 10^4. In the process some rounding took place, and furthermore the results reported in Bredström and Rönnqvist (2007) are rounded. Therefore, reported numbers for the settings 0-0-0 and BR in Table A.5 will not necessarily match on all digits. Also, one should keep in mind that our lower bounds are tighter.

The detailed statistics for the test runs show that there is a clear connection between the run times and the sizes of the branch-and-bound trees, which is intuitively very sensible and expected. It should here be noticed that Bredström and Rönnqvist (2007) use significantly fewer branch-and-bound nodes than our algorithm. This is most probably due to their branching candidate selection which seems to perform very well.

Overall it can be said, that, as expected, run times can be decreased by using visit clustering, but this implies a decreased solution quality. It is difficult to point out the best settings as well as to quantify the speed gain/quality loss trade-off. As mentioned earlier, it is seen that the pair-wise disjoint clustering is by no doubt the fastest, but if quality is also taken into account, the all-preferred clustering scheme tends to perform best. Settings with CS = 1 have at least equally good and in most cases much better run times when compared to the settings 0-0-0 and do only have a significant loss in quality for the test group br-[01-05][S,M,L]-td0. The loss in quality is due to three out of 15 instances in the

Test case	Settings	Root LP value	Objective value	Uncovered visits	B&B nodes	B&B depth	Subproblems solved	Generated columns	LP solve time (s)	Subproblem time (s)	Running time (s)
br-05S-td0	0-0-0	-76277.00	-76277	0	11	4	284	687	0.31	0.17	0.60
br-05S-td0	1-0-0	-71289.00	923612	1	3	1	64	124	0.02	0.02	0.05
br-05S-td0	1-1-0	-71289.00	923612	1	3	1	88	154	0.04	0.02	0.07
br-05S-td0	1-0-1	-71289.00	923612	1	3	1	64	124	0.02	0.03	0.05
br-05S-td0	1-1-1	-71289.00	923612	1	3	1	88	154	0.03	0.02	0.07
br-05S-td0	2-0-0	-76277.00	-76277	0	9	3	200	436	0.12	0.08	0.26
br-05S-td0	2-1-0	-76277.00	-76277	0	9	3	152	366	0.11	0.06	0.22
br-05S-td0	2-0-1	-76277.00	-76277	0	9	3	200	436	0.14	0.06	0.28
br-05S-td0	2-1-1	-76277.00	-76277	0	9	3	152	366	0.12	0.06	0.22
br-05S-td0	3-0-0	933849.00	933849	1	1	0	16	38	0.00	0.00	0.01
br-05S-td0	3-1-0	933849.00	933849	1	1	0	16	34	0.00	0.00	0.01
br-05S-td0	3-0-1	933849.00	933849	1	1	0	16	38	0.00	0.00	0.02
br-05S-td0	3-1-1	933849.00	933849	1	1	0	16	34	0.00	0.01	0.02
br-05S-td0	BR	-76290.00	-76290	0	1	-	139	-	-	-	0.64
br-06M-td0	0-0-0	-380509.00	-379854	0	353	33	8989	8941	54.91	35.10	107.18
br-06M-td0	1-0-0	-380509.00	-379854	0	419	58	10284	8359	34.44	26.10	72.35
br-06M-td0	1-1-0	-379287.00	-378589	0	431	48	10904	8148	32.17	26.71	70.64
br-06M-td0	1-0-1	-380509.00	-379854	0	419	58	10284	8359	34.38	26.32	72.53
br-06M-td0	1-1-1	-379287.00	-378589	0	431	48	10904	8148	32.17	26.49	69.67
br-06M-td0	2-0-0	-376764.00	649853	1	77	38	1910	1399	2.67	4.16	8.22
br-06M-td0	2-1-0	-374594.00	-332648	0	109	54	2520	1701	4.04	5.23	11.37
br-06M-td0	2-0-1	-376764.00	641531	1	91	40	2240	1642	2.83	4.49	8.95
br-06M-td0	2-1-1	-374594.00	653339	1	177	43	4070	2254	5.11	7.72	15.67
br-06M-td0	3-0-0	-362005.00	686191	1	51	25	1060	530	0.22	1.73	2.42
br-06M-td0	3-1-0	-352443.00	-301188	0	33	16	840	464	0.14	1.41	1.87
br-06M-td0	3-0-1	-362005.00	1662244	2	47	13	880	436	0.20	1.31	1.81
br-06M-td0	3-1-1	-352443.00	3680521	4	31	15	520	282	0.08	0.81	1.12
br-06M-td0	BR	-386860.00	-379880	0	101	-	1861	-	-	-	247.88
hh	1-0-0	6851842.00	6851850	5	187	21	16755	12032	27.74	587.64	639.90
hh	1-1-0	6851859.00	6851867	5	141	15	11910	9090	17.65	422.74	458.90
hh	1-0-1	6851842.00	6851850	5	171	19	15915	11629	22.73	517.76	564.61
hh	1-1-1	6851859.00	6851867	5	139	15	11640	8792	19.51	613.39	694.34
hh	2-0-0	6858829.00	6858843	5	167	21	16890	20070	42.30	549.17	617.44
hh	2-1-0	7860857.00	7860868	6	121	21	6630	6896	17.90	235.45	266.05
hh	2-0-1	6858829.00	6858843	5	167	21	16305	19558	43.97	569.96	639.66
hh	2-1-1	7860857.00	7860868	6	115	21	5880	6012	13.21	186.02	209.15
hh	3-0-0	11869075.00	10870068	9	23	11	1650	1594	1.06	45.22	48.64
hh	3-1-0	12871112.00	11872103	10	21	10	1080	1012	0.48	29.91	31.94
hh	3-0-1	11869075.00	13869078	10	21	10	1140	1223	0.60	29.90	32.29
hh	3-1-1	12871112.00	14871115	11	19	9	810	855	0.36	22.95	24.50
ll1-td1-B	1-0-0	36920146.00	37926295	17	21	10	952	1487	2.05	31.07	35.56
ll1-td1-B	1-1-0	36920146.00	37926294	17	17	8	992	1494	2.08	23.22	26.91
ll1-td1-B	1-0-1	36920146.00	44921229	18	25	12	792	1128	1.43	21.31	24.26
ll1-td1-B	1-1-1	36920146.00	48924236	19	27	13	888	1230	1.80	18.80	22.20
ll1-td1-B	2-0-0	33245315.00	35937179	15	65	32	2216	3688	33.05	86.91	128.25
ll1-td1-B	2-1-0	33245315.00	34937306	17	51	25	1856	3524	28.04	60.63	95.23
ll1-td1-B	2-0-1	33245315.00	40932305	17	39	19	1448	2962	21.55	65.10	94.11
ll1-td1-B	2-1-1	33245315.00	41932341	18	73	36	2104	3535	32.74	65.87	106.49
ll1-td1-B	3-0-0	38767254.00	39934260	16	17	8	856	1074	0.88	15.47	17.73
ll1-td1-B	3-1-0	38767255.33	39934260	16	21	10	816	1035	1.02	13.13	15.56
ll1-td1-B	3-0-1	38767254.00	48931257	19	19	9	552	825	0.53	8.86	10.31
ll1-td1-B	3-1-1	38767255.33	48932218	19	25	12	576	721	0.58	8.39	9.96
ll2-td4-C	0-0-0	940394.00	940400	1	341	22	15393	29336	204.53	103.45	333.50
ll2-td4-C	1-0-0	940394.00	940400	1	153	14	6279	8202	29.95	19.74	56.44
ll2-td4-C	1-1-0	940398.75	940400	1	27	7	938	1488	4.19	3.10	8.33
ll2-td4-C	1-0-1	940394.00	940400	1	153	14	6202	8114	29.61	19.24	55.40
ll2-td4-C	1-1-1	940398.75	940400	1	27	7	854	1331	3.88	2.83	7.72
ll2-td4-C	2-0-0	940412.00	4941459	2	3	1	203	582	0.31	0.79	1.26
ll2-td4-C	2-1-0	940415.00	4941423	2	3	1	203	553	0.23	0.70	1.09
ll2-td4-C	2-0-1	940412.00	4941459	2	3	1	203	582	0.31	0.79	1.25
ll2-td4-C	2-1-1	940415.00	4941423	2	3	1	203	553	0.25	0.67	1.09
ll2-td4-C	3-0-0	3949532.00	3949532	4	1	0	84	233	0.03	0.22	0.30
ll2-td4-C	3-1-0	25951558.00	25951558	8	1	0	56	152	0.01	0.14	0.21
ll2-td4-C	3-0-1	3949532.00	3949532	4	1	0	84	233	0.02	0.23	0.31
ll2-td4-C	3-1-1	25951558.00	25951558	8	1	0	56	152	0.02	0.15	0.22

Table A.5: *Key statistics for selected test runs. Settings are written as CS-RA-ER.*

group having a single uncovered visit in the solutions with CS = 1. Focusing on the all-preferred clustering scheme, it seems to be slightly better for the solution quality to allow cluster expansion in every node of the branch-and-bound tree.

Lastly, it should be mentioned that we have also compared our solutions of hh, ll1, ll2, and ll3 with the current practice. Current practice is based partly on an automated heuristic and partly on manual planning. Unfortunately, it is not straight forward to make a comparison. It is clearly indicated, though, that we are able to enhance the service level. There is a significant decrease in the number of uncovered visits and a truly dramatic decrease in the number of necessary constraint adjustments. Constraint adjustments are another way of dealing with an uncovered visit, so that it is possible to fit the visit into the schedule anyway. Possible options are to: reduce the duration of the visit, extend the time window of the visit or extend the work shift of one of the home carers. This is done a lot in practice, and it makes comparison very difficult. However, any of these adjustments will naturally decrease the overall quality of the schedule. In the presented solution method, we have chosen to keep all the original constraints intact, and let the constraint adjustment be a manual post-processing task. This decision is supported by the fact that it is hard to put a quantitative penalty on all possible adjustments before solving.

A.8 Conclusion and future work

Initiated by the method's successful use in the VRPTW context, we have formulated the Home Care Crew Scheduling Problem as a set partitioning problem with side constraints and developed a branch-and-price solution algorithm. All temporal dependencies are modelled as generalised precedence constraints, and these constraints are enforced through the branching. To our knowledge, we are the first to enforce generalised precedence constraints in the branching for real-life problems. Based on the preference parameters, we have devised different visit clustering schemes. The visit clustering schemes for the exact branch-and-price framework are novel. We have compared the visit clustering schemes in order to survey how much they decrease run times, and how much they compromise optimality. The visit clustering schemes have been tested both on real-life problem instances and on generated test instances inspired by realistic settings. The tests have shown that by using clusters with only preferred visits, run times were significantly decreased, while there was only a loss of quality for few instances. The clustering schemes have allowed us to find solutions to instances that could not be solved to optimality. Summarised, our main contributions are: Development of visit clustering schemes for the Home Care Crew Scheduling Problem, and enforcement of generalised precedence constraints in

the branching for real-life problems.

We see a number of directions in which future work on this problem could go. One direction is improvement of the algorithm presented in this paper. New visit clustering schemes could be devised accompanied by cluster expansion schemes. For the clustering scheme with a fixed cluster size, it could be interesting to look into what determines a good cluster size for a given instance. It might be possible to express the cluster size as a function of number of visits and number of home carers.

Other very interesting and yet unexplored planning problems in home care are long-term planning and disruption management. In the long-term planning problem, the goal is to present a plan that spans e.g. half a year. The long-term problem does not decide how the visits should be assigned to the specific home carers, but only how to distribute the visits optimally on the weekdays and possibly in time windows.

In a disruption management or recovery situation the original plan has become infeasible due to unforeseen circumstances. Therefore, rescheduling of the home carers for the remains of the planning period (most likely the rest of the day) must take place. The goal of the rescheduling is to provide a new, feasible plan very fast, i.e. within minutes, with as few alterations to the original plan as possible. In many cases the disruption will only directly influence a smaller subset of the home carers, and an approach could be inspired by what Rezanova and Ryan (2010) do for train driver rescheduling.

References

Akker, J. van den, J. Hoogeveen, and J. van Kempen (2006). "Parallel machine scheduling through column generation: Minimax objective functions (extended abstract)". *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 4168 LNCS, pp. 648–659.

Begur, S. V., D. M. Miller, and J. R. Weaver (1997). "An integrated spatial DSS for scheduling and routing home-health-care nurses". *Interfaces* 27.4, pp. 35–48.

Bertels, S. and T. Fahle (2006). "A hybrid setup for a hybrid scenario: combining heuristics for the home health care problem". *Computers and Operations Research* 33.10, pp. 2866–2890.

Bredström, D. and M. Rönnqvist (Feb. 2007). *A branch and price algorithm for the combined vehicle routing and scheduling problem with synchronization*

constraints. Discussion Papers 2007/7. Department of Finance, Management Science, Norwegian School of Economics, and Business Administration.

Bredström, D. and M. Rönnqvist (2008). "Combined vehicle routing and scheduling with temporal precedence and synchronization constraints". *European Journal of Operational Research* 191.1, pp. 19–31.

Chabrier, A. (2006). "Vehicle Routing Problem with elementary shortest path based column generation". *Computers and Operations Research* 33.10, pp. 2972–2990.

Cheng, E. and J. L. Rich (1998). *A Home Health Care Routing and Scheduling Problem*. Tech. rep. Department of CAAM, Rice University.

Cordeau, J.-F., G. Desaulniers, J. Desrosiers, M. M. Solomon, and F. Soumis (2002). "VRP with Time Windows". In: *The Vehicle Routing Problem*. Ed. by P. Toth and D. Vigo. Society for Industrial and Applied Mathematics. Chap. 7, pp. 176–213.

Dohn, A., E. Kolind, and J. Clausen (2009a). "The manpower allocation problem with time windows and job-teaming constraints: A branch-and-price approach". *Computers and Operations Research* 36.4, pp. 1145–1157.

Dohn, A., M. S. Rasmussen, and J. Larsen (2011). "The Vehicle Routing Problem with Time Windows and Temporal Dependencies". *Networks*. (Accepted for publication).

Dror, M. (1994b). "Note on the complexity of the shortest path models for column generation in VRPTW". *Operations Research* 42.5, pp. 977–8.

Eveborn, P., P. Flisberg, and M. Rönnqvist (2006). "Laps Care—an operational system for staff planning of home care". *European Journal of Operational Research* 171.3, pp. 962–976.

Feillet, D., P. Dejax, M. Gendreau, and C. Gueguen (2004). "An exact algorithm for the elementary shortest path problem with resource constraints: application to some vehicle routing problems". *Networks* 44.3, pp. 216–29.

Foster, B. A. and D. M. Ryan (1976). "An Integer Programming Approach to the Vehicle Scheduling Problem". *Operational Research Quarterly* 27.2, pp. 367–384.

Garey, M. and D. Johnson (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York: W.H. Freeman and Co.

Gélinas, S., M. Desrochers, J. Desrosiers, and M. Solomon (1995). "A new branching strategy for time constrained routing problems with application to backhauling". *Annals of Operations Research* 61, pp. 91–109.

Ioachim, I., J. Desrosiers, F. Soumis, and N. Bélanger (1999). "Fleet assignment and routing with schedule synchronization constraints". *European Journal of Operational Research* 119.1, pp. 75–90.

Kallehauge, B., J. Larsen, O. B. Madsen, and M. Solomon (2005). "The Vehicle Routing Problem with Time Windows". English. In: *Column Generation*. Ed. by G. Desaulniers, J. Desrosiers, and M. M. Solomon. GERAD 25th anniversary series. New York: Springer. Chap. 3, pp. 67–98.

Lessel, C. R. (2007). "Ruteplanlægning i hjemmeplejen [Routing in Home Care]". Master's thesis. Department of Informatics and Mathematical Modelling, Technical University of Denmark.

Li, Y., A. Lim, and B. Rodrigues (2005). "Manpower allocation with time windows and job-teaming constraints". *Naval Research Logistics* 52.4, pp. 302–311.

Lim, A., B. Rodrigues, and L. Song (2004). "Manpower allocation with time windows". *Journal of the Operational Research Society* 55.11, pp. 1178–1186.

Lougee-Heimer, R. (2003). "The Common Optimization INterface for Operations Research: Promoting Open-Source Software in the Operations Research Community". *IBM Journal of Research and Development* 47.1, pp. 57–66.

Rezanova, N. J. and D. M. Ryan (2010). "The train driver recovery problem-A set partitioning based model and solution method". *Computers and Operations Research* 37.5, pp. 845–856.

Ryan, D. M. and B. Foster (1981). "An integer programming approach to scheduling". *Computer Scheduling of Public Transport. Urban Passenger Vehicle and Crew Scheduling. Proceedings of an International Workshop*, pp. 269–280.

Sekretariatet for ministerudvalget vedrørende kvalitet i den offentlige sektor (Nov. 2006). *Fakta på ældreområdet [Facts about the elderly care sector]*.

Thomsen, K. (2006). "Optimization on Home Care". Master's thesis. Department of Informatics and Mathematical Modelling, Technical University of Denmark.

Appendix B

The Vehicle Routing Problem with Time Windows and Temporal Dependencies

---

Anders Dohn, Matias Sevel Rasmussen, and Jesper Larsen

Accepted for publication in: *Networks* (2011).

The Vehicle Routing Problem with Time Windows and Temporal Dependencies*

Anders Dohn[1], Matias Sevel Rasmussen[1], and Jesper Larsen[1]

In this paper, we formulate the vehicle routing problem with time windows and temporal dependencies. The problem is an extension of the well studied vehicle routing problem with time windows. In addition to the usual constraints, a scheduled time of one visit may restrain the scheduling options of other visits. Special cases of temporal dependencies are synchronization and precedence constraints. Two compact formulations of the problem are introduced and the Dantzig-Wolfe decompositions of these formulations are presented to allow for a column-generation-based solution approach. Temporal dependencies are modeled by generalized precedence constraints. Four different master problem formulations are proposed and it is shown that the formulations can be ranked according to the tightness with which they describe the solution space. A tailored time window branching is used to enforce feasibility on the relaxed master problems. Finally, a computational study is carried out to quantitatively reveal strengths and weaknesses of the proposed formulations. It is concluded that, depending on the problem at hand, the best performance is achieved either by relaxing the generalized precedence constraints in the master problem, or by using a time-indexed model, where generalized precedence constraints are added as cuts when they become severely violated.

Keywords: Vehicle routing with time windows, Temporal dependency, Generalized precedence constraints, Time window branching, Relaxation, Column generation, Branch-and-price, Branch-and-cut-and-price, Set partitioning, Set covering, Integer programming

*Accepted for publication in: *Networks* (2011).

[1]Department of Management Engineering, Technical University of Denmark, Produktionstorvet, Building 424, DK-2800 Kgs. Lyngby, Denmark.

B.1 Introduction

The vehicle routing problem with time windows and temporal dependencies (VRPTWTD) is an extension of the vehicle routing problem with time windows (VRPTW). Given is a fixed set of customers with individual demands and with time windows specifying when each customer accepts service. The objective is to find routes for a number of vehicles, all starting and ending at a central depot in such a way that the total cost is minimized. The extension that we present here is concerned with temporal dependencies between customers. A temporal dependency which is often encountered in practical instances and that has received the most attention in the literature, is the rather strict requirement of synchronization between two visits. Synchronization on visits is also used to model rendezvous between vehicles. Other, less restrictive, dependencies are constraints on minimum overlap between visits and limits on minimum or maximum gaps between visits.

In this paper, a context-free approach to VRPTWTD is presented for the first time. We apply time window branching combined with time window reductions to restore feasibility with respect to temporal dependencies. We prove that the standardized modeling of temporal dependencies as generalized precedence constraints does not affect the efficiency of the solution method. Along with a direct formulation and a relaxed formulation, we introduce a time-indexed formulation with an implicit representation of generalized precedence constraints. We are able to rank the formulations theoretically, according to the tightness with which they describe the solution space. For computational testing, we introduce a fourth model, which is a hybrid of the relaxed formulation and the time-indexed formulation. Finally, we introduce a new set of context-free benchmark instances which enables a thorough quantitative analysis and which we hope will facilitate future research in this area. The main contribution of this paper is the formulation and comparison of models for VRPTWTD along with a generic and efficient solution approach.

There is a vast amount of literature on VRPTW and its variants. VRPTW is known to be NP-hard (Savelsbergh, 1985); nevertheless exact solution of the problem has received a lot of attention. The most successful approach is based on a Dantzig-Wolfe decomposition (Dantzig and Wolfe, 1960a) of the mathematical model using column generation in a branch-and-cut-and-price framework. The method was first proposed by Desrochers et al. (1992a). The most promising recent work is based on solution of the subproblem as an elementary shortest path problem with time windows and capacity constraints. Feillet et al. (2004) were the first to apply this idea and were followed by Chabrier (2006), Danna and Pape (2005), Jepsen et al. (2008), and Desaulniers et al. (2008) among others. The approach that we present here for VRPTWTD builds on the same

idea. See Kallehauge et al. (2005) for a recent review of the literature and a thorough description of the technique.

The motivation behind this work is the many practical applications of VRPTWTD. With the inclusion of temporal dependencies in the model, we are able to describe numerous concrete problems. As Kilby et al. (2000) point out, there is a need for more sophisticated models for the vehicle routing problem. They mention synchronization and precedence constraints as some of the relevant extensions.

Ioachim et al. (1999) describe a fleet assignment and routing problem with synchronization constraints. The problem is solved by column generation. A similar problem with synchronization is described by Bélanger et al. (2006). Rousseau et al. (2003) present the synchronized vehicle dispatching problem (SVDP), which is a dynamic vehicle routing problem with synchronization between vehicles. Constraint programming and local search are applied to arrive at high-quality feasible solutions. Lim et al. (2004) and Li et al. (2005) study a problem from the Port of Singapore, where technicians are allocated to service jobs. For each job, a certain combination of technicians with individual skills is needed. The technicians must be present at the same time, and hence the schedule for each technician must respect a number of synchronization constraints with other schedules. The problem is solved using metaheuristics. Another application with synchronization between visits is in ground handling at airports. Teams drive around at the airport and are assigned tasks on the parked aircraft. Dohn et al. (2009b) describe this setup and present exact solutions to the instances considered. Oron et al. (2008) consider ground handling with synchronization constraints as well, and present computational results for a tailored heuristic applied to data instances from an in-flight caterer in Malaysia. Bredström and Rönnqvist (2007) present an application of vehicle routing with synchronization constraints in home health care. A branch-and-price algorithm is applied to a realistic home care routing problem and yields promising results.

The generalization of synchronization to other temporal dependencies has been described for a few applications. Lesaint et al. (1998) present a workforce scheduling software from a practical perspective. In the problem described, both synchronization and various other sequencing constraints occur. Fügenschuh (2006) describes a problem in school bus routing. Busses must wait for each other at various intermediate stops and hence precedence relations are introduced for such stops. Fügenschuh refers to the problem as the vehicle routing problem with coupled time windows. Doerner et al. (2008) describe an application in blood collection from satellite locations for a central blood bank. Multiple visits at each location have to be scheduled with a certain slack between them. They refer to the vehicle problem as having interdependent time windows. Bredström and Rönnqvist (2008) modeled temporal dependencies for a

home care routing problem in a mixed integer programming model (MIP) which was solved with a standard MIP solver. In Justesen and Rasmussen (2008) and Dohn et al. (2008c) a similar application is described and solved using branch-and-price. Bredström and Rönnqvist (2008) have also continued their work in this direction. An application with general temporal dependencies in machine scheduling is described by Akker et al. (2006). Column generation is used to solve the problem. The pricing problem is primarily solved heuristically by local search and occasionally to optimality using a standard solver on an integer programming formulation of the pricing problem. van den Akker et al. (2000) and Bigras et al. (2008) describe machine scheduling problems and propose to apply column generation approaches to time-indexed formulations. Hence, their models have some similarities to the time-indexed formulation presented in this paper.

The paper is organized as follows. In Section B.2, we present two valid compact formulations of VRPTWTD. Possible decompositions of the compact formulations are presented and compared in Section B.3. For the decomposed models, a tailored branching method is required, which is described in Section B.4. A set of test instances are introduced in Section B.5 and the test results for these are found in Section B.6. Finally, we conclude on our findings and discuss possible areas for future research in Section B.7.

B.2 Model

In the following, we present two valid model formulations for VRPTWTD, namely a mixed-integer formulation that we refer to as the direct formulation and a time-indexed formulation. The mixed-integer formulation is an extension of the model commonly used for VRPTW, whereas a time-indexed model has not received the same amount of attention.

In the traditional vehicle routing problem with time windows, the objective is to find the cheapest set of routes to a set, $\mathcal{C}$, of n customers. Given is a fleet of identical vehicles, $\mathcal{V}$, which are located at a central depot. Typically, the depot is represented as two locations, namely a start depot, 0, and an end depot, $n+1$. Together with all customers, they form the set, $\mathcal{N}$. All vehicles have a capacity of q. Each customer, i, has a demand, d_i, and a time window, where it accepts service $[\alpha_i, \beta_i]$. α_i is the first possible service time. The vehicle is allowed to arrive before this time, but must then wait at the customer for the time window to open. β_i is the latest possible time of initiation at customer i. $[\alpha_0, \beta_0]$ denotes the scheduling horizon of the problem. Vehicles start at the depot at time α_0 and must return to the end depot no later than β_0. τ_{ij} gives the travel time

between any two customers, i and j. This may include service time at customer i. Traveling between the two customers also incurs a certain cost given by c_{ij}. We assume that q, d_i, α_i, β_i, and c_{ij} are non-negative integers and that τ_{ij} are positive integers, respecting the triangular inequality.

B.2.1 Direct formulation

The mathematical model of VRPTW is presented below. x_{ijk} are binary variables with $x_{ijk} = 1$, if vehicle k drives directly from customer i to customer j, $x_{ijk} = 0$, otherwise. s_{ik} are continuous variables and are defined as the start time for service at customer i, if the customer is serviced by vehicle k. Otherwise, $s_{ik} = 0$. Without restricting the model, we can fix $s_{0k} = \alpha_0, \forall k \in \mathcal{V}$ and $s_{n+1,k} = \beta_0, \forall k \in \mathcal{V}$.

$$\min \sum_{i\in\mathcal{N}} \sum_{j\in\mathcal{N}} \sum_{k\in\mathcal{V}} c_{ij} x_{ijk} \tag{B.1}$$

$$\sum_{j\in\mathcal{N}:j\neq i} \sum_{k\in\mathcal{V}} x_{ijk} = 1 \qquad \forall i \in \mathcal{C} \tag{B.2}$$

$$\sum_{i\in\mathcal{C}} d_i \sum_{j\in\mathcal{N}} x_{ijk} \leq q \qquad \forall k \in \mathcal{V} \tag{B.3}$$

$$\sum_{j\in\mathcal{N}} x_{0jk} = 1 \qquad \forall k \in \mathcal{V} \tag{B.4}$$

$$\sum_{i\in\mathcal{N}} x_{ihk} - \sum_{j\in\mathcal{N}} x_{hjk} = 0 \qquad \forall h \in \mathcal{C}, \forall k \in \mathcal{V} \tag{B.5}$$

$$\sum_{i\in\mathcal{N}} x_{i,n+1,k} = 1 \qquad \forall k \in \mathcal{V} \tag{B.6}$$

$$s_{ik} + \tau_{ij} - M(1 - x_{ijk}) \leq s_{jk} \qquad \forall i, j \in \mathcal{N}, \forall k \in \mathcal{V} \tag{B.7}$$

$$\alpha_i \sum_{j\in\mathcal{N}} x_{ijk} \leq s_{ik} \leq \beta_i \sum_{j\in\mathcal{N}} x_{ijk} \qquad \forall i \in \mathcal{C}, \forall k \in \mathcal{V} \tag{B.8}$$

$$x_{ijk} \in \{0, 1\} \qquad \forall i, j \in \mathcal{N}, \forall k \in \mathcal{V} \tag{B.9}$$

The objective is to minimize the total cost of all edges traveled (B.1). All customers must be visited by exactly one vehicle (B.2) and the route for each vehicle must respect the capacity of that vehicle (B.3). (B.4) and (B.6) ensure that each route starts and ends at the depot. We also need to ensure that routes are not segmented, i.e. if a vehicle arrives at a customer, it eventually leaves that customer again (B.5). If a vehicle is set to travel between two customers, there has to be enough time between the two visits (B.7). Finally, we need to make

sure that all time windows are respected (B.8). (B.8) also ensure that $s_{ik} = 0$ when vehicle k does not visit customer i. (B.9) are the integrality constraints on x_{ijk}.

In VRPTWTD, we furthermore have a number of temporal dependencies between customers. We are able to express all of these by generalized precedence constraints. We introduce the parameter δ_{ij} which specifies the minimum difference in time from customer i to customer j. The set Δ defines all customer pairs (i, j) for which a temporal dependency exists. The generalized precedence constraints are formulated as follows, where $\sum_{k \in \mathcal{V}} s_{ik}$ is the start time of service at customer i.

$$\sum_{k \in \mathcal{V}} s_{ik} + \delta_{ij} \leq \sum_{k \in \mathcal{V}} s_{jk} \qquad \forall (i, j) \in \Delta \tag{B.10}$$

Constraint (B.10) can be used to model all the temporal dependencies that were observed in the literature review. There may be dependencies between several customers, e.g. synchronization of three or more customers. Such dependencies are modeled by applying the corresponding pair wise dependencies. In this paper, we will focus on five kinds of temporal dependencies that are commonly found in practice. These are visualized in Figure B.1.

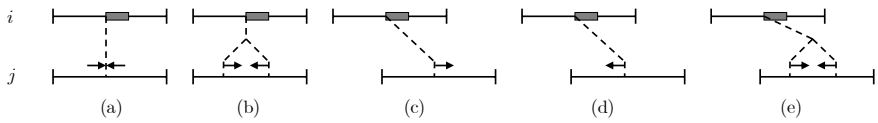


Figure B.1: *Five kinds of temporal dependencies that are often encountered in practice. Each of the five subfigures shows the time windows of two customers i and j with a temporal dependency between them. Assuming some start time for customer i, the dashed line together with the arrows give the corresponding feasible part of the time window of customer j. (a) synchronization, (b) overlap, (c) minimum difference, (d) maximum difference, (e) minimum+maximum difference.*

It is straight forward to model the temporal dependencies of Figure B.1 using constraints (B.10). The correct values for δ_{ij} and δ_{ji} are listed in Table B.1.

	Temporal dependency	δ_{ij}	δ_{ji}
(a)	Synchronization	0	0
(b)	Overlap	$-\text{dur}_j$	$-\text{dur}_i$
(c)	Minimum difference	$\text{diff}_{\min}$	N/A
(d)	Maximum difference	N/A	$-\text{diff}_{\max}$
(e)	Minimum+maximum difference	$\text{diff}_{\min}$	$-\text{diff}_{\max}$

Table B.1: *Parameter values for the five temporal dependencies of Figure B.1. dur_i is the service time at customer i. $diff_{\min}$ and $diff_{\max}$ are, respectively, the minimum and maximum differences required.*

B.2.2 Time-indexed formulation

Time-indexed formulations have not received much attention in the column generation context of VRPTW. A time-indexed formulation is usually disregarded because of its vast size. It is, however, popular in the formulation of machine scheduling problems, as it gives a tight description of precedence constraints. Here, we present the time-indexed model of VRPTWTD, as it will be used to strengthen the bounds in the branch-and-price algorithm. We introduce the index $t \in \mathcal{T}$ on the x-variable, with $\mathcal{T} = \{\alpha_0, \ldots, \beta_0\}$. x_{ijkt} is defined as: $x_{ijkt} = 1$, if vehicle k services customer i at time t and then drives directly to customer j. $x_{ijkt} = 0$, otherwise. Further, define the auxiliary sets $\mathcal{T}^{\tau}_{tij} = \{\alpha_0, \ldots, \min\{\beta_0, t + \tau_{ij} - 1\}\}$ and $\mathcal{T}^{\delta}_{tij} = \{\alpha_0, \ldots, \min\{\beta_0, t + \delta_{ij} - 1\}\}$.

$$\min \sum_{i\in\mathcal{N}}\sum_{j\in\mathcal{N}}\sum_{k\in\mathcal{V}}\sum_{t\in\mathcal{T}} c_{ij}x_{ijkt} \tag{B.11}$$

$$\sum_{j\in\mathcal{N}:j\neq i}\sum_{k\in\mathcal{V}}\sum_{t\in\mathcal{T}} x_{ijkt} = 1 \qquad \forall i \in \mathcal{C} \tag{B.12}$$

$$\sum_{i\in\mathcal{C}} d_i \sum_{j\in\mathcal{N}}\sum_{t\in\mathcal{T}} x_{ijkt} \leq q \qquad \forall k \in \mathcal{V} \tag{B.13}$$

$$\sum_{j\in\mathcal{N}}\sum_{t\in\mathcal{T}} x_{0jkt} = 1 \qquad \forall k \in \mathcal{V} \tag{B.14}$$

$$\sum_{i\in\mathcal{N}}\sum_{t\in\mathcal{T}} x_{ihkt} - \sum_{j\in\mathcal{N}}\sum_{t\in\mathcal{T}} x_{hjkt} = 0 \qquad \forall h \in \mathcal{C}, \forall k \in \mathcal{V} \tag{B.15}$$

$$\sum_{i\in\mathcal{N}}\sum_{t\in\mathcal{T}} x_{i,n+1,kt} = 1 \qquad \forall k \in \mathcal{V} \tag{B.16}$$

$$\sum_{k\in\mathcal{V}}\sum_{t'=t,\ldots,\beta_0} x_{ijkt'} + \sum_{h\in\mathcal{N}}\sum_{k\in\mathcal{V}}\sum_{t'\in\mathcal{T}^{\tau}_{tij}} x_{jhkt'} \leq 1 \qquad \forall i,j \in \mathcal{N}, \forall t \in \mathcal{T} \tag{B.17}$$

$$\sum_{h\in\mathcal{N}}\sum_{k\in\mathcal{V}}\sum_{t'=t,\ldots,\beta_0} x_{ihkt'} + \sum_{h\in\mathcal{N}}\sum_{k\in\mathcal{V}}\sum_{t'\in\mathcal{T}^{\delta}_{tij}} x_{jhkt'} \leq 1 \qquad \begin{array}{l}\forall (i,j) \in \Delta,\\ \forall t \in \mathcal{T}\end{array} \tag{B.18}$$

$$x_{ijkt} = 0 \qquad \begin{array}{l}\forall i \in \mathcal{C}, j \in \mathcal{N}, \forall k \in \mathcal{V},\\ \forall t \in \{\alpha_0,\ldots,\alpha_i - 1\}\\ \cup\{\beta_i + 1,\ldots,\beta_0\}\end{array} \tag{B.19}$$

$$x_{ijkt} \in \{0,1\} \qquad \begin{array}{l}\forall i,j \in \mathcal{N}, \forall k \in \mathcal{V},\\ \forall t \in \mathcal{T}\end{array} \tag{B.20}$$

Constraints (B.11)–(B.16) are similar to Constraints (B.1)–(B.6), where we now sum over the time index as well. Constraints (B.17) provide the required travel time between customers. If any vehicle goes directly from customer i to customer j, and if customer i is scheduled at time t or later, then j cannot be scheduled at time $t+\tau_{ij}-1$ or earlier. The strength of this model lies in the formulation of generalized precedence constraints (B.18). Constraints (B.18) are the equivalent of Constraints (B.10) of the former model. Similarly to the former constraints, these constraints state that if customer i is scheduled anywhere from time t and onward, then customer j is not scheduled before time $t+\delta_{ij}$. This is valid for all $t \in \mathcal{T}$. Constraints (B.19) enforce the time windows and (B.20) are the integrality constraints.

B.3 Decomposition

As described earlier, Dantzig-Wolfe decomposition has been very successful in exact optimization of VRPTW. The decomposition splits the problem into a set-partitioning master problem and a resource constrained shortest path subproblem. See e.g. Kallehauge et al. (2005) for a thorough exposition. In the traditional VRPTW formulation, Constraints (B.2) are the only constraints that link the vehicles. Without these, we can solve the problem separately for each vehicle. Hence, the problem is split into a subproblem, where feasible routes are generated and a master problem, where these routes are combined.

B.3.1 Master problem

We propose four applicable formulations of the master problem and rank them according to the tightness with which they describe the solution space.

Direct formulation

The introduced generalized precedence constraints apply to routes from separate vehicles, and hence these will be part of the new master problem. In the full master problem, we have the set of all feasible routes, $\mathcal{R}$. Each route has a cost of c_r and is defined by the customers visited and the time of each such visit, described by two parameters, a_i^r and s_i^r. For each route, r, and each customer, i, if customer i is in the route r, we set $a_i^r = 1$ and set s_i^r equal to the time of that visit. If the customer is not in the route, $a_i^r = 0$ and $s_i^r = 0$. In column generation, the variables of the master problem are generated iteratively and the set of variables available in a specific iteration is denoted $\mathcal{R}'$. Decision variables for the master problem are denoted λ_r, with $\lambda_r = 1$, if route r is used, and $\lambda_r = 0$, otherwise. The LP-relaxation of the master problem defined by a subset of the decision variables, $\mathcal{R}'$, is denoted the *restricted master problem* and is formulated below. The master problem is obtained by decomposing the compact direct formulation (B.1)-(B.10).

$$\min \sum_{r \in \mathcal{R}'} c_r \lambda_r \tag{B.21}$$

$$\sum_{r \in \mathcal{R}'} a_i^r \lambda_r = 1 \qquad \forall i \in \mathcal{C} \tag{B.22}$$

$$\sum_{r \in \mathcal{R}'} s_i^r \lambda_r + \delta_{ij} \leq \sum_{r \in \mathcal{R}'} s_j^r \lambda_r \qquad \forall (i,j) \in \Delta \tag{B.23}$$

$$\lambda_r \geq 0 \qquad \forall r \in \mathcal{R}^{'} \tag{B.24}$$

The corresponding subproblem is that of generating negative reduced cost routes for the master problem (B.21)–(B.24). In this context, we refer to the model as the *direct formulation*. The main disadvantage of the model is that it introduces linear time costs in the subproblem, namely the dual variables of Constraints (B.23). Hence, the subproblem is a resource constrained shortest path problem with linear node costs. Another issue is that s_i^r is a non-binary parameter, and the introduction of non-binary parameters in the master problem is usually a feature that leads to highly fractional solutions.

Time-indexed formulation

In the time-indexed formulation, the master problem contains only binary parameters. Constraints (B.12) and (B.18) link the vehicles and must therefore remain in the master problem. The parameters of the time-indexed master problem are defined as $a_{it}^r = 1$ if customer i is scheduled at time t in route r, and $a_{it}^r = 0$ otherwise. The decision variable λ_r has the same definition as in the previous model. The relation to the decision variables of model (B.11)-(B.20) is: $\sum_{j \in \mathcal{N}} \sum_{k \in \mathcal{V}} x_{ijkt} = \sum_{r \in \mathcal{R}'} a_{it}^r \lambda_r, \forall i \in \mathcal{C}, \forall t \in \mathcal{T}$. The restricted master problem of the *time-indexed formulation* is:

$$\min \sum_{r \in \mathcal{R}'} c_r \lambda_r \tag{B.25}$$

$$\sum_{r \in \mathcal{R}'} \sum_{t' \in \mathcal{T}} a_{it'}^r \lambda_r = 1 \qquad \forall i \in \mathcal{C} \tag{B.26}$$

$$\sum_{r \in \mathcal{R}'} \sum_{t'=t,\ldots,\beta_0} a_{it'}^r \lambda_r + \sum_{r \in \mathcal{R}'} \sum_{t' \in \mathcal{T}_{tij}^{\delta}} a_{jt'}^r \lambda_r \leq 1 \qquad \forall (i,j) \in \Delta, \forall t \in \mathcal{T} \tag{B.27}$$

$$\lambda_r \geq 0 \qquad \forall r \in \mathcal{R}^{'} \tag{B.28}$$

The obvious problem with the time-indexed restricted master problem (B.25)–(B.28) is the number of constraints of type (B.27). The scheduling horizon is usually large enough to make this model intractable in realistic problems. The subproblem is a resource constrained shortest path problem with time-dependent costs. The costs may be different for each time step. This is very unlikely, however. Most of the constraints of type (B.27) will be non-binding and this leaves the corresponding dual variables equal to 0. For the same reason, we may choose to introduce them, only when they become violated.

Relaxed formulation

A third way of approaching the problem is to simply disregard the temporal dependencies in the master problem. The dependencies must then be enforced by the branching scheme. This approach is used for synchronization by Dohn et al. (2009b) and Bredström and Rönnqvist (2007) and for generalized precedence constraints by Justesen and Rasmussen (2008). It leaves the following master problem, which is identical to the master problem of the VRPTW decomposition. Here, we refer to it as the *relaxed formulation*.

$$\min \sum_{r \in \mathcal{R}'} c_r \lambda_r \tag{B.29}$$

$$\sum_{r \in \mathcal{R}'} a_i^r \lambda_r = 1 \qquad \forall i \in \mathcal{C} \tag{B.30}$$

$$\lambda_r \geq 0 \qquad \forall r \in \mathcal{R}^{'} \tag{B.31}$$

Limited time-indexed formulation

In the time-indexed formulation (B.25)-(B.28), it is possible to include only a subset of Constraints (B.27) and this idea is implemented in a limited version of the time-indexed formulation. The formulation can be seen as a hybrid of the time-indexed formulation and the relaxed formulation. Obviously, all generalized precedence constraints must be respected in a feasible solution. Therefore, if a violation occurs for a generalized precedence constraint, which is not in the subset of included constraints, the constraint is instead enforced by branching, like in the relaxed formulation.

In our case, we have chosen to define the subset of generalized precedence constraints dynamically. More specifically, we only add cuts if they are maximally violated, i.e. if the left hand side of constraint (B.27) is equal to 2. When a cut has been added it stays in the model. Smaller violations are handled by

the branching scheme. How to identify violated constraints is described in more detail in Section B.3.2.

Strength of the formulations

The relaxed formulation is obviously a relaxation of both the direct formulation, the full time-indexed formulation, and the limited time-indexed formulation. An interesting result is that the direct formulation is also a relaxation of the time-indexed formulation, and we are hence able to rank the models according to their strength.

Proposition B.1 *[The time-indexed master problem formulation is a stronger formulation than the direct master problem formulation.]*

Proof. In the following, we assume that we have a solution to (B.25)–(B.28) and prove that the solution is also feasible for Constraints (B.21)–(B.24). For all problems with a feasible solution, it holds that $\alpha_0 + \delta_{ij} - 1 \leq \beta_0, \forall (i,j) \in \Delta$ and hence a special case of (B.27) with $t = \alpha_0$ is:

$$\sum_{r \in \mathcal{R}'} \sum_{t' = \alpha_0, \ldots, \beta_0} a^r_{it'} \lambda_r + \sum_{r \in \mathcal{R}'} \sum_{t' = \alpha_0, \ldots, \alpha_0 + \delta_{ij} - 1} a^r_{jt'} \lambda_r \leq 1 \qquad \forall (i,j) \in \Delta \quad \text{(B.32)}$$

Using (B.26) this entails the rather obvious:

$$\sum_{r \in \mathcal{R}'} \sum_{t' = \alpha_0, \ldots, \alpha_0 + \delta_{ij} - 1} a^r_{jt'} \lambda_r = 0 \quad \forall (i,j) \in \Delta \quad \text{(B.33)}$$

$$\Downarrow$$

$$\sum_{r \in \mathcal{R}'} \sum_{t' = \alpha_0, \ldots, t} a^r_{jt'} \lambda_r = 0 \quad \begin{array}{l} \forall (i,j) \in \Delta, \\ t = \alpha_0, \ldots, \alpha_0 + \delta_{ij} - 2 \end{array} \quad \text{(B.34)}$$

$$\Downarrow$$

$$\sum_{r \in \mathcal{R}'} \sum_{t' \in \mathcal{T}} a^r_{it'} \lambda_r + \sum_{r \in \mathcal{R}'} \sum_{t' = \alpha_0, \ldots, t} a^r_{jt'} \lambda_r = 1 \quad \begin{array}{l} \forall (i,j) \in \Delta, \\ t = \alpha_0, \ldots, \alpha_0 + \delta_{ij} - 2 \end{array} \quad \text{(B.35)}$$

Summing Constraints (B.26), (B.35), and (B.27) over t, we get the following for $(i,j) \in \Delta$:

$$\sum_{r\in\mathcal{R}'}\sum_{t'\in\mathcal{T}} a^r_{it'}\lambda_r = 1$$

For $t = \alpha_0, \ldots, \alpha_0 + \delta_{ij} - 2$:

$$\sum_{r\in\mathcal{R}'}\sum_{t'\in\mathcal{T}} a^r_{it'}\lambda_r + \sum_{r\in\mathcal{R}'}\sum_{t'=\alpha_0,\ldots,t} a^r_{jt'}\lambda_r = 1$$

For $t = \alpha_0, \ldots, \beta_0$:

$$\sum_{r\in\mathcal{R}'}\sum_{t'=t,\ldots,\beta_0} a^r_{it'}\lambda_r + \sum_{r\in\mathcal{R}'}\sum_{t'=\alpha_0,\ldots,\min\{\beta_0,t+\delta_{ij}-1\}} a^r_{jt'}\lambda_r \leq 1$$

$$\sum_{r\in\mathcal{R}'}\sum_{t'\in\mathcal{T}}(t' - \alpha_0 + 1 + \delta_{ij})a^r_{it'}\lambda_r + \sum_{r\in\mathcal{R}'}\sum_{t'\in\mathcal{T}}(\beta_0 + \delta_{ij} - t')a^r_{jt'}\lambda_r \leq \beta_0 - \alpha_0 + \delta_{ij} + 1$$

Therefore, for any feasible solution of (B.25)–(B.28), we have for $(i,j) \in \Delta$:

$$\begin{aligned}
0 \leq\ & \beta_0 - \alpha_0 + \delta_{ij} + 1 - \sum_{r\in\mathcal{R}'}\sum_{t'\in\mathcal{T}}(t' - \alpha_0 + 1 + \delta_{ij})a^r_{it'}\lambda_r \\
& - \sum_{r\in\mathcal{R}'}\sum_{t'\in\mathcal{T}}(\beta_0 + \delta_{ij} - t')a^r_{jt'}\lambda_r && \text{(B.36)}\\
=\ & \beta_0 - \alpha_0 + \delta_{ij} + 1 - \sum_{r\in\mathcal{R}'}\sum_{t'\in\mathcal{T}} t' a^r_{it}\lambda_r - (-\alpha_0 + 1 + \delta_{ij})\sum_{r\in\mathcal{R}'}\sum_{t'\in\mathcal{T}} a^r_{it'}\lambda_r \\
& - (\beta_0 + \delta_{ij})\sum_{r\in\mathcal{R}'}\sum_{t'\in\mathcal{T}} a^r_{jt'}\lambda_r + \sum_{r\in\mathcal{R}'}\sum_{t'\in\mathcal{T}} t' a^r_{jt'}\lambda_r && \text{(B.37)}\\
=\ & \beta_0 - \alpha_0 + \delta_{ij} + 1 - \sum_{r\in\mathcal{R}'}\sum_{t'\in\mathcal{T}} t' a^r_{it'}\lambda_r + \alpha_0 - 1 - \delta_{ij} \\
& - \beta_0 - \delta_{ij} + \sum_{r\in\mathcal{R}'}\sum_{t'\in\mathcal{T}} t' a^r_{jt'}\lambda_r && \text{(B.38)}\\
=\ & \sum_{r\in\mathcal{R}'}\sum_{t'\in\mathcal{T}} t' a^r_{jt'}\lambda_r - \sum_{r\in\mathcal{R}'}\sum_{t'\in\mathcal{T}} t' a^r_{it'}\lambda_r - \delta_{ij} && \text{(B.39)}\\
=\ & \sum_{r\in\mathcal{R}'} s^r_j\lambda_r - \sum_{r\in\mathcal{R}'} s^r_i\lambda_r - \delta_{ij} && \text{(B.40)}
\end{aligned}$$

The result in (B.38) is based on (B.26). The final result in (B.40) comes from the following relation between the parameters of the models (B.21)–(B.24) and

(B.25)–(B.28): $s_i^r = \sum_{t \in \mathcal{T}} t a_{it}^r$. The result in (B.40) proves that any feasible solution of (B.25)–(B.28) also respects (B.23). (B.22) is trivially respected as $a_i^r = \sum_{t' \in T} a_{it'}^r$, and hence (B.21)–(B.24) is a relaxation of (B.25)–(B.28).

To illustrate that the two formulations are not equally strong, we consider the following small example. Take two customers $i = 1$ and $j = 2$ with $\delta_{12} = 2$. Three simple routes cover these two customers with $a_1^1 = 1, a_2^2 = 1, a_2^3 = 1$ and $s_1^1 = 1, s_2^2 = 2, s_2^3 = 4$ for model (B.21)–(B.24). In model (B.25)–(B.28) this corresponds to $a_{11}^1 = 1, a_{22}^2 = 1, a_{24}^3 = 1$. A solution with $\lambda_1 = 1, \lambda_2 = 0.5, \lambda_3 = 0.5$ is feasible in (B.21)–(B.24) but not in (B.25)–(B.28). This is verified by inspecting (B.23) for $i = 1, j = 2$:

$$\sum_{r \in \mathcal{R}'} s_1^r \lambda_r + \delta_{12} \leq \sum_{r \in \mathcal{R}'} s_2^r \lambda_r \Rightarrow 1 + 2 \leq 3$$

and (B.27) for $i = 1, j = 2, t = 1$:

$$\sum_{r \in \mathcal{R}'} (a_{11}^r \lambda_r + a_{12}^r \lambda_r + a_{13}^r \lambda_r + a_{14}^r \lambda_r) + \sum_{r \in \mathcal{R}'} (a_{21}^r \lambda_r + a_{22}^r \lambda_r) = 1 + 0.5 \not\leq 1$$

□

Using the above result, we can conclude that the full time-indexed formulation is a stronger formulation than the direct formulation. The direct formulation in turn is stronger than the relaxed formulation. In the same way, we also know that the full time-indexed formulation is a stronger formulation than the limited time-indexed formulation, which is stronger than the relaxed formulation. It is not possible to rank the direct formulation and the limited time-indexed formulation.

A nice property of the time-indexed model is that it has only been relaxed with respect to integrality and this means that if we can restore integrality, we have a feasible solution. In this paper, we will only consider branching to restore integrality, and hence the advantage may not seem immediate. For VRPTW, a significant amount of work has been done on cut generation to remove fractional solutions. Such cuts could be added to the time-indexed model of VRPTWTD as well, and this may restore integrality without the need of branching. See e.g. the work of Kohl et al. (1999), Cook and Rich (2001), Lysgaard et al. (2004), and Jepsen et al. (2008) for more on the subject.

B.3.2 Identifying violated cuts

As described earlier, the generalized precedence constraints (B.27) of the time-indexed master problem (B.25)–(B.28) are only represented implicitly. The constraints are added as cuts, as they become violated. The constraint is repeated below.

$$\sum_{r\in\mathcal{R}'}\sum_{t'=t,\ldots,\beta_0} a^r_{it'}\lambda_r + \sum_{r\in\mathcal{R}'}\sum_{t'\in\mathcal{T}^{\delta}_{tij}} a^r_{jt'}\lambda_r \leq 1 \qquad \forall(i,j)\in\Delta, \forall t\in\mathcal{T} \quad \text{(B.27)}$$

In theory, we have to check for violations for all $t \in \mathcal{T}$, but actually it is possible to do with significantly less. As a^r_{it} is a binary parameter and $\lambda_r \geq 0$, the sum $\sum_{r\in\mathcal{R}'}\sum_{t'=t,\ldots,\beta_0} a^r_{it'}\lambda_r$ is non-increasing for increasing t. Correspondingly, the sum $\sum_{r\in\mathcal{R}'}\sum_{t'=\alpha_0,\ldots,\min\{\beta_0,t+\delta_{ij}-1\}} a^r_{jt'}\lambda_r$ is non-decreasing. Constraints (B.27) are never violated for $t = \alpha_0$ as such violations are prevented by preprocessing the time windows, see Section B.4.1. Therefore, for customer i, we only need to check for violations with any t where $\exists r \in \mathcal{R}' : a^r_{it}\lambda_r > 0$, i.e. any point in time where customer i is scheduled (possibly with a fractional value). It is easy to generate a list of all t where $\exists r \in \mathcal{R}' : a^r_{it}\lambda_r > 0$, by running through the routes of all variables with positive values and registering the time of service for each customer. By separating cuts as described, we are not adding all violated cuts, but we are sure to add at least one cut for each customer, if any cuts are violated for that customer.

B.3.3 Subproblem

The subproblem of the Dantzig-Wolfe decomposition of VRPTW can be solved as an elementary shortest path problem with time windows and capacity constraints (ESPPTWCC). Any feasible solution of the subproblem with negative cost represents a column with negative reduced cost in the master problem and may therefore enter the basis. The subproblem consists of Constraints (B.3)–(B.9). The variables are defined as in the compact formulation, but now for the single vehicle under consideration, i.e. the index k has been removed. The objective function of the subproblem becomes:

$$\min \sum_{i\in\mathcal{N}}\sum_{j\in\mathcal{N}} (c_{ij} - \pi_i)x_{ij} \quad \text{(B.41)}$$

$\pi_i, i \in \mathcal{N}$, are the dual variables of Constraints (B.30) of the VRPTW master problem. Dror (1994a) proves that ESPPTW is NP-hard in the strong sense and hence no pseudo-polynomial algorithms are likely to exist. The subproblem is usually solved with a dynamic label setting algorithm. Desrochers et al. (1992a) presented a dynamic algorithm for the non-elementary version of the subproblem. This algorithm was adjusted to handle the elementary problem by Feillet et al. (2004) and superior results based on this method have been presented recently, see e.g. Desaulniers et al. (2008). The idea in the label setting algorithm is to represent partial paths by labels. Given a label for some partial path, it is possible to expand the path by creating new labels in nodes that can possibly extend the current partial path. The length of the path is increased by one, and the process continues iteratively.

The subproblem of the direct formulation must consider the dual variables of Constraints (B.22), π_i, and additionally the dual variables of Constraints (B.23), σ_{ij}, and the objective function becomes:

$$\min \sum_{i \in \mathcal{N}} \sum_{j \in \mathcal{N}} (c_{ij} - \pi_i)\, x_{ij} - \sum_{(i,j) \in \Delta} \sigma_{ij} s_i + \sum_{(j,i) \in \Delta} \sigma_{ji} s_i \tag{B.42}$$

As described previously, the subproblem is now a resource constrained shortest path problem with linear node costs, which makes it much harder to solve. Ioachim et al. (1998) describe a dynamic algorithm to solve the acyclic version of this problem. A similar cyclic problem is solved as a subproblem by Christiansen and Nygreen (2005).

The subproblem of the time-indexed formulation has the following objective function which we split in three parts for easy reference:

$$\min \sum_{i \in \mathcal{N}} \sum_{j \in \mathcal{N}} (c_{ij} - \pi_i) \sum_{t \in \mathcal{T}} x_{ijt} \tag{B.43a}$$

$$- \sum_{(i,j) \in \Delta} \sum_{t \in \mathcal{T}} \sum_{t' = \alpha_0, \ldots, t} \rho_{ijt'} x_{ijt} \tag{B.43b}$$

$$- \sum_{(j,i) \in \Delta} \sum_{t \in \mathcal{T}} \sum_{t' = \max\{\alpha_0, t - \delta_{ji} + 1\}, \ldots, \beta_0} \rho_{jit'} x_{ijt} \tag{B.43c}$$

where π_i are the dual variables of Constraints (B.26) and ρ_{ijt} are the non-positive dual variables of Constraints (B.27). In the worst case, this objective function introduces a distinct cost for each time step. In a label setting algorithm

this means that we have to create a label for each time step and hence the number of labels explodes immediately. In practice, only a few constraints of type (B.27) are binding and therefore only few ρ_{ijt} have non-zero values.

The idea in the basic label setting algorithm is to create and keep only labels that are not dominated by better labels. With the objective function (B.43), we have a lot of potential labels. It is, however, only an advantage to postpone a visit, if it can possibly decrease the objective value. As $\rho_{ijt} \leq 0$ and $x_{ijt} \in \{0, 1\}$, adjusting time for a certain visit can only decrease the objective, if it removes terms in (B.43b) or (B.43c). (B.43a) is neutral to service time of customers, i.e. for a given transition (i, j) the contribution to the objective function coefficient of x_{ijt} is the same for all $t \in \mathcal{T}$. The smallest contribution from (B.43b) is obtained by the smallest possible value of t as $\sum_{t'=\alpha_0,\ldots,t} \rho_{ijt'}$ is non-increasing over t for given i, j. Therefore, for customer i, we need a label for the earliest possible time t^0. Only the value of (B.43c) will decrease as t is increased (for given i, j) and only when terms with $\rho_{jit'} < 0$ are excluded. The value of (B.43c) for t is lower than the corresponding sum for $t - 1$ when $\rho_{ji(t-\delta_{ji})} < 0$, i.e.:

$$\rho_{ji(t-\delta_{ji})} < 0 \Rightarrow \sum_{t'=\max\{\alpha_0, t-\delta_{ji}\},\ldots,\beta_0} \rho_{jit'} < \sum_{t'=\max\{\alpha_0, t-\delta_{ji}+1\},\ldots,\beta_0} \rho_{jit'}$$

The full objective function possibly decreases for such t and hence we need one label for each $t \in \{t^0 + 1, \ldots, \beta_i\}$, where $\exists (j, i) \in \Delta : \rho_{ji(t-\delta_{ji})} < 0$. For all other potential labels, there will always be a label earlier in time with the same or less cost.

A small improvement, that we found to have a significant effect, is to include knowledge of mutually exclusive customers. Some temporal dependencies like e.g. synchronization and overlap make it impossible to include both customers in the same route. In these methods, such a restriction is imposed in the master problem or in the branching scheme. Hence, routes could be generated that would never occur in a feasible solution. By excluding the occurrence of mutually exclusive customers in all routes generated in the subproblem, the LP-bounds get stronger and as a consequence the algorithm is more efficient. The dominance scheme in the subproblem solver is also modified to utilize this knowledge. By visiting one of two mutually exclusive customers, the other becomes unreachable. Hence by updating the set of unreachable customers appropriately, two labels which have visited two different mutually exclusive costumers can still be compared in the dominance check, as their possible extensions are identical.

B.4 Branching

The master problem models presented in the previous section are relaxations. Therefore, we may need to apply branch and bound in order to get to a feasible solution. The λ-variables of the master problems were introduced as binary variables, but the integer property has been relaxed to allow solution by an LP-solver. Therefore, integrality needs to be restored by branching. A lot of work has already been done for VRPTW in this respect. See e.g. Kallehauge et al. (2005) for a review. In the relaxed formulation, the generalized precedence constraints have also been relaxed. Therefore, in this model, we need a branching method that will also restore feasibility with respect to temporal dependencies.

Gélinas et al. (1995) proposed to branch on time variables in order to arrive at integer-feasible solutions. This type of branching was also used to enforce synchronization by Ioachim et al. (1999), Dohn et al. (2009b), and Bredström and Rönnqvist (2007), and for general temporal dependencies by Justesen and Rasmussen (2008). Time window branching is not complete with respect to integer feasibility and hence has to be complemented by another branching scheme, e.g. traditional flow variable branching.

B.4.1 Time window reduction

Before describing the actual branching scheme, we introduce a simple reduction technique based on the generalized precedence constraints. For any two customers, i and j with $(i, j) \in \Delta$, it is possible to reduce the time windows as follows:

	Customer i	Customer j
Old time windows	$[\alpha_i, \beta_i]$	$[\alpha_j, \beta_j]$
New time windows	$[\alpha_i, \min\{\beta_i, \beta_j - \delta_{ij}\}]$	$[\max\{\alpha_j, \alpha_i + \delta_{ij}\}, \beta_j]$

These reductions are illustrated in Figure B.2. The reductions are used to preprocess the time windows and may also be used anywhere in the branching tree. This technique is essential when applying time window branching.

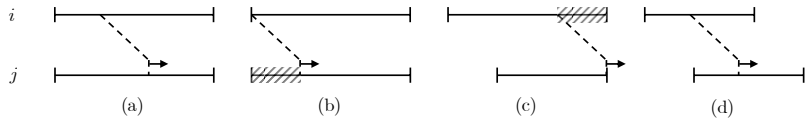


Figure B.2: *Time window reductions. (a) The original time windows. (b) Using the generalized precedence constraint, the first part of the time window of customer j is removed. (c) In a similar way, the last part of the time window of customer i is removed. (d) The time windows after the reduction.*

B.4.2 Time window branching

In a feasible solution of VRPTWTD, all visits are scheduled at exactly one point in time and all generalized precedence constraints are respected. In the relaxed formulation, a solution may be integer feasible, but could still violate precedence constraints. In the direct formulation and the time-indexed formulation, an integer feasible solution will also respect precedence constraints. As for VRPTW, we may still use time window branching to get integral solutions. In the following, we use the relaxed formulation as a basis for introducing time window branching, but it transfers easily to the other models.

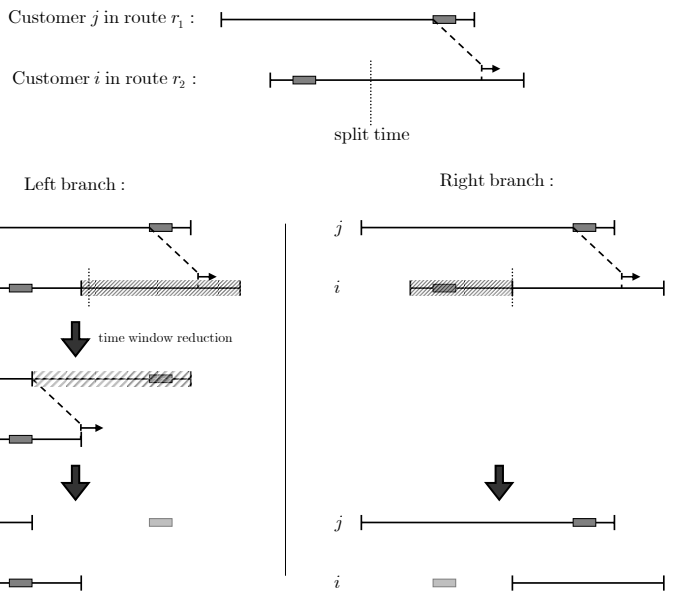


Figure B.3: *Branching to avoid a violation of a precedence constraint.*

Figure B.3 shows a violation of the precedence constraint between customers j and i, in routes r_1 and r_2. By branching on the time window of customer i and using the time window reduction rule of Section B.4.1 for (j, i), r_1 and r_2 are prohibited in the left and right branch, respectively. Note that there is no overlap between the time window of customer i in the left branch and the corresponding time window in the right branch.

Later, we describe a strategy to wisely select the point in time, t^s, where the time window is split. If $s_i + 1 \leq t^s \leq s_j + \delta_{ji}$, the current solution will be excluded from the solution space by using the reduction rule for (j, i). The modified time windows of customer i become: $[\alpha_i, t^s - 1]$ in the left branch and $[t^s, \beta_i]$ in the right branch.

The tightest formulation is reached if time windows are reduced as much as possible. Therefore in both branches, we run through all relevant precedence constraints with the new time window of customer i and reduce time windows where possible. This may also reduce the time windows of other customers than i and j, and this process is repeated iteratively, until no further reduction is possible.

An interesting result is that this branching strategy is as strong as the one formerly proposed specifically for synchronization, by e.g. Ioachim et al. (1999). In the less general context, the time windows of two synchronized customers are, naturally, always identical. Branching is done on the two time windows simultaneously, so they always stay identical. Synchronization modeled by two generalized precedence constraints, also has this property when time window reductions are applied. Assume that we have a synchronization constraint between customers i and j, i.e. $\delta_{ij} = 0$ and $\delta_{ji} = 0$ and hence $\alpha_j = \alpha_i \wedge \beta_j = \beta_i$. For a given split time t^s for customer i, the time windows become:

	Customer i	Customer j
Old time windows	$[\alpha_i, \beta_i]$	$[\alpha_i, \beta_i]$
TW (left branch)	$[\alpha_i, t^s - 1]$	$[\alpha_i, \min\{\beta_i, t^s - 1 - \delta_{ji}\}] = [\alpha_i, t^s - 1]$
TW (right branch)	$[t^s, \beta_i]$	$[\max\{\alpha_i, t^s + \delta_{ij}\}, \beta_i] = [t^s, \beta_i]$

This is illustrated in Figure B.4. The time windows of i and j are identical in each of the branches after applying time window reduction.

Usually, there are several branching candidates to choose from and we need a strategy to choose one of these. Gélinas et al. (1995) elaborate further on this subject. When using strong branching (see e.g. Achterberg et al. (2005a)) a few candidates are chosen for further probing. In any case, we need to specify a

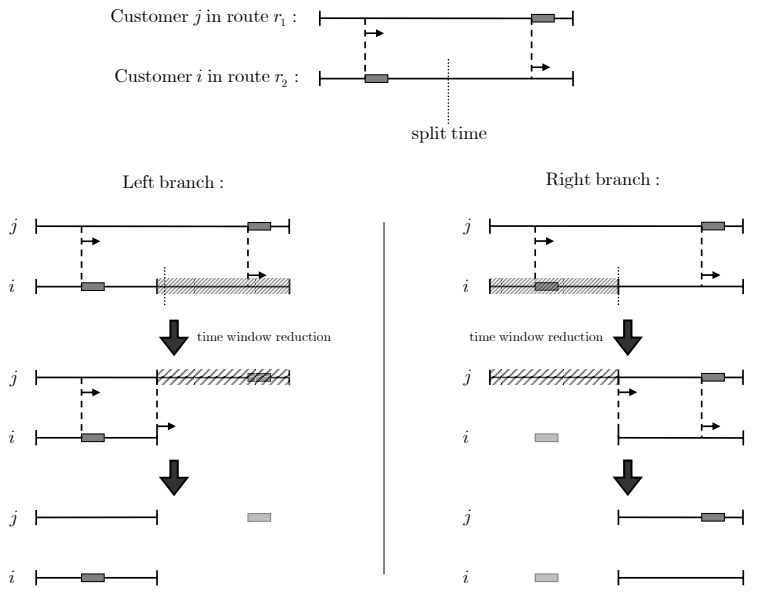


Figure B.4: *Branching on a generalized precedence constraint of a synchronization constraint.*

priority ordering of candidates. First, we need to find the potential branching candidates. In theory, we could branch on any time window and split it at an arbitrary position. In practice, however, we limit this choice. We do not want to consider candidates where the branching is without effect in one of the branches, i.e. where one of the branches does not prohibit any columns of the current solution. Also, many of the remaining candidates have an identical effect on the current solution. They may still have a different effect on new columns, but it is very hard to predict this impact. Figure B.5 (a) depicts some of the potential branching candidates in the time window of customer i. Customer i is a part of two routes that have been included in the solution with fractional values and hence it appears at multiple positions within its own time window. In this example we assume that the routes r_2 and r_3 are both in the solution with a value of 0.5 and r_1 and r_4 with a value of 1. The effect on the current solution of each candidate is shown in Table B.2. Candidates 1 and 6 are examples of ineffective candidates. Candidates 2 and 3 have an identical effect on the current solution.

In our approach, when choosing between candidates with an identical immediate effect, it is optimal to select the candidate which splits at the latest possible time. For customer i, that split time coincides with either s_i^r or with $s_j^r + \delta_{ji}$ for a

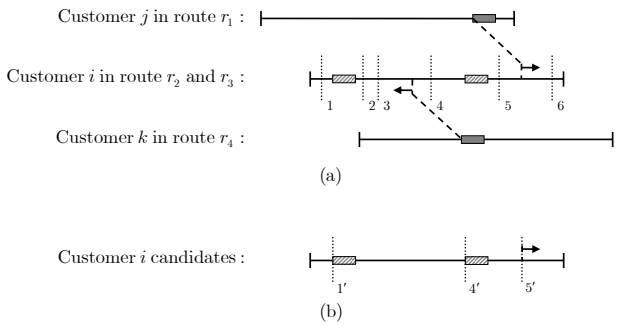


Figure B.5: *Some potential branching candidates in the time window of customer i.*

	Infeasible routes		Sum of excluded variables		
	Left branch	Right branch	Left branch	Right branch	Preference
Candidate 1	r_1, r_2, r_3		2	0	0
Candidate 2	r_1, r_3	r_2	1.5	0.5	0.5
Candidate 3	r_1, r_3	r_2	1.5	0.5	0.5
Candidate 4	r_1, r_3	r_2, r_4	1.5	1.5	1.5
Candidate 5	r_1	r_2, r_3, r_4	1	2	1
Candidate 6		r_2, r_3, r_4	0	2	0

Table B.2: *Effect of the branching candidates of Figure B.5.*

route r, which is in the current basis of the master problem.

In Figure B.5 (a), we prefer candidate 3 to candidate 2 as there is less chance that r_2 can be adapted to the new time window of the right branch. We prefer candidate 4 to candidate 3 as it excludes the same or more in both branches. Figure B.5 (b) shows the three candidates that we would actually consider for the time window of customer i. The candidates $1'$, $4'$, and $5'$ get the same values as 1, 4, and 5, respectively, in Table B.2. Remember that these are just the candidates of customer i. There will be similar candidates for each of the other customers. We find the candidates for customer i by running through all routes that are included in the solution with a positive value. If customer i is in the route, the start time of the customer is a candidate ($t^s = s_i$). If the route includes a customer j, where $(j, i) \in \Delta$ the route contributes with a candidate for customer i with split time $t^s = s_j + \delta_{ji}$.

The preference of late split times is due to the following algorithmic considerations: The label setting algorithm, which is used to generate routes, schedules visits at the earliest possible time in any route. Hence, it is impossible to sched-

ule the visit earlier without rerouting. In the left branch, the service time of a customer will be forced to decrease by at least one time unit and hence rerouting is required. In the right branch, the current conflict is resolved, as the start of the new time window is equal to s_i^r or $s_j^r + \delta_{ji}$ which generated the branching candidate in the first place.

In this paper, the problems are solved to optimality, which means that every node in the branch-and-bound tree must be either explored or pruned. Hence, we aim for a small, but at the same time, balanced tree. To achieve this, we rank the branching candidates according to the corresponding sums of excluded variables in the two branches. A candidate gets the value of the minimum of the two sums, and hence only the worst of the branches counts. A large value of a candidate is equal to a high preference. Hence, we prefer branching candidates which exclude as much as possible in the least effective branch. In the example, candidate 4 is preferred, as it excludes 1.5 routes in each branch, giving it a preference ranking of 1.5.

If the aim is to get high-quality solutions, but not necessarily optimal solutions, in a short time, it may be better to choose branching candidates where one of the branches is more promising than the other. This may then be utilized in a heuristic search of the branch-and-bound tree. This idea has been used in several other contexts, see e.g. Ryan (1992a).

B.5 Benchmark instances

A set of benchmark instances has been used in the following quantitative analysis of the problem and in a comparison of the different models. The instances are extensions of the 56 well known VRPTW-instances of Solomon (1987a). Solomon's VRPTW-instances have been used extensively in existing literature and new solution algorithms for VRPTW are often tested on these to indicate algorithm performance. The data sets are well suited for the tests, as they represent a wide range of problems with varying structure. The locations of customers are in some instances uniformly distributed over whole area. In others, customers are located in clusters. The time windows of customers are also varied to test both very tight and very loose time window constraints. The data sets consist of a number of customers with a geographical location, a time window, service time, and a demand along with the number of available vehicles, their capacity, and the scheduling horizon. The instances are publicly available. We take the original instances and introduce temporal dependencies of various types to these instances. We have chosen to look at the instances with 25 and 50 customers, as these are small enough to allow quick solution of the basic

problem. Some of them still prove hard to solve as temporal dependencies are introduced. A thorough analysis is carried out on the instances with 25 customers, as most of these can be solved within one hour. Tests on instances with 50 customers are also included, to assess how the findings for 25 customers scale to larger instances.

Five sets of instances were made: one for each of the five temporal dependencies of Figure B.1, except *maximum difference*, and one set with a random combination of the other four (Random Combination). The reason for omitting *maximum difference* is its similarity to *minimum difference*. When we generate instances randomly, the two types are actually identical. For all instances, a list of temporal dependencies is created for each of the five types. All random values are drawn from uniform distributions, and the list is generated randomly in the following way:

1. Determine the type of the next temporal dependency to be added (For Random Combination this choice is random, and for all other types it is fixed).
2. Choose, at random, two visits, i and j, which are not already directly or indirectly interdependent. Visits are indirectly interdependent if there is a chain of dependencies from one to the other, e.g. if they both have a dependency on a certain visit, but not directly on each other. We require independency between the visits to avoid infeasible cycles of dependencies.
3. Check if it is possible to impose a temporal dependency between i and j. If not possible, go to 2. If it is impossible to add more temporal dependencies of this type, go to 1. If it is impossible to add more temporal dependencies altogether, exit.
4. Draw random values for the temporal dependency. $\text{diff}_{\min}$ and $\text{diff}_{\max}$ are random numbers drawn from all values that do not make the problem infeasible and that impose a constraint which is more strict than that already given by the time windows. For Synchronization and Overlap all values are fixed.
5. Set values of δ_{ij} and δ_{ji} according to Table B.1.
6. Reduce time windows as explained in Section B.4.1. This is necessary to ensure that all instances are feasible.
7. Go to 1.

As we do not allow cycles of dependencies, it is not possible to add more than $n - 1$ temporal dependencies, where n is the number of customers. In some

cases, the number may be smaller than this. E.g. for the very strict synchronization constraint, it is obviously not possible to impose $n-1$ synchronizations, corresponding to a synchronization of all visits, if some of the visits have non-overlapping time windows. This results in fewer test instances for some instances sets.

The addition of temporal dependencies to the instance set leads to a very large number of new test instances. For every one of the original instances we have five sets of dependencies and for each of these sets we can choose to use from 0 to $n-1$ dependencies. Hence for instances with 25 customers, we are able to run 125 tests for each of the original instances (except for a few cases, where it was not possible to generate $n-1$ dependencies). This totals to 7000 tests and gives a good statistical foundation for the test results presented in the next section. The data files containing the parameters for the randomly generated temporal dependencies are available from the authors on request. We have also generated similar files for instances of size 50 and 100.

B.6 Test results

The intention of this section is to give a general overview of the complexity of vehicle routing problems with temporal dependencies. The tests are summarized in graphs that capture the trends we see in the tests overall.

The algorithms are implemented in the branch-and-cut-and-price framework of COIN-OR (Lougee-Heimer, 2003; Coin, 2006). The tests have been run on 2.2 GHz AMD processors with 2 GB RAM. Based on preliminary tests, the algorithm is set to do strong branching with three candidates and add up to five variables with negative reduced cost per iteration. For as long as possible, columns are generated by a heuristic version of the label setting algorithm similar to the one proposed by Chabrier (2006).

The direct formulation is expected to lead to highly fractional solutions, as generalized precedence constraints can be respected by linearly combining routes, where a particular customer has varying start times. Furthermore, the subproblem is a resource constrained shortest path problem with linear node costs, which is significantly harder to solve, than the other subproblems presented. To our knowledge, no exact solution methods for this problem exist in the literature. The method of Ioachim et al. (1998) could probably be adapted to the cyclic case, but the efficiency is questionable. Therefore, the computational experiments of this paper do not include the direct formulation.

	Instances in total	Time-indexed formulation (all cuts) Solved in the root	Time-indexed formulation (all cuts) Solved before timeout	Time-indexed formulation (limited) Solved in the root	Time-indexed formulation (limited) Solved before timeout	Relaxed formulation Solved in the root	Relaxed formulation Solved before timeout
Synchronization	1148	483	1027	448	1141	138	1143
Overlap	1324	351	1058	322	1207	127	1240
Minimum difference	1400	741	1350	703	1377	226	1381
Min+max difference	1400	531	1226	465	1361	105	1384
Random mix	1400	506	1271	459	1382	155	1383

Table B.3: *Overview of the test results for instances with 25 customers. Time out is one hour.*

To give an idea of the overall performance of the remaining three approaches, the test results are summarized in Table B.3. As described in Section B.5, five sets of instances were generated, and the table shows a clear tendency for all five types. The full time-indexed formulation solves the largest number of instances in the root node, as expected. The instances solved in the root node by the relaxed formulation are a subset of those solved in the root node by the limited time-indexed model which again is a subset of those of the full time-indexed formulation. The numbers in the table illustrate this relationship. When looking at the number of instances solved before timeout, the tendency is reversed. The relaxed formulation is capable of solving the largest number of instances for all five types. However, the performance of the limited time-indexed model is in all cases almost as good as that of the relaxed model. Interestingly, the Overlap instances seem harder to solve than the other types.

Table B.4 summarizes the results for the 50 customer instances. Incrementing the number of temporal dependencies with one between each test, would result in 14000 tests, as the temporal dependency generation scheme can generate 49 dependencies for each of the 5 types for all 56 original instances. To limit the extend of the test, we have chosen to test only for 5, 15, 25, 35, and 45 temporal dependencies. This limits the maximum number of tests to 1400 in total. As can be observed from Table B.4, the results are similar to those of Table B.3. In all cases, a smaller ratio of the instances can be solved within an hour and a significant drop, in the number of instances solved in the root, is observed, compared to the instances with 25 customers. Interestingly, the limited time-indexed model now performs slightly better than the relaxed model.

In the remainder of this section, we have chosen to focus on two of the 25 cus-

	Instances in total	Time-indexed formulation (all cuts) Solved in the root	Time-indexed formulation (all cuts) Solved before timeout	Time-indexed formulation (limited) Solved in the root	Time-indexed formulation (limited) Solved before timeout	Relaxed formulation Solved in the root	Relaxed formulation Solved before timeout
Synchronization	242	56	158	45	201	5	196
Overlap	276	24	121	23	156	2	150
Minimum difference	280	66	196	58	205	7	202
Min+max difference	280	50	108	44	135	3	134
Random mix	280	33	140	28	189	1	183

Table B.4: *Overview of the test results for instances with 50 customers. Time out is one hour.*

tomer instance sets, namely instances with only synchronization relations and a set with a random mix of the five temporal dependencies of Table B.1. These two sets have been chosen as the first represents a large group of practical applications and the latter does not hold any particular structure. The statements made in the following are in full accordance with the other instance sets.

The root node lower bound sometimes coincides with the value of the optimal solution. In such cases, we often find the optimal solution at the root node. As this results in low computation times, it is interesting to see how often it happens.

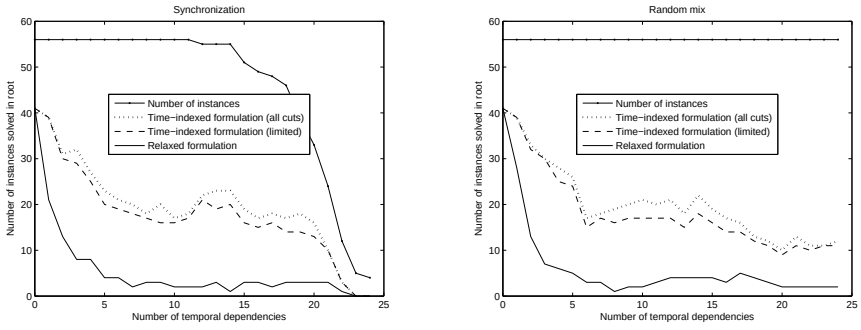


Figure B.6: *Number of instances solved at the root node of the branch-and-bound tree.*

In Figure B.6 the total number of instances solved at the root node is given,

summarized over all 56 instances. We clearly observe the strength of the time-indexed formulation. There is a significant increase in the number of instances solved at the root node compared to the relaxed formulation. Interestingly, there is not much difference from the full formulation to the limited version. In the relaxed formulation, if a problem can be solved at the root node, it means that all temporal dependencies were respected by chance, and hence they would not have been very constraining. Figure B.7 gives the number of nodes in the branch-and-bound tree (the mean over all instances) and the conclusions are the same as for Figure B.6. As we would expect for the relaxed formulation, we see that the number of nodes increases with the number of temporal dependencies. This does not seem to be the case for the time-indexed formulation.

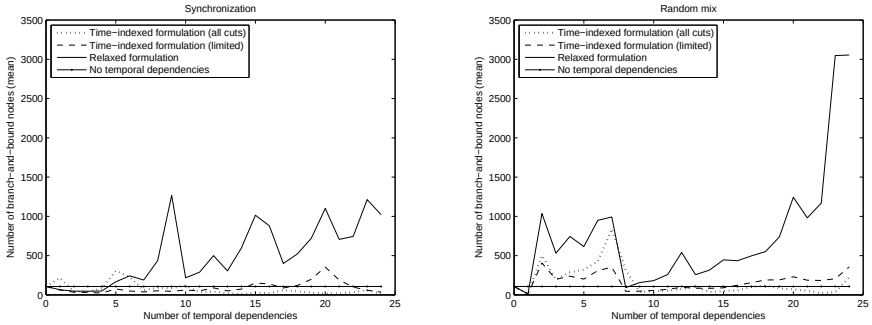


Figure B.7: *Number of nodes in the branch-and-bound tree.*

Another interesting aspect is the solution time. We examine the solution time for each of the instances individually and we also consider the general trend. The variation on solution time is large between the instances. Hence, an average of these values would emphasize the harder instances. We want all instances to count equally and therefore, we normalize the values by comparing each computation time to the solution time for the same problem without temporal dependencies. The mean over all instances is shown in Figure B.8.

Looking at Figure B.8, it is clear that the time-indexed formulation is worse than the other two with respect to solution time. Closer inspection shows that the full time-indexed formulation has a few instances where computation time is excessive and this has a major impact on the mean value.

In connection with solution time, it is also interesting to make a direct comparison between the approaches for each instance. For each number of temporal dependencies, we count the number of instances where the limited time-indexed approach is faster than the relaxed formulation and vice versa. The results are summarized in Figure B.9. Looking at the instances individually, the limited

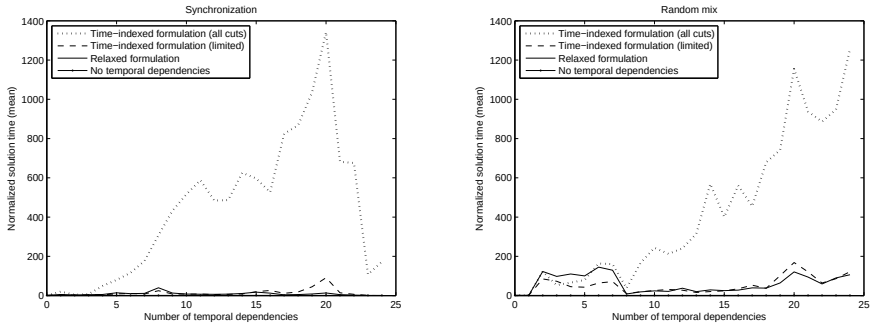


Figure B.8: *Normalized solution time (mean).*

time-indexed approach seems a little better.

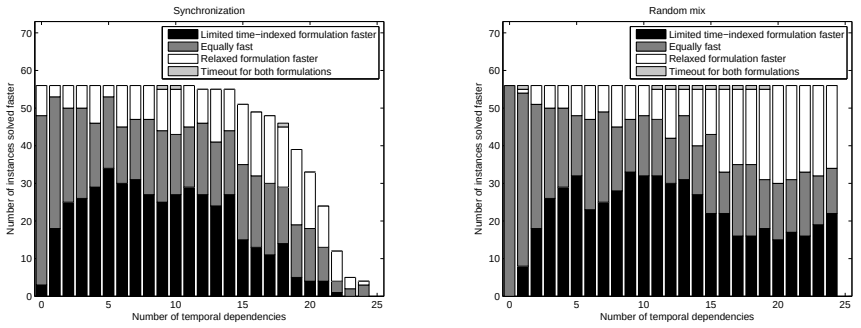


Figure B.9: *Number of tests where one of the approaches is faster than the other. The two approaches are considered equally fast if they are within 20% of each other.*

Finally, we look at the distribution of time spent in the algorithm. This is illustrated in Figure B.10. The solution procedure in each node of the branch-and-bound tree consists of three parts, namely cut generation, variable (column) generation, and solution of the LP master problem. The times of Figure B.10 sum the time spent in all nodes of the tree. The branching time reported is the time used to select branching candidates. As strong branching is applied, this selection also involves the solution of a limited number of LP problems. There may be an overhead on time from memory management, primarily. Therefore, the four components illustrated of the figure do not sum to exactly 100%.

From Figure B.10, we observe that for the time-indexed formulation, the portion of time spent in the LP-solver increases as problems with more temporal depen-

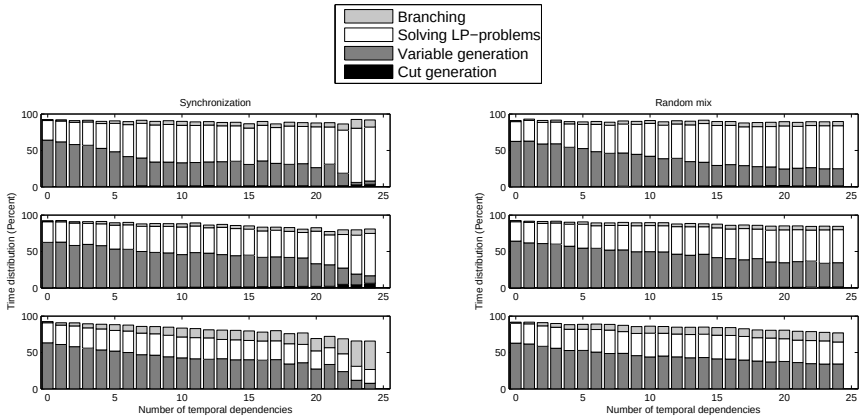


Figure B.10: *Distribution of solution time for the time-indexed model (top), the limited time-indexed model (middle), and the relaxed model (bottom).*

dencies are considered. This is due to the fact that more cuts are added and hence the size of the LP-model increases. For the relaxed formulation, the tendency is, not surprisingly, that more time is spent branching when the number of temporal dependencies increases. The share of time spent by the LP-solver is, in this case, stable.

On the basis of the tests, we are able to conclude that the temporal dependencies introduce additional complexity to the problem, as expected. The time-indexed formulation has the worst immediate performance, but may be more useful for large instances with harder pricing problems. The performance of the limited time-indexed approach and the relaxed formulation is comparable. A few instances of each type turn out to be very hard to solve, no matter what method is used. The time-indexed formulation does have a number of nice features that could be utilized in future development. It has tighter bounds, both theoretically and in the practical instances that we have examined. The tighter bounds mean that more instances are solved at the root node of the branch-and-bound tree, and in these cases this formulation gives better results. Also, for instances where the solution is not found at the root node, the branch-and-bound tree is still significantly smaller than the corresponding tree for the relaxed formulation. The number of variables that has to be generated is also generally smaller for the time-indexed formulation. For most realistic problems, variable generation is the dominating factor of the overall solution time, and in these cases the time-indexed formulation may be the better choice, as the LP solution time becomes less important.

B.7 Conclusions and future work

The vehicle routing problem with time windows and temporal dependencies has been introduced. The problem has previously been treated in various practical contexts in different forms, but this is the first generic analysis presented in the literature. Four different models were presented and ranked according to their theoretical strength. The time-indexed model has the tightest formulation and hence gives the best bounds, but the number of constraints is too large for them to be included explicitly. Instead, the model was implemented in a branch-and-cut-and-price framework, where both constraints and variables are generated dynamically. As this approach is novel, it was described how to efficiently identify violated cuts and the necessary adjustments in the pricing problem were introduced. The branching scheme was presented next. The scheme is based on the traditional time window branching, where the scheme is also used to restore feasibility with respect to temporal dependencies. The branching scheme is as strong as and more general than the previously presented branching scheme for routing with synchronization. Finally, benchmark instances were introduced and a quantitative analysis was carried out.

The analysis showed that, even though the time-indexed model has some nice properties, it also retains its major drawback, namely the number of constraints. As a consequence, a hybrid method was implemented, where only a limited number of the violated cuts are added. This approach kept most of the nice features of the time-indexed model, while at the same time lowering the solution time to the same level as that of the relaxed model.

The model presented in this paper is general and is therefore applicable to various practical problems. Future work could be on an adaption to real world problems. Another very interesting direction for future research is to include additional cuts. Using the time-indexed formulation, we were able to solve many instances at the root node of the branch-and-bound tree, and this number could be increased by introducing additional cuts. From e.g. Desaulniers et al. (2008) it is clear that the number of nodes can be limited severely by including cuts, especially for large instances. In many cases, the problems are solved in the root node. The performance of the time-indexed model was clearly better than the relaxed model for the instances, where the optimal solution was obtained at the root node.

Acknowledgements

The authors thank the anonymous referees for constructive comments on an earlier version of this paper. The first author was supported by the EliteForsk-grant of the Danish Ministry of Science, Technology and Innovation.

References

Achterberg, T., T. Koch, and A. Martin (2005a). "Branching Rules Revisited". *Operations Research Letters* 33.1, pp. 42–54.

Akker, J. van den, J. Hoogeveen, and J. van Kempen (2006). "Parallel machine scheduling through column generation: Minimax objective functions (extended abstract)". *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 4168 LNCS, pp. 648–659.

Bélanger, N., G. Desaulniers, F. Soumis, and J. Desrosiers (2006). "Periodic airline fleet assignment with time windows, spacing constraints, and time dependent revenues". *European Journal of Operational Research* 175.3, pp. 1754–1766.

Bigras, L.-P., M. Gamache, and G. Savard (2008). "Time-Indexed Formulations and the Total Weighted Tardiness Problem." *INFORMS Journal on Computing* 20.1, p. 133.

Bredström, D. and M. Rönnqvist (Feb. 2007). *A branch and price algorithm for the combined vehicle routing and scheduling problem with synchronization constraints*. Discussion Papers 2007/7. Department of Finance, Management Science, Norwegian School of Economics, and Business Administration.

Bredström, D. and M. Rönnqvist (2008). "Combined vehicle routing and scheduling with temporal precedence and synchronization constraints". *European Journal of Operational Research* 191.1, pp. 19–31.

Chabrier, A. (2006). "Vehicle Routing Problem with elementary shortest path based column generation". *Computers and Operations Research* 33.10, pp. 2972–2990.

Christiansen, M. and B. Nygreen (2005). "Robust Inventory Ship Routing by Column Generation". In: *Column Generation*. Ed. by G. Desaulniers, J. Desrosiers, and M. Solomon. New York: Springer. Chap. 7, pp. 197–224.

Coin (2006). *COmputational INfrastructure for Operations Research (COIN-OR)*. http://www.coin-or.org/.

Cook, W. and J. L. Rich (2001). *A Parallel Cutting-Plane Algorithm for the Vehicle Routing Problem With Time Windows*. Tech. rep. Rice University.

Danna, E. and C. L. Pape (2005). "Branch-and-Price Heuristics: A Case Study on the Vehicle Routing Problem with Time Windows". In: *Column Generation*. Ed. by J. D. Guy Desaulniers and M. M. Solomon. Springer. Chap. 4, pp. 99–129.

Dantzig, G. B. and P. Wolfe (1960a). "Decomposition Principle for Linear Programs". *Operations Research* 8.1, pp. 101–111.

Desaulniers, G., F. Lessard, and A. Hadjar (2008). "Tabu Search, Partial Elementarity, and Generalized k-Path Inequalities for the Vehicle Routing Problem with Time Windows". *Transportation Science* 42.3, p. 387.

Desrochers, M., J. Desrosiers, and M. Solomon (1992a). "A new optimization algorithm for the vehicle routing problem with time windows". *Operations Research* 40.2, pp. 342–354.

Doerner, K. F., M. Gronalt, R. F. Hartl, G. Kiechle, and M. Reimann (2008). "Exact and heuristic algorithms for the vehicle routing problem with multiple interdependent time windows". *Computers and Operations Research* 35.9, pp. 3034–3048.

Dohn, A., M. S. Rasmussen, T. Justesen, and J. Larsen (2008c). "The Home Care Crew Scheduling Problem". In: *ICAOR'08 - Proceedings, 1st International Conference on Applied Operational Research*. Ed. by K. Sheibani. Tadbir Institute for Operational Research, pp. 1–8.

Dohn, A., E. Kolind, and J. Clausen (2009b). "The manpower allocation problem with time windows and job-teaming constraints: A branch-and-price approach". *Computers and Operations Research* 36.4, pp. 1145–1157.

Dror, M. (1994a). "Note on the Complexity of the Shortest Path Models for Column Generation in VRPTW". *Operations Research* 42.5, pp. 977–978.

Feillet, D., P. Dejax, M. Gendreau, and C. Gueguen (2004). "An exact algorithm for the elementary shortest path problem with resource constraints: application to some vehicle routing problems". *Networks* 44.3, pp. 216–29.

Fügenschuh, A. (2006). "The vehicle routing problem with coupled time windows". *Central European Journal of Operations Research* 14.2, pp. 157–176.

Gélinas, S., M. Desrochers, J. Desrosiers, and M. Solomon (1995). "A new branching strategy for time constrained routing problems with application to backhauling". *Annals of Operations Research* 61, pp. 91–109.

Ioachim, I., S. Gelinas, F. Soumis, and J. Desrosiers (1998). "A dynamic programming algorithm for the shortest path problem with time windows and linear node costs". *Networks* 31.3, pp. 193–204.

Ioachim, I., J. Desrosiers, F. Soumis, and N. Bélanger (1999). "Fleet assignment and routing with schedule synchronization constraints". *European Journal of Operational Research* 119.1, pp. 75–90.

Jepsen, M., B. Petersen, S. Spoorendonk, and D. Pisinger (2008). "Subset-Row Inequalities Applied to the Vehicle-Routing Problem with Time Windows". *Operations Research* 56.2, pp. 497–511.

Justesen, T. and M. S. Rasmussen (2008). "The Home Care Crew Scheduling Problem". Master's thesis. Department of Informatics, Mathematical Mod-

elling, Technical University of Denmark, and Department of Computer Science, University of Copenhagen.

Kallehauge, B., J. Larsen, O. B. Madsen, and M. Solomon (2005). "The Vehicle Routing Problem with Time Windows". English. In: *Column Generation*. Ed. by G. Desaulniers, J. Desrosiers, and M. M. Solomon. GERAD 25th anniversary series. New York: Springer. Chap. 3, pp. 67–98.

Kilby, P., P. Prosser, and P. Shaw (2000). "A Comparison of Traditional and Constraint-based Heuristic Methods on Vehicle Routing Problems with Side Constraints". *Constraints* 5.4, pp. 389–414.

Kohl, N., J. Desrosiers, O. B. G. Madsen, M. M. Solomon, and F. Soumis (1999). "2-Path Cuts for the Vehicle Routing Problem with Time Windows". *Transportation Science* 33.1, pp. 101–116.

Lesaint, D., N. Azarmi, R. Laithwaite, and P. Walker (1998). "Engineering Dynamic Scheduler for Work Manager". *BT Technology Journal* 16.3, pp. 16–29.

Li, Y., A. Lim, and B. Rodrigues (2005). "Manpower allocation with time windows and job-teaming constraints". *Naval Research Logistics* 52.4, pp. 302–311.

Lim, A., B. Rodrigues, and L. Song (2004). "Manpower allocation with time windows". *Journal of the Operational Research Society* 55.11, pp. 1178–1186.

Lougee-Heimer, R. (2003). "The Common Optimization INterface for Operations Research: Promoting Open-Source Software in the Operations Research Community". *IBM Journal of Research and Development* 47.1, pp. 57–66.

Lysgaard, J., A. N. Letchford, and R. W. Eglese (2004). "A New Branch-and-Cut Algorithm for the Capacitated Vehicle Routing Problem". *Mathematical Programming* 100.2, pp. 423–445.

Oron, D., S.-N. Sze, and A. S.-F. Ng (2008). "A Heuristic Manpower Scheduling for In-Flight Catering Service". In: The 13th International Conference of Hong Kong Society for Transportation Studies.

Rousseau, L.-M., M. Gendreau, and G. Pesant (2003). *The Synchronized Vehicle Dispatching Problem*. Tech. rep. CRT-2003-11. Conference paper, Odysseus 2003. Centre de Recherche sur les Transports, Université de Montréal, Canada.

Ryan, D. M. (1992a). "The Solution of Massive Generalized Set Partitioning Problems in Aircrew Rostering". *Journal of the Operational Research Society* 43.5, pp. 459–467.

Savelsbergh, M. (1985). "Local search in routing problems with time windows". *Annals of Operations Research* 4.1-4, pp. 285–305.

Solomon, M. M. (1987a). "Algorithms for the Vehicle Routing and Scheduling Problems with Time Window Constraints". *Operations Research* 35.2, pp. 254–265.

van den Akker, J. M., C. A. J. Hurkens, and M. W. P. Savelsbergh (2000). "Time-indexed formulations for machine scheduling problems: column generation". *INFORMS Journal on Computing* 12.2, pp. 111–124.

Appendix C

Subsequence Generation for the Airline Crew Pairing Problem

Matias Sevel Rasmussen, Richard M. Lusby, David M. Ryan, and Jesper Larsen

Submitted to: *Transportation Research Part E* (2011).

Subsequence Generation for the Airline Crew Pairing Problem*

Matias Sevel Rasmussen[1], Richard M. Lusby[1], David M. Ryan[2], and Jesper Larsen[1]

Good and fast solutions to the airline crew pairing problem are highly interesting for the airline industry, as crew costs are the biggest expenditure after fuel for an airline. The crew pairing problem is typically modelled as a set partitioning problem and solved by column generation. However, the extremely large number of possible columns naturally has an impact on the solution time.

In the solution method of this work we severely limit the number of allowed subsequent flights, i.e. the subsequences, thereby significantly decreasing the number of possible columns. Set partitioning problems with limited subsequence counts are known to be easier to solve, resulting in a decrease in solution time.

The problem though, is that a small number of deep subsequences might be needed for an optimal or near-optimal solution and these might not have been included by the subsequence limitation. Therefore, we try to identify or generate such subsequences that potentially can improve the solution value.

We benchmark the subsequence generation approach against a classical column generation approach on real-life test instances. We consider the LP relaxation and compare the quality and the integrality of the solutions. The LP solutions from the subsequence generation approach are less fractional, but it comes at the cost of a worse solution quality.

The approach in the present paper is novel. To our knowledge generation of subsequences have not been described and tested previously in the literature.

Keywords: Airline crew pairing, Subsequence generation, Column generation, Limited subsequence, Crew scheduling, Real-life application, Set partitioning, LP relaxation

*Submitted to: *Transportation Research Part E* (2011).

[1]Department of Management Engineering, Technical University of Denmark, Produktionstorvet, Building 424, DK-2800 Kgs. Lyngby, Denmark.

[2]Department of Engineering Science, The University of Auckland, New Zealand.

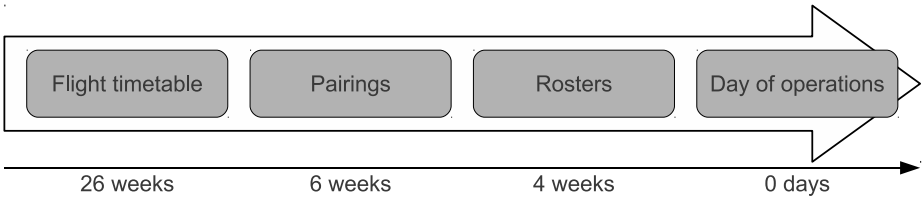


Figure C.1: *The airline crew scheduling process. The times are for Air New Zealand's domestic scheduling.*

C.1 Introduction

Crew costs are the second largest expense for an airline company. Only fuel costs are higher, see Gopalakrishnan and Johnson (2005). Here it is also reported that, for instance, American Airlines spent USD 1.3 billion on crew in 1991. The expenditures for an airline can roughly be divided equally between three areas: Fuel, crew, and other costs (buildings, maintenance, administrative staff, etc.). As fuel costs cannot be controlled by an airline, crew costs are probably the most important area for potential savings. Therefore, airline crew scheduling has received a lot of attention in the literature, and consequently, optimisation is heavily used by the airlines. With such a large amount of money being spent on crew, even small improvements in how they are scheduled can result in significant savings.

The airline crew pairing problem which is dealt with in this work is a part of a larger series of optimisation problems. The first step is *flight timetabling*. In this step a schedule of all the flights that the airline will fly is constructed. The next steps are *fleet assignment*, where aircraft types are allocated to the flights, and *aircraft routing*, where the aircraft routes are laid. These steps, however, do not directly influence the crew scheduling. The crew pairing step (which is the focus of this paper) finds sequences of flights that can be flown in a feasible way at a minimum cost. These sequences of flights are called *pairings* and are anonymous, that is they are not associated with a specific crew member. The crew pairing problem can be solved separately for cockpit crew and cabin crew, and it can also be solved separately per aircraft type qualification. The last step is *crew rostering* where pairings are combined to form actual rosters for individual crew members. The crew pairing and the crew rostering steps are together called *airline crew scheduling*, see Figure C.1.

This paper presents a novel *subsequence generation* approach to solving the crew pairing problem. The subsequence generation approach is to our knowledge not

found elsewhere in the literature. We consider the linear programming (LP) relaxation of a set partitioning formulation of the problem. The idea is to generate subsequences of flights that appears in the optimal pairings instead of—as in classic column generation—to generate the actual pairings. Whenever a subsequence is found, i.e. generated, a whole set of pairings containing that subsequence is enumerated and added to the LP relaxation. The devised solution algorithm is tested on real-life data instances and benchmarked against classic column generation.

In Gopalakrishnan and Johnson (2005) a recent survey of airline crew scheduling can be found. The authors describe the different approaches that have been used over the last two decades, and point out promising directions for future work in the area. The crew pairing problem is treated separately and in detail. Barnhart et al. (2003) give a text book description of airline crew scheduling and also have a detailed section on crew pairing with examples. They formulate the crew pairing problem as a set partitioning problem and describe how the problem can be solved as a weekly problem or a dated problem. The weekly problem approach exploits repetitive patterns of flights over the weekdays, and is thus able to break the problem into smaller parts, which are then combined. This division of the problem is of course a trade-off against optimality. The dated problem approach on the other hand solves the problem directly, and is necessary for flight timetables where flights are not repeated several times a week. The complex cost structures for pairings are described by Gopalakrishnan and Johnson (2005) and Barnhart et al. (2003). Andersson et al. (1998) describe different approaches to crew pairing and give a detailed introduction to the Carmen (now Jeppesen) system for solving the crew pairing problem. The Carmen system uses a priori column generation; however, it has separated the checking of the pairing requirements into a special rules language. The Carmen system uses the algorithm described by Wedelin (1995). Desaulniers et al. (1998) present the crew pairing model as a special case of a generic air crew scheduling model, that also covers, for instance, rostering. They solve the crew pairing problem with column generation. AhmadBeygi et al. (2009) develop an integer programming model for generating pairings. The model can be used especially in research to overcome the time-consuming task of implementing a pairing generator. Butchers et al. (2001) describe airline optimisation problems in general and the crew pairing problem in particular for Air New Zealand's domestic and international schedule. Also here the crew pairing problem is formulated as a set partitioning problem. Lavoie et al. (1988) use a set covering formulation and perform column generation on a duty period network. A duty period is a sequence of flights that corresponds to a day's work, see more in Section C.2. Graves et al. (1993) use a set partitioning formulation and do column generation on a network of flights. Vance et al. (1997) use a two-stage approach. First flights are combined to form duty periods, and next duty periods are combined to form pairings. Using dynamic constraint aggregation crew scheduling can be

solved in an integrated approach, see Saddoune et al. (2011). In this way all constraints are virtually present in the master problem, but in an aggregated form, where basically constraints belonging to the same pairing are just represented by one active constraint. The update of these active constraints leads to a complex setup in the interplay with the column generator. This, though, does at present remain a very complex and time-consuming approach limited to academic environments only.

The remainder of this paper is organised as follows. In Section C.2, we present a formal definition of the airline crew pairing problem. In Section C.3, we introduce the concept of subsequence limitation and the motivation behind it. In Section C.4, we develop the suggested subsequence generation solution algorithm. In Section C.5, we present real-life test instances, and we show benchmark results from the comparison between the subsequence generation approach and a classical column generation approach. Finally, in Section C.6, we conclude on the work and point out directions for future research.

C.2 Problem formulation

Let $\mathcal{F}$ denote the set of flights in the flight schedule for an airline. A *duty period* is a sequence of flights from $\mathcal{F}$ which can be flown by an anonymous crew member. A duty period must comply with several rules and regulations in order to be feasible. A crew member can either be *operating* or *passengering* (sometimes called *deadheading*) on a flight. Passengering allows crew members to be repositioned in order to operate other flights. A duty period consists of *flying time*, where the crew member is operating the flight, and *idle time*, which together give the *elapsed time*. Each duty period has a maximum flying time and a maximum elapsed time, as well as a maximum number of flights that can be operated. Duty periods must also respect meal break regulations. Duty periods are separated by *rest periods*, which must have a minimum length. Starting and ending a duty period impose a *sign-on* and *sign-off* time, respectively.

A *pairing* (sometimes called a *tour-of-duty*) is a sequence of duty periods and rest periods. Every airline has a set of *crew bases*, i.e. airports from where crew can start working. A feasible pairing must start and end at the same crew base. Pairings can only contain up to a maximum number of duties, and a pairing is only allowed to stretch over a certain number of *mandays*. The manday count is increased every time midnight is passed in the time zone where the pairing originates. Different airlines use different and quite complex ways of calculating the cost of a pairing, for examples of this see Gopalakrishnan and Johnson (2005) and Barnhart et al. (2003). For the research carried out in the present

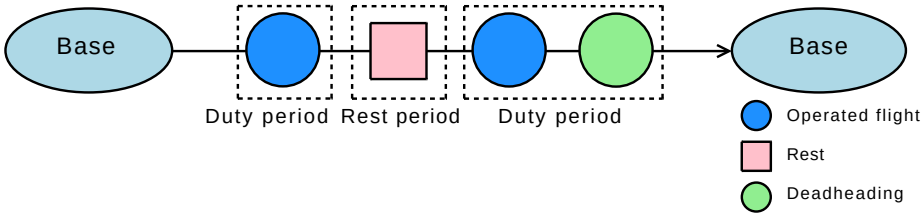


Figure C.2: *Illustration of a pairing.*

paper, we use the pairing's idle time as the cost of the pairing. That way crew utilisation is maximised. An illustration of a pairing can be seen on Figure C.2.

The *airline crew pairing problem* is then to find the set of pairings that covers all flights exactly once at minimum cost. Let $\mathcal{P}$ be the set of feasible pairings. The problem is modelled as a *set partitioning problem*. Each row corresponds to a flight and each column corresponds to a pairing. Let $\bar{m} = |\mathcal{F}|$ be the number of flights and $n = |\mathcal{P}|$ be the number of pairings. Now, the pairings can be represented by a binary $\bar{m} \times n$ matrix $\boldsymbol{A}$, where the entries are defined by $a_{ij} = 1$ if flight $i \in \{1, \ldots, \bar{m}\}$ is contained in pairing $j \in \{1, \ldots, n\}$, and $a_{ij} = 0$ otherwise. Let c_j be the cost of pairing $j \in \{1, \ldots, n\}$. The decision variables x_j for $j \in \{1, \ldots, n\}$ govern the inclusion of pairing j in the solution and are binary. The mathematical programme can then be written as

$$\begin{aligned} &\text{minimise} && \boldsymbol{c}^\top \boldsymbol{x} \\ &\text{subject to} && \boldsymbol{A}\boldsymbol{x} = \boldsymbol{1} \\ & && \boldsymbol{x} \in \{0,1\}^n \ . \end{aligned}$$

Most airlines, however, extend this standard model to include the so-called *base constraints*. These constraints are required for distributing the pairings amongst the crew bases in a way that matches the actual distribution of where the crew is located geographically. Base constraints can be defined in many different ways. In order to simplify matters, we have chosen to include only one type of base constraint. A base constraint puts a lower or an upper bound on the number of mandays that can be worked out of a set of crew bases in a given time period. A pairing contributes to a base constraint, if the pairing origins from that set of crew bases in the specified time period. The pairing's contribution to the base constraint is the manday count of the pairing and given as d_j, where $j \in \{1, \ldots, n\}$.

Let $\mathcal{B}$ denote the set of base constraints and set $m = \bar{m} + |\mathcal{B}|$. We can then

augment $\boldsymbol{A}$ to an $m \times n$ matrix where

$$a_{ij} = \begin{cases} d_j & \text{if pairing } j \text{ originates from the set of crew bases and in the} \\ & \text{time period specified by base constraint } i, \\ 0 & \text{otherwise} \end{cases}$$

for $i \in \{m'+1, \ldots, m\}$ and $j \in \{1, \ldots, n\}$. The base constraints are of less-than-or-equal or greater-than-or-equal type, and most often have non-unit right hand sides. We therefore end up with a *generalised set partitioning model*, where slack and surplus columns are included to convert the inequality base constraints to equality constraints:

$$\begin{array}{ll} \text{minimise} & \boldsymbol{c}^\top \boldsymbol{x} \\ \text{subject to} & \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} \\ & \boldsymbol{x} \in \{0,1\}^{\bar{n}} \ . \end{array}$$

Here, the flight set partitioning constraints for $i \in \{1, \ldots, \bar{m}\}$ have $b_i = 1$, and the base constraints for $i \in \{\bar{m}+1, \ldots, m\}$ have $b_i \in \mathbb{Z}_+ \cup \{0\}$. The dimension $\bar{n}$ is equal to n plus the number of slack and surplus columns.

We allow for the possibility of leaving flights uncovered at a high objective value penalty, and we allow for the violation of base constraints, also with a high penalty. This is modelled by having feasibility singleton columns for flights and for base constraints in the model.

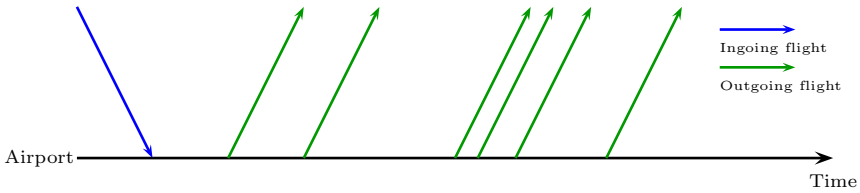
The number of possible pairings in the set partitioning formulation is very large, so the pairings are typically only enumerated implicitly by column generation. In the present approach we will, however, not perform column generation, but subsequence generation.

C.3 Subsequence limitation

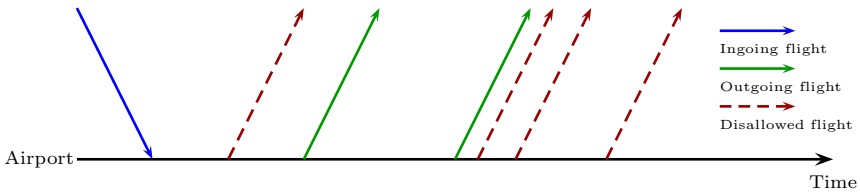
The *subsequences* for a flight $f \in \mathcal{F}$ are the set of pairs $(f, g) \in \mathcal{F}^2$ where $g \in \mathcal{F}$ is a subsequent flight that can follow f in a feasible way in a pairing. We denote this set $\mathcal{S}(f) \subset \mathcal{F}^2$. Subsequences are illustrated on Figure C.3(a). In general terms for an $m \times n$ zero-one matrix $\boldsymbol{A}$ with entries a_{ij}, the subsequence set $\mathcal{S}(s)$, for any row s is given by

$$\mathcal{S}(s) = \{(s,t) : [\exists j \in \{1, \ldots, n\} : a_{sj} = 1, a_{ij} = 0 \text{ for } s < i < t, a_{tj} = 1]\} \ .$$

In the example on Figure C.4 we have $\mathcal{S}(1) = \{(1,3), (1,4), (1,6)\}$. Matrices where the *subsequence count* $|\mathcal{S}(s)| \leq 1$ for all $s \in \{1, \ldots, m\}$ are said to



(a) Subsequences.



(b) Severely limited subsequences.

Figure C.3: *Subsequences for an ingoing flight.*

$$\begin{pmatrix} \boxed{1} & \boxed{1} & 1 & 1 & \boxed{1} & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \boxed{1} & 1 & 0 \\ \boxed{1} & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & \boxed{1} & 1 & 0 & 0 & 1 & 0 \end{pmatrix}$$

Figure C.4: *Subsequences for row 1.*

have *unique subsequence*, and such matrices are balanced, see Ryan and Falkner (1988). Exploiting results from graph theory, see Conforti et al. (2001), we know that the LP relaxation of a set partitioning problem with a balanced $\boldsymbol{A}$ matrix has an integral optimal solution. Intuitively, the closer we get towards unique subsequence, the closer we get to naturally integral LP solutions. Ryan and Falkner (1988) show experimental results to support this.

Therefore, we severely limit the subsequence count for each flight when generating pairings, see Figure C.3(b). In this example the first disallowed flight is removed, because there is not enough ground time for a robust aircraft change. The three last disallowed flights are removed, because they have a lot of ground idle time, so it is not likely (though still possible) that they will end up in an optimal solution.

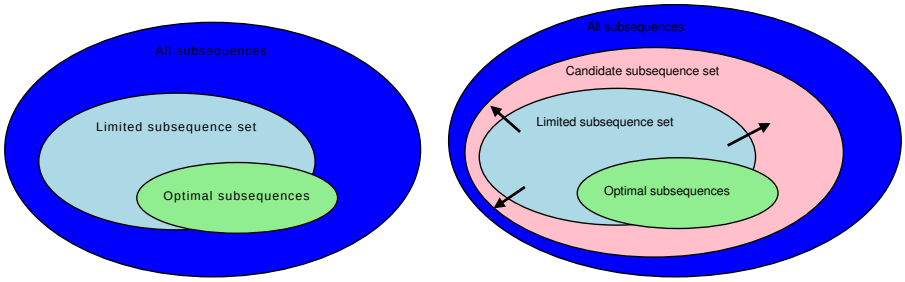
The possible subsequent outgoing flights for an ingoing flight f are now restricted to be in the *limited subsequence set* $\mathcal{L}(f) \subseteq \mathcal{S}(f)$. Let $\mathcal{S} = \bigcup_{f \in \mathcal{F}} \mathcal{S}(f)$ denote the set of all subsequences for all flights, let $\mathcal{L} = \bigcup_{f \in \mathcal{F}} \mathcal{L}(f)$ denote the set of limited subsequences for all flights, and let $\mathcal{O}$ denote an optimal subsequence set, i.e. an optimal solution, for all flights. Naturally, $\mathcal{O}$ has unique subsequence due to the set partitioning constraints. The relations between these three sets can be illustrated by a Venn diagram, see Figure C.5(a). There could, of course, be more than one set of optimal subsequences, but we only show one set on the figure.

The disadvantage of this limited subsequence approach is that some optimal subsequences might be excluded. However, the approach results in significantly fewer possible pairings, and therefore a total enumeration of the pairings in $\mathcal{L}$ can be carried out. Moreover, when the LP relaxation is solved, fewer fractions are expected, as the subsequence count of all flights per construction is low.

C.4 Subsequence generation

To remedy the possible lack of optimal subsequences, the limited subsequence set is made to be dynamic. The core idea is to generate subsequences that will decrease the objective value. We exploit the fact, that in crew pairing the chosen subsequences will most often be close in time, which is natural, keeping the pairing cost definition in mind. Therefore, the hope is that only relatively few subsequences with much idle time have to be generated.

A *candidate subsequence set* $\mathcal{C}(f)$ is defined for all flights $f \in \mathcal{F}$, and again we define $\mathcal{C} = \bigcup_{f \in \mathcal{F}} \mathcal{C}(f)$. We have $\mathcal{L}(f) \subseteq \mathcal{C}(f) \subseteq \mathcal{S}(f)$ for all $f \in \mathcal{F}$ and $\mathcal{L} \subseteq \mathcal{C} \subseteq$



(a) Optimal subsequences might be missing. (b) Limited subsequence set is dynamic.

Figure C.5: *The relations between the set of all subsequences $\mathcal{S}$, the limited subsequence set $\mathcal{L}$, the candidate subsequence set $\mathcal{C}$, and an optimal subsequence set $\mathcal{O}$.*

$\mathcal{S}$, which is shown in Figure C.5(b). The idea is now to expand $\mathcal{L}$ with attractive subsequences from $\mathcal{C}$. A subsequence $s \in \mathcal{C}$ is attractive if it is likely that $s \in \mathcal{O}$, where again $\mathcal{O}$ is a set of optimal subsequences. Iteratively an *attractive subsequence set* $\mathcal{A} \subseteq \mathcal{C}$ is found and added to $\mathcal{L}$. Although Figure C.5(b) shows the set of optimal subsequences to be contained in the candidate subsequence set, there is no guarantee for this. In Algorithm 1 the outline of the algorithm for solving the LP relaxation of the pairing problem can be seen.

Algorithm 1 Subsequence generation

1: Find an initial limited subsequence set $\mathcal{L}$
2: Enumerate all pairings over $\mathcal{L}$
3: Solve the LP relaxation on these pairings
4: **while** stop criteria not met **do**
5: Based on the LP dual vector, identify a set of attractive subsequences $\mathcal{A} \subseteq \mathcal{C}$
6: Enumerate pairings for each of the subsequences in $\mathcal{A}$
7: Set $\mathcal{L} := \mathcal{L} \cup \mathcal{A}$
8: Expand the LP relaxation with the enumerated pairings and re-solve
9: **end while**

The means that is used to identify attractive subsequences is the dual vector from the LP solution. The dual vector is passed on to a pairing generator that produces negative reduced cost columns on a the candidate subsequence set $\mathcal{C}$. The pairing generator is a resource constrained shortest path solver, which is run on subsequences from $\mathcal{C}$. The shortest path solver is a labelling algorithm, see for instance Irnich and Desaulniers (2005). The negative reduced cost columns, that are returned from the pairing generator, are analysed in order to collect

statistics about the subsequences in $\mathcal{C}\backslash\mathcal{L}$.

The pairing generator is run sequentially on N different networks consisting of subsequences $\mathcal{C}^k \subseteq \mathcal{C}$ for $k \in \{1, \ldots, N\}$ with $\bigcup_{k=1}^{N} \mathcal{C}^k = \mathcal{C}$. The networks are kept small, so that the shortest path solver can execute very fast. In crew pairing there are four classes of subsequences that are very important to recognise:

1. *Follow-the-aircraft subsequences*: A follow-the-aircraft subsequence is a subsequence, where the crew flies out on the same aircraft as they flew in with. This type of subsequence is very robust towards possible delays, and one would expect the majority of the subsequences in an optimal crew pairing solution to be follow-the-aircraft. This expectation is supported by data from Air New Zealand. The follow-the-aircraft subsequence is unique, as there can only be one subsequent flight on the same aircraft. Most often the follow-the-aircraft subsequence will be low-cost, because the minimum sit time for crew is close to the minimum turnaround time for the aircraft. Being unique, robust, and low-cost, the follow-the-aircraft subsequence is the most attractive subsequence class.

2. *Robust subsequences*: A crew coming in on flight f can leave on flight g, if the *minimum sit time* is respected. However, if flight f is delayed and the time difference between arrival and departure of the two flights is exactly the minimum sit time, then flight g will also be delayed. A way to try avoid this delay propagation, is to add some *buffer time* to the minimum sit time. This of course comes at a higher pairing cost, as the crew might get unnecessary idle time. Studies in Ehrgott and Ryan (2002) show that delays increase during the day (and reset at midnight), so the buffer time should also increase during the day. We can now define a robust subsequence, as a subsequence, where the time difference between arrival and departure respects the buffer time needed at the given time of day.

3. *Meal break subsequences*: Naturally, crew is entitled to meal breaks, which is controlled by complex regulations. A meal break subsequence is a subsequence, where there is sufficient time for a meal break either inflight or on the ground between the flights.

4. *Overnight subsequences*: An overnight subsequence is a subsequence, where the time difference between arrival and departure is longer than the *minimum rest time*. An overnight subsequence is needed for a crew that flies in to a non-base airport late at night, where there is no subsequent flight to a home base. The overnight subsequence is also needed for a crew to fly out of a non-base airport early in the morning, where there has been no preceding incoming flight.

Follow-the-aircraft and robust subsequences are preferred from a pairing cost and robustness point of view, but the meal break and overnight subsequences are needed in order to make the pairings feasible and cover all flights. We will work with three subset of the candidate subsequence set $\mathcal{C}$ (so we have $N = 3$), based on these classes. The set $\mathcal{C}^1$ consists of follow-the-aircraft subsequences and robust subsequences. The set $\mathcal{C}^2$ consists of follow-the-aircraft subsequences and meal break subsequences, and $\mathcal{C}^3$ consists of follow-the-aircraft subsequences and overnight subsequences. The motivation for this setup, is, that we will now have networks for the pairing generator, that search specifically for robust, meal break, or overnight subsequences.

In order for the pairing generator to solve quickly, the subsequence count for all flights is kept low, that is $|\mathcal{C}^k(f)| \leq n^k$, where n^k is a small integer less than, say, five for all $k \in \{1, \ldots, N\}$. It is important to note that n^k is an upper bound on the subsequence count in the given set for a flight. Consider for instance a flight going in to a non-busy airport. If the first robust outgoing flight (other than the follow-the-aircraft flight) departs, say, nine hours after the arrival of the ingoing flight, we do not include the flight as a robust subsequence, because it is unlikely that a pairing with such excessive idle time will end up in an optimal solution. Similar reasoning goes for the meal break and the overnight subsequence sets.

The set $\mathcal{C}^1$ is used as the initial limited subsequence set $\mathcal{L}$, where total enumeration is carried out.

For each subsequence $s \in \mathcal{C}$ we maintain four measures that are accumulated over all iterations and updated after analysis of the set of negative reduced cost columns returned by the pairing generator:

1. Count of columns containing s.
2. Count of different dual vectors that have produced columns containing s.
3. Sum of the reduced cost of columns that contain s.
4. Sum of the contribution from s to the negative reduced cost of columns containing s.

These measures can all be computed and updated quickly, which is important with respect to keeping the computational overhead of the approach at a minimum. The measures are correlated, so a high rank in one measure could also give a high rank in some of the other measures. In each iteration some subsequences are identified as attractive based on these four measures and added to $\mathcal{A}$. The goal is, of course, to be able to, as early as possible, identify the subsequences that potentially could end up in an optimal or near-optimal solution. Ideally

one would identify subsequences that were non-dominated on all four measures and add these to $\mathcal{A}$. However, finding non-dominated points in four dimensions is very time-consuming, so instead we use just add one of the four measures, or we alternate between them. The idea of using dual information to identify attractive flights is also used by Barnhart et al. (1995). Here, only passengering flights are searched for, and added to a standard column generation approach.

Whenever a subsequence s is identified as an attractive subsequence, a whole set of columns which include the new subsequence is added to the LP relaxation. Enumeration is carried out in one of the following two ways:

1. Enumeration of all feasible pairings in $\mathcal{C}$ containing s.

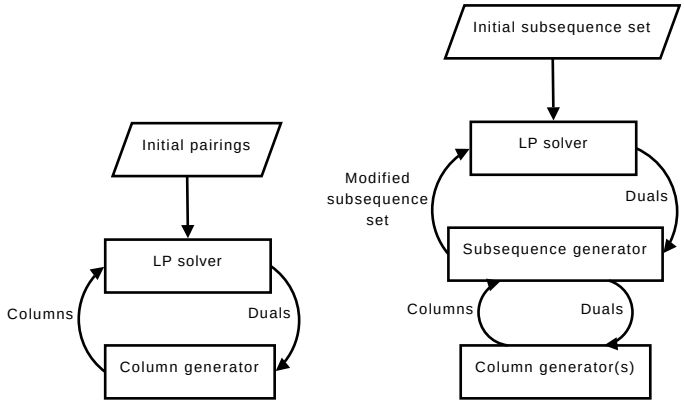
2. Enumeration of all feasible pairings in $\mathcal{L}$ containing s.

The reason why a relatively large set of columns is added to the LP relaxation, is, that whenever a subsequence is identified as attractive, it is believed that it is likely to end up in an optimal solution. And hence, the optimal solution will contain one of the enumerated columns. The first way of enumerating gives rise to more columns in the LP relaxation. This is beneficial when it is strongly believed that a "right" subsequence is identified. The second way of enumerating is more restrictive on the set of columns that are added to the LP relaxation. Hence, the second enumeration scheme is expected to be better at keeping the good integer properties of the initial limited subsequence set. With this scheme, adding a subsequence only increases the subsequence count with one for a single flight, namely for the first flight in the added subsequence. The subsequence generation algorithm is terminated, when the objective value has not improved significantly over a given span of iterations.

The differences between classic column generation and subsequence generation can be illustrated as the flowchart comparison in Figure C.6. Subsequence generation has an extra part where columns are analysed.

C.5 Computational results

The goal in this section is to provide a benchmark analysis of the devised solution algorithm. We will benchmark the subsequence generation approach against a classical column generation approach. In this way, we will investigate the trade-off between solution quality and integrality of the solutions.



(a) Classic column generation. (b) Subsequence generation.

Figure C.6: *Difference between classic column generation and subsequence generation.*

	w08r01a	w08r01b	w08r01c	w08r01d	w08r01e	w08r02a	w08r02b	w08r02c	w08r02d	w08r02e	w08r03a	w08r03b	w08r03c	w08r03d	w08r03e	w08r04a	w08r04b	w08r04c	w08r04d
$\lvert\mathcal{F}\rvert$	450	430	430	370	400	380	400	320	450	350	320	420	420	450	320	400	400	350	320
$\lvert\mathcal{B}\rvert$	5	5	5	5	4	5	5	5	5	4	5	5	5	5	4	5	5	5	5

Table C.1: *Characteristics for the test instances.* $|\mathcal{F}|$ *is number of flights and* $|\mathcal{B}|$ *is number of base constraints.*

Air New Zealand has provided us with 19 real-life data instances from their domestic timetable. It should be noted that the domestic timetable for Air New Zealand covers Australia and various destinations in the Pacific Ocean. In order to allow the algorithm to terminate in reasonable time, we have limited the number of flights in the instances. Characteristics for these instances can be seen in Table C.1.

We consider two quality measures for a solution. The first quality measure is the LP objective value. The objective value is value is the sum of idle time for all pairings plus penalties for leaving flights uncovered and penalties for violating base constraints. The second quality parameter is the number of uncovered flights. A flight is uncovered, if the feasibility column for that flight is chosen (perhaps fractionally) in the LP solution. Both quality parameters should be minimised.

To gauge the integrality of a solution, we consider two integer measures. Let $\boldsymbol{x}^* = (x_1^*, \ldots, x_n^*)^\top$ denote an LP solution to the crew pairing problem. The first integer measure counts the number of variables at value one in the solution and is calculated as $|\{i \in \{1, \ldots, n\} : x_i^* = 1\}| \cdot 100/|\{i \in \{1, \ldots, n\} : x_i^* > 0\}|$. The second integer measure counts the number of "nice fraction"-variables in the solution, where a nice fraction is nonzero rational number smaller than or equal to one and a multiple of $1, 1/2, \ldots, 1/8$. It is calculated as $|\{i \in \{1, \ldots, n\} : x_i^* \in \{a/b : a, b \in \{1, \ldots, 8\}\}\}| \cdot 100/|\{i \in \{1, \ldots, n\} : x_i^* > 0\}|$. Integer measures should be maximised. The integer measure should give an indication of how easy or how difficult the fractions in the LP solution would be to resolve in a branch-and-bound framework. The use of such integer measures are based on the results from Ryan and Falkner (1988). The last measure we compare is run time.

We test different settings of the subsequence algorithm:

1. Maximum sizes for the candidate subsequence sets for a flight n^k: Experiments are carried out with n^k set to 2, 3, or 4.

2. Subsequence identification scheme: Either 1) count of columns, 2) count of different duals, 3) sum of reduced costs, 4) sum of subsequence contribution, or 5) an alternation of the previous is used.

3. Pairing enumeration scheme: Either enumeration is done over 1) $\mathcal{C}$ or 2) $\mathcal{L}$.

We identify one subsequence per iteration, and the algorithm is terminated, when the improvement in the LP objective value is less than 1% over a span of 100 iterations. The pairing generator returns up to eight columns with negative reduced cost each time it is run. For all of the 19 instances we run the algorithm for all 30 combinations of the settings described above. For all instances we also run a classic column generation algorithm where no limitation of subsequences is used. In order to make a fair comparison on quality, we set the robust buffer time to zero, so that we in effect do not search for robust subsequences. The time-out for all test runs is set to one hour, and all tests are run on 2.67 GHz Intel Xeon X5550 CPUs with 23.5 GB of memory. The algorithm is implemented in C++ and compiled with g++ 4.4.0 on a Linux computer. LP relaxations are solved with the LP solver from MOSEK version 6 using an academic license.

For all instances and for all 30 combinations of settings, we calculate ratios R/R_{ben}, where R denotes the value for the subsequence generation algorithm, we want to benchmark, and R_{ben} denotes the value for classic column generation to benchmark against. Table C.2 shows the averages over all instances for each

of the settings. For the solve time ratio it is preferable for our algorithm to have a ratio less than 1.00, meaning that subsequence generation is faster. For the root LP value ratio, and the uncovered ratio it is preferable for our algorithm to have a ratio as close to 1.00 as possible, as the optimal values from classic column generation is a lower bound. However, classic column generation times out on all instances, and therefore it would actually be possible to have a ratio less than 1.00. For the "at 1"-measure ratio and the "nice fraction"-measure ratio it is preferable for our algorithm to have a ratio larger than 1.00. From Table C.2 it can be seen that the subsequence generation algorithm is always clearly faster than classic column generation. Classic column generation is still producing negative reduced cost columns at the one hour time-out limit. This is probably due to high degeneracy. From the table it can also be seen that the subsequence generation algorithm performs worse on the two quality parameters, LP objective value and number of uncovered flights. This was expected, as the subsequence generation algorithm is working on a limited subsequence set and therefore is more restricted than classic column generation, which has the full set of subsequences to choose from. Still, the conclusion that must be drawn from these averages, is, that more work on the subsequence identification procedure must be carried out, as there are some optimal subsequences missing.

One should note that the number of uncovered flights have a very large impact on both the 'Root LP value ratio' and the 'Uncovered ratio'. If, for instance, one setting results in two out of 400 flights to be uncovered, and a second setting leaves three out of 400 flights uncovered, then the second setting would have a 'Uncovered ratio' of 1.5 when comparing to the first setting. The same holds for the 'Root LP value ratio', due to the high and dominant penalty for violating the flight constraints.

Lastly, from Table C.2, it can be seen, that the subsequence generation algorithm has better integer measures on average. Therefore, we have very good reason to believe that integer solution can be found faster than when classic column generation is used.

Comparing the different settings of the subsequence generation algorithm, Table C.2 shows, as expected, that the more the candidate subsequence set $\mathcal{C}$ is limited, the worse the solution quality gets, but the solutions also get slightly more integral, which is also expected. The different measures to identify attractive subsequences seem to perform almost equally good. This is probably due to a high correlation between them. Enumerating pairings over $\mathcal{C}$ gives a better solution quality than enumerating pairings over $\mathcal{L}$, but the integrality of the solution is higher when enumerating over $\mathcal{L}$. This is also expected, as $\mathcal{L}$ is more restricted than $\mathcal{C}$. Again, improving the subsequence identification would lead to increased solution quality, while integrality benefits of the restrictions could be kept.

Settings	Solve time ratio	Root LP value ratio	Uncovered ratio	"At 1"-measure ratio	"Nice fraction"-measure ratio
CG	1.00	1.00	1.00	1.00	1.00
2/1/1	0.00	1.77	1.63	1.95	1.98
2/1/2	0.00	1.84	1.80	1.83	2.02
2/2/1	0.00	1.75	1.66	1.75	2.11
2/2/2	0.00	1.84	1.82	1.57	1.94
2/3/1	0.00	1.76	1.73	1.34	1.81
2/3/2	0.00	1.85	1.80	1.47	1.90
2/4/1	0.00	1.75	1.64	2.10	2.10
2/4/2	0.00	1.90	1.87	1.47	1.90
2/5/1	0.00	1.75	1.74	1.76	1.92
2/5/2	0.00	1.83	1.79	1.92	2.05
3/1/1	0.01	1.25	1.25	1.22	2.08
3/1/2	0.00	1.30	1.21	1.48	2.08
3/2/1	0.01	1.25	1.27	1.06	1.89
3/2/2	0.00	1.30	1.30	1.43	1.89
3/3/1	0.01	1.25	1.29	0.92	1.74
3/3/2	0.00	1.28	1.23	1.43	1.89
3/4/1	0.00	1.25	1.35	1.13	2.06
3/4/2	0.00	1.29	1.31	1.98	2.04
3/5/1	0.01	1.25	1.32	1.09	1.85
3/5/2	0.00	1.28	1.34	1.47	1.94
4/1/1	0.01	1.23	1.22	1.07	1.71
4/1/2	0.01	1.26	1.21	1.72	1.57
4/2/1	0.01	1.23	1.24	1.02	1.52
4/2/2	0.01	1.26	1.28	1.49	1.73
4/3/1	0.01	1.24	1.27	0.88	1.80
4/3/2	0.01	1.24	1.30	1.26	1.80
4/4/1	0.01	1.21	1.26	0.81	1.51
4/4/2	0.00	1.25	1.26	1.62	2.06
4/5/1	0.01	1.23	1.31	0.80	1.55
4/5/2	0.01	1.24	1.21	1.44	1.75
2/*/*	0.00	1.81	1.75	1.72	1.97
3/*/*	0.00	1.27	1.29	1.32	1.95
4/*/*	0.01	1.24	1.25	1.21	1.70
/1/	0.01	1.44	1.39	1.55	1.91
/2/	0.01	1.44	1.43	1.39	1.85
/3/	0.01	1.44	1.44	1.22	1.82
/4/	0.00	1.44	1.45	1.52	1.95
/5/	0.01	1.43	1.45	1.41	1.84
//1	0.01	1.41	1.41	1.26	1.84
//2	0.00	1.47	1.45	1.57	1.90
//*	0.01	1.44	1.43	1.42	1.87

Table C.2: *Benchmarking of the subsequence generation algorithm against classic column generation. Settings are read as Maximum-candidate-set-size / Subsequence-identification-scheme / Pairing-enumeration-scheme. An asterisk means that an average is taken over that setting.*

Table C.3 shows statistics for the test runs on the w08r03b instance. The statistics are representative for the other instances as well. The table reveals that almost no computation time is spent with analysis of columns and subsequence identification. As these are the two core parts of the solution approach, this clearly points out an area for future research with a great potential gain. If the right subsequences are identified, the solution quality would naturally increase.

Figure C.7 shows a typical graph of the LP objective value per main loop iteration. The flat lines of the graph are especially interesting. Whenever there is a flat line, it means that the subsequences added in those iterations did not lower the LP objective value. This could mean one of two things: Either we have identified the wrong subsequence, or we have identified the right subsequence, but the subsequence cannot be used, so we do not get the gain of adding it. The latter is most easily seen in the case with unique subsequence for all flights. Let (f_1, f_2) be the subsequence selected in the current solution and let (f_1, f_3) be the new subsequence that is identified as attractive in the current iteration. However, (f_1, f_3) cannot be selected by the LP relaxation, before a new subsequence for (f_4, f_2) for f_2 is added. The dual vector will point out a suggestion for (f_4, f_2) eventually, but an early "subsequence partner"-prediction of (f_4, f_2) would be beneficial. The problem is, though, that this problem propagates much further than to just one other subsequence. Still, both cases support that an improvement of the subsequence identification would benefit the overall algorithm a lot.

C.6 Conclusion and future work

We have contributed a novel solution approach based on generation of subsequences of flights for solving the well-known airline crew pairing problem. We have developed a method for solving the LP relaxation of the crew pairing problem, and the method aims at keeping the LP solution as close to integral as possible, thereby providing a good starting point for branch-and-bound. We have benchmarked the new method against a classic column generation algorithm. The benchmarking has revealed that the subsequence generation approach indeed is less fractional, but this comes at the price of a decrease in solution quality. The benchmarking has been carried out on real-life instances, and on all instances the subsequence generation approach was clearly faster than classic column generation.

There is a number of directions that future work on the subsequence generation method could go. Obviously, nesting the method in a branch-and-bound

Instance	Settings	Solve time (s)	LP solver (%)	Col. generation (%)	Col. analysis (%)	Subs. identification (%)	Col. enumeration (%)	Root LP value	Uncovered flights	"At 1" (%)	"Nice fraction" (%)	Main loop iterations	Subsequences added	Columns added	Stop reason
w08r03b	CG	3604.75	0	100	0	0	0	2.00034e+08	2	26	33	249	0	19811	TO
w08r03b	2/1/1	11.42	25	27	1	0	47	2.18032e+08	2	100	100	164	163	6298	NI
w08r03b	3/1/1	29.45	53	14	1	0	32	2.08365e+08	2	29	99	144	143	31669	NI
w08r03b	4/1/1	91.58	49	7	0	0	43	2.05233e+08	3	18	67	174	173	99385	NI
w08r03b	2/1/2	12.32	15	26	2	0	57	2.18698e+08	2	50	99	196	195	4075	NI
w08r03b	3/1/2	24.22	38	22	2	0	38	2.10430e+08	2	61	97	194	193	13904	NI
w08r03b	4/1/2	22.91	46	22	1	0	31	2.09529e+08	2	36	100	129	128	16949	NI
w08r03b	2/2/1	10.72	25	25	2	0	48	2.17366e+08	3	55	99	163	162	6590	NI
w08r03b	3/2/1	28.23	53	14	1	0	32	2.08366e+08	2	43	99	142	141	26443	NI
w08r03b	4/2/1	74.89	54	7	0	0	39	2.05306e+08	2	19	25	136	135	92115	NI
w08r03b	2/2/2	12.42	16	26	2	0	56	2.18032e+08	4	42	100	196	195	4374	NI
w08r03b	3/2/2	14.53	45	22	1	0	32	2.12528e+08	3	37	99	112	111	12596	NI
w08r03b	4/2/2	23.42	53	19	1	0	27	2.10027e+08	2	69	99	117	116	27339	NI
w08r03b	2/3/1	10.61	25	26	2	0	47	2.17432e+08	3	49	99	167	166	6356	NI
w08r03b	3/3/1	27.17	52	14	1	0	33	2.08366e+08	2	22	58	141	140	29918	NI
w08r03b	4/3/1	105.70	64	5	0	0	31	2.05233e+08	3	17	90	146	145	153388	NI
w08r03b	2/3/2	8.25	18	27	2	0	53	2.20247e+08	2	28	79	138	137	3606	NI
w08r03b	3/3/2	23.74	43	21	1	0	35	2.10363e+08	2	38	67	174	173	17832	NI
w08r03b	4/3/2	23.70	55	19	1	0	25	2.09630e+08	3	19	72	109	108	20837	NI
w08r03b	2/4/1	8.84	24	27	2	0	47	2.18367e+08	3	23	61	144	143	4679	NI
w08r03b	3/4/1	23.95	49	16	1	0	34	2.08365e+08	2	13	65	135	134	22741	NI
w08r03b	4/4/1	59.02	53	8	0	0	38	2.05533e+08	2	21	89	125	124	71852	NI
w08r03b	2/4/2	7.10	16	28	3	0	53	2.22532e+08	2	22	72	120	119	2840	NI
w08r03b	3/4/2	11.26	36	26	1	0	37	2.14027e+08	4	54	99	105	104	8726	NI
w08r03b	4/4/2	19.76	51	21	1	0	26	2.08531e+08	2	49	99	107	106	16682	NI
w08r03b	2/5/1	8.21	23	27	1	0	49	2.18032e+08	2	54	99	135	134	5060	NI
w08r03b	3/5/1	21.49	47	15	1	0	36	2.09363e+08	2	27	66	123	122	23275	NI
w08r03b	4/5/1	64.11	46	8	0	0	46	2.05233e+08	3	12	37	127	126	85375	NI
w08r03b	2/5/2	11.20	15	26	3	0	55	2.18586e+08	4	20	25	177	176	3675	NI
w08r03b	3/5/2	12.15	36	24	1	0	38	2.12027e+08	3	51	98	109	108	12143	NI
w08r03b	4/5/2	23.85	53	19	1	0	27	2.09529e+08	2	58	99	117	116	20007	NI

Table C.3: *Test statistics for the subsequence generation algorithm and classic column generation. Settings are read as Maximum-candidate-set-size / Subsequence-identification-scheme / Pairing-enumeration-scheme. NI abbreviates that no improvement is found over the given iteration span, and TO abbreviates time-out.*

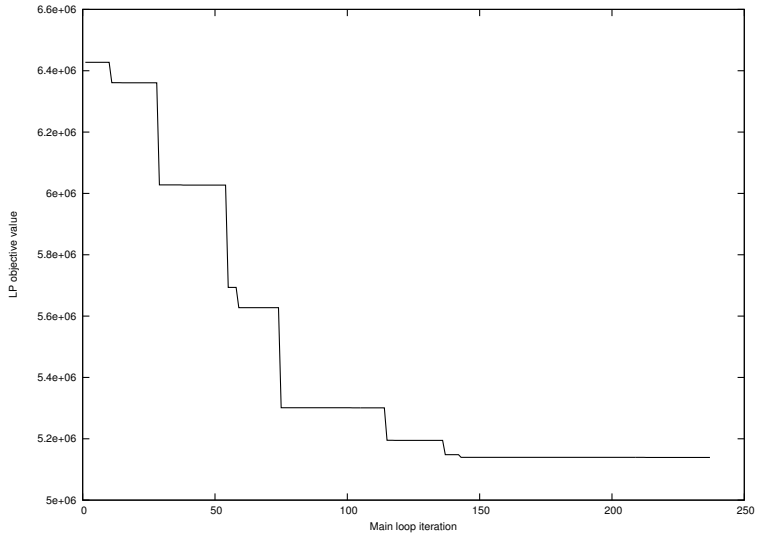


Figure C.7: *The LP objective value per main loop iteration for the instance w08r03b with settings 3/5/2.*

framework in order to find integer solutions would be interesting. This could be extended with delayed subsequence generation in each node of the branching tree. It should be noted that generation of subsequences in all tree nodes would mean that the objective value of a particular node is not a lower bound on the solution value for the children nodes.

The subsequence identification step is very important for the method to be successful. As mentioned earlier, the gain from adding a subsequence might not appear before another subsequence is added. Therefore, some kind of "subsequence partner"-prediction could prove beneficial. Perhaps also new measures for subsequence identification could be invented to complement the existing four measures. Still, as mentioned earlier, such measures should be computationally fast in order to avoid a large overhead of the approach. It could also be interesting to experiment with the outcome of identifying more than a single subsequence per iteration.

To use of historic data could also be a key to success. If one has a set of pairings that make up a good solution for June, then many of the subsequences in these pairings would probably be repeated in a good solution for July, as flight timetables and crew resources are relatively stable. Therefore, last month's subsequences could be put in the initial limited subsequence set, or a measure that in some way took these into account could be used.

As it is now the limited subsequence set $\mathcal{L}$ can only expand. It could speed up computation and maybe improve integrality, if the set could also shrink. Subsequences that have not appeared, i.e. columns containing the subsequence have not been selected even fractionally, in the LP solution for a given span of iterations could be removed. When a subsequence is removed, all columns containing the subsequence should be removed.

At the moment the candidate subsequence set $\mathcal{C}$ is static. However, there might be a gain in making it dynamic, as is the case for the limited subsequence set. Regarding the column enumeration, it is possible to generate duplicate columns. These could also be removed from the problem, if the overhead for doing this is not to severe.

A final suggestion for future work is to let pairing generation happen in parallel. Each of the $\mathcal{C}^k$ candidate subsequence subsets could be run on its own processor, and then return negative reduced cost columns to a single controlling process. As the pairing generation is faster than solving the LP relaxation, subsequences could be enumerated and added to the LP relaxation, before the LP solver had reached its optimum. Dual stabilisation should then be added in order to make the duals reliable as early as possible.

Acknowledgements: The authors would like to thank Paul Keating from Air New Zealand for providing the data instances and explaining them thoroughly, as well as his participation in many stimulating discussions about the solution approach.

References

AhmadBeygi, S., A. Cohn, and M. Weir (2009). "An integer programming approach to generating airline crew pairings". *Computers & Operations Research* 36.4, pp. 1284–1298.

Andersson, E., E. Housos, N. Kohl, and D. Wedelin (1998). "Crew Pairing Optimization". In: *Operations Research in the Airline Industry*. Ed. by G. Yu. Kluwer Academic Publishers. Chap. 8, pp. 228–258.

Barnhart, C., L. Hatay, and E. L. Johnson (1995). "Deadhead selection for the long-haul crew pairing problem". *Operations Research* 43.3, pp. 491–499.

Barnhart, C., A. M. Cohn, E. J. Johnson, D. Klabjan, G. L. Nemhauser, and P. H. Vance (2003). "Airline Crew Scheduling". In: *Handbook of Transportation Science*. Ed. by R. W. Hall. 2nd ed. Norwell: Kluwer Academic Publishers. Chap. 14, pp. 517–560.

Butchers, E. R., P. R. Day, A. P. Goldie, S. Miller, J. A. Meyer, D. M. Ryan, A. C. Scott, and C. A. Wallace (2001). "Optimized crew scheduling at Air New Zealand". *Interfaces* 31.1, pp. 30–56.

Conforti, M., G. Cornuéjols, A. Kapoor, and K. Vuskovic (2001). "Perfect, ideal and balanced matrices". *European Journal of Operational Research* 133.3, pp. 455–461.

Desaulniers, G., J. Desrosiers, M. Gamache, and F. Soumis (1998). "Crew Scheduling in Air Transportation". In: *Fleet Management and Logistics*. Ed. by T. G. Crainic and G. Laporte. Boston: Kluwer Academic Publishers, pp. 169–185.

Ehrgott, M. and D. M. Ryan (2002). "Constructing robust crew schedules with bicriteria optimization". *Journal of Multi-Criteria Decision Analysis* 11.3, pp. 139–150.

Gopalakrishnan, B. and E. L. Johnson (2005). "Airline Crew Scheduling: State-of-the-Art". *Annals of Operations Research* 140.1, pp. 305–337.

Graves, G. W., R. D. McBride, I. Gershkoff, D. Anderson, and D. Mahidhara (1993). "Flight Crew Scheduling". *Management Science* 39.6, pp. 736 –745.

Irnich, S. and G. Desaulniers (2005). "Shortest Path Problems with Resource Constraints". In: *Column Generation*. Ed. by G. Desaulniers, J. Desrosiers, and M. M. Solomon. GERAD 25th Anniversary Series. Springer. Chap. 2, pp. 33–65.

Lavoie, S., M. Minoux, and E. Odier (1988). "A new approach for crew pairing problems by column generation with an application to air transportation". *European Journal of Operational Research* 35.1, pp. 45 –58.

Ryan, D. M. and J. C. Falkner (1988). "On the integer properties of scheduling set partitioning models". *European Journal of Operational Research* 35.3, pp. 442 –456.

Saddoune, M., G. Desaulniers, I. Elhallaoui, and F. Soumis (2011). "Integrated airline crew scheduling: A bi-dynamic constraint aggregation method using neighborhoods". *European Journal of Operational Research* 212.3, pp. 445–454.

Vance, P. H., C. Barnhart, E. L. Johnson, and G. L. Nemhauser (1997). "Airline Crew Scheduling: A New Formulation and Decomposition Algorithm". *Operations Research* 45.2, pp. 188–200.

Wedelin, D. (1995). "An algorithm for large scale 0-1 integer programming with application to airline crew scheduling". *Annals of Operations Research* 57, pp. 283–301.

Appendix D

An IP Framework for the Crew Pairing Problem using Subsequence Generation

---

Matias Sevel Rasmussen, Richard M. Lusby, David M. Ryan, and Jesper Larsen

Submitted to: *Journal of the Operational Research Society* (2011).

An IP Framework for the Crew Pairing Problem using Subsequence Generation*

Matias Sevel Rasmussen[1], Richard M. Lusby[1], David M. Ryan[2], and Jesper Larsen[1]

In this paper we consider an important problem for the airline industry. The widely studied crew pairing problem is typically formulated as a set partitioning problem and solved using the branch-and-price methodology. Here we develop a new integer programming framework, based on the concept of subsequence generation, for solving the set partitioning formulation. In subsequence generation one restricts the number of permitted subsequent flights, that a crew member can turn to after completing any particular flight. By restricting the number of subsequences, the number of pairings in the problem decreases. The aim is then to dynamically add attractive subsequences to the problem, thereby increasing the number of possible pairings and improving the solution quality. Encouraging results are obtained on 19 real-life instances supplied by Air New Zealand and show that the described methodology is a viable alternative to column generation.

Keywords: Airline crew pairing, Subsequence generation, Set partitioning, Integer programming

D.1 Introduction

For an airline company crew costs can be identified as the second largest expense, typically only fuel costs are higher. In 1991 it was reported that American Airlines spent USD 1.3 billion on crew (see Gopalakrishnan and Johnson (2005)). Unlike fuel costs, crew costs can in some sense be controlled by an airline. The inefficient use of crew may lead to unnecessary expenditure. As a result, finding the optimal use of an airline's crew is a topic that has received significant attention in the literature, and now, as a result, optimisation tools are heavily

*Submitted to: *Journal of the Operational Research Society* (2011).

[1]Department of Management Engineering, Technical University of Denmark, Produktionstorvet, Building 424, DK-2800 Kgs. Lyngby, Denmark.

[2]Department of Engineering Science, The University of Auckland, New Zealand.

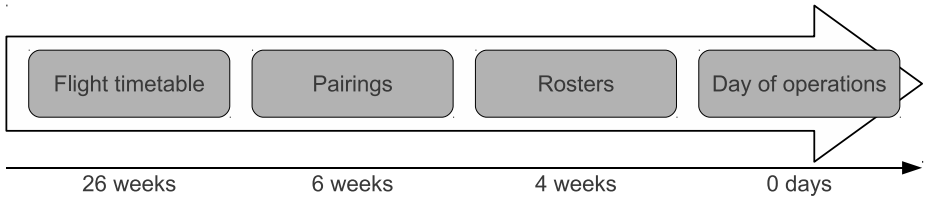


Figure D.1: *The airline crew scheduling process. The times are for Air New Zealand's domestic scheduling.*

used by airlines in their planning operations. Due to the high cost associated with crews, even minor improvements in the schedules can result in significant savings.

The focus of this paper is on the so-called *airline crew pairing problem*. This problem is one of the core optimisation problems encountered by an airline company. Its position in the planning horizon, as well as the other main optimisation problems can be seen in Figure D.1. While the airline crew pairing problem will be discussed in more detail in Section D.2, it essentially requires one to find sequences of flights that crew members will fly, at minimum cost. The sequences of flights are termed *pairings* and are anonymous. The pairing problem succeeds the *flight timetabling* step. Here, a schedule of all flights that will be flown by the airline must be constructed. This is then followed by *fleet assignment*, which requires one to assign an aircraft type to each of the flights. Finally, as a last step from an aircraft perspective, one must determine aircraft routes, termed the *aircraft routing problem*. A solution to the flight timetabling phase is required as input for the pairing problem; however, one can solve the pairing problem separately for cockpit and cabin crew, or even separately for each aircraft type. For example, here we consider the domestic Boeing 737 fleet for Air New Zealand. The final step in the planning horizon is *crew rostering*. Here, pairings, training, and vacation are combined to form actual rosters for individual crew member. The crew pairing and the crew rostering steps are together called *airline crew scheduling*. The steps for airline crew scheduling can be seen in Figure D.1.

A recent survey on airline crew scheduling can be found in Gopalakrishnan and Johnson (2005). The authors provide an overview of the different approaches that have been used over the last two decades to solve this problem. In addition to this, some promising directions for future work are described. The crew pairing problem has also been treated separately and in detail. Barnhart et al. (2003) give a text book description of airline crew scheduling and also have a detailed section on crew pairing with examples. The crew pairing problem

is formulated as a set partitioning problem and the authors describe how the problem can be solved as a weekly problem or a dated problem. In the weekly problem approach, one exploits repetitive patterns of flights over the weekdays, and can thus break the problem into smaller parts, which are then combined. This division of the problem is, of course, a trade-off against optimality. The dated problem approach, on the other hand, solves the problem directly, and is necessary for flight timetables where flights are not repeated several times a week. The complex cost structures for pairings are described by Gopalakrishnan and Johnson (2005) and Barnhart et al. (2003). Andersson et al. (1998) describe different approaches to crew pairing and give a detailed introduction to the Carmen (now Jeppesen) system for solving the crew pairing problem. The Carmen system uses a priori column generation; however, it has separated the checking of the pairing requirements into a special rules language. The Carmen system uses the optimisation approach described by Wedelin (1995). Desaulniers et al. (1998) present the crew pairing model as a special case of a generic air crew scheduling model that also covers, for instance, rostering. The crew pairing problem is solved by column generation. AhmadBeygi et al. (2009) develop an integer programming model for generating pairings. The model can be used, especially in research, to overcome the time-consuming task of implementing a pairing generator. Butchers et al. (2001) describe airline optimisation problems in general and the crew pairing problem in particular for Air New Zealand's domestic and international schedule. The authors also formulate the crew pairing problem as a set partitioning problem. Lavoie et al. (1988) use a set covering formulation and perform column generation on a duty period network. Graves et al. (1993) use a set partitioning formulation and do column generation on a network of flights. Vance et al. (1997) use a two-stage approach. First flights are combined to form duty periods, and next duty periods are combined to form pairings. Using dynamic constraint aggregation crew scheduling can be solved in an integrated approach, see Saddoune et al. (2011). In this way all constraints are virtually present in the master problem, but in an aggregated form, where constraints belonging to the same pairing are just represented by one active constraint. The update of these active constraints leads to a complex setup in the interplay with the column generator. This, though, does at present remain a very complex and time-consuming approach limited to academic environments only.

In this paper we extend the work of Rasmussen et al. (2011e), where the idea of using subsequence generation for solving the pairing problem was first proposed. Given the encouraging results from a linear programming (LP) perspective, here we extend this methodology to a full integer programming framework. The framework itself is similar to that of the well known *branch-and-price* approach for solving large scale optimisation problems; however, it does possess certain key differences. In particular, pairings found during the pricing phase of the algorithm are not directly added to the master problem, but instead are anal-

ysed in order to identify attractive subsequences. This is then followed by an enumeration step in which all pairings containing the identified subsequence(s) are enumerated and simultaneously added to the master problem.

The paper is organised as follows. In Section D.2 we provide a definition of the airline crew pairing problem and present the conventional generalised set partitioning formulation of it. Section D.3 describes a subsequence generation based solution approach for solving the linear programming model. This methodology is built into an integer programming framework in Section D.4 and computational results for the complete algorithm are presented in Section D.5. The main conclusions and directions for future work are summarised in Section D.6.

D.2 Problem formulation

In this section we formally define the airline crew pairing problem and present a mathematical formulation of it. We begin by introducing the required terminology and sets.

The flight schedule of an airline consists of a set of flights, $\mathcal{F}$. A *duty period* is a sequence of flights in $\mathcal{F}$ that can be flown by a crew member. Any duty period must adhere to several rules and regulations in order to be feasible. A crew member can either be *operating* or *passengering* (sometimes called *deadheading*) on a flight. Passengering allows crew members to be repositioned in order to operate other flights. A duty period consists of *flying time*, where the crew member is operating the flight, and *idle time*, which together give the *elapsed time*. Each duty period has a maximum flying time and a maximum elapsed time, as well as a maximum number of flights that can be operated. Duty periods must also respect meal break regulations. Consecutive duty periods are separated by a *rest period*, which must have a minimum duration. Finally, a so-called *sign-on* (*sign-off*) time, is imposed when starting (respectively, terminating) a duty period.

A *pairing* (sometimes called a *tour-of-duty*) is a sequence of duty periods and rest periods. Every airline has a set of *crew bases*, i.e. airports from where crew can start working. A feasible pairing must start and end at the same crew base. Pairings can only contain up to a maximum number of duties, and a pairing is only allowed to stretch over a certain number of *mandays* (where one manday is equivalent to the amount of work one person can produce in a day). The manday count is increased every time midnight is passed in the time zone where the pairing originates. Different airlines use different and quite complex ways of calculating the cost of a pairing, for examples of this see Gopalakrishnan

and Johnson (2005) and Barnhart et al. (2003). For the research described in this paper, the pairing's idle time is used as the cost of a pairing. This simple measure ensures the crew utilisation is maximised. An illustration of a pairing can be seen on Figure D.2.

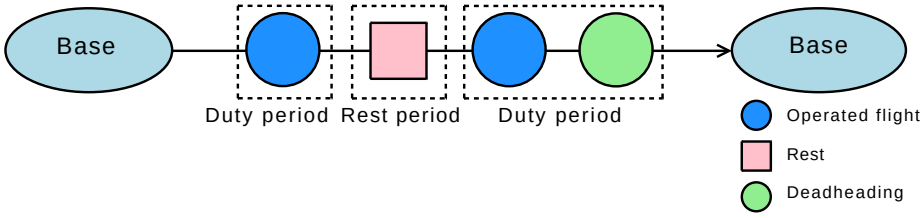


Figure D.2: *Illustration of a pairing.*

The airline crew pairing problem entails finding a set of pairings that covers all flights exactly once at minimum cost. Let $\mathcal{P}$ be the set of feasible pairings. The problem is modelled as a *set partitioning problem*. Each row corresponds to a flight $f \in \mathcal{F}$ and each column corresponds to a pairing $p \in \mathcal{P}$. Let $\bar{m} = |\mathcal{F}|$ be the number of flights and $n = |\mathcal{P}|$ be the number of pairings. Now, the pairings can be represented by a binary $\bar{m} \times n$ matrix $\boldsymbol{A}$, where the entries are defined by $a_{ij} = 1$ if flight $i \in \{1, \ldots, \bar{m}\}$ is contained in pairing $j \in \{1, \ldots, n\}$, and $a_{ij} = 0$ otherwise. Let c_j denote the cost of pairing $j \in \{1, \ldots, n\}$.

In addition to the flight partitioning constraints, most airlines include so-called *base constraints*. Such constraints are necessary in order to distribute the pairings across the crew bases in such a way that is consistent with where the actual crew are located. In other words, a base constraint is associated with a set of crew bases and puts a lower limit and/or an upper limit on the number of mandays that can be worked out of the associated bases over a given time horizon. A pairing contributes to a base constraint if the pairing originates, in the specified time horizon, from one of the bases associated with the constraint. The pairing's contribution to the base constraint is the manday count of the pairing and given as d_j, where $j \in \{1, \ldots, n\}$. Here we denote the set of base constraints that enforce an upper limit on manday count as $\mathcal{B}_1$, while those that enforce a lower limit are given by the set $\mathcal{B}_2$. The two matrices $\mathcal{B}_1$ and $\mathcal{B}_2$, with dimension $|\mathcal{B}_1| \times n$ and $|\mathcal{B}_2| \times n$, respectively, reflect the manday coverage of each pairing for each type of base constraint. Finally, vectors $\boldsymbol{b}_1 \in \mathbb{N}_0^{|\mathcal{B}_1|}$ and $\boldsymbol{b}_2 \in \mathbb{N}_0^{|\mathcal{B}_2|}$ give the corresponding upper and lower limits on manday count.

The decision variables x_j for $j \in \{1, \ldots, n\}$ govern the inclusion of pairing j in the solution and are binary. We also allow the possibility for leaving flights uncovered. The decision variables s_i for $i \in \{1, \ldots, m\}$ give the possibility of

leaving flight $i \in \mathcal{F}$ uncovered. Such variables are necessary to ensure feasibility and to reflect their unattractiveness each is assigned a high objective coefficient, M. In a similar way we also allow the base constraints to be violated. In reality one would prefer to violate the base constraints in preference to cancelling some of the flights. The decision variables u_k where $k = 1, \ldots, |\mathcal{B}_1|$ and o_k where $k = 1, \ldots, |\mathcal{B}_2|$ give the number of mandays one violates the respective base constraints by. Again, such decision variables are assigned a high cost, p (where $p < M$), to make them unattractive. The mathematical programme can then be written as follows:

$$\begin{aligned}
&\text{minimise} && \boldsymbol{c}^\top \boldsymbol{x} + M\boldsymbol{s} + p\boldsymbol{u} + p\boldsymbol{o} && \text{(D.1)}\\
&\text{subject to} && \boldsymbol{A}\boldsymbol{x} + \boldsymbol{I}\boldsymbol{s} = \boldsymbol{1} && \text{(D.2)}\\
& && \boldsymbol{B}_1\boldsymbol{x} + \boldsymbol{I}_1\boldsymbol{u} = \boldsymbol{b}_1 && \text{(D.3)}\\
& && \boldsymbol{B}_2\boldsymbol{x} - \boldsymbol{I}_2\boldsymbol{o} = \boldsymbol{b}_2 && \text{(D.4)}\\
& && \boldsymbol{x} \in \{0,1\}^n && \text{(D.5)}\\
& && \boldsymbol{s} \in \mathbb{R}^n && \text{(D.6)}\\
& && \boldsymbol{u} \in \mathbb{R}_+^{|\mathcal{B}_1|} && \text{(D.7)}\\
& && \boldsymbol{o} \in \mathbb{R}_+^{|\mathcal{B}_2|}\ . && \text{(D.8)}
\end{aligned}$$

where $\boldsymbol{I}, \boldsymbol{I}_1$ and $\boldsymbol{I}_2$ are identity matrices of appropriate size.

The number of possible pairings in the set partitioning formulation is very large and the above model is typically only solved using branch-and-price methodology. In what follows, however, we will present a new integer programming framework for efficiently solving this problem based on the subsequence methodology described in Rasmussen et al. (2011e).

D.3 Subsequence methodology

The idea of using subsequence generation to solve the airline crew pairing problem was first proposed in Rasmussen et al. (2011e). The fundamental idea with this approach is to cleverly limit the number of subsequent flights that can feasibly follow any flight $f \in \mathcal{F}$ and then dynamically introduce attractive so-called *subsequences* during the solution process. Formally stated, the subsequences of a particular flight $f \in \mathcal{F}$ are the set of pairs $(f, g) \in \mathcal{F}^2$, where $g \in \mathcal{F}$ is a feasible subsequent flight of f. Figure D.3 illustrates this concept. We denote the set of subsequences of any flight $f \in \mathcal{F}$ as $\mathcal{S}(f)$. The *subsequence count* of

any flight $f \in \mathcal{F}$ is then given as $|\mathcal{S}(f)|$ and the term *unique subsequence* refers to the situation in which the subsequence count of all flights $f \in \mathcal{F}$ is at most one. We denote the set of all subsequences as $\mathcal{S} = \bigcup_{f \in \mathcal{F}} \mathcal{S}(f)$, and $\mathcal{O}$ is used to refer to a set of optimal subsequences.

The premise is that by considering a limited subsequence set, one can reduce the complexity of the mathematical model and, if done in an intelligent manner, the approach should not reduce the quality of the solution. Due to the set partitioning constraints of the model, one can observe that an optimal integer solution must have unique subsequence. In what follows, we discuss how one first limits the number of subsequences and then how one identifies attractive subsequences to add to the problem.

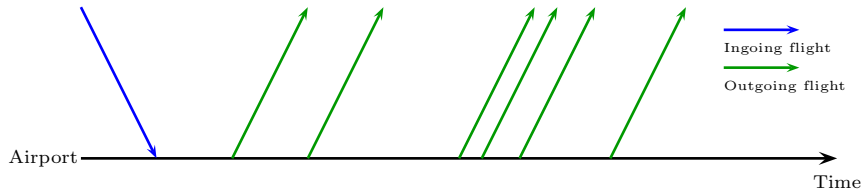


Figure D.3: *Subsequences.*

D.3.1 Subsequence limitation

Limiting the number of possible subsequences for each flight in the problem is beneficial from a computational perspective. Naturally it reduces the total number of pairings that must be considered. In addition to this, however, from a graph theoretical perspective problems with a limited subsequence set produce balanced or close to balanced constraint matrices. That is, the constraint matrix defines a polytope that has integral or near integral vertices (see Ryan and Falkner (1988) and Conforti et al. (2001) for details). Figure D.4 provides an example of a limited subsequence set for an incoming flight $f \in \mathcal{F}$. The set $\mathcal{L}(f) \subseteq \mathcal{S}(f)$ is used to denote the limited subsequence set for flight $f \in \mathcal{F}$. The set $\mathcal{L} = \bigcup_{f \in \mathcal{F}} \mathcal{L}(f)$ contains all limited subsequences for all flights. In the example $|\mathcal{L}(f)| = 2$ and $|\mathcal{S}(f)| = 6$. Being able to accurately identify which subsequences to include in $\mathcal{L}(f)$ for every flight $f \in \mathcal{F}$ is of crucial importance with this approach. Obviously, one must determine which subsequences to initially include. However, the limited subsequence set for each flight is a dynamic set in the sense that one can add attractive subsequences during the solution process. Here, attractive refers to a subsequence's ability to reduce the objective function of the *relaxed* master problem. The relaxed master problem is problem of the

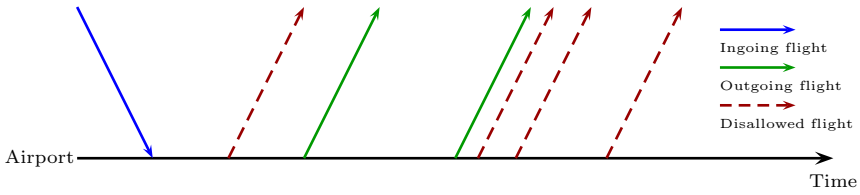


Figure D.4: *Limited subsequences.*

form of Model (D.1)–(D.8) that contains only a subset of all pairings and has no integral restrictions. The overall aim of the subsequence approach is to be able to identify a set of optimal subsequences without considering all possible subsequences in the pairing construction.

D.3.2 Subsequence generation

The purpose of subsequence generation is to use successive solutions to the relaxed master problem to identify an attractive subsequence (or possibly more than one) to add to the respective limited subsequence sets in order to enrich the set of possible pairings in the problem. Given the nature of the objective function it is unlikely that many *deep* subsequences will be in an optimal solution. A deep subsequence is one with a lengthy duration between the incoming and subsequent outbound flight. Therefore, we begin by defining *candidate* subsequence sets $\mathcal{C}(f)$ for each flight $f \in \mathcal{F}$. Again we define $\mathcal{C} = \bigcup_{f \in \mathcal{F}} \mathcal{C}(f)$. We have $\mathcal{L}(f) \subseteq \mathcal{C}(f) \subseteq \mathcal{S}(f)$ for all $f \in \mathcal{F}$ and $\mathcal{L} \subseteq \mathcal{C} \subseteq \mathcal{S}$. The relationship between these sets can be seen in Figure D.5. Ideally we would like $\mathcal{O} \subset \mathcal{C}$. Subsequence generation is an iterative procedure noticeably similar to column generation. At every iteration the relaxed master problem is solved and the dual vector is used to identify negative reduced cost pairings. Finding a negative reduced cost pairing entails solving a resource constrained shortest path problem on a connection network, see Clausen et al. (2010). Each node of this network corresponds to a particular flight $f \in \mathcal{F}$, while an arc between any two nodes indicates that it is possible for one flight to follow the other feasibly in a pairing. Unlike column generation, however, the aim is not to return a pairing from the pairing generation step, but rather to identify a good subsequence that should be contained in a pairing. For this reason we set up a number of sparse networks specifically designed to identify particular subsequences. The following classes of subsequences are important to recognise in crew pairing:

1. Follow-the-aircraft

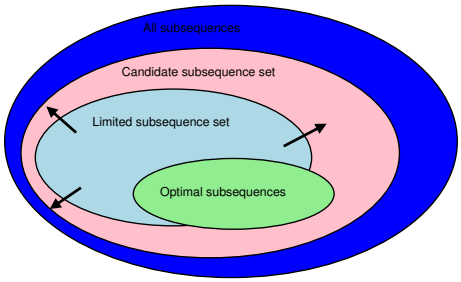


Figure D.5: *The relations between the set of all subsequences $\mathcal{S}$, the limited subsequence set $\mathcal{L}$, the candidate subsequence set $\mathcal{C}$, and an optimal subsequence set $\mathcal{O}$.*

2. Robust
3. Meal break
4. Overnight

For any flight $f \in \mathcal{F}$, the follow-the-aircraft subsequence is the one where the crew flies out on the same aircraft as they flew in with. This type of subsequence is very robust with respect to delays, because if crew is delayed on the ingoing flight that can still operate the outgoing flight. Expectations and data from Air New Zealand show that the majority of the optimal subsequences are of this type. A robust subsequence is one in which the time difference between the arrival of the incoming flight and departure of the subsequent flight exceeds the minimum sit time for the crew plus a certain degree of buffer time. Robust subsequences guard against delay propagation just as follow-the-aircraft subsequences. All crew members are entitled to meal breaks at certain time intervals. A meal break subsequence is one which provides crew with the possibility of taking a meal (either in the air or on the ground). Finally, since we typically construct pairings that are longer than three days in generation, it is inevitable that crew will have to overnight somewhere (base or non-base) during this time interval. An overnight subsequence is one in which the crew's idle time on the ground exceeds the minimum rest time.

The above classes of subsequences collectively give the candidate set of subsequences $\mathcal{C}$. We combine the follow-the-aircraft subsequences with each of the other three classes to give three subsets of the candidate set in order to construct networks that allow us to search specifically for robust, meal break, and overnight subsequences. The set $\mathcal{C}^1$ contains subsequence classes 1 and 2, $\mathcal{C}^2$ contains classes 1 and 3, and $\mathcal{C}^3$ consists of classes 1 and 4. We note for completeness that $\bigcup_{k=1}^{3} \mathcal{C}^k = \mathcal{C}$. To ensure that the shortest path solve on each of

the networks is quick, we restrict the number of subsequences for each flight in each of the subsets. That is $|\mathcal{C}^k(f)| \leq n^k$, where n^k is a small integer less than, say, five. The set $\mathcal{C}^1$ is used as the initial limited subsequence set $\mathcal{L}$.

While the pairing generation mirrors what is done in column generation (albeit on slightly different networks), unlike column generation the pairings are not returned directly to the relaxed master problem. Instead, all negative reduced cost columns are passed through a subsequence analysis phase. For each subsequence $s \in \mathcal{C}$ the analyser maintains four measures that are accumulated over all iterations and updated after analysis of the set of negative reduced cost columns returned by the pairing generators. These statistics are:

1. Count of columns containing s.

2. Count of different dual vectors that have produced columns containing s.

3. Sum of the reduced cost of columns that contain s.

4. Sum of the contribution from s to the negative reduced cost of columns containing s.

The measures are correlated, so a high rank in one measure could also give a high rank in some of the other measures. In fact, three out of the four measures utilise the dual information of the relaxed master problem. Utilising dual information to identify favourable flights is also the topic of Barnhart et al. (1995). Here the authors limit the search to deadhead flights only and incorporate it into a standard column generation procedure.

At each iteration some subsequences are identified as attractive based on these four measures and added to the set $\mathcal{L}$. Once a subsequence s has been identified as an attractive subsequence, an enumeration procedure is performed to generate a whole set of new pairings, which all contain the identified subsequence. All enumerated pairings are then added to the relaxed master problem. The enumeration returns feasible pairings in $\mathcal{L}$ containing s. The reason why a relatively large set of columns is added to the LP model, is, that whenever a subsequence is identified as attractive, it is believed that it is likely to end up in an optimal solution. That is, the optimal solution should contain one of the enumerated columns.

D.4 Integer programming framework

Given the preceding section on subsequence generation, one can obtain a high quality solution to the relaxed master problem as shown by Rasmussen et al. (2011e). Since only a subset of the subsequences are considered this approach cannot provide a certificate of optimality. In order to solve the airline crew pairing problem we need a solution that satisfies the integral restrictions of Model (D.1)–(D.8). In this section we present an integer programming framework, which utilises follow-on branching, to force the x_j variables to assume integer values. In Section D.4.1 we formalize the methodology of Section D.3.2 via a flow-chart, before introducing the idea of constraint branching in Section D.4.2. Finally, in Section D.4.3, we provide an overview of the complete integer programming approach.

D.4.1 Solving the LP relaxation

Figure D.6 provides a schematic view of how the subsequence generation procedure solves an instance of the relaxed master problem. To initialise the algorithm all possible pairings from the subsequence set $\mathcal{C}^1$ are enumerated and given to the LP solver to obtain an initial solution. The dual solution to this problem is then passed to the subsequence generation routine, which executes a series of pairing generators. Each pairing generator returns a set of negative reduced cost pairings which are then analysed in order to identify one or more attractive subsequences. Upon identifying such a subsequence an enumeration procedure is performed to generate all pairings that contain the specified subsequence. Again, the enumeration is done only with subsequences from the limited set $\mathcal{L}$. The relaxed master problem is then re-optimised with the additional set of pairings. One iterates in this fashion until one of the following situations occurs:

1. No significant improvement in the objective function for a specified number of iterations.

2. No negative reduced cost pairings are returned from the pairing generators.

It can be observed that the process is quite similar to column generation. However, with column generation, the dual solution is passed to the pairing generators and any negative reduced cost pairings are added directly to the relaxed master problem.

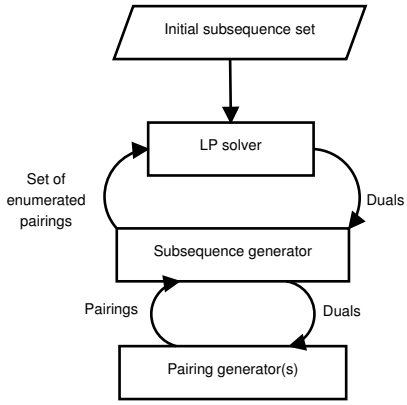


Figure D.6: *Subsequence generation.*

D.4.2 Follow-on branching

Fractional solutions to the relaxed master problem arise when two or more pairings compete to cover the same flight(s). Due to the set partitioning structure of the model, one knows that in any optimal solution at most one pairing can cover any flight. In most airline crew pairing applications the branching method of choice is the so-called *follow-on* branching rule. This rule is a variation of the *constraint branching* technique developed by Ryan and Foster (1981) and is also what is implemented in this paper. In follow-on branching one must identify two flights that are flown consecutively and contained in a pairing that is covered fractionally (i.e. at a value greater than zero, but strictly less than one) in a solution to the relaxed master problem. This branching strategy partitions the solution space into two disjoint subspaces (or branches). The first ensures that the two flights are flown consecutively, while the second ensures that they are covered by different pairings. Since one is branching on consecutive flights, this rule is particularly easy to incorporate in the pairing generators as it only requires the modification of arcs associated with two flights in the network. Furthermore, the follow-on branching concept is closely related to the notion of a subsequence—identifying two flights to be flown consecutively amounts to identifying a subsequence.

To identify the subsequence to branch on given a fractional solution to a relaxed master problem we simply find the subsequence that is covered fractionally at maximum value (i.e. at a value greater than zero, but strictly less than one) and create two new nodes to be solved, as outlined above. Imposition of a branch requires one to first remove those pairings that violate the branch from

the relaxed master problem. Here, we simply bound all such variables to zero. As we mentioned in Section D.2, we allow the partitioning flight constraints and base constraints to be violated, with an appropriate penalty. The artificial variables are never bounded to zero and ensure we always have a starting basis after this bounding step. By retaining the artificial variables, we do not need to implement a time consuming phase 1/phase 2 approach.

Through modifications to the pairing generators, any pairing that violates the branch is prevented from entering the problem. If the branch states that the subsequence should not be contained in any pairing (i.e. the two flights cannot be flown consecutively), then the corresponding arc is removed from any pairing generators it appears in. If, on the other hand, one is forcing a subsequence to be contained in a pairing, then one must ensure that the corresponding arc is contained in the solutions to the respective resource constrained shortest paths. Here, to enforce a particular subsequence, we remove all other *conflicting* subsequences from the relevant networks. This ensures that the desired subsequence is contained and prevents us from having to introduce a new resource in the shortest path solve. A conflicting subsequence is one in which either the inbound or outbound flight is different to that stated in the subsequence to branch on. Figure D.7 illustrates how a subsequence (consisting of flights one and two) is enforced. The arcs given in red are all subsequences that must be removed in

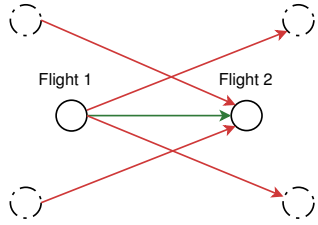


Figure D.7: *Forcing a subsequence.*

order to ensure that flight 1 and flight 2 are flown consecutively (or not flown at all) in a pairing.

D.4.3 Solving the integer programme

To produce a high quality solution to Model (D.1)–(D.8), we combine the follow-on branching strategy of the preceding section with the subsequence generation methodology of Section D.4.1 to implement a kind of branch-and-price algorithm. Branch-and-price is a well-known technique, which utilises column generation, for solving the crew pairing problem (see Barnhart et al. (1998) for de-

tails). Due to the fact that we identify, and add dynamically, new subsequences to the problem as we proceed, even during the branching phase of the approach, the integer programming framework we propose does not strictly adhere to traditional branch-and-bound principles. In particular, one cannot guarantee that a child node will have an objective function value that is at least that of its parent. We are prepared to make this sacrifice, since as it is hoped that by identifying good subsequences at the root node high quality integer solutions can be obtained quickly (without too much branching).

When solving nodes of the branch-and-bound tree we adopt a depth-first strategy, each time enforcing the identified subsequence, since this often produces a good integer solution quickly. Upon finding the first integer solution, however, we switch to a best-first search. That is, we evaluate the unexplored nodes in increasing order of their parent's objective value. Figure D.6, with two modifications, can be considered the solution procedure for any node. When initialising the algorithm all pairings that do not satisfy the branch to enforce must be removed. Furthermore, all networks must be modified to ensure only feasible pairings (i.e. all necessary branches are enforced) are generated. The branch-and-bound procedure terminates when all nodes have been evaluated or the incumbent integer solution is within a degree of tolerance of the best, unexplored node. While this integer programming approach does not provide valid lower bounds, the idea of subsequence generation is to provide a good integer solution quickly. In the computational results of Section D.5 we compare our integer solutions to the optimal solution of the relaxed master problem.

D.5 Computational results

In this section we analyse the performance of the proposed solution approach on 19 real-life data instances that were made available to us by Air New Zealand. The data sets are taken from Air New Zealand's domestic timetable. The domestic timetable does, however, also include destinations in Australia and the Pacific Islands. To perform the computational analysis we restrict the number of flights in each of the instances. This is done to ensure that they terminate in reasonable time. Table D.1 states the number of flights ($|\mathcal{F}|$) and the total number of base constraints for each instance ($|\mathcal{B}_1| + |\mathcal{B}_2|$).

There are a number of parameters one must determine when implementing the subsequence generation. These include the following:

1. Which criterion does one use to identify a subsequence to add to the problem?

	w08r01a	w08r01b	w08r01c	w08r01d	w08r01e	w08r02a	w08r02b	w08r02c	w08r02d	w08r02e	w08r03a	w08r03b	w08r03c	w08r03d	w08r03e	w08r04a	w08r04b	w08r04c	w08r04d
$\lvert\mathcal{F}\rvert$	450	430	430	370	400	380	400	320	450	350	320	420	420	450	320	400	400	350	320
$\lvert\mathcal{B}_1\rvert + \lvert\mathcal{B}_2\rvert$	5	5	5	5	4	5	5	5	5	4	5	5	5	5	4	5	5	5	5

Table D.1: *Characteristics for the test instances.*

2. How many subsequences should be contained in each of the sets $\mathcal{C}^1$, $\mathcal{C}^2$, and $\mathcal{C}^3$?

Based on the results of Rasmussen et al. (2011e), the selection strategy that is used to identify an entering subsequence is a round-robin procedure which loops over the four measures described in Section D.3.2. At each iteration the subsequence which has the highest score in the measure under consideration is added to the problem. Enumeration is then performed. Here, we test and compare the impact on solution time and quality by increasing the number of subsequences that can be contained in the sets $\mathcal{C}^1$, $\mathcal{C}^2$, and $\mathcal{C}^3$. In the first case we restrict the sets to contain at most three subsequences, while in the second this limit is set to four. The branching routine terminates when the incumbent integer solution is within 1% of the "best" unexplored node. We impose a time limit of 3600 seconds on the complete algorithm. The penalties associated with not covering a flight and violating a base constraint are 10^8 and 10^6, respectively. All tests are run on 2.67 GHz Intel Xeon X5550 CPUs with 23.5 GB of memory. The algorithm is implemented in C++ and compiled with g++ 4.4.0 on a Linux computer. LP relaxations are solved with the LP solver from MOSEK 6.0 using an academic license.

Table D.2 gives the results for the case in which each of the subsequence classes $\mathcal{C}^1$, $\mathcal{C}^2$, and $\mathcal{C}^3$ contains at most three subsequences. For each instance we state the instance name, the objective value of the best integer solution, and the time at which this solution was obtained. Furthermore, we provide an indication of the quality of this solution through a comparison with objective function value of the relaxed master problem (root LP) as well as the objective function value of the relaxed master problem obtained using a conventional column generation procedure. The column generation procedure has no restrictions on the number of subsequences each flight can have and is also given a time limit of 3600 seconds. For a fair comparison, we also hot start this procedure with an enumeration of pairings on the $\mathcal{C}^1$ subsequence class. On all instances the column generation procedure timed out and what is given in the table is the objective function value of the root node at termination (cg LP). We also provide the percentage gap between the objective value of our integer solution and the solution obtained using column generation, the number of uncovered flights in

our solution (UF), the number of nodes explored in the branching phase of the algorithm, and the time required for the algorithm to execute. The subsequence generation procedure terminates when the incumbent integer solution is within 1% of the "best" unexplored. That is, within 1% of objective value of the best node's parent.

One can observe from the table that the subsequence generation procedure provides, with a few exceptions, good quality integer solutions (within a few percent of the objective value obtained using column generation) given the one hour time limit. In all cases an integer solution is obtained within 11 minutes of computation time. The column generation procedure, on the other hand, does not converge within the same time frame. It is encouraging to see that for some instances (i.e. w08r02a and w08r04a) we are within 2% of the column generation approach extremely quickly. However, instances w08r01e, w08r02c, w08r03a, w08r03e, and w08r04d show that there is room for improvement in the subsequence identification phase of the algorithm The large percentage gaps can be explained by the fact that we have more uncovered flights than the column generation procedure. For example, for instance w08r03a we have twice as many uncovered flights. However, to put this in perspective, w08r03a is a flight schedule containing 320 flights and we uncover eight of them. Comparing the time at which the best integer solution was found with the time it took the algorithm to conclude, we note that all instances terminated upon finding the first integer solution. One can also see that in several cases the integer solution obtained has a better objective function value than the root node. As we mentioned in Section D.4.3, this is possible as subsequences are dynamically added during the branching phase of the algorithm.

Table D.3 gives the results for the case in which each of the subsequence classes $\mathcal{C}^1$, $\mathcal{C}^2$, and $\mathcal{C}^3$ contains at most four subsequences. While this table reinforces many of the conclusions from Table D.2, one can also observe that the integer solutions are slightly better than those obtained in Table D.2. Increasing the class sizes does, however, slow the method down. This can be explained by the following. A larger candidate set of subsequences creates larger networks for the pairing generators and in doing so creates more feasible pairings. As a result, the enumeration procedure not only takes longer, but there are also more pairings in the relaxed master problem making the optimisation slower. Interestingly, instance w08r04d is the only instance for which we uncover fewer flights by increasing the candidate subsequence set size. This could be a result of increased flexibility given the additional flights or a result of the subsequence generation taking a different path in the execution of the algorithm. That is, subsequences are identified in a different order, prompting a different sequence of events in the algorithm. Finally, the fact that in some cases we undercover more flights than would appear necessary would suggest that a more sophisticated process of including subsequences in the candidate set might be required.

instance	Best Integer		Statistics					
	obj	time (s)	root LP	cg LP	gap (%)	UF	nodes	time (s)
w08r01a	6.12041e+08	323.53	6.13042e+08	6.02042e+08	1.66	6	105	323.54
w08r01b	3.07034e+08	246.09	3.07598e+08	3.00034e+08	2.33	3	13	246.09
w08r01c	3.05040e+08	634.48	5.06037e+08	3.00031e+08	1.67	3	195	634.55
w08r01d	2.10030e+08	117.34	2.10785e+08	2.03031e+08	3.45	2	5	117.34
w08r01e	8.21038e+08	280.26	8.22034e+08	6.05033e+08	35.70	8	49	280.27
w08r02a	5.00023e+08	19.67	5.00023e+08	5.00021e+08	0.00	5	3	19.67
w08r02b	3.00023e+08	249.96	3.00023e+08	3.00023e+08	0.00	3	47	249.96
w08r02c	8.00019e+08	202.31	8.00521e+08	4.00019e+08	100.00	8	35	202.31
w08r02d	3.00029e+08	468.04	3.00030e+08	3.00025e+08	0.00	3	87	468.06
w08r02e	5.18034e+08	281.61	5.17605e+08	5.12029e+08	1.17	5	41	281.62
w08r03a	8.01024e+08	241.71	8.01023e+08	4.00020e+08	100.25	8	79	241.73
w08r03b	2.11028e+08	184.79	2.12027e+08	2.00034e+08	5.50	2	15	184.79
w08r03c	2.12030e+08	360.95	2.14029e+08	2.02035e+08	4.95	2	77	360.97
w08r03d	3.12030e+08	435.32	3.11531e+08	3.00038e+08	4.00	3	69	435.34
w08r03e	8.19022e+08	102.97	8.19023e+08	4.11028e+08	99.26	8	13	102.97
w08r04a	4.09027e+08	12.69	4.09027e+08	4.01031e+08	1.99	4	1	12.69
w08r04b	4.17025e+08	345.34	4.17026e+08	4.07032e+08	2.46	4	29	345.34
w08r04c	3.16025e+08	8.22	3.16025e+08	3.05033e+08	3.60	3	1	8.22
w08r04d	8.09021e+08	16.29	8.10021e+08	4.00023e+08	102.24	8	3	16.29

Table D.2: *Classes $\mathcal{C}^1$, $\mathcal{C}^2$, and $\mathcal{C}^3$ have at most three subsequences.*

instance	Best Integer		Statistics					
	obj	time (s)	root LP	cg LP	gap (%)	UF	nodes	time (s)
w08r01a	6.10039e+08	234.22	6.13038e+08	6.02042e+08	1.33	6	15	234.22
w08r01b	3.04036e+08	75.48	3.05202e+08	3.00034e+08	1.33	3	3	75.48
w08r01c	3.02037e+08	1107.07	3.04541e+08	3.00031e+08	0.67	3	25	1107.09
w08r01d	2.09031e+08	1371.26	2.08913e+08	2.03031e+08	2.96	2	115	1371.43
w08r01e	8.19041e+08	295.70	8.21035e+08	6.05033e+08	35.37	8	15	295.71
w08r02a	5.00022e+08	59.37	5.00022e+08	5.00021e+08	0.00	5	7	59.37
w08r02b	3.00023e+08	307.40	3.00023e+08	3.00023e+08	0.00	3	23	307.40
w08r02c	8.00018e+08	48.34	8.00018e+08	4.00019e+08	100.00	8	5	48.35
w08r02d	3.00026e+08	1207.20	3.00026e+08	3.00025e+08	0.00	3	33	1207.22
w08r02e	5.17030e+08	699.73	5.18198e+08	5.12029e+08	0.98	5	23	699.74
w08r03a	8.00021e+08	14.55	8.00021e+08	4.00020e+08	100.00	8	1	14.55
w08r03b	2.06033e+08	1343.80	2.09529e+08	2.00034e+08	3.00	2	171	1344.01
w08r03c	2.08033e+08	1080.62	2.10031e+08	2.02035e+08	2.97	2	71	1080.66
w08r03d	3.11031e+08	19.50	3.11031e+08	3.00038e+08	3.66	3	1	19.50
w08r03e	8.15024e+08	574.02	8.16523e+08	4.11028e+08	98.29	8	41	574.05
w08r04a	4.07028e+08	1333.81	4.07196e+08	4.01031e+08	1.50	4	147	1349.18
w08r04b	4.14028e+08	183.32	4.14029e+08	4.07032e+08	1.72	4	11	183.32
w08r04c	3.13027e+08	12.62	3.13027e+08	3.05033e+08	2.62	3	1	12.62
w08r04d	7.05023e+08	159.30	8.08021e+08	4.00023e+08	76.25	7	15	159.31

Table D.3: *Classes $\mathcal{C}^1$, $\mathcal{C}^2$, and $\mathcal{C}^3$ have at most four subsequences.*

D.6 Conclusion and future work

In this paper we have described a new integer programming framework for solving the well-known airline crew pairing problem. At the core of the methodology is subsequence generation. This approach limits the number of subsequences in the problem and dynamically adds attractive subsequences as needed. A follow-on branching strategy is described for obtaining integer solutions. Encouraging results are presented for 19 real-life instances supplied by Air New Zealand. In comparison to a column generation procedure that fails to converge to the optimal solution of the relaxed master problem within in an hour of computation time, the methodology presented in this paper produces good quality integer solutions well within the same time limit. This indicates the method could potentially be a viable alternative to the conventional column generation approach to this problem.

While the results are encouraging, they also suggest that improvements are necessary. For instance, as it is now, the candidate set of subsequences $\mathcal{C}$ is a static set. If this does not contain the optimal subsequences (or at least a close to optimal set), then it is unlikely the method will do well. One promising improvement would be to be make this set dynamic so that one could add new candidate subsequences during the solution process, or even remove some unpromising ones. In this way one can keep a good, small set of subsequences. Furthermore, one can also improve the subsequence identification step. This is the core process in the approach and dictates how many pairings will be added to the problem. Improvements here will positively impact the run time of the approach.

Acknowledgements: The authors would like to thank Paul Keating from Air New Zealand for providing the data instances used in this paper.

References

AhmadBeygi, S., A. Cohn, and M. Weir (2009). "An integer programming approach to generating airline crew pairings". *Computers & Operations Research* 36.4, pp. 1284–1298.

Andersson, E., E. Housos, N. Kohl, and D. Wedelin (1998). "Crew Pairing Optimization". In: *Operations Research in the Airline Industry*. Ed. by G. Yu. Kluwer Academic Publishers. Chap. 8, pp. 228–258.

Barnhart, C., L. Hatay, and E. L. Johnson (1995). "Deadhead selection for the long-haul crew pairing problem". *Operations Research* 43.3, pp. 491–499.

Barnhart, C., E. Johnson, G. Nemhauser, M. Savelsbergh, and P. Vance (1998). "Branch-and-Price: Column Generation For Solving Huge Integer Programs". *Operations Research* 46.3, pp. 316–329.

Barnhart, C., A. M. Cohn, E. J. Johnson, D. Klabjan, G. L. Nemhauser, and P. H. Vance (2003). "Airline Crew Scheduling". In: *Handbook of Transportation Science*. Ed. by R. W. Hall. 2nd ed. Norwell: Kluwer Academic Publishers. Chap. 14, pp. 517–560.

Butchers, E. R., P. R. Day, A. P. Goldie, S. Miller, J. A. Meyer, D. M. Ryan, A. C. Scott, and C. A. Wallace (2001). "Optimized crew scheduling at Air New Zealand". *Interfaces* 31.1, pp. 30–56.

Clausen, J., A. Larsen, J. Larsen, and N. J. Rezanova (2010). "Disruption management in the airline industry—Concepts, models and methods". *Computers & Operations Research* 37.5. Disruption Management, pp. 809–821.

Conforti, M., G. Cornuéjols, A. Kapoor, and K. Vuskovic (2001). "Perfect, ideal and balanced matrices". *European Journal of Operational Research* 133.3, pp. 455–461.

Desaulniers, G., J. Desrosiers, M. Gamache, and F. Soumis (1998). "Crew Scheduling in Air Transportation". In: *Fleet Management and Logistics*. Ed. by T. G. Crainic and G. Laporte. Boston: Kluwer Academic Publishers, pp. 169–185.

Gopalakrishnan, B. and E. L. Johnson (2005). "Airline Crew Scheduling: State-of-the-Art". *Annals of Operations Research* 140.1, pp. 305–337.

Graves, G. W., R. D. McBride, I. Gershkoff, D. Anderson, and D. Mahidhara (1993). "Flight Crew Scheduling". *Management Science* 39.6, pp. 736 –745.

Lavoie, S., M. Minoux, and E. Odier (1988). "A new approach for crew pairing problems by column generation with an application to air transportation". *European Journal of Operational Research* 35.1, pp. 45 –58.

Rasmussen, M. S., R. M. Lusby, D. M. Ryan, and J. Larsen (2011e). *Subsequence Generation for the Airline Crew Pairing Problem*. Tech. rep. Department of Management Engineering, Technical University of Denmark.

Ryan, D. M. and J. C. Falkner (1988). "On the integer properties of scheduling set partitioning models". *European Journal of Operational Research* 35.3, pp. 442 –456.

Ryan, D. M. and B. Foster (1981). "An integer programming approach to scheduling". *Computer Scheduling of Public Transport. Urban Passenger Vehicle and Crew Scheduling. Proceedings of an International Workshop*, pp. 269–280.

Saddoune, M., G. Desaulniers, I. Elhallaoui, and F. Soumis (2011). "Integrated airline crew scheduling: A bi-dynamic constraint aggregation method using neighborhoods". *European Journal of Operational Research* 212.3, pp. 445–454.

Vance, P. H., C. Barnhart, E. L. Johnson, and G. L. Nemhauser (1997). "Airline Crew Scheduling: A New Formulation and Decomposition Algorithm". *Operations Research* 45.2, pp. 188–200.

Wedelin, D. (1995). "An algorithm for large scale 0-1 integer programming with application to airline crew scheduling". *Annals of Operations Research* 57, pp. 283–301.