Rostering and Task Scheduling

Applications in Manpower Planning

Dohn, Anders Høeg; Larsen, Jesper; Clausen, Jens

Publication date:
2010

Document Version
Publisher's PDF, also known as Version of record

Rostering and Task Scheduling

– Applications in Manpower Planning

PhD thesis 9.2010

DTU Management Engineering

Anders Dohn

August 2010

DTU Management Engineering
Department of Management Engineering

Rostering and Task Scheduling

Applications in Manpower Planning

Anders Dohn

Kgs. Lyngby, 2010

Anders Dohn
Rostering and Task Scheduling - Applications in Manpower Planning
(Vagtplanlægning og opgaveallokering - Anvendelser i mandskabsplanlægning)
PhD Thesis, 2010

ISBN 978-87-90855-84-0

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

Resumé (Danish Summary)

I et moderne samfund er mandskab ofte en både begrænset og bekostelig ressource. Kvalificerede medarbejdere er efterspurgte og udgør en stor del af de samlede omkostninger i mange virksomheder. For at minimere omkostninger og for at maksimere anvendelsen af det tilgængelige personale er det nødvendigt med avancerede planlægningsværktøjer. Disse værktøjer gør det samtidig muligt at sikre en høj tilfredshed blandt medarbejderne, da deres ønsker kan inkluderes på struktureret vis. Denne ph.d.-afhandling er udarbejdet med baggrund i operationsanalyse, som bl.a. beskæftiger sig med udvikling af algoritmer til automatiseret planlægning. Med en struktureret tilgang skabes forbedrede løsninger til komplekse praktiske optimeringsproblemer.

Afhandlingen indeholder seks videnskabelige artikler og en sammenfatning af de vigtigste bidrag og konklusioner fra disse. En række industrielle problemstillinger indenfor mandskabsplanlægning præsenteres med særlig fokus på generaliseret vagtplanlægning og på opgaveallokering med tidslige afhængigheder mellem opgaver. De betragtede problemstillinger stammer fra sundhedssektoren, lufthavne, transportvirksomheder og produktionsvirksomheder. De vigtigste bidrag fra afhandlingen inkluderer udviklingen af en alsidig tilgang til generaliseret vagtplanlægning. Endvidere præsenteres adskillige udvidelser af vagtplanlægningsproblemer. For opgaveallokering præsenteres en generel modellering af tidslige afhængigheder, der inkluderes i en såkaldt søjlegenereringsmetode. Søjlegenerering er en iterativ metode, der baseret på lineær programmering, kan give løsninger, som er beviseligt optimale. Denne metode anvendes i afhandlingen på størstedelen af de praktiske problemer med lovende resultater. Endelig præsenteres en ny metode til kranstyring og lageroptimering i stålproduktion. Også her er resultaterne yderst lovende.

Summary

In a modern society, manpower can be both a scarce and an expensive resource. Skilled personnel is usually in high demand and accounts for a significant part of total expenses in many companies. When the work is divided in shifts, a roster is compiled to allocate these to the employees. The rostering process is non-trivial and especially when service is required around the clock, *rostering* may involve considerable effort from a designated planner. Therefore, in order to minimize costs and overstaffing, to maximize the utilization of available staff, and to ensure a high level of satisfaction among the employees, sophisticated scheduling methods are required. When approaching the day of operation, the detail level of the planning becomes finer. With a given allocation of shifts to employees, the focus is turned to *tasks scheduling* within those shifts. The objective is to assign as much work as possible to the available staff, while respecting various requirements and rules and while including possible transportation time between tasks.

This thesis presents a number of industrial applications in rostering and task scheduling. The applications exist within various contexts in health care, the aviation industry, transportation, and production. The focus regarding rostering is both on a generalized rostering problem, which captures most realistic settings, and also on a more specific case, where particular issues and extensions are examined. In task scheduling, the focus is restricted to scheduling problems with temporal dependencies between tasks. However, these problems appear in various contexts and with different properties. A group of the problems considered are related to vehicle routing problems, where transportation and time windows are important factors that must be accounted for.

Mathematical and logic-based models are presented for the problems considered. Novel components are added to existing models and the modeling decisions are justified. In one case, the model is solved by a simple, but efficient greedy construction heuristic. In the remaining cases, column generation is applied. Column generation is an iterative exact solution method based on the theory of linear programming and is capable of providing provably optimal solutions. In some of the applications, the approach is modified to provide feasible solutions of high-quality in less time. The exceptional solution quality of column generation is maintained, but the certificate of optimality is compromised.

The contribution of this thesis is partly in the introduction, extension, and refinement of mathematical models for practical planning problems. Further, the contribution is in the proposed solution methods, which produce applicable and superior results to a range of realistic manpower planning problems. The contributions are presented in six scientific papers, which are compiled in the thesis. These include the development of a versatile approach to generalized rostering, building on an idea of compile-time customization. Several extensions of practical rostering problems are presented. For task scheduling, a general modeling of temporal dependencies is introduced and included in the methodology of column generation. The approach is applied to several practical problems with promising results. Lastly, a novel approach to crane scheduling with superior results is presented.

Preface

This dissertation is submitted to DTU Management Engineering, Technical University of Denmark in partial fulfillment of the requirements for acquiring the PhD degree. The work has been supervised by Professor Jens Clausen and Associate Professor Jesper Larsen.

The thesis consists of an introduction to the project and a collection of six research papers prepared during the period from December 2006 to April 2010. Generally, American spelling rules are used in this thesis, but a few of the research papers use British spelling, due to preferences of co-authors.

Kgs. Lyngby, Denmark, May 2010

Anders Dohn

Scientific papers composed

The following papers have been composed as a part of this PhD. The papers have been divided into two categories, where the first consists of the most significant contributions, which are also found in the appendices of this thesis. The remaining papers and reports have been produced during the work on the projects. They may contain information that partially overlaps with what is found in the papers of the first group.

Papers in appendices

- **Paper A:** Anders Dohn and Andrew Mason (2010). "A Nested Column Generation Based Approach to the Generalized Rostering Problem using Compile-time Customization". In: *INFORMS Journal on Computing* (Submitted)
- **Paper B:** Richard Lusby, Anders Dohn, Troels Martin Range, and Jesper Larsen (2010). "An Integrated Approach to the Ground Crew Rostering Problem with Work Patterns". In: *Journal of the Operational Research Society* (Submitted)
- **Paper C:** Anders Dohn, Esben Kolind, and Jens Clausen (2009*b*). "The manpower allocation problem with time windows and job-teaming constraints: A branch-and-price approach". In: *Computers and Operations Research* 36.4, pp. 1145–1157.
- **Paper D:** Matias Sevel Rasmussen, Tor Justesen, Anders Dohn, and Jesper Larsen (2010). "The Home Care Crew Scheduling Problem: Preference-Based Visit Clustering and Temporal Dependencies". In: *European Journal of Operational Research* (Submitted)
- **Paper E:** Anders Dohn, Matias Sevel Rasmussen, and Jesper Larsen (2010*c*). "The Vehicle Routing Problem with Time Windows and Temporal Dependencies". In: *Networks* (Conditionally accepted)
- **Paper F:** Anders Dohn and Jens Clausen (2010). "Optimising the Slab Yard Planning and Crane Scheduling Problem using a two-stage heuristic". In: *International Journal of Production Research* 48.15, pp. 4585–4608

Other papers, reports, and abstracts

- **Journal paper:** Anders Dohn, Matias Sevel Rasmussen, Tor Justesen, and Jesper Larsen (2008*b*). "The Home Care Crew Scheduling Problem". In: *ORbit* 13, pp. 19–23

- **Journal paper:** Anders Dohn and Esben Kolind (2009). "A Practical Branch and Price Approach to the Crew Scheduling Problem with Time Windows". In: *ORbit* 14, pp. 23–27

- **Conference paper (ICAPS'07):** Anders Dohn (2007). "Optimizing the Steel Plate Storage Yard Crane Scheduling Problem Using a Two Stage Planning/Scheduling Approach". In: *ICAPS 2007 - Doctoral Consortium*

- **Conference paper (ICAPS'07):** Anders Dohn, Esben Kolind, and Jens Clausen (2007*a*). "The Manpower Allocation Problem with Time Windows and Job-Teaming Constraints". In: *ICAPS 2007 - Proceedings, Seventeenth International Conference on Automated Planning and Scheduling*, pp. 120–127

- **Conference abstract (NOS'07):** Anders Dohn, Esben Kolind, and Jens Clausen (2007*b*). "The Manpower Allocation Problem with Time Windows and Job-Teaming Constraints". In: *Nordic Optimization Symposium*

- **Conference abstract (MetMat'08):** Anders Dohn and Jens Clausen (2008*a*). "A Two-stage Planning/Scheduling Model". In: *Workshop: MetMat*

- **Conference abstract (CG'08):** Anders Dohn, Matias Sevel Rasmussen, and Jesper Larsen (2008*a*). "Manpower Routing and Scheduling with Temporal Dependencies Between Tasks". In: *International Workshop on Column Generation*

- **Conference abstract (IFORS'08):** Anders Dohn and Jens Clausen (2008*d*). "Optimizing the Steel Slab Yard Crane Scheduling Problem Using a Two Stage Planning/Scheduling Approach". In: *International Federation of Operational Research Societies Conference*

- **Conference paper (ICAOR'08):** Anders Dohn, Matias Sevel Rasmussen, Tor Justesen, and Jesper Larsen (2008*c*). "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 (ORSNZ'08):** Anders Dohn and Esben Kolind (2008). "Optimizing Manpower Allocation for Ground Handling Tasks in Airports using Column Generation". In: *ORSNZ'08 - Proceedings - 43rd*

Annual Conference of the Operational Research Society of New Zealand, pp. 2–11

- **Conference paper (ORSNZ'09):** Andrew J. Mason, David Ryan, and Anders Dohn (2009). "Customised Column Generation for Rostering Problems: Using Compile-time Customisation to create a Flexible C++ Engine for Staff Rostering". In: *ORSNZ'09 - Proceedings - 44rd Annual Conference of the Operational Research Society of New Zealand*

- **Technical report:** Anders Dohn, Andrew Mason, and David Ryan (2010*a*). *A Generic Solution Approach to Nurse Rostering.* Tech. rep. Department of Management Engineering, Technical University of Denmark, Kgs. Lyngby, Denmark

- **Technical report:** Anders Dohn, Esben Kolind, and Jens Clausen (2007*c*). *The Manpower Allocation Problem with Time Windows and Job-Teaming Constraints: A Branch-and-Price Approach.* Tech. rep. Richard Petersens Plads, Building 321, DK-2800 Kgs. Lyngby: Informatics and Mathematical Modelling, Technical University of Denmark, DTU

- **Technical report:** Anders Dohn, Matias Sevel Rasmussen, and Jesper Larsen (2009*a*). *Technical Report: The Vehicle Routing Problem with Time Windows and Temporal Dependencies.* Tech. rep. Department of Management Engineering, Technical University of Denmark, Kgs. Lyngby, Denmark

- **Technical report:** Anders Dohn and Jens Clausen (2008*b*). *Optimizing the Slab Yard Planning and Crane Scheduling Problem using a Two-Stage Approach.* Tech. rep. DTU Management Engineering, Technical University of Denmark

Projects supervised

- **Master's Thesis:** Tor Justesen, Matias Sevel Rasmussen (2008). "The Home Care Crew Scheduling Problem", Department of Informatics and Mathematical Modeling, Technical University of Denmark.
- **Master's Thesis:** Thomas Vermehren Lins, Jonas Salee Vinther Jeppesen (2010). "Manpower planning for Air Traffic Controllers", Department of Management Engineering, Technical University of Denmark.
- **Master's Thesis:** Brian Ravn, Helene Martine Overø (2008). "Integrated Crew Scheduling for Airlines", Department of Informatics and Mathematical Modeling, Technical University of Denmark.
- **Bachelor Thesis:** Thomas Vermehren Lins, Jonas Salee Vinther Jeppesen (2008). "Skedulering af male-robotter", Department of Management Engineering, Technical University of Denmark.
- **Bachelor Thesis:** Julie Louise Munk Vejborg (2008). "The Crane Scheduling Problem: Heuristic approach", Department of Management Engineering, Technical University of Denmark.
- **Individual Course:** Henrik Alsing Pedersen (2010). "Optimizing shifts for Ground Staff", Department of Management Engineering, Technical University of Denmark.

Acknowledgements

First of all, I would like to thank my two supervisors, Professor Jens Clausen and Associate Professor Jesper Larsen for their guidance throughout the three years. I thank Jens for always believing in me, for being a constant source of inspiration - all the way through my education, and for nominating me for the EliteForsk Travel Grant. I thank Jesper for his (mostly) positive management and for extensive and patient support. I am thankful for his constant willingness to let me participate in the many interesting projects that emerged during the last three years. Also, I am thankful for getting the chance to co-supervise numerous student projects run by my two supervisors. I thank everybody in the Section of Operations Research for three enjoyable years.

I thank Professor David Ryan for inviting me to New Zealand and for his exceptional hospitality during the stay. We enjoyed all the visits to the house in Remuera, the bach on Waiheke, and the lodge in Whakapapa. We were very honored by the time he and his wife, Ruth, took to ensure that our stay in New Zealand was as joyful as possible. Further, I thank everybody at the University of Auckland for making the trip to New Zealand an unforgettable experience. In particular, I thank Senior Lecturer Andrew Mason for his commitment to my research and for his great enthusiasm that was characteristic for all the work, we did together. Furthermore, I am thankful to the Danish Agency for Technology and Innovation for supporting my stay in New Zealand through the EliteForsk Travel Grant.

I am thankful to all my co-authors for their contributions to individual projects. Their contributions were essential to the existence of this thesis. In particular, I thank Esben Kolind and Richard Lusby for proof reading the thesis. Special thanks to Richard for letting us borrow his car, while we lived in New Zealand.

Finally, I would like to thank my fiancée, Henriette, for her unlimited support and for always being there, when I needed her. I thank her for quitting her job to follow me to New Zealand and for always being eager to travel the world with me.

Contents

Part I

Background and synopsis

Chapter 1

Introduction

The optimization of complex planning problems is the main focus of the science of operations research. Using mathematical modeling, simulation, and computer based optimization, operations research practitioners increase the quality of solutions to practical planning and scheduling problems. Increased solution quality is a result of a structured approach towards practical problems combined with well founded optimization theory. This thesis is concerned with a subfield of operations research focused on manpower planning. Manpower planning is an area of constantly increasing importance in an industrialized and knowledge intensive society.

There is a large potential gain in applying optimization theory to practical manpower planning problems. E.g. Abbink et al. (2005) report annual savings of USD 4.8 million for a crew scheduling application at Netherlands railways. Likewise, Butchers et al. (2001) state that their system has saved Air New Zealand NZD 15.7 million (USD 11.2 million) per year. British Telecommunications implemented an automated workforce allocation system that according to Lesaint et al. (2000) introduced annual savings of USD 150 million. There are many other similar success stories in the area of manpower planning and scheduling.

When tasks are fixed to a specific time or in other ways restricted in time, the scheduling problem becomes nontrivial, and a structured approach to task allocation is profitable. Tasks may occur around the clock and the rostering problem in turn becomes nontrivial, as one needs to consider a range of laws and union rules on rest hours between shifts, maximum number of consecutive working days, etc. Rostering and task scheduling problems are similar in many aspects, as is apparent from the work presented in this thesis. A simplistic definition of the two types of problems is:

Rostering:	Assigning shifts to employees.
Task Scheduling:	Assigning tasks to employees within shifts.

As evident from the above, the main difference is the level of detail considered in the two types of problems. In rostering, the scheduling horizon (the roster period) is typically a month or more and shifts of several hours are considered. Usually, at most one shift is allocated per employee per day and the rostering problem consists of distributing the shifts between the employees over the days of the roster period. In task scheduling problems, the scheduling horizon is typically 24 hours or less and each employee is already assigned to a specific shift. Within this shift, a number of tasks can be allocated and the aim of task scheduling problems is to use the available manpower as efficiently as possible.

In practice, many rostering and task scheduling problems have traditionally been solved and verified manually. The motivation for using a decision support system, based on the theory of operations research, is a potentially increased utilization of the available manpower. At the same time, a decision support system will be able to incorporate other desirable features and produce solutions, which are superior in several aspects. Such features may include minimizing total expenses, maximizing efficiency, increasing robustness, maximizing employee satisfaction, and ensuring fairness between employees. The idea of decision support is to provide superior solution suggestions, while also saving time for the planner. Especially in disruption management, readily available solutions are essential.

In this thesis, we consider a number of specific manpower planning problems, which can all be characterized as rostering or task scheduling problems. We present problems from various contexts and explain how these are associated. As well as being related to each other, the practical problems are also related to established problems from the literature. The first of the specific cases considered is from the health care sector, namely the nurse rostering problem. Nurse rostering is related to airline ground crew rostering, where we describe another application. This is followed by another interesting manpower planning problem. The problem is concerned with the allocation of airplane cleaning tasks to mobile cleaning crews in some of Europe's major airports. A closely related problem, from the health sector, is the routing of home care personnel in Danish municipalities. The next problem described is the vehicle routing problem with time windows and temporal dependencies. It represents a context-free generalization of the routing problems encountered for ground crew in airports and for the home care crew. A fairly related problem, with some interesting similarities to the other applications, is the crane scheduling problem in steel production, where daily work plans for crane operators are generated.

To solve practical problems like the ones listed above, we have a wide variety of methods available. We expect the reader of the thesis to be familiar with the basic terminology and methodology of *operations research*, *linear programming*, and *integer programming*.

An intuitive idea, which is particularly useful for very hard problems, is that of greedy heuristics. Greedy heuristics to some extent imitate the approach that a manual planner is using. By utilizing the computational power of modern computers, we may be able to arrive at superior solutions in less time. Greedy heuristics are often extended to metaheuristics, which are able to search through a greater diversity of potential solutions. A few greedy heuristics will be described in detail later, whereas metaheuristics are not part of the focus of this thesis.

A different approach to solving the manpower planning problems is to describe the problems by mathematical models and thereafter apply mathematically based optimization methods to the models, which in our case will be integer programming models. A straight forward approach is to use a standard solver directly on the model, but the complexity of most problems call for sophisticated customized approaches. The Dantzig-Wolfe decomposition principle has formerly been applied to manpower planning problems and other related problems with great success. Therefore, much of the effort in this thesis has been focused on the use of column generation in Dantzig-Wolfe decomposed models.

The aim of this thesis is not to capture all existing rostering and task scheduling problems in a unified analysis. It is an exposition of certain practical problems along with their suggested models and solution methods. The choice of problems has been based on the selection of interesting cases existing with industrial partners. However, as will be apparent, the cases have many characteristics that relate them and justify their inclusion in the collection here.

1.1 Important terms

In the following, we establish some basic terminology which is used throughout the thesis. The individual papers may use variations of the terms, but will in general follow the definitions given here.

Roster-line A line of shifts for a single employee for the full time horizon.

Roster A schedule/plan for all employees for the full time horizon. A set of roster-lines, one for each employee, make up the roster.

Rostering The process of making a roster of shift assignments to employees.

Work-line A line of tasks for a single employee within a shift. This may also be referred to as a *route* in vehicle routing.

Schedule The given task allocation to employees and time specifications of all tasks, i.e. the collection of work-lines for all employees.

Task scheduling The process of scheduling tasks. The scheduling of tasks includes the allocation of tasks to employees.

Workload A measure sometimes used in rostering to specify the expected amount of work in a given time interval.

Demand / Shift demand / Cover demand / Requested cover The number of people needed in a given time interval, for a given duty, or on a given shift. The demands may be explicitly based on a forecasted workload.

Cover / Shift Cover The actual number of people working in a specific time interval.

Employee / Worker / Worker-group / Team These terms are used interchangeably to refer to the units for which a roster is made or tasks are scheduled. As the terms indicate, the unit may consist of several people, which, however, does not affect the structure of the model.

Temporal dependency A constraint expressing a relationship between two tasks, where the scheduled time of one task may restrict when the other task can be scheduled.

Attributes Attributes are defined for roster-lines and components of a roster-line. Attributes are used to count the number of paid hours, days on, weekends on, etc. Rules are defined on the values of the attributes.

1.2 Thesis structure

This thesis consists of two major parts. Part I is a digest of the findings of the thesis. A foundation for the work in this thesis is established and the most important contributions and conclusions are summarized and put into perspective. Part II consists of six individual scientific papers, which have been or will be published in international scientific journals.

Part I is divided into several chapters. In Chapter 2, rostering and task scheduling is described in broad terms and using generic models. The models are not related to particular practical instances, but instead give a context-free description of the structures of the problems and allow for a comparison to well established problems from the literature. The models are concretized for the problems at hand in Chapter 3. The emphasis of this chapter is on a detailed description of the problems as they exist in practice. The chapter also includes a comparison of the problems. In the scientific papers of the second part of the thesis, the reader is assumed to be familiar with common optimization methods. Therefore, the purpose of Chapter 4 is to describe the methods in more detail, to establish the foundation for the individual projects. The contents of the individual papers are summarized in Chapter 5. As the applications were introduced earlier, the main focus of this chapter is on solution approaches, results, and conclusions. The main conclusions of the thesis are found in Chapter 6.

Part II consists of the six scientific papers, which constitute the major part of the work in this thesis. The papers appear in their most recent version, which is either the version submitted or the post-print version. The papers have been adjusted to the layout of this thesis and may therefore appear slightly different from the published version.

Readers who are interested in the full thesis and all included papers, should start with the synopsis and subsequently read the individual scientific papers in the order in which they appear. Readers with a thorough knowledge of operations research may skip Chapter 4. Readers with particular interest in one of the areas considered in the thesis may just read the selected papers of Part II, possibly supplemented by the corresponding descriptions in Chapter 3. One may also refer to Chapter 5 to select particular papers of interests.

CHAPTER 2

Modeling

In this thesis, various practical planning problems are considered. Many of these problems are inherently related by the practical context in which they arise. In the following, we will establish a common model for these problems, to show that they are also closely related theoretically. The models are more general than those of the individual papers and are hence less suitable as starting points for a solution method. Instead, the models are intended to capture all of the manpower planning problems of the thesis, to allow for a common understanding of these. Furthermore, we will describe variants and related prototype problems from the literature to establish the theoretical context in which we are working.

2.1 Rostering

Rostering is the problem of allocating shifts to employees to cover a prespecified workload. There are various ways of modeling this. In the following, a generalized set partitioning model will be used. Given is a set of *employees* $\mathcal{E}$, which is divided into *groups* $\mathcal{G}$. The size of group $\mathcal{G}$ is m_g. In many rostering problems, employees are uniquely defined and the model has one employee in each group (the notion of groups can hence be disregarded). A set of *demands / work requests*, $\mathcal{D}$, is defined. Demands specify a requested number of people, b_d, for a particular duty, where μ_d^- and μ_d^+ quantify the *under-coverage* or *over-coverage*, respectively. Under-coverage refers to a shortage compared to the requested number of employees. Similarly, over-coverage refers to an excess in the number of employees. Under-coverage usually introduces larger costs than over-coverage. Tasks may be defined within shifts or may span multiple shifts and may have skill requirements. Demands do in practice often coincide with shifts. Let the decision variable $x_{gd} \in \mathbb{Z}_+$ count the number of employees of group $g \in \mathcal{G}$ that contribute to cover the demand $d \in \mathcal{D}$. $\mathcal{X}_g$ defines for each group $g \in \mathcal{G}$ all feasible combinations of shifts that the group may be given. A generic rostering model can then be formulated:

$$\begin{aligned}
\min \quad & f(x, \mu_i^-, \mu_i^+) & & \quad (2.1)\\
\sum_{g \in \mathcal{G}} \sum_{d \in \mathcal{D}} x_{gd} + \mu_d^- - \mu_d^+ &= b_d & \forall d \in \mathcal{D} & \quad (2.2)\\
x_{g\cdot} \in \mathcal{X}_g &\subseteq \mathbb{Z}_+^{|\mathcal{D}|} & \forall g \in \mathcal{G} & \quad (2.3)\\
\mu_d^- \in \mathbb{R}_+, \mu_d^+ &\in \mathbb{R}_+ & \forall d \in \mathcal{D} & \quad (2.4)
\end{aligned}$$

The objective (2.1) is a function of all decision variables. Constraints (2.2) ensure that all demands are satisfied or the corresponding slack and surplus variables are set appropriately. Constraints (2.3) define the roster-line feasibility for each group. The constraints are not easily described in a mathematical model. The constraints are particularly complex in the current format, where groups of employees are considered. Often, the roster-line feasibility is decomposed into a description of feasibility for a single roster-line and the rostering problem is decomposed into two parts, where one part generates feasible roster-lines and the other part combines these into a feasible roster. *Attribute-based* descriptions have shown to be especially well suited for definition of feasibility and cost in roster-lines. Attributes are used to count the number of paid hours, days on, weekends on, etc. Rules are defined on the values of the attributes. Constraints (2.4) set the domains of the remaining decision variables.

We will not go into more detail with the rostering model here. Paper A describes a generic approach to rostering and gives details on the modeling as well.

2.2 Task scheduling

All of the task scheduling problems considered are basically asking the question:

"In an optimal schedule: Who does what, when?"

It is possible to formulate a general mathematical model which expresses exactly this question. Given is a workforce, i.e. a set of *workers* to which tasks must be allocated. Workers may, according to the context, refer to employees, crews, vehicles, operators, etc. $\mathcal{K}$ denotes the set of workers. $\mathcal{I}$ is the set of tasks. Finally, $\mathcal{T}$ is a set of decision times, i.e. a discretization of time, where the elements represent all points in time, when tasks can be scheduled. The magnitude of the discretization may vary from seconds to hours.

With the sets given, a binary decision variable $x_{kit}, k \in \mathcal{K}, i \in \mathcal{I}, t \in \mathcal{T}$ is introduced, where $x_{kit} = 1$ if task i is allocated to worker k at time t. $x_{kit} = 0$ otherwise. The following generic model captures all the practical problems that we have worked with. We refer to it as the *compact formulation* of task scheduling problems:

$$\min \quad f(x) \tag{2.5}$$
$$\text{s.t.} \quad x \in \mathcal{X} \subseteq \mathbb{B}^{|\mathcal{K}||\mathcal{I}||\mathcal{T}|} \tag{2.6}$$

The objective function is given in (2.5) and may be defined differently for each application. $\mathcal{X}$ defines the feasible solution space. Neither the objective function nor the solution space are easily described in this model. Therefore, the model is not introduced as a means to solve any problems, but rather as a common way of describing them. Admittedly, the task scheduling problems can be modeled in various ways. The model presented here is concise and it is chosen as it has a suitable detail level for the intended description.

Naturally, this very general model does capture other types of problems as well, but the important property to notice, is that we can now describe any solution to any of the task scheduling problems using only the x-variables. For the task scheduling problems at hand, we may even refine the description of the feasible region slightly. We introduce μ_i^- and μ_i^+ to represent the amount of under-coverage and over-coverage for task i, respectively. b_i is the *demand* (the requested number of employees) for task i. We replace (2.5)-(2.6) by:

$$\min \quad f(x, \mu_i^-, \mu_i^+) \tag{2.7}$$
$$\sum_{k \in \mathcal{K}} \sum_{t \in \mathcal{T}} x_{kit} + \mu_i^- - \mu_i^+ = b_i \qquad \forall i \in \mathcal{I} \tag{2.8}$$
$$x_{k\cdot\cdot} \in \mathcal{X}_k \subseteq \mathbb{B}^{|\mathcal{I}||\mathcal{T}|} \qquad \forall k \in \mathcal{K} \tag{2.9}$$
$$\mu_i^- \in \mathbb{Z}_+, \mu_i^+ \in \mathbb{Z}_+ \qquad \forall i \in \mathcal{I} \tag{2.10}$$

The objective (2.7) is a function of all the decision variables. Constraints (2.8) are the generalized assignment constraints, which ensure that all tasks are allocated or marked as unallocated by setting the slack and surplus variables appropriately. Constraints (2.9) define the feasible schedules of each worker. The extent and complexity of these constraints vary considerably from one context to another. Constraints (2.10) set the domains of the slack and surplus variables.

As is obvious from (2.7)-(2.10) the task scheduling model is very similar to the rostering model (2.1)-(2.4). Actually, rostering may be seen as a special case of task scheduling where the tasks (shifts) are fixed in time. Workers need to be considered uniquely (not in groups) to fit the definition of task scheduling problems and the decision variables are hence binary variables. The big difference in the two formulations lies in the definition of feasible lines of work, respectively Constraints (2.3) and Constraints (2.9). Very different rules apply to the two sets of problems. However, using this unified view on the two problem types, we are in the remainder of this thesis able to sketch ideas and solution methods for task scheduling problems which are easily transferred to rostering problems as a special case.

We will in this thesis focus particularly on task scheduling problems with *temporal dependencies* between tasks. Temporal dependencies represent restrictions between tasks, where the scheduled time of one task may restrict when another task can be schedule. Let $\mathcal{T}^x$ be a set that describes exclusions between two tasks i and j. An element (i, j, t_i, t_j) in $\mathcal{T}^x$ prohibits the scheduling of task i at time t_i if task j is scheduled at time t_j and vice versa. Constraints (2.11) describe *temporal dependencies* between tasks and are added to model (2.7)-(2.10):

$$x_{k_1 i t_i} + x_{k_2 j t_j} \leq 1 \qquad \forall k_1, k_2 \in \mathcal{K}, \forall (i, j, t_i, t_j) \in \mathcal{T}^x \tag{2.11}$$

However, the temporal dependencies considered in this thesis can all be described by *generalized precedence constraints* or *disjunctive generalized precedence constraints*, which are less general than Constraints (2.11). To formulate the generalized precedence constraints, we introduce an auxiliary variable $s_{ki} = \sum_{t \in \mathcal{T}} t x_{kit}$. Assuming that $\sum_{t \in \mathcal{T}} x_{kit} \leq 1, \forall i \in \mathcal{I}, \forall k \in \mathcal{K}$, s_{ki} denotes the scheduled time of task i in the schedule of worker k. $s_{ki} = 0$ if task i is not allocated to worker k. The generalized precedence constraints can be formulated as:

$$s_{k_1 i} + \delta_{ij} \leq s_{k_2 j} \qquad \forall k_1, k_2 \in \mathcal{K}, \forall (i,j) \in \Delta \tag{2.12}$$

Δ is the set of all task-pairs (i, j) for which generalized precedence constraints exist and δ_{ij} is the corresponding parameter that defines the necessary time-difference between two tasks. For $b_i = 1$ and $\mu_i^- = \mu_i^+ = 0, \forall i \in \mathcal{I}$, we may use a compact formulation of the constraints:

$$\sum_{k \in \mathcal{K}} s_{ki} + \delta_{ij} \leq \sum_{k \in \mathcal{K}} s_{kj} \qquad \forall (i,j) \in \Delta \tag{2.13}$$

$$\Updownarrow$$

$$s_i + \delta_{ij} \leq s_j \qquad \forall (i,j) \in \Delta \tag{2.14}$$

Constraint (2.14) is a simplification achieved by redefining $s_i = \sum_{k \in \mathcal{K}} s_{ki}$. The generalized precedence constraints may also be modeled with the original x-variables. This and other details are presented in Paper E.

Disjunctive generalized precedence constraints describe a relationship where two tasks are forced to keep a certain distance, but where it is up to the model to decide which one is first. For the simple case (with $b_i = 1$ and $\mu_i^- = \mu_i^+ = 0, \forall i \in \mathcal{I}$) the disjunctive constraints are formulated below. $\Delta_\vee$ is the set of task-pairs (i, j) for which disjunctive generalized precedence constraints exist.

$$s_i + \delta_{ij} \leq s_j \vee s_j + \delta_{ji} \leq s_i \qquad \forall (i,j) \in \Delta_\vee \tag{2.15}$$

The generic mathematical model of task scheduling problems allows for a common graphical presentation of solutions. For this, we use a chart similar to a gantt-chart, with time running on the horizontal axis. The workers are listed on the vertical axis, one under the other. If we, additionally, show the duration of each task, we get a chart like the one shown in Figure 2.1. The chart may be refined for each specific case, as we may want to include additional information or as the problem may have a certain structure that we can make use of in a graphical representation. The papers of Part II show various refinements of the gantt-chart.

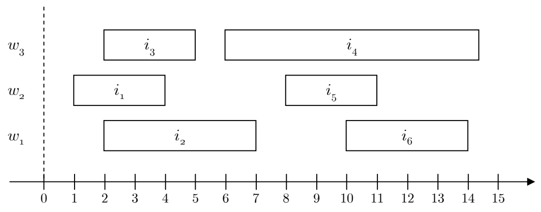


Figure 2.1: A generic graphical representation of a solution to a task scheduling problem.

2.3 Interconnectivity

As mentioned, generalized precedence constraints are not part of all task scheduling problems. Without these constraints, only Constraints (2.8) link the individual work-lines of the workers. We define the interconnectivity of a task scheduling problem in three levels, defined by the order of the number of elements in Δ (and $\Delta_\vee$).

Level 0 $\Delta = \emptyset$.

Level 1 The number of elements in Δ is in the order of $|\mathcal{I}|$.

Level 2 The number of elements in Δ is in the order of $|\mathcal{I}|^2$.

This measurement of interconnectivity is introduced, as we find that the amount of interconnectivity has a significant effect on the complexity of the problem and hence on the choice of solution method. Obviously, the difficulty of a task scheduling problem also depends on the complexity of the single work-line generation and on the size of the instances. Hence, when considering these factors, we may quantify the difficulty of a problem. We will do this for our practical problems and reflect on the result in Section 3.3.

2.4 Related problems

With the models for manpower planning problems in place, it is possible to find a range of related problems and problems with overlapping definitions in the literature. The intention of this section is to give an overview of the related literature and to position the manpower planning problems in relation to similar optimization problems.

2.4.1 The assignment problem

A specification of both rostering problems and task scheduling problems is the *assignment problem*, which is a well established problem in operations research. A number of tasks is given along with the same number of workers. Each combination of worker and task has a given cost, c_{ki} and the objective is to assign each worker to a task at minimum total cost. One set of decision variables, x_{ki}, is used, where $x_{ki} = 1$ if worker $k \in \mathcal{K}$ is assigned to task $i \in \mathcal{I}$, and $x_{ki} = 0$ otherwise.

$$\begin{aligned}
\min \quad & c_{ki}x_{ki} && && (2.16)\\
\text{s.t.} \quad & \sum_{k\in\mathcal{K}} x_{ki} = 1 && \forall i \in \mathcal{I} && (2.17)\\
& \sum_{i\in\mathcal{I}} x_{ki} = 1 && \forall k \in \mathcal{K} && (2.18)\\
& x \in \mathbb{B}^{|\mathcal{K}||\mathcal{I}|} && && (2.19)
\end{aligned}$$

There is no temporal dimension in the assignment problem. We may consider it as having a set of times with a single element, e.g. $\mathcal{T} = \{0\}$. The objective (2.16) is to minimize the sum of costs. It is clearly a specification of objective (2.7). Constraints (2.17) and (2.18) give the one-to-one correspondence between workers and tasks. The constraints are specifications of Constraints (2.8) and (2.9), respectively. The domain of the x-variables (2.19) is the same as in the general model. Hence, the assignment problem may be considered as a very simple task scheduling problem. The same is true for the rostering model (2.1)-(2.4).

The assignment problem can be solved in polynomial time by e.g. The Hungarian Method of Kuhn (1955). Many extensions and variants of the classical assignment problem has been examined in the literature, including the quadratic assignment problem (see Pardalos and Wolkowicz, 1993) and the generalized assignment problem (see Cattrysse and Van Wassenhove, 1992). In the latter, each worker can be assigned to multiple tasks, i.e. Constraints (2.18) are changed to 'less than or equal to'-constraints and the right hand sides can be integer values larger than 1. In both rostering and task scheduling, workers can certainly be assigned to multiple tasks and the model of the generalized assignment problem is therefore closer to those of rostering and task scheduling. However, it is still a very simple problem, compared to the other two. For a recent book on the assignment problem that contains a thorough survey of the literature and an exposition of numerous solution methods and problem variants see Burkard et al. (2009).

2.4.2 Crew rostering

Crew rostering (also referred to as *crew scheduling*) comes in many different variants from various contexts. In essence, most of these are extensions or variations of the rostering model presented earlier (2.1)-(2.4).

An area which has had substantial impact on the practice of the industry is in crew rostering for cabin crew in airlines. These problems are particularly complex, as they contain an inherent geographical dimension, where rosters need to ensure that employees end up in their home town before off-days can be assigned. The complexity of the problems makes it very hard to generate good rosters by hand and extensive savings are usually observed when the rostering process is automated. Butchers et al. (2001) report savings of NZD 15.7 million (USD 11.2 million) per year for Air New Zealand, which is more than 6% of the estimated annual crew costs in the airline. Similarly, Anbil et al. (1991) report savings in excess of USD 20 million at American Airlines. Naturally, because of the large potential cost reductions, the rostering problem for cabin crew has received a lot of attention in the literature. A good introduction to the problem is found in Barnhart et al. (2003). Gopalakrishnan and Johnson (2005) present a resent survey of approaches and solution methodology of the area. Kohl and Karisch (2004) give a comprehensive description of the problem and describe the challenges faced when implementing a crew rostering system in practice.

In this thesis, the focus is on rostering problems without the transportation component. This still includes a variety of problems in areas such as call centers, health care systems, emergency services, civic services, venue management, tourism, and manufacturing. Ernst et al. (2004*a*) present a long list of applications along with numerous references to problem descriptions and proposed solutions approaches in the literature. Other recent surveys of the rostering literature are found in Burke et al. (2004); Ernst et al. (2004*b*); Cheang et al. (2003). In the literature, many different rostering problems have been presented and the solution methods, as well, are plentiful. Typically, specific problems are solved using explicit models and tailored solution methods. This makes it complicated to adapt to similar problems or variants of the same problem, which is also declared by Ernst et al. (2004*b*). This is also partially the reason why, it can be hard, to get the industry to appreciate the solutions proposed in academia, as pointed out by Kellogg and Walczak (2007). To support the development of solution approaches that can eventually be applied in realistic settings, it is important to solve the concrete problem and not a simplification hereof. At the same time, the approaches must be versatile enough to allow transformation between similar setups.

Rostering problems are typically characterizes by having many preferences and *soft constraints*, i.e. constraints that may be violated, if necessary. Preferences are in many cases individual to the employees and the employees can therefore not be treated in a unified perspective. The soft constraints stem from a number of rules that have typically been introduced to describe some desired features of a roster, where it is well known that no roster will be able to accommodate all requirements.

2.4.3 The vehicle routing problem

Another well established problem in operations research is the vehicle routing problem. A subset of the vehicle routing problems are constrained by time windows, i.e. the vehicle routing problem with time windows (VRPTW). The time window constraints give the vehicle routing problem a structure much similar to the scheduling problems considered in this thesis. In VRPTW a set of customers is served by multiple vehicles. The vehicles are constrained by their capacity and must serve each customer within a predefined time window. VRPTW can be considered as a task scheduling problem, where vehicles correspond to workers and customers correspond to tasks. We use the notation from before, and refer to customers in $\mathcal{I}$ and vehicles in $\mathcal{K}$. We use decision variable y_{ijk} in the model, as it is slightly different from the variable x_{kit} of the general model. The variable is defined so that $y_{ijk} = 1$ if vehicle k visits customer j immediately after visiting customer i, and $y_{ijk} = 0$ otherwise. An integer variable, s_{ik}, specifies the start time of service for vehicle k at customer i. If customer i is not visited by vehicle k, $s_{ik} = 0$. An auxiliary set of locations, $\mathcal{N}$, is introduced, which contains all customers and two representations of the depot, one where each route must begin and one where each route must end.

$$\min \sum_{i\in\mathcal{N}}\sum_{j\in\mathcal{N}}\sum_{k\in\mathcal{K}} c_{ij} y_{ijk} \tag{2.20}$$

$$\sum_{j\in\mathcal{N}}\sum_{k\in\mathcal{K}} y_{ijk} = 1 \qquad \forall i \in \mathcal{I} \tag{2.21}$$

$$\sum_{i\in\mathcal{I}} d_i \sum_{j\in\mathcal{N}} y_{ijk} \le q \qquad \forall k \in \mathcal{K} \tag{2.22}$$

$$\sum_{j\in\mathcal{N}} y_{0jk} = 1 \qquad \forall k \in \mathcal{K} \tag{2.23}$$

$$\sum_{i\in\mathcal{N}} y_{ihk} - \sum_{j\in\mathcal{N}} y_{hjk} = 0 \qquad \forall h \in \mathcal{I}, \forall k \in \mathcal{K} \tag{2.24}$$

$$\sum_{i\in\mathcal{N}} y_{i,n+1,k} = 1 \qquad \forall k \in \mathcal{K} \tag{2.25}$$

$$s_{ik} + \tau_{ij} - M(1 - y_{ijk}) \le s_{jk} \qquad \forall i, j \in \mathcal{N}, \forall k \in \mathcal{K} \tag{2.26}$$

$$\alpha_i \sum_{j\in\mathcal{N}} y_{ijk} \le s_{ik} \le \beta_i \sum_{j\in\mathcal{N}} y_{ijk} \qquad \forall i \in \mathcal{I}, \forall k \in \mathcal{K} \tag{2.27}$$

$$y_{ijk} \in \{0, 1\} \qquad \forall i, j \in \mathcal{N}, \forall k \in \mathcal{K} \tag{2.28}$$

$$s_{ik} \in \mathbb{R}_+ \qquad \forall i \in \mathcal{N}, \forall k \in \mathcal{K} \tag{2.29}$$

VRPTW falls under the definition of task scheduling problems (2.7)-(2.10), with the following variable transformation:

$$x_{kit} = \begin{cases} \sum_{j\in\mathcal{N}} y_{ijk} & \text{, if } t = s_{ik} \\ 0 & \text{, otherwise} \end{cases} \tag{2.30}$$

With this definition, Constraints (2.21) reformulate to:

$$\sum_{k\in\mathcal{K}}\sum_{t\in\mathcal{T}} x_{kit} = 1 \qquad \forall i \in \mathcal{I} \tag{2.31}$$

It is clear that Constraints (2.31) are specializations of Constraints (2.8) of the general task scheduling model. Constraints (2.22)-(2.29) define the individual routes and together are specializations of Constraints (2.9). The objective function (2.20) is not compactly described in terms of x_{kit}, but the objective can be deducted from x_{kit} and hence is a specialization of (2.7). There are no temporal dependencies between vehicles, i.e. no constraints of type (2.11), in the ordinary VRPTW. Hence, VRPTW has interconnectivity of level 0. An extension with temporal dependencies is described in detail in Paper E. As the objective and all constraints are specializations of Constraints (2.7)-(2.10), VRPTW may also be considered as a task scheduling problem. In practice, some manpower planning problems are indeed very similar to VRPTW. We show a few examples in the applications considered in this thesis. An introduction to VRPTW is given in Kallehauge et al. (2005). Early approaches to the problem are among others presented by Desrosiers et al. (1984), Solomon (1987), and Kolen et al. (1987). The field of research can be seen as going in two directions, where one direction considers realistic problems, typically producing solutions with a pragmatic approach using heuristic solution methods. The other direction is concerned with exact solution of prototype problems. The latter is the most relevant to the work presented in this thesis. An early breakthrough was made with the column generation approach presented by Desrochers et al. (1992). Recent contributions have developed the concepts significantly, see e.g. Chabrier (2006), Jepsen et al. (2008), and Desaulniers et al. (2008).

Another variation of the vehicle routing problem with relevant similarities to the problems considered here, is the vehicle routing problem with split deliveries (VRPSD) as introduced by Dror and Trudeau (1989). When split deliveries are allowed, multiple vehicles can serve the same customer to jointly meet the customer demand. A survey of the literature is found in Lee et al. (2006). Frizzell and Giffin (1995) were the first to include time windows in VRPSD to form the vehicle routing problem with time windows and split deliveries. Desaulniers (2009) present a recent exact approach to the problem.

2.4.4 Parallel machine scheduling

Parallel machine scheduling is another classic operations research problem with a close relation to the applications considered in this thesis. Job shop scheduling is perhaps the best know of these machine scheduling problems. A set of jobs, $\mathcal{J}$, each consisting of a sequence of operations, $\mathcal{I}$, is to be scheduled on a set of machines, $\mathcal{K}$. The processing order of the operations in each job is predetermined and each job can only be processed on one machine at a time. Each machine has a maximum capacity of one and each operation has processing time t_i. Minimizing makespan is a frequently used objective. Variants of job shop scheduling are open shop scheduling and flow shop scheduling. In all three variants, the task allocation is predetermined. They are all specializations of our general definition of task scheduling problems.

A general model of parallel machine scheduling which, among many others problems, captures the three scheduling problems mentioned above, can be formulated as follows. Given is a set of operations, $\mathcal{I}$, and a set of machines, $\mathcal{K}$. Each operation has a release date α_i and a deadline β_i, which together define a time window for the initiation of operation i, s_i. Each operation is allocated to a machine. The allocation is represented by a binary variable x_{ik}, where $x_{ik} = 1$ if operation i is allocated to machine k, $x_{ik} = 0$ otherwise. The operations may be subject to generalized precedence constraints. Some of these describe the sequencing of operations in jobs, because one operation of a job must finish before another operation of the same job is started. With such precedence constraints, the interconnectivity of the parallel machine scheduling is of level 2. Minimizing makespan, we get the following model:

$$\min \max_{i \in \mathcal{I}} s_i + t_i \tag{2.32}$$

$$\text{s.t.} \quad \sum_{k \in \mathcal{K}} x_{ik} = 1 \qquad \forall i \in \mathcal{I} \tag{2.33}$$

$$\alpha_i \leq s_i \leq \beta_i \qquad \forall i \in \mathcal{I} \tag{2.34}$$

$$s_i + \delta_{ij} \leq s_j \qquad \forall (i,j) \in \Delta \tag{2.35}$$

$$x_{ik} = 1 \wedge x_{jk} = 1 \Rightarrow s_i + t_i \leq s_j \vee s_j + t_j \leq s_i \qquad \forall i,j \in \mathcal{I}, \forall k \in \mathcal{K} \tag{2.36}$$

$$x_{ik} \in \{0,1\} \qquad \forall i \in \mathcal{I}, \forall k \in \mathcal{K} \tag{2.37}$$

$$s_{ik} \in \mathbb{R}_+ \qquad \forall i \in \mathcal{I}, \forall k \in \mathcal{K} \tag{2.38}$$

The objective function (2.32) minimizes makespan, but various objectives can be used. Constraints (2.33) enforce that each operation is allocated to exactly one machine. All time windows must be respected (2.34) and likewise the generalized precedence constraints (2.35). For operations on the same machine, there is a strong precedence relation, i.e. one operation is finished before the next is initiated (2.36). Constraints (2.37) and (2.38) give the domains of the variables.

The model again is a special case of the general task scheduling model stated earlier. The objective function (2.32) is a specialization of (2.7). Constraints (2.33) are specializations of Constraints (2.8). Constraints (2.34) and (2.36) can be considered individually for each machine and are therefore together with Constraints (2.37) and (2.38) a specialization of Constraints (2.9). Finally, Constraints (2.35) describe temporal dependencies, which can also be modeled with Constraints (2.11). We refer to Baptiste et al. (2001) for more on literature in the context of constraint programming and for other variants of the problem. van den Akker et al. (1999) are the first to present a column generation based approach to the problem. Allahverdi et al. (2008) present a recent survey of scheduling problems relevant to the work presented in this thesis.

2.4.5 Cutting stock

The cutting stock problem can be found in various contexts in the industry, where e.g. rolls of paper need to be cut into smaller pieces with as little waste as possible. Obviously, the material is not important in the mathematical model and it can be used to describe any problem, where something is cut into smaller pieces. The problem can also be used in an abstract understanding, where *something* needs to be divided, e.g. as stated in Paper B some rostering problems may been seen as 'cutting' a roster-line into requested 'pieces' being the shifts.

A traditional formulation of the cutting stock problem is given below. Defined is a set of possible patterns $\mathcal{R}$ which defines how a roll is cut into pieces that belong to the set $\mathcal{I}$. Each piece $i \in \mathcal{I}$ has a request of b_i. The decision variables $x_r, r \in \mathcal{R}$ set the number of times pattern $r \in \mathcal{R}$ is used, at a cost of c_r for each. Parameter $a_{ri}, r \in \mathcal{R}, i \in \mathcal{I}$ describes how many pieces of type i is cut in pattern r.

$$\sum_{r \in \mathcal{R}} c_r x_r \tag{2.39}$$

$$\text{s.t.} \quad \sum_{r \in \mathcal{R}} a_{ri} x_r \geq b_i \qquad \forall i \in \mathcal{I} \tag{2.40}$$

$$x_r \in \mathbb{Z}_+ \qquad \forall r \in \mathcal{R} \tag{2.41}$$

An early exposition of the cutting stock problem, which illustrates that optimization of the problem has attracted attention for several decades, is given by Gilmore and Gomory (1961). A more recent description of a column generation solution method is given by Amor and Carvalho (2005).

2.4.6 Hoist scheduling and container stacking

The scheduling of cranes/hoists has attracted significant attention, particularly in production of electronics, where production has usually been automated and where a high efficiency is essential for a profitable operation. E.g., in circuit board production, hoists are moving products between tanks with various chemicals. When hoists run on shared tracks or in other ways may block each other, it is important to respect so-called anti-collision requirements, as described by Lamothe et al. (1996). For more on hoist scheduling see Leung and Zhang (2003) or refer to Zhu and Wilhelm (2006) for a recent literature review on general scheduling problems with sequence-dependent setup times.

In the context of this thesis, crane scheduling is considered in connection with a container-stacking-like problem. Container stacking has, as a separate subject, received a lot of attention. Container stacking is a complex problem, even when only one crane is involved. Stacks of containers have to build so that the total number of moves is minimized, while respecting a number of side-constraints. See Steenken et al. (2004) for a recent literature review.

Chapter 3

Applications

The work in this thesis is based on a number of practical manpower planning problems. In the following, we will describe each of the applications in their context of either rostering or task scheduling. Following, the problems will be compared to prototype problems from the literature and to each other, in order to establish the context of this thesis.

Figure 3.1 illustrates the usual workflow in manpower planning. The workflow is divided into the tasks that are usually part of the planning process.

In long term planning, the workload is forecasted and strategic decisions on manpower capacity are made. The available manpower will typically have to follow trends in forecasted demand and hiring/firing employees may be necessary. From an estimated workload, a shift layout can be produced. Generating the layout includes a specification of shift types with start and end times. Typically, the shifts are matched with a requested cover which is deduced from the forecasted workload. The next step in the process is the rostering. Rosters are made to closely fit the requested cover while respecting laws, union rules and internal agreements. Figure 3.2 illustrates the relation between the workload, the demand (requested cover) and the actual cover.

With the rosters available, the tasks are scheduled. This is often done when approaching the day of operation. From the roster, the number of people working

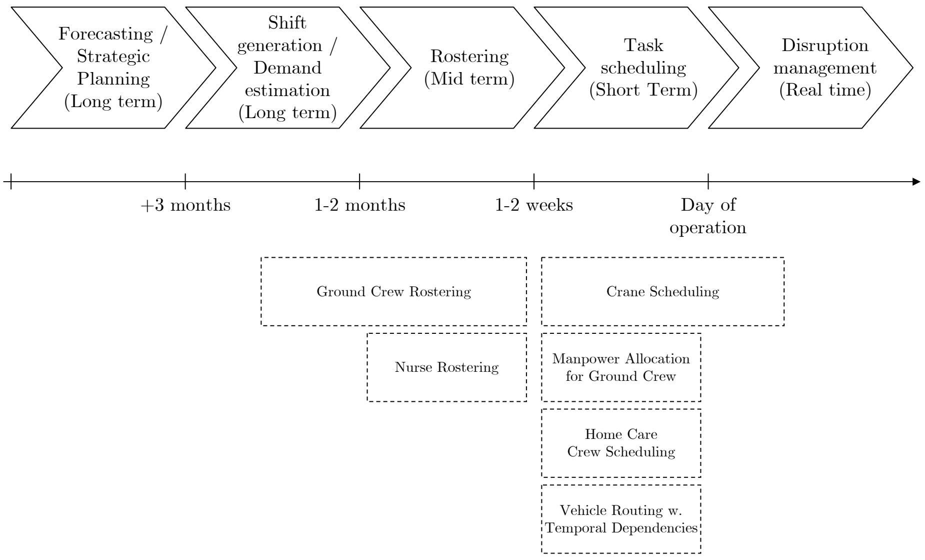

Forecasting / Strategic Planning (Long term)
Shift generation / Demand estimation (Long term)
Rostering (Mid term)
Task scheduling (Short Term)
Disruption management (Real time)
+3 months
1-2 months
1-2 weeks
Day of operation
Ground Crew Rostering
Crane Scheduling
Nurse Rostering
Manpower Allocation for Ground Crew
Home Care Crew Scheduling
Vehicle Routing w. Temporal Dependencies

Figure 3.1: The usual workflow in manpower planning.

each shift is known and tasks are distributed between the available employees. There may be various dependencies internally between tasks, but the overall goal is usually to schedule as many of the tasks as possible with the available manpower. The last phase of the workflow is concerned with the real time operation. Often, task schedules have to be adjusted according to disturbances on the day of operation and this is done in an online planning module.

Obviously, there are variations and specifications of the workflow. A step which is not included in the figure is concerned with roster disruption, which happens anywhere from when the first roster is generated until the day of operation. Also, the phases are not necessarily considered independently. There may be advantages in combining e.g. shift generation with rostering or rostering with task scheduling, as illustrated in Paper B.

Below the workflow, boxes with the practical problems that have been considered in this thesis are given. The practical problems are located according to their position in the workflow. The problems are described in more detail below. From the figure, it can be observed that the focus of this thesis has been on mid term and short term planning. As illustrated, two of the projects also include components of long term planning and real time planning, respectively.

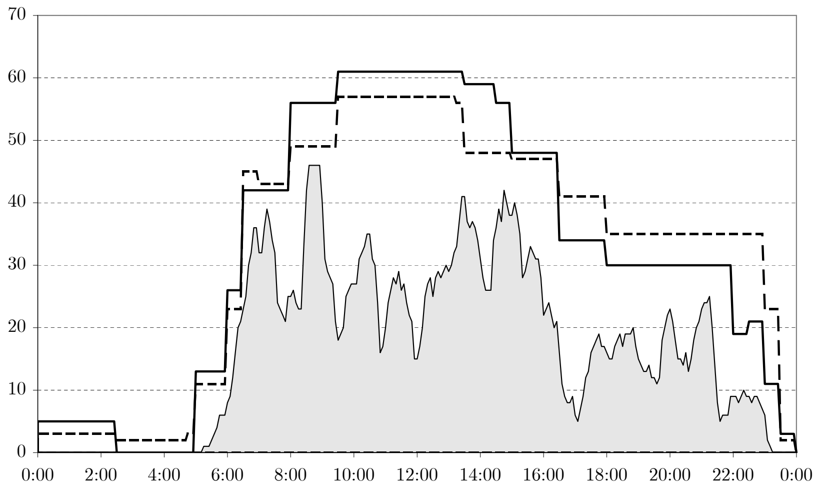


Figure 3.2: Example of a workload graph. The workload (gray area) is forecasted. Based on the workload, a requested cover is determined (dashed line). The actual cover of the roster (full line) should fit the requested cover as closely as possible. The unit on the vertical axis is 'number of people'.

3.1 Rostering

Rosters are generated from a fixed set of shifts and from a requested cover. Rosters consist of a number of roster-lines that are combined into a full roster, where as many demands as possible are satisfied. Individual roster-lines are generated for each employee or alternatively for a group of employees working under the same contractual restrictions. Each roster-line can be decomposed into a sequence of *work-stretches*. Work-stretches consist of a segment of days on (a so called *on-stretch*) followed by a segment of days off (an *off-stretch*). This relationship is illustrated in Figure 3.3.

The roster-lines have to respect a number of constraints on their shift composition, the on-days/off-days structure, as well as general restrictions on the roster-line. The constraints can be defined for each of the components, i.e. some constraints define feasibility of on-stretches, others define feasibility of work-stretches and only rules that are concerned with multiple work-stretches are defined on the roster-line level. In this thesis, only non-transportational rosters are considered. There may be transportation during a shift, but nothing that affects the roster as such. In e.g. rostering of cabin crew in the airline industry, rosters have to incorporate a geographical dimension, as employees may end a

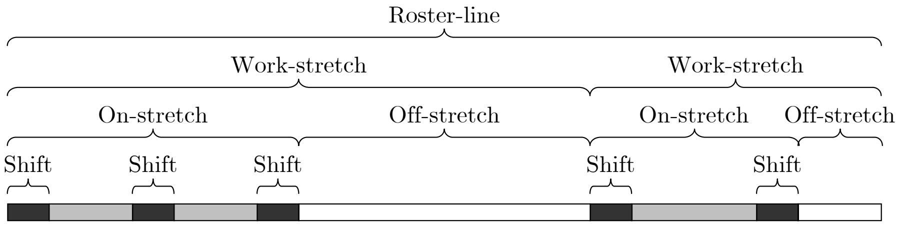


Figure 3.3: A roster-line and its components.

shift in a location far from the one, they started their shift in. This introduces a number of complications that are disregarded here.

It takes a lot of experience to build good rosters manually and even with experience, the process of building the rosters is very time consuming. Therefore, there is a significant demand for automated rostering tools. Within the last decade the supply of software products has increased significantly to meet this demand. It is not straight forward to create a solution, which is generic enough to capture the many variations of rostering problems that one may encounter. Therefore, it is important to have as few assumptions as possible.

In the work presented in this thesis, the following is assumed for the rostering problems. A predefined set of shifts exists, where shifts have fixed start and end times. Prespecified demands or workload estimates are defined and the demands must be met, as closely as possible, by assigning employees to shifts. A simple example of a demand is a single-shift-demand introducing a bound on a single shift, e.g. a lower bound on a specific morning shift. Often demands will involve several shifts, e.g. a lower bound on the number of employees on all types of morning shifts. Demands can relate to anything from small 5-minutes intervals to full day or monthly requirements. Demands may also involve skill requirements.

All constraints on roster-lines and their components are described by *attributes*. The values of the attributes are calculated as the roster-line is built. It is assumed that all roster-line constraints can be described by attributes. Both the literature and the examples considered here, show that almost any possible constraint can indeed be formulated this way.

We confine our focus to rostering problems which can be described by a set of demands and by the roster-line feasibility criteria and costs. The demands may be declared as soft constraints, i.e. under-coverage and over-coverage is permitted at a certain cost. Admittedly, this specification is not broad enough to cover any imaginable rostering problem, but it does capture most realistic problems including those encountered in practice and those described in broad literature surveys of e.g. Cheang et al. (2003) and Burke et al. (2009). See also Ernst et al. (2004*b*) for a survey of the rostering literature.

3.1.1 Nurse rostering

The nurse rostering instances considered in Paper A are instantiations of a generalized rostering model, which captures rostering problems from many industries. Nurse rostering problems are interesting as they contain most of the complexities considered in any non-transportational rostering problem. There is usually a 24-hour coverage requirement and the nurses have various skills and competences. Often they are employed under varying contractual conditions and have individual preferences. Preferences are on specific shifts as well as on shift composition in the roster-line. On top of the individual preferences, nurse rostering problems, typically, include an array of soft constraints, i.e. desirable features of the roster that do not make the rosters infeasible, if they cannot be satisfied. All instances considered here look at a 30-day roster period. Burke et al. (2004) present a survey of the literature on nurse rostering.

3.1.2 Ground crew rostering

In Paper B, one specific ground crew rostering problem from the industry is considered. The problem has a few distinctive characteristics. The crew is working a 6-on/3-off pattern, meaning that all on-stretches are 6 days and are always followed by a 3-day off-stretch. No individual preferences are considered, and anonymous roster-lines are generated for 9 different employee groups, one for each possible offset of the pattern. Demands are defined for 5-minute intervals and are inferred directly from the corresponding airport workload.

As can be observed from Figure 3.1, the problem also considers shift generation to a limited extent. A number of shift proposals are included in the model with no requirements on which of these must be used or with what cover. Instead, the objective is to cover the workload using the proposed shifts as desired. It is not a full shift generation scheme, as it still requires a larger set of potential shifts to choose from.

The roster period is half a year. To be able to solve for such a long roster period, it is split into blocks of 2-3 weeks and each block can be solved separately with certain conditions applying in the overlap of blocks.

Rostering for ground crew has not received nearly the same attention in the literature as nurse rostering. A few important references are Brusco et al. (1995), Dowling et al. (1997), and Chu (2007).

3.2 Task scheduling

From a given roster and hence a specification of workforce availability during the day, tasks can be scheduled and assigned to employees. This is the focus in the following applications.

3.2.1 Manpower allocation for ground crew

The manpower allocation problem with time windows and job-teaming constraints (MAPTWTC) is considered for ground crew in airports. It is the problem of assigning a number of teams to a set of tasks in a daily schedule. The teams drive from one task to the next and therefore the necessary travel time has to be included in the schedule. The teams are working individual shifts, which give them an individual start and end times during the day. The tasks are also constrained by time windows, restricting the time at which they can be scheduled. MAPTWTC was first described by Lim et al. (2004) and Li et al. (2005). What differentiates MAPTWTC from the uncapacitated VRPTW is the job-teaming, where several teams may cooperate on a single task. Also, it is usually impossible to allocate all tasks to teams and therefore the objective is to allocate as many as possible. The classical variants of VRPTW that most resemble MAPTWTC are the vehicle routing problem with split deliveries and the vehicle routing problem with a limited number of vehicles. In the first of these, several vehicles may service each customer, and in the latter some customers may not be visited. MAPTWTC also contains a number of characteristics from crew scheduling, e.g. staff skills and mandatory breaks.

The job-teaming constraint may be interpreted as a synchronization constraint between several identical sub-tasks that together make up the original task. With this interpretation, MAPTWTC nicely falls under the definition of task scheduling problems of Chapter 2. The manpower allocation problem considered in Paper C concerns schedules of ground handling crew in airports.

Between arrival and subsequent departure of an aircraft, numerous jobs including baggage handling and cleaning must be performed. It may be necessary to have several teams cooperating on one task in order to complete it within the time window. Hence, job-teaming is required. Often, airlines buy the ground handling service from so called ground handling companies. The ground handling companies end up with a large amount of tasks on each day and hence some interesting optimization problems.

The number of synchronization constraints is between 10% and 40% of the total number of tasks (interconnectivity of level 1).

3.2.2 Home care crew scheduling

The home care crew scheduling problem (HCCSP) is a variant of MAPTWTC and is considered in Paper D. In home care, a number of citizens request services in their own home. The services may include cleaning and laundry assistance 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, preparing food, and taking medicine. As a consequence of the variety of services offered, people with different competences are employed as caretakers.

HCCSP deals with the assignment of visits to caretakers and the scheduling of these visits. The main goal is to cover all visits, if possible. In addition, 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, as this establishes confidence with the citizen.

The problem also contains shared visits. These are visits requiring the presence of more than one caretaker, and consequently each visit must be included in the route of several caretakers, where the interconnected visits must be synchronized. Other tasks have internal temporal dependencies, e.g. some tasks have to be separated by a prespecified gap to allow time for a washing machine to finish or because of medical prescriptions. The number of temporal dependencies is always lower than the number of tasks (interconnectivity of level 1).

The home care crew scheduling problem has previously been described in several papers. This work builds on top of the findings of Thomsen (2006) and Lessel (2007), who describe home care problems in Denmark. A similar study in Sweden is described by Eveborn et al. (2006), Bredström and Rönnqvist (2007), and Bredström and Rönnqvist (2008). Studies in other countries are conducted by e.g. Bertels and Fahle (2006) and Begur et al. (1997).

3.2.3 Vehicle routing with time windows and temporal dependencies

The vehicle routing problem with time windows and temporal dependencies (VRPTWTD) is closely related to MAPTWTC and has in this thesis been used to test if the conclusions from the task allocation problems also hold for the well studied vehicle routing problems. VRPTWTD is described in Paper E. VRPTWTD is an extension of VRPTW where temporal dependencies between customers exist. As in the traditional VRPTW all customers must be visited using an unrestricted number of vehicles. The objective is to visit all customers, while minimizing the total distance travelled. As VRPTWTD is a new problem, the relevant literature is on similar variants of vehicle routing (see e.g. Toth and Vigo (2001)).

The instances considered are randomly generated based on well known VRPTW instances from the literature. Various temporal dependencies are added and the number of temporal dependencies in the test instances is varied between 0 and the number of tasks (interconnectivity of level 1).

3.2.4 Slab yard planning and crane scheduling

Paper F describes the slab yard planning and crane scheduling problem. It has its origin in steel production facilities, where steel slabs are stored in stacks in a large yard. Two gantry cranes on a shared track operate in the yard, where slabs are hoisted and moved one by one using electromagnets. Slabs are added to the yard from trains, which arrive in one end of the yard. The cranes move the slabs from the trains to stacks in the yard, where they may be moved around, until they are eventually needed in production. When asked for, the slabs are moved to a roller table on which they leave the yard. Each slab has distinct properties and is therefore, in practice, considered to be unique. Figure 3.4 sketches the layout of the slab yard. The work presented here is based on the research of Hansen (2003), who also gives a detailed problem description.

The problem can be considered in two stages. The first stage consists of the slab yard planning, which is concerned with the layout of the yard. Decisions are made on the movement of slabs. The goal is to arrange the yard so that slabs are not blocking each other, thereby minimizing the total number of slab moves needed. Also, the gantry cranes run on a shared track and can therefore not pass each other. In the planning stage it is ensured that the cranes will not wait too much for each other while moving slabs. The planning problem is related to similar problems in container yard planning, but it is slightly different from

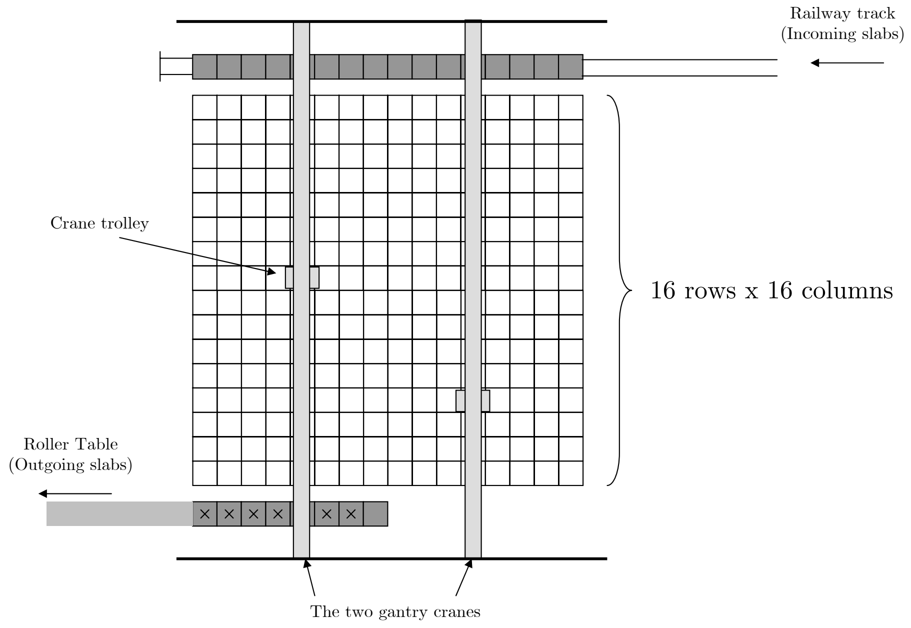


Figure 3.4: Overview of a slab yard.

the other problems considered in this thesis.

The second part of the problem is concerned with the scheduling of the cranes and can be considered as a task scheduling problem. Assuming that all move-tasks are generated in the planning stage, the crane scheduling problem consists of scheduling these tasks, i.e. allocating the tasks to the cranes and finding a feasible schedule for each of the cranes. It is possible to model the problem as a regular machine scheduling problem. Disjunctive precedence constraints are common in scheduling problems and represent the property that either task A is scheduled and finished before task B or vice versa. Such a relationship may exist internally on a machine as well as between tasks on different machines. In the crane scheduling problem, disjunctive generalized precedence constraints are used to ensure that one crane enters and leaves an area of the yard before the other crane enters it, or vice versa. The crane scheduling problem can hence be interpreted as a task scheduling problem with an interconnectivity of level 2, as the number of temporal dependencies for each task is in the order of the number of tasks.

3.3 Comparison

As demonstrated in the previous sections, the manpower planning problems are related to various problems from the literature. The relation to existing literature is sketched in Figure 3.5. The figure is meant to give a rough idea of the resemblance between the individual problems that have been considered in this thesis and well established problems from the literature. A bold arrow indicates that the problem has many properties similar to the prototype problem. A regular full arrow means that there are some similarities between the problems. Finally, a dashed arrow indicates that there are partial or vague similarities.

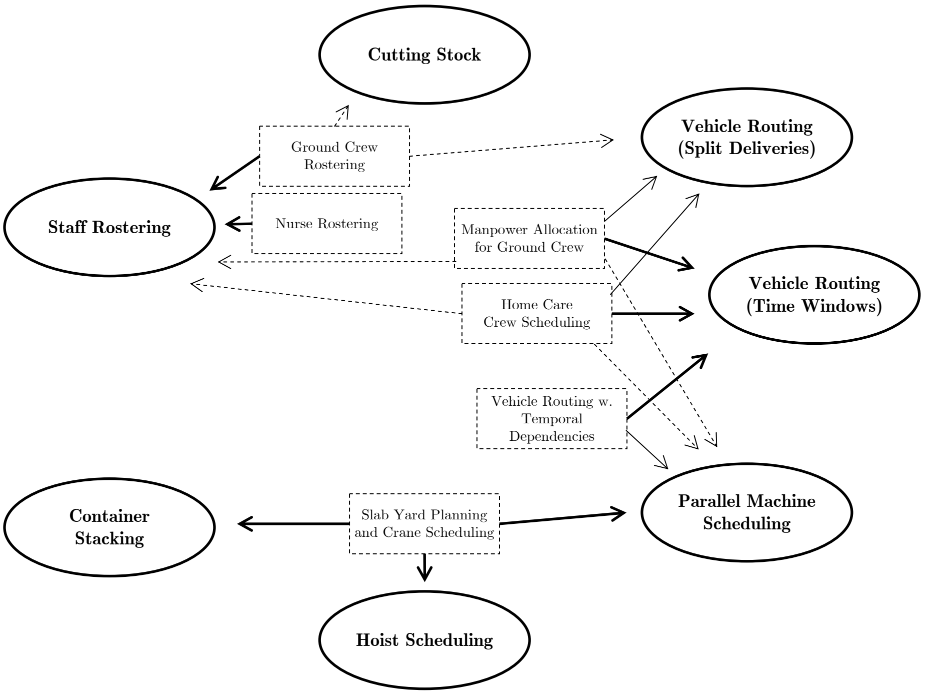

Cutting Stock
Ground Crew Rostering
Vehicle Routing (Split Deliveries)
Staff Rostering
Nurse Rostering
Manpower Allocation for Ground Crew
Vehicle Routing (Time Windows)
Home Care Crew Scheduling
Vehicle Routing w. Temporal Dependencies
Container Stacking
Slab Yard Planning and Crane Scheduling
Parallel Machine Scheduling
Hoist Scheduling

Figure 3.5: Relations between the applications considered and established problems from the literature.

From Figure 3.5, it is evident that the problems of this thesis have much in common with prototype problems of the literature. The figure can also be used to consider the similarities between the applications of this thesis. The division in rostering problems and task scheduling problems is evident, and again the slab yard planning and crane scheduling problem stands out as a slightly different application. It is an interesting observation that several of the problems are related both to routing and scheduling and actually one may question what differentiates routing from scheduling, except from the practical context in which the problems usually appear. A detailed investigation is conducted by Beck et al. (2003); Beck et al. (2006).

Problem	Interconnectivity (level)	Complexity of work-line generation	Instance size
Nurse rostering	0	$\mathcal{NP}$	Medium
Ground crew rostering	0	$\mathcal{P}$	Large
Manpower allocation for ground crew	1	$\mathcal{NP}^+$	Medium-Large
Home care crew scheduling	1	$\mathcal{NP}^+$	Medium-Large
Vehicle routing with time windows and temporal dependencies	1	$\mathcal{NP}^+$	Small-Medium
Slab yard planning and crane scheduling	2	$\mathcal{NP}^+$	Large

Table 3.1: Properties that affect the expected computational difficulty of the problems at hand.

In Table 3.1, we try to estimate how hard it is to find good solutions for the individual problems at hand, i.e. the expected computational difficulty. We assume that the computational difficulty depends primarily on three factors, namely: interconnectivity between workers, complexity of the work-line generation problem, and the size of the instances. The level of interconnectivity is based on the definition of Section 2.3. The complexity of the work-line generation is stated as the complexity of the subproblem in a Dantzig-Wolfe decomposition, i.e. the problem of finding the best possible work-line for an employee, given costs on each task/shift. $\mathcal{P}$ means that it is solvable in polynomial time. $\mathcal{NP}$ means that it is an $\mathcal{NP}$-hard problem and therefore no polynomial algorithms exist (unless $\mathcal{P} = \mathcal{NP}$). $\mathcal{NP}^+$ means that the problem is $\mathcal{NP}$-hard in the strong sense and therefore no pseudo-polynomial solution algorithms exist (unless $\mathcal{P} = \mathcal{NP}$). The instance size is a rough estimate and includes both the number of employees and the number of tasks/shifts.

The ground crew rostering and nurse rostering problems have no interconnectivity and the patterns and the few resources in the ground crew rostering instances make work-line generation easy. The ground crew rostering instances are however significantly larger than the nurse rostering problems at hand, both in terms of number of employees and in terms of the scheduling horizon.

The vehicle routing problems with temporal dependencies may have up to one temporal dependency for each task. It is a routing problem, and therefore the creation of single work-lines is harder, as the underlying graph is not inherently acyclic. However, the size of the instances has been chosen to allow optimal solution of the majority of the problems. The work-line creation of home care crew scheduling is slightly complicated by incomplete transportation networks. On the other hand, the manpower allocation problem for ground crew is slightly easier as one can utilize the fact that many tasks are similar and that the cost is only on tasks and not on transport. For both applications, it is possible to find optimal solutions in most cases. Finally, crane scheduling has a higher level of interconnectivity with many tasks and therefore stands out as the hardest problem to solve.

Chapter 4

Solution Methods

In this chapter, we describe a range of solution methods. The most detailed descriptions are given for the methods that have been applied in one or more of the projects. In the individual papers, it is assumed that the reader is familiar with the standard methods and with the related terminology. Therefore, we will describe the basic concepts and review the relevant theory that has been applied in the projects. Column generation has been the method of choice in several of the projects and hence receives significant attention in this chapter.

Heuristics are applied to find near optimal solutions of optimization problems and may be very different in their approach. In operations research, the term heuristic is used for methods which are not guaranteed to find optimal solutions. Heuristics are typically used for hard problems as a means to find high-quality solutions in reasonable time. Some heuristics mimic the corresponding manual planning process or apply simple rules to build and modify solutions. Such heuristics are typically easier to implement than optimization based methods and they may be the method of choice for that reason. Other heuristics are built from extensive theory and are applied mainly for their computational properties. In the following, we distinguish between four types of heuristics: Meta-heuristics, approximation algorithms, greedy heuristics, and optimization based heuristics.

A meta-heuristic is a method which combines a generic methodology with a problem specific setup. Meta-heuristics describe ideas and techniques in generic terms. For each problem, depending on the choice of heuristic, certain problem specific components have to be defined. With such definitions in place, the algorithm follows a procedure dictated by the meta-heuristic.

Several meta-heuristics have been proposed in the last 50 years. A group of these rely on the definition of neighborhoods. From a feasible solution (or in some cases an almost feasible solution) a neighborhood defines a set of similar solutions. Neighborhoods are attractive, as it is often possible to find better solutions in the neighborhood of already good solutions. The difference between neighborhood based meta-heuristics usually stems from their way of ensuring diversification, i.e. how effective they are at exploring large parts of the solution space. Examples of neighborhood based meta-heuristics are: simulated annealing (Kirkpatrick et al., 1983), tabu search (Glover, 1989), and greedy randomized adaptive search procedure (Feo and Resende, 1995). Other meta-heuristics are genetic algorithms (Holland, 1975) and ant colony algorithms (Dorigo, 1992). As the names indicate, many meta-heuristics take their names from the natural phenomenon that they have been inspired by. Meta-heuristics have not received much attention in this thesis. We refer to Glover and Kochenberger (2003) for a general introduction to meta-heuristics.

Approximation algorithms are applied to find near optimal solutions to specific optimization problems. Approximation algorithms are able to give a guarantee on the solution quality as well as the run time. They are, however, less generic and, as they are defined for a specific problem, are typically less useful for realistic practical problems. Hence, approximation algorithms have not been part of the focus of the work in this thesis. See Vazirani (2001) for more on approximation algorithms.

Greedy heuristics and optimization based heuristics are described in the following.

4.1 Greedy heuristics

A greedy heuristic is a simple algorithm, where decisions are made one after the other and when a decision has been made, it is fixed and will not be revised. This approach typically leads to very fast algorithms, but the solution quality may suffer from the restricted perspective in which decisions are made. A greedy heuristic which is used to build a new solution from scratch is also referred to as a construction heuristic.

Most heuristics that mimic manual planning are greedy, as this is a natural approach to manually solve realistic planning problems. A planner would typically know about critical parts of the problem and will therefore fix the most important decisions first. Local decisions are made with the rest of the problem in mind. Doing it this way, does not ensure that the complete solution is feasible, but it makes it more likely. The same is true for costs. The solution is obviously not guaranteed to be optimal, but even though decisions are made iteratively, their effect on future decisions is not disregarded completely. Greedy heuristics are usually designed to reasonably balance the immediate effects and the potential implied effects on future decisions. The heuristics are typically tailor-made for each application and as a result are not very versatile. For further details, we refer to Cormen et al. (2001).

4.2 Column generation

Column generation is a methodology developed specifically for optimization problems with certain characteristics. It has its foundation in mathematical programming and has been shown to perform extremely well on rostering and routing problems, because they naturally decompose into a master problem and a subproblem, as described in the following. Column generation considers a limited set of columns, which, depending on the context, refer to roster-lines, work-lines, paths, etc. A *master problem* combines the available columns to cover a given demand as tightly as possible. A *pricing problem* is called iteratively, to generate new promising columns. The iterative procedure may be continued until an optimal solution is found. Column generation can be embedded in a *branch-and-bound* structure and the resulting algorithm is referred to as a *branch-and-price* algorithm. We describe the main principles of column generation, based on its use in the context of rostering and task scheduling. We have deliberately chosen to keep the description specific to the focus of this thesis. For a general introduction to column generation, we refer to the extensive literature available in the area. See e.g. Barnhart et al. (1998) or Desrosiers and Lübbecke (2005) for an introduction to column generation.

4.2.1 The master problem

In Chapter 2, we defined the so called compact formulation of rostering and task scheduling problems. It is possible to use an alternative formulation, where they are formulated as the problem of finding good individual work-lines and subsequently combining these into the optimal overall rosters/schedules. The problem of combining work-lines can be modeled as a mixed integer program. In the following, we will treat rostering and task scheduling in a unified model.

Given is a set of groups/employees, $\mathcal{K}$, and a set of all tasks/shifts $\mathcal{I}$. Initially, assume that for each group, $k \in \mathcal{K}$, we have a set, $\mathcal{R}_k$, of all feasible work-lines for that group. Each task requires a certain number of workers, where a violation of this requirement may either be considered infeasible or may introduce a penalty. In many applications, exactly one worker is required per task.

Associated with each line of work, r, is a cost c_k^r. a_{kit}^r describes the individual lines of work, i.e. $a_{kit}^r = 1$ if task i is scheduled at time t in work-line r of group k, $a_{kit}^r = 0$ otherwise. A decision variable λ_k^r is introduced. λ_k^r specifies the number of workers of group k working line r. The relation to the variables of the compact formulation (2.7)-(2.10) of Chapter 2 is: $x_{kit} = \sum_{r \in \mathcal{R}_k} a_{kit}^r \lambda_k^r$. In some applications under-coverage and over-coverage is allowed. μ_i^- and μ_i^+ are decision variables representing the amount of under-coverage and over-coverage, which comes at a cost, which is denoted c_i^- and c_i^+, respectively. μ_i^- and μ_i^+ are identical to the corresponding variables of the compact formulation.

With the introduced parameters and variables, we can formulate the task scheduling problem below. The model corresponds closely to model (2.7)-(2.11) of Chapter 2. The cost is linear in the cost of the work-lines plus the slack/surplus costs. This is a specialization of the cost defined in (2.7). Nonetheless, it can be used for all applications that we consider. We refer to the model as the *master problem.*

$$\min \sum_{k \in \mathcal{K}} \sum_{r \in \mathcal{R}_k} c_k^r \lambda_k^r + \sum_{i \in \mathcal{I}} \left(c_i^- \mu_i^- + c_i^+ \mu_i^+ \right) \tag{4.1}$$

$$\text{s.t.} \quad \sum_{r \in \mathcal{R}_k} \lambda_k^r = m_k \qquad \forall k \in \mathcal{K} \tag{4.2}$$

$$\sum_{t \in \mathcal{T}} \sum_{k \in \mathcal{K}} \sum_{r \in \mathcal{R}_k} a_{kit}^r \lambda_k^r + \mu_i^- - \mu_i^+ = b_i \qquad \forall i \in \mathcal{I} \tag{4.3}$$

$$\sum_{r \in \mathcal{R}_{k_1}} a_{k_1 i t_i}^r \lambda_{k_1}^r + \sum_{r \in \mathcal{R}_{k_2}} a_{k_2 j t_j}^r \lambda_{k_2}^r \leq 1 \qquad \forall k_1, k_2 \in \mathcal{K}, \forall (i, j, t_i, t_j) \in \mathcal{T}^x \tag{4.4}$$

$$\lambda_k^r \in \mathbb{Z}_+ \qquad \forall k \in \mathcal{K}, \forall r \in \mathcal{R}_k \tag{4.5}$$

$$\mu_i^- \geq 0, \mu_i^+ \geq 0 \qquad \forall i \in \mathcal{I} \tag{4.6}$$

In this model, it is assumed that only the individual costs of work-lines together with costs from under-coverage and over-coverage, contribute to the objective function (4.1). Constraints (4.2) enforce the inclusion of the correct number of workers of each group. Constraints (4.3) ensure that all tasks are allocated and correspond directly to Constraints (2.8) of the compact model. Constraints (4.4) describe the temporal dependencies and correspond to Constraints (2.11). Constraints (4.5) and (4.6) set the domains of the variables.

We observe that (4.1)-(4.6) is a mixed integer program. This is essential for column generation to be applied. However, column generation is based on linear programming and we therefore need to relax the integer variables in order to obtain a linear programming (LP) problem, referred to as the *relaxed master problem.* In the relaxed master problem, Constraints (4.5) are replaced by:

$$\lambda_k^r \geq 0 \qquad \forall k \in \mathcal{K}, \forall r \in \mathcal{R}_k \tag{4.7}$$

Another important observation is that the sets of work-lines, $\mathcal{R}_k$, are prohibitively large. In all practical applications the sets are so large that it is impossible to represent them explicitly. The idea in column generation is to add work-lines to a restricted set of work-lines, $\mathcal{R}_k^{'}$, when they are needed. A *pricing problem* is solved in order to identify work-lines, which may contribute to the full schedule. Columns corresponding to variables μ_i^- and μ_i^+ are represented explicitly. The relaxed master problem with a restricted set of work-lines, $\mathcal{R}_k^{'}$, is denoted the *restricted master problem.* Formally, column generation is a technique to find the optimal solution to the relaxed master problem by repeatedly solving the restricted master problem. All columns are implicitly considered and added when needed.

As the restricted master problem is an LP-problem, we are able to utilize fundamental simplex theory to solve it. Consider a general LP-problem ($\mathcal{P}$): $z = \min\{c^\top x | Ax \geq b \wedge x \in \mathbb{R}_+^n\}$. For a basic feasible solution of $\mathcal{P}$, it is possible to calculate the reduced cost of all variables as $\bar{c} = c - c_B B^{-1} A$, where c is the original cost of the variables. c_B is the original cost of the current basic variables and B is the current basis matrix. The value of the dual variables are $y = c_B B^{-1}$ and hence the reduced costs can be calculated as $\bar{c} = c - yA$. Consequently, for a single variable x_j the reduced cost is $\bar{c}_j = c_j - \sum_{i \in \mathcal{I}} y_i A_{ij}$. In the simplex method, variables with negative reduced cost are iteratively added to the basis until no more negative reduced costs exist. Therefore, in order to solve the relaxed master problem to optimality, we must be able to identify such variables, if they exist. In column generation, this is exactly the purpose of the pricing problem. For any variable x_j with cost c_j and the column of matrix coefficients $A_{\cdot j}$ the calculation of $\bar{c}_j$ is straight forward. The challenge is to implicitly consider all variables, without having to represent them explicitly.

We refer to Hillier and Lieberman (2001) for a detailed exposition of simplex theory.

For the restricted master problem, we are able to find an optimal primal solution (λ, μ^-, μ^+) along with a corresponding dual solution (τ, π, σ). τ, π, and σ are the dual variables of Constraints (4.2), (4.3), and (4.4), respectively. A variable λ_k^r has reduced cost $\bar{c}_k^r$:

$$\begin{aligned}\bar{c}_k^r = c_k^r - \Bigg(& \tau_k + \sum_{i \in \mathcal{I}} \sum_{t \in \mathcal{T}} a_{kit}^r \pi_i \\ & + \sum_{k' \in \mathcal{K}} \sum_{(i,j,t_i,t_j) \in \mathcal{T}^x} \left(a_{kit_i}^r \sigma_{kk'(i,j,t_i,t_j)} + a_{kjt_j}^r \sigma_{k'k(i,j,t_i,t_j)} \right) \Bigg) \end{aligned} \tag{4.8}$$

When considering a problem with $b_i = 1, \forall i \in \mathcal{I}$ and $\mu_i^- = \mu_i^+ = 0, \forall i \in \mathcal{I}$ a compact formulation of Constraints (4.4) may be used:

$$\sum_{k \in \mathcal{K}} \sum_{r \in \mathcal{R}_k} \left(a_{kit_i}^r \lambda_k^r + a_{kjt_j}^r \lambda_k^r \right) \leq 1 \qquad \forall (i,j,t_i,t_j) \in \mathcal{T}^x \tag{4.9}$$

This in turn gives a slightly simplified reduced cost calculation:

$$\bar{c}_k^r = c_k^r - \left(\tau_k + \sum_{i \in \mathcal{I}} \sum_{t \in \mathcal{T}} a_{kit}^r \pi_i + \sum_{(i,j,t_i,t_j) \in \mathcal{T}^x} \left(a_{kit_i}^r + a_{kjt_j}^r \right) \sigma_{(i,j,t_i,t_j)} \right) \tag{4.10}$$

As described earlier, some problems do not have temporal dependencies and hence have no constraints of Type (4.4). Also, for problems with temporal dependencies, we may choose to relax Constraints (4.4), and let the constraints be enforced by e.g. the branching scheme instead of by the master problem. In both cases, the relaxed master problem consists of Constraints (4.1)-(4.3),(4.6),(4.7) and the reduced cost calculation becomes:

$$\bar{c}_k^r = c_k^r - \left(\tau_k + \sum_{i \in \mathcal{I}} \sum_{t \in \mathcal{T}} a_{kit}^r \pi_i \right) \tag{4.11}$$

4.2.2 The pricing problem

From a current dual solution of the restricted master problem, the pricing problem finds one or more variables with negative reduced cost to enter the basis or proves that no such variables exists. One way of doing that is by solving the optimization problem:

$$\min \quad \bar{c}_k(x_k) \tag{4.12}$$

$$\text{s.t.} \quad x_{k\cdot\cdot} \in \mathcal{X}_k \subseteq \mathbb{B}^{|\mathcal{I}||\mathcal{T}|} \qquad \forall k \in \mathcal{K} \tag{4.13}$$

Constraints (4.13) define the feasible schedules of each worker like Constraints (2.9) of the general model in Chapter 2.

For all task scheduling problems considered in this thesis, it is possible to formulate the pricing problem as a number of *resource constrained shortest path problems*. Nodes represent tasks/shifts and resources are used to track attributes like time and capacity. Each task i is represented by a node with cost $-\pi_i$. The reduced cost contributions from Constraints (4.4) or (4.9) depend on resource consumption. Two additional nodes are introduced, namely a source node and a sink node. A shortest path problem is formulated for each group $k \in \mathcal{K}$. A fixed cost, $-\tau_k$, applies to all columns of the group and is typically set as a cost in the source node. The problem is to find a feasible path from the source to the sink which minimizes the total cost.

The resource constrained shortest path problem is usually solved with a label setting algorithm built on the concept of dynamic programming. In short, the algorithm keeps track of a set of labels representing partial paths trough the graph. Labels are compared and unpromising labels are eliminated along with infeasible ones. The promising labels are extended to longer partial paths and ultimately to full paths. Desrochers et al. (1992) presented a dynamic algorithm for the non-elementary version of the problem. The algorithm was adapted to the elementary resource constrained shortest path problem by Feillet et al. (2004). An important contribution to the acyclic variant of the problem is described by Dumitrescu and Boland (2003). See Irnich and Desaulniers (2005) for a recent survey of the literature on resource constrained shortest path problems.

The definition of the subproblem varies greatly from case to case and the methodology is usually customized to fit with individual settings. In this thesis, when column generation is used, the corresponding pricing problem is described individually for the specific setup. As the solution of the pricing problem usually constitutes a significant part of the total solution time, it is important to have as efficient methods as possible. This is achieved by the use of efficient customizations for individual problems building on top of a well founded basic algorithmic setup.

4.2.3 Branch-and-bound

So far we have focused on the solution of the relaxed master problem. Ultimately, we want to solve the original mixed integer problem. Therefore, the column generation procedure is embedded in a *branch-and-bound* framework, which will reintroduce the constraints that were relaxed. In branch-and-bound, a dynamic decision tree is built. Starting from a root node, the solution space is split whenever the optimal solution of the relaxed formulation is found to be infeasible in the original formulation. Each of the subspaces is stored in a child node of the root node and the corresponding restricted problem is solved in the child node. A complete branching scheme has the property that the relaxed solution space is split into two or more subsets, where the union of the subsets contains all feasible solution of the original formulation. The current infeasible solution is not in any of the subsets and hence each of the subsets will provide a new solution, which is either feasible or where another branching decision can be applied. A bounding scheme limits the amount of fruitless work. For a detailed exposition of branch-and-bound, we refer to Wolsey (1998).

In a standard column generation setup, only the integer properties of some variables are relaxed, e.g. Constraints (4.5). As mentioned earlier, in this thesis, it has also been tested to relax interconnecting constraints, i.e. Constraints (4.4). Either way, the relaxed constraints can be implicitly enforced by applying branching decisions to the problem. Branching decisions are applied repeatedly until no violations of the original constraints exist.

When column generation is included in branch-and-bound, it is usually referred to as *branch-and-price*. It is also possible to reintroduce relaxed constraints or generate new ones, by the generation of *cuts*. An example of constraint reintroduction can be found in Paper E. When using both column- and cut generation, the approach is referred to as a *branch-and-cut-and-price* method. The flow of a branch-and-cut-and-price algorithm has been sketched in Figure 4.1. See Ralphs et al. (2003) for an introduction to branch-and-cut-and-price algorithms and their implementation.

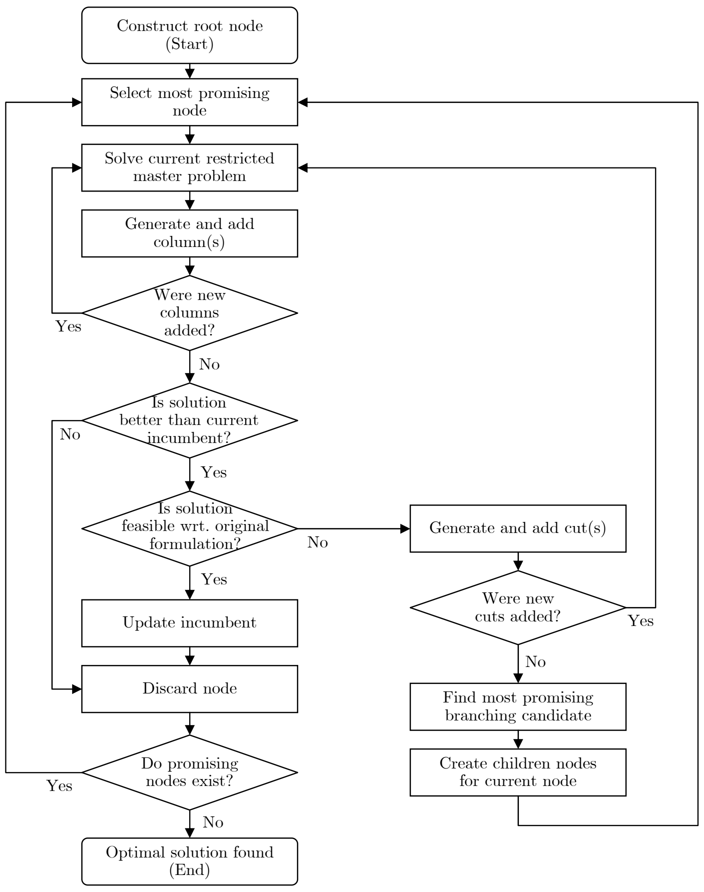

Construct root node (Start)
Select most promising node
Solve current restricted master problem
Generate and add column(s)
Were new columns added?
Yes
No
Is solution better than current incumbent?
No
Yes
Is solution feasible wrt. original formulation?
No
Yes
Generate and add cut(s)
Were new cuts added?
Yes
No
Update incumbent
Discard node
Find most promising branching candidate
Create children nodes for current node
Do promising nodes exist?
Yes
No
Optimal solution found (End)

Figure 4.1: Flowchart of a branch-and-cut-and-price algorithm. The flowchart of a branch-and-price algorithm is identical, where the processing step "Generate and add cut(s)" has no effect.

4.2.4 Optimization based heuristics

Traditionally, LP-based methods are used to find the exact solution of optimization problems. As we turn our attention towards practical problems, it is usually ineffective to solve problems to optimality. In this thesis, various approaches have been applied to achieve significant speedups. When heuristic speedups are applied, the methods lose the theoretical property of providing an exact solution and hence compete on equal terms with other heuristics. In a practical context, the lost theoretical property is seldom of concern. If exact solutions are required, heuristics may still be used at an early stage in the algorithm. They must eventually be replaced by exact components to prove optimality.

One possible heuristic approach is to remove parts of the solution space which appear to be unattractive, before initiating the algorithm. If we disallow certain unpromising combinations of workers and tasks, the sizes of the subproblems decrease and the generation of columns is accelerated. Another positive effect on solution time is that the diversity of columns decreases, which in turn leads to a less fractional LP-solution and hence to less branching. However, decreased diversity may also reduce solution quality, and the excluded parts of the solution space must therefore be chosen carefully. A workaround to this is an iterative method, where the solution space is reduced significantly from the outset of the algorithm. Certain parts of it are later reintroduced, when it becomes clear which features the best solutions are likely to have. The algorithm finds solutions in the reduced solution space, along with a list of complications which may be resolved by reintroducing parts of the solution space. See Paper D for an example. Other authors applying this idea are e.g. Rezanova and Ryan (2010).

Premature branching and pruning can be introduced to speed up the solution of the master problem. Traditionally, the optimal solution of the restricted master problem is found in each iteration. If the simplex algorithm contributes with a significant part of the total solution time, it may be preempted to decrease solution time. Another option is to declare the master problem solved even when more variables could be priced out. The solution value of the master problem often converges much slower, when it is close to the optimal value, and premature branching and pruning may be applied to avoid this effect. If bounds are updated appropriately, optimal solutions may still be provided ultimately. The technique is applied in Paper A.

Heuristic pricing may introduce significant speed improvements in the pricing problem. First of all, we do not need to always find the least cost variable. What we do need in order to prove optimality of a solution is a sound certificate for the non-existence of variables with negative reduced cost. This can be done by using exact pricing only when heuristic pricing fails. If exact pricing fails to produce columns with negative reduced cost, it is because there are no such columns. This idea is applied in Paper C. We may also choose to compromise overall optimality and use only heuristic pricing. In Paper A, this approach gave very good results. It has previously also been applied in vehicle routing with success, see e.g. Savelsbergh and Sol (1998) or Chabrier (2006).

Another way to speed up the algorithm is by limiting the part of the branch-and-bound tree which is explored. Nodes of the branch-and-bound tree are removed heuristically to allow for a quicker exploration of the tree. The selection may be based on bounds or other features of the branching decision. In the extreme case, we may apply a greedy search of the branch-and-bound tree, where the algorithm from the outset dives in the branch-and-bound tree. This leads to a much faster algorithm, but may also have a very bad influence on solution quality. The greedy approach has often been used in the literature, especially for rostering; see e.g. Day and Ryan (1997). See Paper B for a successful application in this thesis.

Paper B also describes a decomposition of the problem, where a rolling horizon is considered. A limited part of the full horizon is considered and a solution is found to only that part of the problem. The rolling horizon is pushed forward and partial problems are solved iteratively. Finally, the partial solutions are merged to form a solution of the full problem. Similar ideas have been described in the literature by e.g. Eveborn and Rönnqvist (2004).

The suggested heuristic components may be combined to achieve better run times than what can be obtained by each of them individually. As solutions are no longer guaranteed to be optimal, it may be valuable to subsequently reoptimize in order to improve the found solution. To summarize the possible heuristic improvements, we list the elements of a traditional branch-and-price framework along with the heuristic modifications that may be introduced.

- Problem description: Define a reduced solution space or alternatively decompose the problem and merge parts to a full solution.
- Master problem: Premature branching and pruning.
- Pricing problem: Heuristic pricing.
- Branch-and-bound: Heuristically remove parts of the tree.

CHAPTER 5

Papers of Part II

In the following, we describe the papers which together constitute the major part of this thesis. The problem instances considered were introduced in Chapter 3. Here, we describe the implemented solution approaches, results and conclusions.

5.1 Paper A: A Nested Column Generation Based Approach to the Generalized Rostering Problem using Compile-time Customization

This paper describes a solution approach to the generalized rostering problem. The paper builds on a comprehensive technical report (Dohn et al., 2010*a*), which describes the setup in more detail. Implementation details, in particular, are discussed thoroughly and the technical report makes a good starting point, if one wants to build a solution algorithm based on the ideas presented. The work is also described in the conference proceedings of ORSNZ'09 (Mason et al., 2009).

A column generation approach is applied to solve a generalized rostering problem. The setup is generic and can therefore solve a wide variety of rostering problems. Problem specific knowledge is included using an attribute-based description of rules and costs. This idea is similar to that described by Kohl and Karisch (2004) for a commercial system. A traditional decomposition of the problem is used, where feasible roster-lines for individual employees are combined using a generalized set covering model solved with column generation. The set covering model is generic by nature and seldom needs modification to fit with any realistic rostering problem. Contrary to the master problem, the roster-line generation in the pricing problem needs to be customized for each individual problem. The idea, presented in the paper, is to automatically include the attribute-based description of rules and costs at compile time. The description is parsed into the code and in this way the algorithm is customized at compile-time. Any problem describable with the attribute-based language can hence be solved, solely by parsing the new description to the compiler.

Three different practical nurse rostering problems are solved to illustrate the versatility of the setup. In a former report by Engineer (2003) it is shown that the setup easily models rostering problems from other industries as well. The test results from the three instances are very promising. High-quality solutions are found in less than 15 minutes for all three. The algorithm can be setup to prioritize exact solution over runtime and optimal solutions are found for two of the three instances. The algorithm is compared to a former approach, where customization happens at runtime and the results show that the compile-time customization is on average 20 times faster, for the specific instance available to both implementations. A better bounding procedure for the subproblem is also presented, and the test shows that the number of labels in roster-line creation is reduced by more than 60%.

The contribution of the paper is mainly in the novel approach to the generalized rostering problem. The idea allows an approach which is more versatile than what has been seen before. Any constraint and cost observed in the literature can be modeled in the attribute-based setup and the corresponding instances are therefore solved without any further modifications necessary in the algorithm. Not only is the method more versatile, it is also much more efficient in its implementation and significant speed ups compared to a similar former version of the algorithm are observed. In a survey of the literature, Ernst et al. (2004*b*) point out that:

> *Another important area requiring further work is generalisation of models and methods. Currently, models and algorithms often require significant modification when they are to be transferred to a different application area, or to accommodate changes within an organisation.*

The presented approach alleviates exactly this problem.

5.2 Paper B: An Integrated Approach to the Ground Crew Rostering Problem with Work Patterns

A crew rostering problem from a major European airline is considered in Paper B.

The problem considered may be seen as a specification of the generalized rostering problem. However, the focus of the paper is slightly different from that of Paper A. Like in the generalized rostering project, the rostering problem is solved by column generation on a generalized set covering master problem with a resource constrained shortest path subproblem. An integrated approach is applied, where the rosters are generated directly on the estimated workload, while including robustness. The number of employees is large and the variables are not restricted to binary values. This makes it very hard to apply a traditional branching. Instead a greedy column bounding scheme is applied, where the value of fractionally selected columns are iteratively pushed upwards to integer values. This approach is much faster than a traditional branch-and-price approach, but it comes at the cost of a decreased solution quality. As the roster period is half a year, modeling the problem as a single set covering problem makes the problem intractable. Instead, the roster period is decomposed into blocks which are solved iteratively. Each block is solved with column generation and integer solutions are found for the block. Overlaps between the blocks ensure continuity throughout the roster period. Decisions from one block fix some of the decisions in the next block. The decomposition makes the problem tractable, but solution quality may again be affected.

To evaluate the efficiency of the approach, a number of test cases are introduced. Three realistic instances from a major European Airline are available. From these, another 10 random instances are generated. The tests clearly illustrate the capability of the approach. On the three realistic instances, the workload is covered almost perfectly and robustness is built into the solution. It is showed that it is even possible to cover the workload well with fewer people, and hence increase efficiency. The results indicate that neither the decomposition into blocks nor the aggressive variable fixing affect the solution quality significantly.

The contribution of the paper is in the solution of an interesting real-life problem as well as in the development of efficient new heuristic techniques that decrease the needed solution time significantly without compromising solution quality considerably. The integration of demand estimation and rostering by directly creating the roster from a workload estimation is novel and the results indicate that it is a very valuable idea. The objective function includes a robustness measure which is easily incorporated into the model. By introducing a large set of shifts that the rostering process can choose from, the cover is fitted nicely to the estimated workload and includes robustness, as well.

5.3 Paper C: The Manpower Allocation Problem with Time Windows and Job-Teaming Constraints: A Branch-and-Price Approach

Paper C describes the manpower allocation problem with time windows and job-teaming constraints. The paper has been published in Computers & Operations Research. An early version of the paper exists as a technical report (Dohn et al., 2007*c*). Furthermore, the project has been presented at several conferences and is described in conference proceedings of ICAPS'07 (Dohn et al., 2007*a*) and ORSNZ'08 (Dohn and Kolind, 2008). Finally, the project is also described in the journal of the Danish Operations Research Society (Dohn and Kolind, 2009).

A compact mathematical model is presented along with a Dantzig-Wolfe decomposition of the model. The approach is inspired by a similar successful approach to VRPTW. The decomposition results in a master problem with the structure of a generalized set covering problem and a pricing problem that can be solved as a resource constrained shortest path problem. The LP-relaxation of the master problem is solved with a standard LP-solver. Synchronization between tasks is enforced by time window branching. Integrality is further enforced by a second branching scheme, where employee/task allocations are forced to binary values. The pricing problem is solved with a label setting algorithm.

The computational results are based on 12 full-size realistic instances from ground handling companies in two of Europe's major airports. The focus is on solution optimality, rather than on solution speed, and for the same reason, the timeout limit is 10 hours. In 11 of the 12 instances, the optimal solution is found. In the remaining case, the best solution is very close to optimality. Interestingly, only half of the optimal solutions could be found in an initial run. When the algorithm is restarted with a good initial solution, the method is able to make a better choice on the branching candidates resulting in quick solution of all but one instance.

The main contributions of the paper are a novel approach to MAPTWTC, which is able to find optimal solutions to full-size realistic instances in almost all cases. Particularly, the handling of synchronization constraints between tasks had not been approached before in an exact optimization context. The paper also contributes with some algorithmic insights. It is discussed how to evaluate branching candidates against each other and how to select the best possible branching candidates. Also, the paper presents improvements to column generation with non-identical subproblems. It is concluded that the approach is useful, not in a real-time setting, but as a pre-operational tool or as a benchmarking tool for real-time heuristics.

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

The home care crew scheduling problem is described in Paper D. Preliminary results were presented at the ICAOR'08 conference (Dohn et al., 2008*c*), where the paper was awarded with the Best Paper Award. A similar description of the project is found in the journal of the Danish Operations Research Society (Dohn et al., 2008*b*).

The approach taken to solve the HCCSP is similar to the one presented in Paper C. The problem is modeled as a generalized set covering problem and column generation is used to solve it. The structure introduced, by having a limited number of preferred caretakers for each citizen, is utilized in the algorithm, where the network of each caretaker is initially limited to the citizens of which they are preferred. This introduces a significant speedup, but it is not always possible to service citizens with preferred caretakers and therefore citizens are added iteratively to networks of all caretakers, when allocation issues are detected.

The computational test are carried out on a total of 94 realistic instances, including four real-life instances from Danish municipalities, 60 similar instances generated from the four real-life instances and another 30 made available by Bredström and Rönnqvist (2007). The tests compare a number of different settings with and without the clustering scheme and it is clear that the clustering of visits decreases solution time. However, it is also apparent that there is a tradeoff between getting lower solution times and better solution quality. The tests show that by using a cluster with only preferred visits, the run time can be lowered by $50 - 70\%$. With the right settings, the clustering entails a quality loss only in a few instances.

The contribution of the paper is the proposed solution algorithm to the HCCSP and in particular the clustering of visits which results in a subproblem network reduction with iterative expansion. Furthermore, the paper presents a solution method to a practical problem, which is able to produce solution superior to current best practice in an acceptable time.

5.5 Paper E: The Vehicle Routing Problem with Time Windows and Temporal Dependencies

The vehicle routing problem with time windows and temporal dependencies is described in Paper E and in a preceding technical report (Dohn et al., 2009*a*).

The findings from Paper C and Paper D are transferred to generic settings in vehicle routing, considering academic benchmark instances. It is shown that most temporal dependencies can be modeled by generalized precedence constraints. Two different compact mathematical formulations are proposed. One is a traditional VRPTW formulation extended with generalized precedence constraints modeled with a continuous time-variable for each customer. The other is a time-indexed formulation containing solely binary variables. Dantzig-Wolfe decompositions of the compact formulations are presented and four master problem formulations are derived: the direct formulation, the full time-indexed, the limited time-indexed, and the relaxed formulation. The formulations can be ranked according to the tightness with which they describe the solution space. Any of the master problems allows for a solution method based on column generation with a resource constrained shortest path subproblem. A thorough exposition of time window branching is given along with an essential time window reduction technique. Time window branching is used as a means to remove violations of generalized precedence constraints as well as to enforce integrality. It is shown that time window branching with generalized precedence constraints is theoretically as strong as the specialized synchronization branching used in Paper C.

A set of test instances are introduced to evaluate the computational value of each of the models and to assess the consequence of adding temporal dependencies. The test instances are generated from a benchmark data set from the literature with no temporal dependencies, where the dependencies are added gradually. The direct formulation is not included in the tests, as the implementation is not straight forward and as it is expected to perform worse than the other three formulations. The results clearly show that the full time-indexed formulation performs worse than the remaining two. The performance of the limited time-indexed formulation and the relaxed formulation are comparable.

The contribution of the work is in the generic approach to temporal dependencies. In contrast to previous work, this paper describes temporal dependencies in an academic context on modifications of well known benchmark instances. Temporal dependencies are modeled with generalized precedence constraints and the proof that synchronization modeled with two generalized precedence constraints is as strong as a specialized modeling is a significant contribution. Furthermore, the introduction of a time-indexed model is novel and even though the results of the model are comparable to those of the relaxed model, it has some properties that may be utilized in future work.

5.6 Paper F: Optimizing the Slab Yard Planning and Crane Scheduling Problem using a Two-Stage Heuristic

The slab yard planning and crane scheduling problem is described in Paper F. An extensive technical report (Dohn and Clausen, 2008*b*) describes details that were not included in the paper. Preliminary results were presented at the conference ICAPS'07 (Dohn, 2007).

As the name indicates, the slab yard planning and crane scheduling problem can be considered in two stages, namely a planning stage and a scheduling stage. In the planning stage, a greedy heuristic generates move-actions, which takes the slab yard from an initial state to a desired goal state. In the goal state, all slabs that are needed in production have been moved to a roller table, on which they will leave the yard. Also, there are incoming slabs which need to be stored in the stacks of the yard. Additional moves, the so called reshuffle actions, may be needed in order to access slabs which are not on top of their stack. From the generated plan of move-actions, the crane scheduling problem can be formulated. The cranes are regarded as machines in a traditional machine scheduling problem, where the move-actions are the tasks of the machines and disjunctive generalized precedence constraints are introduced to avoid crane collision in the schedule. Another greedy heuristic is applied to do the scheduling. Tasks are considered one by one and are scheduled as early as time windows and generalized precedence constraints allow. Mandatory tasks that will serve the production line with slabs are prioritized. When time allows it, the schedule will include tasks that move incoming slabs to the yard and tasks that reshuffle stacks in the yard preparing the yard for future production.

The introduced two-stage approach is compared to a simulation of manual planning. The simulation is setup to imitate the behavior of the cranes when they are under the control of the individual crane operator. The results of the two-stage approach are significantly better than those of the simulated manual planning. Especially, when the throughput of the yard is gradually increased, the two-stage approach is consistently performing well, while the simulation incurs more and more serious deadline violations.

The main contribution of the paper is a novel modeling of the slab yard planning and crane scheduling problem and a corresponding solution approach. The setup of especially the scheduling problem is not straight forward and requires careful calculation of parameters for the generalized precedence constraints. The structured approach and in particular the decomposition of the problem in two stages facilitates a transparent evaluation of the schedule quality.

Chapter 6

Conclusions

In the following, the main findings of the thesis will be summarized, and the contributions will be highlighted. Finally, we will give our opinion on the most promising directions for future work.

In this thesis, the modeling and solution of complex planning problems in rostering and task scheduling has been described. The work is based on six specific applications. All applications are inspired by industrial problems and in most cases, the analysis and conclusions are based directly on data from the industry. Manpower planning can be seen as a sequence of planning stages, starting with long term strategic planning and ending on the day of operation with real time disruption management. As the title of this thesis indicates, the focus has been on the planning stages in the mid term and short term planning, namely on rostering and task scheduling. Also, to keep a confined focus, the thesis has been limited to certain applications with specific properties.

Two different approaches to rostering were considered. In the first (Paper A), a generic and very versatile setup was described. The model incorporates most realistic rostering problems by being able to include their definition from an external problem description and by not including any problem specific features in the algorithm. The goal of this work was to create an approach which can solve almost any realistic rostering problem, while keeping performance comparable to similar customized methods. The computational tests on three nurse rostering problems clearly demonstrate the versatility of the setup and at the same time show the strength of the approach by its ability to provide applicable rosters of high-quality. The versatility of the setup is further confirmed in a description of all common rules found in the literature. It is therefore concluded that the rather ambitious goal is definitely within reach. The solution speed has to be increased for the algorithm to be fully comparable to state-of-the-art heuristic methods.

The second rostering project (Paper B) had a slightly different focus. The versatility of the setup was not given priority. Instead, a number of surrounding features and extensions were examined. The roster period was extended to half a year, which made it intractable to create a full roster at once. Instead, the full period was divided into overlapping blocks and the problem was solved by considering blocks sequentially. The focus of the project was extended from the traditional rostering context to include shift demand estimation as part of the decision problem. It was established how to quantify demand so that robustness of the rosters was also ensured. Instead of using a predetermined shift demand, the estimated workload was used directly in the rostering process and the process of generating shift demands from the workload was circumvented. The immediate advantage of combining the two stages was a better workload cover, where a roster with the given cover is directly available. The solution time was in all cases acceptable and the quality of the rosters was significantly better than the quality of the rosters which are currently used in practice. Furthermore, it was shown that the solution values were in all cases very close to an established lower bound.

Two variants of manpower allocation problems with temporal dependencies were considered in the thesis. The first problem (Paper C) describes a practical setting from ground handling in airports. A large number of tasks must be allocated to teams of workers respecting time and skill prerequisites. Job-teaming requirements introduce synchronization constraints between teams and the fact that the workforce is not big enough to take on all tasks makes the objective one of minimizing the number of unallocated tasks. Synchronization was relaxed in the initiation of the solution algorithm and reinforced by the branching scheme. The results indicate that the approach is viable. In close to all instances, the optimal solution is found. However, the current solution times prohibit a real-time application of the approach. The solution quality and theoretical properties of the method make it ideal as a pre-operational tool and for benchmarking purposes.

The second manpower allocation application (Paper D) is in home care crew scheduling, where home care personnel is allocated to practical tasks in the homes of elderly citizens. The application represents an extension to the former project in two concerns. The synchronization between tasks is extended to other temporal dependencies. Furthermore, the citizens have preferences on care takers and this is utilized in a problem reduction, where the set of feasible care takers for each citizen is limited to the preferred ones. The set is iteratively expanded when needed. The computational results clearly show that the approach is capable of producing high-quality solutions. Furthermore, it is apparent that the problem reduction constitutes a significant enhancement.

In another project (Paper E), the findings of the two manpower allocation problems are generalized to a generic context of vehicle routing with time windows and temporal dependencies. Numerous benchmark instances are generated with varying characteristics. Generalized precedence constraints are applied to model temporal dependencies. The computational analysis serves two purposes. It is an assessment of the capabilities of the proposed models in the vehicle routing context. At the same time, it is a computational analysis of the instances and their properties. It is shown that the best performance is achieved with the so called, relaxed model, or the limited time-indexed model. Further, the limited time-indexed model has some nice theoretical properties. It is concluded that the temporal dependencies introduce additional complexity to the problems, as expected.

The final application (Paper F) is the slab yard planning and crane scheduling problem. The problem is slightly different from the other ones considered. The focus of the project is in the modeling of the application and a greedy heuristic is applied in computational tests. A novel two-stage approach is presented, where the planning problem and the crane scheduling problem are considered individually. It is shown that the structured approach to the problem is significantly better than a simulation of manual planning. Not only does it give better solutions, it also allows for a much more transparent management of the slab yard, where problems and bottlenecks can be identified earlier.

In summary, six individual projects have been considered in the thesis. A generalized approach to rostering has been examined with great success and extensions and variations have been explored. The approach, based on column generation, proved to be extremely versatile while also producing superior results. Furthermore, it was shown that the approach adapts well, when the prerequisites and objectives are slightly modified. The focus in the task scheduling problems was confined to problems with certain characteristics, namely task scheduling problems with temporal dependencies between tasks. Again, a column generation based approach proved to be of high value. It was shown how to model various realistic task dependencies in a unified manner and high-quality solutions were found for almost all realistic instances. However, the approaches are in none of the above cases able to produce solutions as fast as the best available heuristics. On the other hand, the approaches are able to provide solutions of superior quality and being based on linear programming theory, the optimality of many solutions can be proved. The last project had a slightly different focus, but clearly illustrates the value of a structured and well-founded approach to the slab yard planning and crane scheduling problem.

6.1 Main contributions

The main contributions of this thesis are primarily the contributions of the individual projects and are therefore also discussed in the corresponding papers. Below, the most important contributions are listed.

- A versatile approach to generalized rostering has been developed. The approach represents state-of-the-art in both versatility and solution quality. The value is validated on three realistic nurse rostering instances from the industry.
- On top of its immediate value for the rostering application, the compile-time customization constitutes a contribution in itself. The idea illustrates how generalized approaches to optimization problems can be built so that extensions and variations are easily accommodated. The generalization introduces only minor challenges in the development.
- A roster block structure with partial overlaps has been developed. The blocks allow solutions of rostering problems with long roster periods, where solution time grows only linearly with the size of the problem. The solution quality of the test instances was only marginally affected by the decomposition.
- An integrated approach to shift demand estimation and rostering has been presented. The approach applies the rostering algorithm directly on the workload estimation. The integration proved to be of high value in the realistic instances tested.
- A general modeling of temporal dependencies in task scheduling problems has been introduced. When a presented time reduction technique is applied, it is shown that inclusion of synchronization in the general modeling scheme is as tight as a customized modeling of synchronization.
- An iterative network expansion technique is developed for manpower allocation problems with preferences. The technique allows for solution of larger instances and only compromises solution quality slightly.
- The solution of realistic task scheduling problems in ground handling and home care is a contribution in itself. A novel methodology for the problems

was developed and it was shown that it was capable of providing high-quality solutions.

- A novel, structured approach to the slab yard planning and crane scheduling problem constitutes an important contribution, as well. The computational results clearly indicate the high value of a structured approach based on mathematical and logic-based models.

6.2 Future work

Future work can follow a number of different directions. For all projects, it would be interesting to include more industrial applications. Realistic applications have the ability to validate the computational capabilities of the methods. At the same time, they typically inspire relevant extensions and variations of the problem under consideration. It is particularly important to include additional problems for generalized rostering, where the ultimate goal is to capture the features of all realistic rostering problems. Only by examining a large number of diverse problems can the approach truly prove its versatility. Including additional realistic instances will also contribute to the identification of areas where algorithmic improvements are needed. For the task scheduling problems, it would be interesting to examine how the models behave for other applications. The setup for the slab yard planning and crane scheduling problem has not been tested directly on any industrial instances and this is naturally of high importance in potential future development.

Another prospective improvement which will contribute to all the column generation based methods of the thesis, is the development of column generation based heuristics. One approach is to adapt well known mixed integer programming (MIP) heuristics to the dynamic structure of column generation. Examples of MIP-heuristics are Local Branching (Fischetti and Lodi, 2008), Relaxation Induced Neighbourhood Search (Danna et al., 2005), Pivot and Shift (Balas et al., 2004), and Feasibility Pump (Fischetti and Salvagnin, 2009). The adaption will result in a framework having the modeling capabilities of column generation but with improved computational performance. The ability of providing exact solutions will be sacrificed to allow for a significant speed improvement. Careful research is needed to ensure that the exceptional solution quality inherent in column generation methods is not compromised more than absolutely necessary.

Another promising direction for enhancing the performance of the column generation based methods, is in the advancement of the subproblem solution algorithms. Any improvement of the algorithm transfers directly to the performance of the full column generation and it is therefore worth considering. An area, which has received limited attention, concerns heuristic solutions of subproblems, see e.g. Savelsbergh and Sol (1998) or Chabrier (2006). As described for MIP-heuristics, the ability to provide provably optimal solutions may ultimately be given up. However, for the subproblems, another option exists. By solving subproblems heuristically only in parts of the process, where no theoretical certificates are needed, the full algorithm profits from a speed enhancement without losing solution quality. Finally, an addition which is often used together with column generation is dynamic cut generation. Cut generation has not been given much attention in this thesis and it is therefore an interesting direction of development. The introduction of cuts was predicted to have a considerable effect, especially for the vehicle routing problem with time windows and temporal dependencies.

A very interesting perspective in rostering is to combine the two projects presented in this thesis. The conclusions from the rostering of ground crew transfer to the generalized settings of the first rostering project. Hence, the extension of the roster period and the broadening of the problem focus, to consider workload directly, can be included in the settings of the generalized rostering problem. Ideally, the project on generalized rostering should capture as many extensions and variations as possible.

The idea of compile-time customization which was presented for the generalized rostering problem can be transferred to problems not considered in this thesis. In many practical applications of operations research, problems eventually need to be modified, remodeled, or extended. By using a generic approach, as presented for rostering problems, developed algorithms are easily adapted to extensions of exiting problems or new similar problems. The customization idea allows for an adaption without performance drawbacks and is therefore ideal for problems, where high performance is a requirement.

Additional directions for future work are sketched in the individual papers.

Bibliography

Abbink, E., M. Fischetti, L. Kroon, G. Timmer, and M. Vromans (2005). “Reinventing crew scheduling at Netherlands railways”. In: *Interfaces* 35.5, pp. 393–401.

Allahverdi, A., C. Ng, T. Cheng, and M. Kovalyov (2008). “A survey of scheduling problems with setup times or costs”. In: *European Journal of Operational Research* 187.3, pp. 985–1032.

Amor, H. B. and J. V. de Carvalho (2005). “Cutting Stock Problems”. In: *Column Generation*. Ed. by G. Desaulniers, J. Desrosiers, and M. Solomon. Springer. Chap. 5, pp. 131 –161.

Anbil, R., E. Gelman, B. Patty, and R. Tanga (1991). “Recent advances in crew-pairing optimization at American Airlines”. In: *Interfaces* 21.1, pp. 62 –74.

Balas, E., C. Wallace, and S. Schmieta (2004). “Pivot and shift - A mixed integer programming heuristic”. In: *Discrete Optimization* 1.1, pp. 3–12.

Baptiste, P., C. Le Pape, and W. Nuijten (2001). *Constraint-based Scheduling: Applying Constraint Programming to Scheduling Problems*. International series in operations research & management science. Springer.

Barnhart, C., E. L. Johnson, G. L. Nemhauser, M. W. P. Savelsbergh, and P. H. Vance (1998). “Branch-and-Price: Column Generation for Solving Huge Integer Programs”. In: *Operations Research* 46.3, pp. 316–329.

Barnhart, C., A. M. Cohn, E. L. Johnson, D. Klabjan, G. L. Nemhauser, and P. H. Vance (2003). “Airline Crew Scheduling”. In: *Handbook of Transportation Science*. Ed. by Randolph W. Hall. Kluwer Academic Publishers.

Beck, J., P. Prosser, and E. Selensky (2003). “Vehicle routing and job shop scheduling: what’s the difference?” Ed. by E. Giunchiglia. In: *Proceedings,* *Thirteenth International Conference on Automated Planning and Scheduling*, pp. 267–76.

Beck, J., P. Prosser, and E. Selensky (2006). “A case study of mutual routing-scheduling reformulation”. In: *Journal of Scheduling* 9.5, pp. 469–491.

Begur, S., D. Miller, and J. Weaver (1997). “An integrated spatial DSS for scheduling and routing home-health-care nurses”. In: *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”. Ed. by Louis-Martin Rousseau Michel Gendreau Gilles Pesant. In: *Computers and Operations Research* 33.10, pp. 2866–2890.

Bredström, D. and M. Rönnqvist (2007). *A branch and price algorithm for the combined vehicle routing and scheduling problem with synchronization constraints.* Tech. rep. 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”. In: *European Journal of Operational Research* 191.1, pp. 19–31.

Brusco, M. J., L. W. Jacobs, R. J. Bongiorno, D. V. Lyons, and B. Tang (1995). “Improving Personnel Scheduling at Airline Stations”. In: *Operations Research* 43.5, 741–751 and 172029.

Burkard, R., M. Dell’Amico, and S. Martello (2009). *Assignment problems.* Philadelphia : SIAM, Society for Industrial and Applied Mathematics.

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

Burke, E. K., T. Curtois, R. Qu, and G. Vanden-Berghe (2009). *Problem Model for Nurse Rostering Benchmark Instances.* Tech. rep. http://www.cs.nott.ac.uk/∼tec/NRP/papers/ANROM.pdf. ASAP, School of Computer Science, University of Nottingham, Jubilee Campus, Nottingham, UK.

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

Cattrysse, D. and L. Van Wassenhove (1992). “A survey of algorithms for the generalized assignment problem”. In: *European Journal of Operational Research* 60.3, pp. 260–272.

Chabrier, A. (2006). “Vehicle Routing Problem with elementary shortest path based column generation”. Ed. by Louis-Martin Rousseau Michel Gendreau Gilles Pesant. In: *Computers and Operations Research* 33.10, pp. 2972–2990.

Cheang, B., H. Li, A. Lim, and B. Rodrigues (2003). “Nurse rostering problems–a bibliographic survey”. In: *European Journal of Operational Research* 151.3, pp. 447–460.

Chu, S. C. K. (2007). “Generating, scheduling and rostering of shift crew-duties: Applications at the Hong Kong International Airport”. In: *European Journal of Operational Research* 177, pp. 1764 –1778.

Cormen, T. H., C. E. Leiserson, R. L. Rivest, and C. Stein (2001). *Introduction to Algorithms*. Second Edition. MIT Press and McGraw-Hill.

Danna, E., E. Rothberg, and C. L. Pape (2005). “Exploring relaxation induced neighborhoods to improve MIP solutions”. In: *Mathematical Programming* 102.1, pp. 71–90.

Day, P. R. and D. M. Ryan (1997). “Flight attendant rostering for short-haul airline operations.” In: *Operations Research* 45.5, pp. 649 –661.

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”. In: *Transportation Science* 42.3, p. 387.

Desaulniers, G. (2009). “Branch-and-Price-and-Cut for the Split-Delivery Vehicle Routing Problem with Time Windows”. In: *Operations Research* To appear.

Desrochers, M., J. Desrosiers, and M. Solomon (1992). “A new optimization algorithm for the vehicle routing problem with time windows”. In: *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.M. Solomon. Springer, New York. Chap. 1, pp. 1–32.

Desrosiers, J., F. Soumis, and M. Desrochers (1984). “Routing with time windows by column generation”. In: *Networks* 14.4, pp. 545–565.

Dohn, A. (2007). “Optimizing the Steel Plate Storage Yard Crane Scheduling Problem Using a Two Stage Planning/Scheduling Approach”. In: *ICAPS 2007 - Doctoral Consortium*.

Dohn, A. and J. Clausen (2008*a*). “A Two-stage Planning/Scheduling Model”. In: *Workshop: MetMat*.

Dohn, A. and J. Clausen (2008*b*). *Optimizing the Slab Yard Planning and Crane Scheduling Problem using a Two-Stage Approach*. Tech. rep. DTU Management Engineering, Technical University of Denmark.

Dohn, A. and J. Clausen (2008*d*). “Optimizing the Steel Slab Yard Crane Scheduling Problem Using a Two Stage Planning/Scheduling Approach”. In: *International Federation of Operational Research Societies Conference*.

Dohn, A. and J. Clausen (2010). “Optimising the Slab Yard Planning and Crane Scheduling Problem using a two-stage heuristic”. In: *International Journal of Production Research* 48.15, pp. 4585–4608.

Dohn, A. and E. Kolind (2008). “Optimizing Manpower Allocation for Ground Handling Tasks in Airports using Column Generation”. In: *ORSNZ’08 - Proceedings - 43rd Annual Conference of the Operational Research Society of New Zealand*, pp. 2–11.

Dohn, A. and E. Kolind (2009). “A Practical Branch and Price Approach to the Crew Scheduling Problem with Time Windows”. In: *ORbit* 14, pp. 23–27.

Dohn, A. and A. Mason (2010). “A Nested Column Generation Based Approach to the Generalized Rostering Problem using Compile-time Customization”. In: *INFORMS Journal on Computing* (Submitted).

Dohn, A., E. Kolind, and J. Clausen (2007*a*). “The Manpower Allocation Problem with Time Windows and Job-Teaming Constraints”. In: *ICAPS 2007 - Proceedings, Seventeenth International Conference on Automated Planning and Scheduling*, pp. 120–127.

Dohn, A., E. Kolind, and J. Clausen (2007*b*). “The Manpower Allocation Problem with Time Windows and Job-Teaming Constraints”. In: *Nordic Optimization Symposium.*

Dohn, A., E. Kolind, and J. Clausen (2007*c*). *The Manpower Allocation Problem with Time Windows and Job-Teaming Constraints: A Branch-and-Price Approach.* Tech. rep. Richard Petersens Plads, Building 321, DK-2800 Kgs. Lyngby: Informatics and Mathematical Modelling, Technical University of Denmark, DTU.

Dohn, A., M. S. Rasmussen, and J. Larsen (2008*a*). “Manpower Routing and Scheduling with Temporal Dependencies Between Tasks”. In: *International Workshop on Column Generation.*

Dohn, A., M. S. Rasmussen, T. Justesen, and J. Larsen (2008*c*). “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 (2008*b*). “The Home Care Crew Scheduling Problem”. In: *ORbit* 13, pp. 19–23.

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

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

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

Dohn, A., M. S. Rasmussen, and J. Larsen (2010*c*). “The Vehicle Routing Problem with Time Windows and Temporal Dependencies”. In: *Networks* (Conditionally accepted).

Dorigo, M. (1992). “Optimization, Learning and Natural Algorithms”. PhD thesis. Politecnico di Milano, Italy.

Dowling, D., M. Krishnamoorthy, H. Mackenzie, and D. Sier (1997). “Staff rostering at a large international airport”. In: *Annals of Operations Research* 72, pp. 125 –147.

Dror, M. and P. Trudeau (1989). “Savings by Split Delivery Routing”. In: *Transportation Science* 23.2, pp. 141–149.

Dumitrescu, I. and N. Boland (2003). “Improved preprocessing, labeling and scaling algorithms for the weight-constrained shortest path problem”. In: *Networks* 42.3, pp. 135–153.

Engineer, F. G. (2003). “A Solution Approach to Optimally Solve the Generalized Rostering Problem”. MA thesis. Department of Engineering Science, University of Auckland, New Zealand.

Ernst, A. T., H. Jiang, M. Krishnamoorthy, B. Owens, and D. Sier (2004*a*). “An Annotated Bibliography of Personnel Scheduling and Rostering”. Ed. by Houyuan Jiang, Mohan Krishnamoorthy, and David Sier. In: *Annals of Operations Research* 127.1-4, pp. 21–144.

Ernst, A. T., H. Jiang, M. Krishnamoorthy, and D. Sier (2004*b*). “Staff scheduling and rostering: A review of applications, methods and models”. Ed. by E. Burke and S. Petrovic. In: *European Journal of Operational Research* 153.1, pp. 3–27.

Eveborn, P. and M. Rönnqvist (2004). “Scheduler - A System for Staff Planning”. Ed. by Houyuan Jiang, Mohan Krishnamoorthy, and David Sier. In: *Annals of Operations Research* 128.1-4, pp. 21–45.

Eveborn, P., P. Flisberg, and M. Rönnqvist (2006). “Laps Care—an operational system for staff planning of home care”. Ed. by J. Krarup L. Sakalauskas. In: *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”. In: *Networks* 44.3, pp. 216–229.

Feo, T. A. and M. G. C. Resende (1995). “Greedy Randomized Adaptive Search Procedures”. In: *Journal of Global Optimization* 6.2, pp. 109–133.

Fischetti, M. and A. Lodi (2008). “Repairing MIP infeasibility through local branching”. Ed. by John W. Chinneck. In: *Computers and Operations Research* 35.5, pp. 1436–1445.

Fischetti, M. and D. Salvagnin (2009). “Feasibility pump 2.0”. In: *Mathematical Programming Computation* 1.2-3, pp. 201–222.

Frizzell, P. W. and J. W. Giffin (1995). “The Split Delivery Vehicle Scheduling Problem with Time Windows and Grid Network Distances”. In: *Computers and Operations Research* 22.6, pp. 655–667.

Gilmore, P. C. and R. E. Gomory (1961). “A Linear Programming Approach to the Cutting-Stock Problem”. In: *Operations Research* 9.6, 849–859 and 167051.

Glover, F. (1989). “Tabu search. 1”. In: *ORSA Journal on Computing* 1.3, pp. 190–206.

Glover, F. and G. A. Kochenberger, eds. (2003). *Handbook of Metaheuristics.* Vol. 57. International Series in Operations Research & Management Science. Springer.

Gopalakrishnan, B. and E. L. Johnson (2005). “Airline Crew Scheduling: State-of-the-Art”. Ed. by Monique Guignard-Spielberg and Kurt Spielberg. In: *Annals of Operations Research* 140.1, pp. 305–337.

Hansen, J. (2003). “Industrialised application of combinatorial optimization”. PhD thesis. Richard Petersens Plads, Building 321, DK-2800 Kgs. Lyngby: Informatics and Mathematical Modelling, Technical University of Denmark, DTU.

Hillier, F. S. and G. J. Lieberman (2001). *Introduction to Operations Research.* McGraw-Hill.

Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems.* University of Michigan Press, Ann Arbor.

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

Jepsen, M., B. Petersen, S. Spoorendonk, and D. Pisinger (2008). “Subset-Row Inequalities Applied to the Vehicle-Routing Problem with Time Windows”. In: *Operations Research* 56.2, pp. 497–511.

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

Kellogg, D. L. and S. Walczak (2007). “Nurse Scheduling: From Academia to Implementation or Not?” In: *Interfaces* 37.4, pp. 355 –369.

Kirkpatrick, S., C. D. Gelatt, and M. P. Vecchi (1983). “Optimization by Simulated Annealing”. In: *Science* 220.4598, 671–680 and 1690046.

Kohl, N. and S. E. Karisch (2004). “Airline Crew Rostering: Problem Types, Modeling, and Optimization”. Ed. by Houyuan Jiang, Mohan Krishnamoorthy, and David Sier. In: *Annals of Operations Research* 127.1-4, pp. 223–257.

Kolen, A. W. J., A. H. G. R. Kan, and H. W. J. M. Trienekens (1987). “Vehicle Routing with Time Windows”. In: *Operations Research* 35.2, pp. 266–273.

Kuhn, H. W. (1955). “The Hungarian method for the assignment problem”. In: *Naval Research Logistics Quarterly* 2, pp. 83–87.

Lamothe, J., C. Thierry, and J. Delmas (1996). “A multihoist model for the real time hoist scheduling problem”. In: *Symposium on Discrete Events and Manufacturing Systems. CESA’96 IMACS Multiconference. Computational Engineering in Systems Applications*, pp. 461–6.

Lee, C.-G., M. A. Epelman, C. C. W. III, and Y. A. Bozer (2006). “A Shortest Path Approach to the Multiple-Vehicle Routing Problem with Split Pick-Ups”. In: *Transportation Research Part B* 40, pp. 265–284.

Lesaint, D., C. Voudouris, and N. Azarmi (2000). “Dynamic workforce scheduling for British Telecommunications plc”. In: *Interfaces* 30.1, pp. 45–56.

Lessel, C. R. (2007). “Ruteplanlægning i hjemmeplejen”. MA thesis. Informatics and Mathematical Modelling, Technical University of Denmark.

Leung, J. and G. Zhang (2003). “Optimal cyclic scheduling for printed circuit board production lines with multiple hoists and general processing sequence". In: *IEEE Transactions on Robotics and Automation* 19.3, pp. 480–484.

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

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

Lusby, R., A. Dohn, T. M. Range, and J. Larsen (2010). "An Integrated Approach to the Ground Crew Rostering Problem with Work Patterns". In: *Journal of the Operational Research Society* (Submitted).

Mason, A. J., D. Ryan, and A. Dohn (2009). "Customised Column Generation for Rostering Problems: Using Compile-time Customisation to create a Flexible C++ Engine for Staff Rostering". In: *ORSNZ'09 - Proceedings - 44rd Annual Conference of the Operational Research Society of New Zealand.*

Pardalos, P. M. and H. Wolkowicz, eds. (1993). *Quadratic Assignment and Related Problems.* American Mathematical Society.

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

Rasmussen, M. S., T. Justesen, A. Dohn, and J. Larsen (2010). "The Home Care Crew Scheduling Problem: Preference-Based Visit Clustering and Temporal Dependencies". In: *European Journal of Operational Research* (Submitted).

Rezanova, N. J. and D. M. Ryan (2010). "The train driver recovery problem-A set partitioning based model and solution method". Ed. by Jesper Larsen Jens Clausen Allan Larsen. In: *Computers and Operations Research* 37.5, pp. 845–856.

Savelsbergh, M. and M. Sol (1998). "Drive: Dynamic Routing of Independent Vehicles". In: *Operations Research* 46.4, 474–490 and 223126.

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

Steenken, D., S. Voß, and R. Stahlbock (2004). "Container terminal operation and operations research - a classification and literature review". In: *OR Spectrum* 26.1, pp. 3–49.

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

Toth, P. and D. Vigo (2001). *The Vehicle Routing Problem.* Society for Industrial and Applied Mathematics Philadelphia, PA, USA.

van den Akker, J. M., J. A. Hoogeveen, and S. L. v. d. Velde (1999). "Parallel Machine Scheduling by Column Generation". In: *Operations Research* 47.6, pp. 862–872.

Vazirani, V. V. (2001). *Approximation Algorithms.* Springer-Verlag, Berlin.

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

Zhu, X. and W. Wilhelm (2006). “Scheduling and lot sizing with sequence-dependent setup: a literature review”. In: *IIE Transactions* 38.11, pp. 987–1007.

Part II

Scientific Papers

Appendix A

A Nested Column Generation Based Approach to the Generalized Rostering Problem using Compile-time Customization

Anders Dohn and Andrew Mason

Submitted to: *INFORMS Journal on Computing* (2010).

A Nested Column Generation Based Approach to the Generalized Rostering Problem using Compile-time Customization [*]

Anders Dohn[1], Andrew Mason[2]

We present a novel column-generation based solution approach to the generalized staff rostering problem. The problem is defined by a generic set partitioning model, which is able to capture all commonly occurring problem characteristics from the literature. Columns of the set partitioning problem are generated dynamically by solving a pricing sub-problem and constraint branching in a branch-and-bound framework is used to enforce integrality. The pricing problem is formulated as a novel three-stage nested shortest path problem with resource constraints that exploits the inherent problem structure. Careful use of the C++ pre-processor allows the pricing model to be customized for the target problem at compile-time, resulting in a both versatile and very efficient solution method. The solution method for the pricing problem is presented along with a bounding scheme for resources. Comparison with a more-standard run-time customization approach shows that speedups of around a factor 20 are achieved using our new approach. The adaption to a new problem is simple and the implementation is automatically adjusted internally according to the new definition. We present results for three practical nurse rostering problems. The approach captures all features of each problem and is able to provide high-quality solutions in less than 15 minutes. In two of the three instances, even the optimal solution is found within this time frame.

Keywords: generalized rostering problem, nurse rostering, nurse scheduling, column generation, branch-and-price, set partitioning, set covering, integer programming, linear programming, shortest path problem with resource constraints, dynamic programming, label setting.

[*]Submitted to: *INFORMS Journal on Computing* (2010).

[1]Department of Management Engineering, Technical University of Denmark, Produktionstorvet, 2800 Kongens Lyngby, Denmark.

[2]Faculty of Engineering, University of Auckland, Auckland, New Zealand.

A.1 Introduction

The generalized rostering problem is the problem of generating a feasible high-quality staff schedule ('roster') for a group of staff working collectively to provide some service. The roster will specify, for each staff member the sequence of shifts and days off (termed the 'roster-line') to be worked during the roster period. This roster must satisfy pre-specified demand constraints which typically express minimum staffing requirements for individual shifts or time periods, and may include requirements for staff with certain skills or meeting particular contract conditions. The roster-line worked by each staff member is typically strictly governed by laws, union regulations and internal agreements, and has an associated quality measure. These requirements together can make it hard to create feasible rosters, let alone high quality ones.

Rosters have traditionally been created manually by the head of the section or by an experienced member of the staff. Often, the rosters were made by modifying former rosters or by putting together roster-lines and parts of roster-lines which were known to be good. It takes a lot of experience to build good rosters and even with experience, the process of building the rosters is very time consuming. Therefore, there has been and still is a large demand for automated rostering tools. Within the last decade the supply of software products has increased significantly to meet this demand.

One of the major obstacles to developing rostering software has been the varying requirements from one application to another. As a result, many rostering systems have been custom made to match the exact requirements of a particular company or institution. The main issue with this approach is the time and money needed for development of the system. For the same reason, automated scheduling systems have in the past been reserved for institutions with a large and very apparent need for automation. Our focus here is on developing a flexible but still efficient generic software framework that can be applied to a large variety of rostering problems. Our system has to be adapted to a given application, but this adaptation is easy and fast. Hence, using our approach, an effectively tailor-made product can be produced at significantly lower costs than before.

Staff rostering has already received a lot of attention in the literature. We refer to the extensive literate reviews of Burke et al. (2004), Ernst et al. (2004*b*), and Cheang et al. (2003). Also, Ernst et al. (2004*a*) present a massive collection of references to papers on rostering. The EURO Working Group on Automated Timetabling (Curtois, 2010) provides a good up-to-date overview of the literature.

The solution method presented here builds on the idea of column generation. Column generation is embedded in a branch-and-bound framework resulting in a so called *branch-and-price* algorithm; for an introduction to column generation, see Desrosiers and Lübbecke (2005). Naturally, the literature on column generation based solution methods to rostering is of special interest. Jaumard et al. (1998) solve a nurse rostering problem using column generation. The subproblem is formulated as a shortest path problem with resource constraints, where each possible shift is represented by a node. It is solved with a two-stage algorithm proposed by the authors. Bard and Purnomo (2005*b*) solve a nurse rostering problem with individual preferences for the nurses. Columns are generated by a, so called, double swapping heuristic. High-quality solutions are found within minutes. In Bard and Purnomo (2005*a*) the model is extended to allow downgrading of workers with higher level skills. Beliën and Demeulemeester (2006) schedule trainees in a hospital using branch-and-price. Interestingly, columns are generated for activities instead of the conventional columns for roster-lines of employees. The described problem of scheduling trainees is somewhat simpler than the generalized rostering problem, and only for this reason, is it possible to use the alternative column generation model. In a succeeding paper by the same authors (Beliën and Demeulemeester, 2007), it is concluded that the activity-decomposed approach does not have the same modeling power. However, if the problem allows this model to be used, the performance may be enhanced by doing so.

A column generation approach to the nurse rostering problem is also described in Maenhout and Vanhoucke (2008). The approach has a number of features in common with our approach. The authors describe different pruning and branching strategies, e.g. Lagrangian dual pruning and branching on the residual problem. Beliën and Demeulemeester (2008) extend the nurse rostering model to include scheduling of the operating room and show that considerable savings can be made by integrating the two scheduling problems.

Eitzen et al. (2004) present a set covering model for a crew scheduling problem at a power station. Three column generation based solution methods are proposed to solve the set covering model: the column expansion method, the reduced column subset method, and branch-and-price. In the first, three phases are solved, where the skill set of each employee is gradually enlarged from one phase to the next. In the second, a restricted number of columns are generated randomly with no guarantee on quality. The latter method integrates dynamic column generation and constraint branching in a branch-and-bound framework. Al-Yakoob and Sherali (2008) solve a crew rostering problem for a large number of gas stations using a column generation approach. The model takes the individual preferences of the employees into account. A heuristic founded on the column generation algorithm is able to solve realistic problems.

In the column generating subproblem presented here, two shortest path problems with resource constraints are solved. Much literature has been published on shortest path problems with resource constraints. Desrosiers et al. (1984) present an early version of the algorithm with time as the only resource, which is generalized by Desrochers (1988). Lübbecke (2005) suggests discarding all labels that cannot lead to a column with negative reduced cost. Dumitrescu and Boland (2003) introduce a more extensive preprocessing scheme. Righini and Salani (2006) present a significant improvement in performance by using bidirectional search. Chabrier (2006) utilize an idea on the potential value of each node to improve performance further. See Irnich and Desaulniers (2005) for a literature review.

The idea behind the generic modeling of rules and preferences that we present is similar to that of an industrial system, described by Kohl and Karisch (2004). The authors describe a modeling tool, which is essentially a rule programming language, where the end user can maintain and modify rules in a flexible way. However, they also explain that the optimization method can only 'ask' simple questions about legality and cost.

The work presented here is based primarily on the work of Smith (1995), Nielsen (2003) and Engineer (2003). Smith presents a column generation setup to solve a nurse rostering problem from Middlemore Hospital in Auckland. Following the promising results for this initial application, two projects followed, with the aim of building a generalized rostering framework which would be able to solve various rostering problems. Nielsen describes a general modeling framework for rostering problems. Roster-lines are generated by an enumerative scheme including some constraint programming techniques. Nielsen uses experimental data from Auckland Healthcare. Engineer applies column generation to various rostering problems. He formulates the pricing problem as a three stage nested shortest path problem with resource constraints as described by Mason and Smith (1998) and solves it using label setting. The algorithm developed proved to be both efficient and at the same time versatile enough to allow solution of applications with very different characteristics. The work presented in this paper extends and improves these earlier approaches.

The contribution of this paper is the novel approach to the generalized rostering problem, where any constraint and cost observed in the literature can be modeled in the attribute-based setup. New instances are therefore solved without any modifications in the algorithm. Not only is the method versatile, it is also very efficient in its implementation. In a survey of the literature, Ernst et al. (2004*b*) point out that:

Another important area requiring further work is generalisation of *models and methods. Currently, models and algorithms often require significant modification when they are to be transferred to a different application area, or to accommodate changes within an organisation.*

The presented approach alleviates exactly this problem in a very efficient way.

In Section A.2, we define the generalized rostering problem, and give details of our integer programming formulation and an overview of the column generation sub-problem. We also discuss the generality of our approach. Our solution method is described in Section A.3. The implementation allows compile-time customization by including a user defined problem definition. The details are described in Section A.4. Following, an example is presented in Section A.5 and computational results for three realistic nurse rostering instances are discussed in Section A.6. Finally, our conclusions are presented in Section A.7.

A.2 Model

The inputs to a generalized rostering problem include staff, skills, shifts, and demands. Each shift is defined by a start date, start time and a duration that specifies when staff assigned to this shift will be at work. Each staff member has an associated subset of skills. These skills are combined with shifts to specify demands. For example, the company may specify that 2 or more 'supervisor' and/or 'clinician' staff shall be on duty from 10am to 12pm next Tuesday. If the shifts that cover this time period are the Tuesday 8am-8pm 'D' shift and 10am-7pm 'M' shift, then this demand is expressed as a lower bound on the number of such shifts worked by staff with the clinician or supervisor skill. The earliest and latest shifts together define the scheduling time horizon of interest. The roster produced as output will specify the roster-line (sequence of shifts and days off) worked by each staff member during the scheduling horizon. We generally assume each staff member works no more than one shift per day. A roster is feasible if all demands are satisfied using feasible roster-lines.

The generalized rostering problem can be modeled as a generalized binary set partitioning problem. Each column corresponds to a possible roster-line for a certain staff member, while each row corresponds to a demand constraint. Some demands may be defined as soft constraints and a penalty is applied if the demand is not met. The master problem combines the roster-lines in order to meet the demand constraints, while the column generation subproblem must generate feasible roster-lines.

The difficulty when modeling rostering problems is typically associated with the generation of high quality feasible roster-lines. The rules these roster-lines must satisfy may vary significantly from one problem to another as illustrated in Section A.2.3. A column generator must be sufficiently flexible to handle these rules. We now formally define the master problem for which columns are generated.

A.2.1 Master problem

Given a set of employees, $\mathcal{E}$, and a set of demands, $\mathcal{D}$, the objective of the master problem is to find a combination of roster-lines, one for each employee, such that all demands are met at the lowest possible cost (where a low cost corresponds to a high quality). The set $\mathcal{R}_e$ holds all feasible roster-lines for employee $e \in \mathcal{E}$. Because of the vast number of feasible roster-lines, columns are generated during the solution process. The set $\mathcal{R}'_e$ contains the roster-lines of employee e generated so far. Three sets of decision variables are used. λ_e^r is a binary decision variable, where $\lambda_e^r = 1$ if roster-line r is chosen for employee e and $\lambda_e^r = 0$ otherwise. s_d^- is the amount of under-coverage (slack) for demand d. Similarly, s_d^+ is the amount of over-coverage (surplus) for demand d. The amount of permitted under- and over-coverage is regulated by bounds on s_d^- and s_d^+, respectively.

Three sets of costs apply to the master problem. c_e^r gives the cost of roster-line r of employee e. c_d^- and c_d^+ specify the cost of under- and over-coverage for demand d, respectively. Parameter a_{ed}^r describes the roster-lines, with $a_{ed}^r = 1$ if roster-line r of employee e contributes to demand d, and $a_{ed}^r = 0$ otherwise. b_d is the demand level to be met. The master problem is formulated as:

$$\min \sum_{e\in\mathcal{E}} \sum_{r\in\mathcal{R}'_e} c_e^r \lambda_e^r + \sum_{d\in\mathcal{D}} c_d^- s_d^- + \sum_{d\in\mathcal{D}} c_d^+ s_d^+ \tag{A.1}$$

$$\sum_{r\in\mathcal{R}'_e} \lambda_e^r = 1 \qquad \forall e \in \mathcal{E} \tag{A.2}$$

$$\sum_{e\in\mathcal{E}} \sum_{r\in\mathcal{R}'_e} a_{ed}^r \lambda_e^r + s_d^- - s_d^+ = b_d \qquad \forall d \in \mathcal{D} \tag{A.3}$$

$$\lambda_e^r \in \{0,1\} \qquad \forall e \in \mathcal{E}, \forall r \in \mathcal{R}'_e \tag{A.4}$$

$$0 \le s_d^- \le u_d^-, 0 \le s_d^+ \le u_d^+ \qquad \forall d \in \mathcal{D} \tag{A.5}$$

The objective (A.1) is to minimize the total cost of all roster-lines while also minimizing penalties from under- and over-coverage. A feasible solution contains one roster-line for each employee (A.2). All demands must be met or the appropriate slack and surplus variables are adjusted accordingly (A.3). (A.4) and (A.5) set the domains of the decision variables, where u_d^- and u_d^+ give the maximum under- and over-coverage permitted. The LP-relaxation of (A.1)-(A.5) is denoted the *Restricted Master Problem.*

For any solution of the master problem given by λ_e^r, s_d^-, and s_d^+, a dual solution exists. Let τ_e be the dual variables of Constraints (A.2) and similarly, let π_d be the dual variables of Constraints (A.3). If a primal solution does not exist, the dual problem is unbounded and no meaningful dual values can be found. Instead, τ_e, π_d denote a dual ray that gives information on how to restore primal feasibility.

A.2.2 Subproblem

The subproblem, also referred to as the pricing problem, generates new columns for the master problem. Given a vector of dual values, the subproblem returns a column with negative reduced cost, if one exists. Such a column will enter the basis in the master problem and dual values are updated. If no additional columns exist, the master problem solution is optimal.

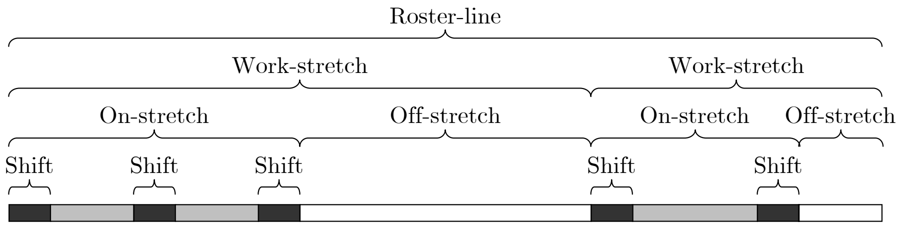


Figure A.1: The entities of a roster and their association.

The subproblem is formulated as a three stage model and follows the setup described by Mason and Smith (1998). A roster-line is considered to be constructed from underlying *entities.* The association between the entities is illustrated in Figure A.1. A *shift* entity defines a period of time during which an employee is working. A series of shifts worked in succession (typically one per day) forms an *on-stretch* entity. The length of on-stretches may be constrained in terms of minimum and maximum number of hours, shifts, days, etc. A period of time where the employee is not working is referred to as an *off-stretch.* An on-stretch followed by an off-stretch forms a *work-stretch.* There are often restrictions on how on-stretches and off-stretches can be combined. Finally, a *roster-line* consists of a sequence of work-stretches. A roster-line spans the full scheduling horizon and typically has to respect constraints on the total amount of hours worked, the number of weekends worked, etc.

Shifts and off-stretches are referred to as simple entities while on-stretches, work-stretches and roster-lines are composite entities. Entities have *attributes* which are tracked as the entities are constructed and extended. Attributes of a simple entity is part of the input data, whereas the attributes of a composite entity are calculated when the entity is created, using the attribute values of its component entities. A set of *rules*, typically stated as bounds on attribute values, define the validity of an entity. Rules may define hard constraints that must be satisfied or soft constraints for which a penalty (termed an *attribute cost*) is applied when a rule violation occurs. The attribute values are also used to calculate the cost associated with each entity. The cost of a new entity is typically calculated using costs derived from the entity's attribute values and/or costs associated with its component entities. These entity costs may be employee dependent if staff have expressed individual preferences such as shifts requests.

Different rules and preferences may apply to each employee; therefore a separate subproblem exists for each of the employees. Given a set of legal shifts, $\mathcal{S}_e$, for employee e, a set of feasible on-stretches, $\mathcal{O}_e$, can be found. A set of legal work-stretches, $\mathcal{W}_e$, is created for all compatible combinations of on-stretches in $\mathcal{O}_e$ and all legal off-stretches, $\mathcal{F}_e$. By sequencing legal work-stretches, roster-lines are generated and together define the set $\bar{\mathcal{R}}_e$.

The objective of the subproblem for employee e is to find the feasible roster-line with the most negative reduced cost or to prove that no roster-line with negative reduced cost exists. For a given employee e, we can compute an effective dual value for each shift by summing the dual values π_d of the demands $d \in \mathcal{D}$ that are contributed to by this shift when worked by employee e. The reduced cost of any entity can then be computed by subtracting from its cost the sum of the effective duals of the shifts contained in that entity. Off-stretches have fixed costs only. In the case of a roster-line, the dual of the corresponding employee constraint, τ_e, must also be included. When the subproblem is used to restore primal feasibility in the master problem, the reduced costs are based solely on the values of the dual ray, with all other costs treated as zero.

As we discuss next, our use of this structure of nested entities, and the corresponding nested column generator we have developed, allows a computationally efficient expression of a wide range of rules and preferences. The user is free to specify what attributes need to be tracked by each entity type, and how these attribute values are calculated and used to determine the feasibility and/or cost of an entity. The use of nested entities allows the rules and preferences to be expressed where they naturally occur, ensuring that illegal or poor quality entities are rejected early in the column generation process.

A.2.3 Applicability of the model

The extent to which our system can model and solve different rostering problems is limited by the modeling power of the master problem and the column generator. As we now discuss, our experience with problems drawn from the literature and from our own industry contacts suggests that this modeling approach, and in particular the nested column generation, can express and efficiently process all common rules and requirements.

In a bibliographic survey by Cheang et al. (2003) a list of commonly occurring constraints in the literature is presented, where the constraints are grouped into 16 different categories. Burke et al. (2009) in a similar way list 26 sets of constraints that occur in practical nurse rostering problems. The list is a slight revision of a list originally formulated by Berghe (2002).

Furthermore, the work of Bliddal and Tranberg (2002), Nielsen (2003), Engineer (2003) and Poulsen (2004) together describe 10 practical problems. The practical problems have been given special attention in this project. From the reports, a number of common characteristics can be found. The problems have a fixed planning period and a fixed set of shifts. Each nurse has individual preferences and their own paid hours targets. Some specific shift transitions are banned and shift assignments may be fixed in advance. These characteristics are all captured by our current model.

Each problem has individual rules. To illustrate, a few examples are listed below.

- On all days: at least one of the nurses was also there the day before (Bliddal and Tranberg, 2002).
- A nurse cannot work two consecutive weekends (Poulsen, 2004).
- Minimize the number of different shifts in an on-stretch (Poulsen, 2004).
- One week with 60 hours or more allows only 16 hours the following week (Poulsen, 2004).
- If working night shifts, at least two consecutive night shifts must be scheduled (Nielsen, 2003).

These constraints can easily be included in the model using appropriately defined attributes. Indeed, by introducing customized attributes, any of the constraints concerning internal roster-line rules can be implemented. A few rules reported in the literature concern roster-lines of multiple employees, and thus impact the master problem. Most of these can be modeled as demand constraints of the form presented in Constraint (A.3). The remaining rule concerns tutorship, where one employee can only work if another employee is working as well. This can be modeled by introducing demands where some employees may have a negative contribution, i.e. $a^r_{ed} = -1$ in Constraint (A.3).

With the suggested extension, the model presented here is able to deal with all the constraints of Cheang et al. (2003) and Burke et al. (2009) and with all constraints seen in the 10 industry examples mentioned earlier.

A.3 Solution method

A.3.1 Master Problem Solution

The relaxed restricted master problem as defined in Section A.2.1 is an LP-problem and a standard solution tool (like CPLEX) can be applied to solve it.

Branching is applied to remove fractional solutions from the solution space of the LP-relaxed master problem. In regular branch-and-bound algorithms, variable branching is the method of choice. It is, however, complex and in most cases highly inefficient to apply variable branching in a branch-and-price algorithm. Instead, we use constraint branching where certain constraints are (implicitly) introduced in the current restricted master problem.

Here, we use a specialization of the constraint branching method proposed by Ryan and Foster (1981). If the solution of the restricted master problem is fractional, the columns of one or more employees are in the solution with a fractional value. As two columns of an employee are never identical, two fractionally selected columns will differ in at least one of the included shifts. This in turn means that the employee is not assigned to that shift with a value of 1. In a feasible integer solution, employees are always assigned to shifts (with a value of 1) or not assigned to them at all (a value of 0). We may therefore branch on the employee/shift assignment.

When branching on an employee/shift assignment, the set of feasible columns of the employee is split into two subsets. One subset contains all columns for that employee, where the shift is in the roster-line. The other subset contains the remaining columns for that employee. The two branches are created by removing the relevant roster-lines from the first and the second subset, respectively. In a branch-and-price setup, the branching decisions are imposed on new columns by forcing the subproblem of the particular employee to either include or exclude the shifts that have been branched on.

During our computational testing (Section A.6), we identified a number of strategies that reduced the solution times for our problems. Firstly, the solutions to the linear programming relaxations often satisfy employee preferences to some fractional degree, meaning that some employee has a mix of roster-lines whose costs come from only a few alternative values. Our experiments showed that where there are many possible branching candidates (fractional employee/shift assignments), one should avoid branches in which the banned columns on either side of the branch have similar costs, and instead prefer branches where these costs differ to the greatest degree. This quickly resolves the fractional preference satisfaction and gives an immediate impact on overall cost and/or feasibility. Finally, when solving an LP after branching and/or adding more columns, it was also found to have a positive effect to stop the solution process as soon as the LP value is equal to the lower bound of the root node. Traditionally, generation continues as long as columns with negative reduced cost exist. However, if the LP value is equal to the lower bound, such columns are only included in the basis because of degeneracy in the LP problem and will never decrease the objective value.

A.3.2 Column Generation Subproblem Solution

As discussed in Section A.2.2, the column generation subproblem involves the following steps as introduced by Mason and Smith (1998).

1. Shifts are combined into on-stretches.
2. On-stretches and off-stretches are paired to form work-stretches.
3. Roster-lines are generated by sequencing work-stretches.

Our column generator always builds the complex entities by combining two simpler entities. For example, instead of describing an on-stretch by all the shifts it contains, we describe it by its last shift and its 'parent' on-stretch which contains all but the last shift in the on-stretch. This is illustrated in

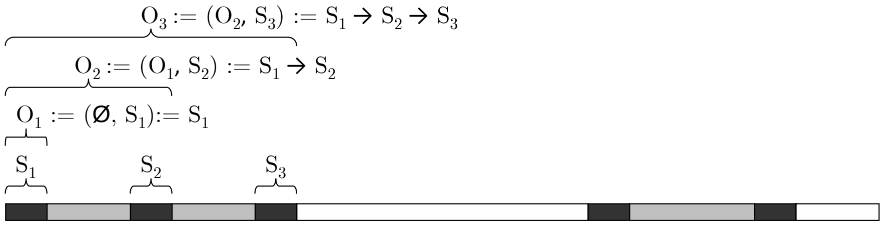


Figure A.2: Illustration of the recursive definition of on-stretches. On-stretch 3, O_3, containing shifts S_1, S_2, S_3, is defined as the combination of the two-shift on-stretch O_2 and the third shift S_3.

Figure A.2. Note that the parent on-stretch may be an infeasible on-stretch, and may be null if the on-stretch contains only one shift. The definition of the composite entities that follows from this approach gives:

on-stretch + shift	→ on-stretch
on-stretch + off-stretch	→ work-stretch
roster-line + work-stretch	→ roster-line

The three stages give rise to their own resource constrained shortest path problems. Attributes are algorithmically represented by resource. Each of shortest path problems is solved by dynamic programming using label setting (Irnich and Desaulniers, 2005), where an entity is labeled using the values calculated for its attributes. Dominance is used to prevent unpromising entities proceeding to the next stage. The dominance rules associated with an attribute form part of the definition of that attribute. Each of these three problems is now addressed in detail.

A.3.3 On-stretch generation

The first stage of the column generation algorithm is to generate on-stretches from shifts. Shifts might have costs given by employee preferences, and will have dual values associated with them, as described earlier. There might also be costs associated with shift transitions, while typical attributes to track might include paid hours or the number of undesirable shifts. On-stretches are considered unique if they differ in their start time or end time, or if they have different non-dominated attribute values. We seek all unique minimum reduced cost on-stretches.

The on-stretch generation problem is modeled as a shortest path problem with resources in a graph where nodes represent shifts. An example of such a graph is shown in Figure A.3. The graph consists of a node for each shift. Arcs between nodes exist when the two shifts are allowed to be consecutive in an on-stretch, i.e. they can be neither too close nor too far apart in time. To generate all on-stretches the all-to-all shortest paths problem is solved. This is done by solving a one-to-all shortest path problem from each of the nodes in the graph. For each of these problems, the start node is selected and the remaining graph is reduced to only allow on-stretches up to the maximum on-stretch length. Figure A.3 shows the shortest path problem with Node 1 as start node. If the maximum on-stretch length is 4 days, all arcs leaving nodes 16-20 are removed.

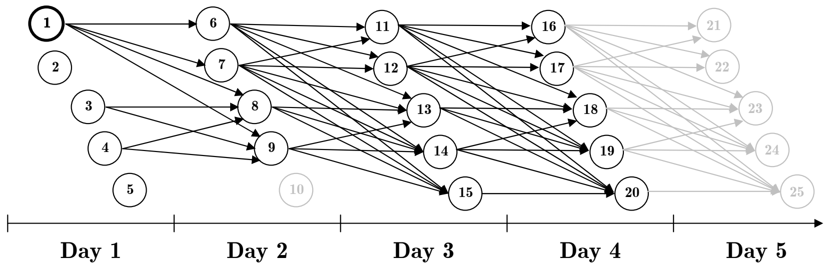


Figure A.3: Graph representation of the on-stretch generation. Note that this employee cannot work shift 10 and the node is therefore removed from the graph.

Nodes may be excluded in the graph in order to generate roster-lines without certain shifts. In Figure A.3, shift 10 has been excluded. Shifts may be disallowed if the employee does not have the appropriate skills and employees may simply be restricted from certain shifts as part of the input data for the problem, e.g. it is common to have employees that can never work night shifts or must have days off on particular days.

The shortest path problems are solved with a label setting algorithm. The graph is acyclic and has an inherent topological order and the nodes are treated accordingly. All on-stretches are validated by checking the attribute values against the feasibility criterion given in the problem definition, and checks are performed to remove dominated entities (see Section A.3.6). All unique non-dominated feasible on-stretches are sent to stage two.

A.3.4 Work-stretch generation

The second stage is the simplest of the three. On-stretches from stage one are combined with off-stretches to form work-stretches. The start time of an off-stretch is not specified exactly, but instead an off-stretch start time window is given along with a minimum off-stretch duration in hours. An on-stretch/off-stretch pair is considered compatible if the on-stretch finish time occurs within the off-stretch's start time window. All compatible on-stretch/off-stretch pairs are constructed, and the resulting work-stretch attribute values are calculated and checked for feasibility. Domination tests are performed (see Section A.3.6) to remove unpromising work-stretches before proceeding to the third stage. The second stage is visualized in Figure A.4.

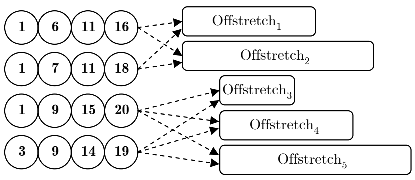


Figure A.4: Visualization of the work-stretch generation problem.

A.3.5 Roster-line generation

The roster-line generation problem is another acyclic shortest path problem with resource constraints. The problem has a node for each day in the horizon. The work-stretches generated in stage two are the transitions between days and hence become the arcs of the graph. Usually, the arc leads to the day after the work-stretch ended.

In the generalized rostering problem, we have a predetermined start and end day, and this transfers to a source node and a sink node in the graph. The problem becomes a one-to-one shortest path problem. The labels applied to the end node represent complete roster-lines. Again, roster-line attributes of these roster-lines have to respect the feasibility criterion given in the problem definition. The best feasible roster-line is also the optimal solution to the pricing problem. A visualization of the shortest path problem is given in Figure A.5.

For some attributes, it is possible to predict infeasibility prior to reaching the end node. Labels that can never lead to a feasible roster-line or a roster-line with negative reduced cost should be removed, as this results in fewer entities being

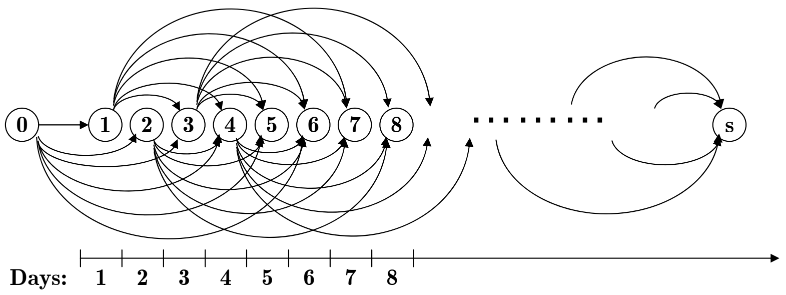


Figure A.5: Graph representation of the roster-line generation problem.

constructed and hence a more efficient algorithm. We describe the approaches we implement; see Dumitrescu and Boland (2003) for a detailed discussion of this area. Note that in the following discussion, reduced cost can be considered a resource like any other and hence needs no special attention. As only columns with negative reduced cost are of interest, the reduced cost is considered to have an imposed upper bound of zero.

In the general case, the feasibility of an attribute can be described by an arbitrary set of values. This feasibility requirement is generally specified for the roster-line as a whole, and so applies to the attribute values at the end node. However, in some cases there are feasibility requirements on intermediate nodes, for e.g. feasibility rules applied at fortnightly intervals. Define the set $\widehat{\mathcal{F}}_i^a$ as the feasible values of attribute a in node i, as determined by the problem formulation. If there are no particular requirements in node i, the set contains all possible values of a. To move the feasibility criterion from a node to a previous node, an inverse accumulation function is required.

The inverse accumulation function of an additive attribute is subtraction. For a particular attribute, each outgoing arc of a node will add a certain amount to the value of this attribute. The arc leads to a succeeding node, and feasible values of the current node are found by subtracting the accumulation value from each of the feasible values of the succeeding node. The complete set of feasible values of the current node is the union of the feasible value sets of all the outgoing arcs. As the graph is acyclic, the feasible values of all nodes are found by running through the nodes in a reverse topological order. For a node i, let δ_i^+ denote the set of outgoing arcs and δ_i^- the set of incoming arcs. acc_w^a is the accumulation function for attribute a on arc w, and correspondingly invacc_w^a is the inverse accumulation function. The arc w represents a transition from a node w_{orig} to a node w_{dest}. The feasibility set for attribute a of the end node is $\mathcal{F}_i^a = \widehat{\mathcal{F}}_i^a$ and for any other node i:

$$\mathcal{F}_i^a = \widehat{\mathcal{F}}_i^a \bigcap \left(\bigcup_{w \in \delta_i^+} \bigcup_{x \in \mathcal{F}^a_{w_{dest}}} \text{invacc}_w^a(x) \right)$$

For additive attributes, the accumulation function is defined by an accumulation value, accval_w^a, which is added when traversing the arc w. If we let the feasibility of an attribute be defined by bounds on the attribute value, the calculations are simplified. The set of feasible values for attribute a is now defined by a lower bound, lb_i^a, and an upper bound, ub_i^a, with initial values $\widehat{lb}_i^a$ and $\widehat{ub}_i^a$, respectively. The lower bound of a node is set to the minimum of all successor lower bounds minus the respective accumulation value of the successor arc. Analogously, the upper bound is the maximum of succeeding upper bounds minus accumulation values. This gives

$$lb_i^a = \max \left\{ \min_{w \in \delta_i^+} \left(lb^a_{w_{dest}} - \text{accval}_w^a \right), \widehat{lb}_i^a \right\}$$

$$ub_i^a = \min \left\{ \max_{w \in \delta_i^+} \left(ub^a_{w_{dest}} - \text{accval}_w^a \right), \widehat{ub}_i^a \right\}$$

Instead of requiring additivity for all attributes, the attributes are divided into two categories: additive attributes with bounds that can be propagated and attributes for which we will not propagate the feasibility criterion. In our system, each attribute, a, has an initialization function init_w^a on arc w, which gives the value of attribute a if the roster-line contains only work-stretch w. For additive attributes, it is assumed that $\text{accval}_w^a = \text{init}_w^a$, and hence the user does not have to provide an inverse accumulation function for the attribute. For non-additive attributes and attributes where $\text{accval}_w^a \neq \text{init}_w^a$, bounds are disabled. This restriction has been introduced solely to limit the input required from the user.

In the same way as bounds can be propagated backward from the end node, we can also propagate the set of possible attribute values forward from the start node. In this way, for each node, i, and each attribute a, we get the value domain, $\mathcal{D}_i^a$, defined as:

$$\mathcal{D}_i^a = \bigcup_{w \in \delta_i^-} \bigcup_{x \in \mathcal{D}^a_{w_{orig}}} \text{acc}_w^a(x)$$

In most cases, the domain may for an attribute a of node i be defined by bounds

$[dl_i^a, du_i^a]$, which are calculated as:

$$dl_i^a = \max\left\{\min_{w\in\delta_i^-}\left(\mathrm{acc}_w^a(dl_{w_{orig}}^a)\right), lb_i^a\right\}$$
$$du_i^a = \min\left\{\max_{w\in\delta_i^-}\left(\mathrm{acc}_w^a(du_{w_{orig}}^a)\right), ub_i^a\right\}$$

The value domains can be used to eliminate arcs from the graph. We check all arcs against the value domain of their tail (origin) combined with the feasibility set of their head (destination). If, using that arc, none of the values of the domain accumulate to values of the feasibility set, the arc can be removed from the graph, i.e. for an arc w to be effectual, the following must hold:

$$\exists x \in \mathcal{D}_{w_{orig}}^a : \mathrm{acc}_w^a(x) \in \mathcal{F}_{w_{dest}}^a.$$

When the sets are described by bounds, this is equivalent to:

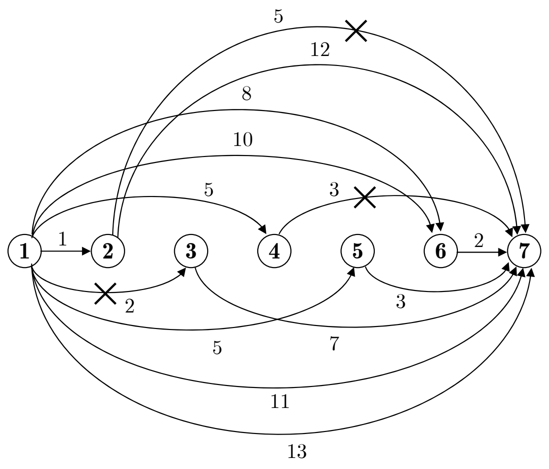
$$\mathrm{acc}_w^a(dl_{w_{orig}}^a) \leq ub_{w_{dest}}^a \wedge \mathrm{acc}_w^a(du_{w_{orig}}^a) \geq lb_{w_{dest}}^a$$

The arc is removed if it is non-effectual for one or more of the attributes. The bound propagation may be repeated to find tighter bounds for the resulting graph. Figure A.6 shows an example of bound and domain propagation.

The bounding scheme utilizes the nested subproblem structure. The arcs are the work-stretches generated in the second stage of the nested subproblem, and as the bounds are calculated based on the arcs of the graph the nested subproblem structure is essential for the bounding scheme.

A.3.6 Domination

Domination is an important concept in all three stages of the subproblem solution algorithm. An entity e_1 is said to be dominated by entity e_2 if any roster-line containing e_1 is worse (or no better) than a new roster-line created by replacing e_1 by e_2. For example, a 3-day on-stretch $O_1 = S_1 \to S_2 \to S_3$ containing shifts S_1, S_2, S_3 might be dominated by $O_2 = S_1 \to S_4 \to S_3$ if S_2 and S_4 are equivalent in paid hours, but S_4 is preferred by the employee. An efficient column generation algorithm must be able to identify and remove dominated entities.



Bounds:	[lb_1;ub_1]	[-2;9]	[3;7]	[2;11]	[4;11]	[8;12]	[10;14]
Domains:	[0;0]	[1;1]	[3;5]	[2;6]	[4;8]	[8;10]	[dl_7;du_7]

Figure A.6: Example of attribute bound and domain propagation. The value given for each arc is the value of a single attribute a. Only selected arcs of the graph are shown. The three arcs which have been crossed out in the figure can be eliminated. The missing values are calculated as:
$lb_1 = \min\{-2-1, 3-2, 2-5, 4-5, 8-8, 8-10, 10-11, 10-13\} = -3$
$ub_1 = \max\{9-1, 7-2, 11-5, 11-5, 12-8, 12-10, 14-11, 14-13\} = 8$
$dl_7 = \max(\min\{0+11, 0+13, 1+5, 1+12, 3+7, 2+3, 4+3, 8+2\}, 10) = \max(5, 10) = 10$
$du_7 = \min(\max\{0+11, 0+13, 1+5, 1+12, 5+7, 6+3, 8+3, 10+2\}, 14) = \min(13, 14) = 13$

In general, entity e_1 is dominated by e_2 if all e_1's attribute values are equal to, or worse than, e_2's. The definition of 'equal to or worse than' is attribute dependent, and thus dominance rules must be specified by the user as part of the attribute definition. Careful attention must be paid to the dominance rules specified for attributes that are used to calculate other attribute values.

A.4 Implementation

A major contribution of this work is the design of a software framework that allows an efficient implementation of our nested column generation framework.

This implementation must be versatile enough to capture any common property of rostering problems, while at the same time being as efficient as a tailored implementation. This rather ambitious goal is met by the creation of a customizable software framework, where the problem definition is part of the input to the compiler, and thus the compiled code implicitly includes the problem definition. This setup requires the code to be recompiled whenever a new problem is encountered, or new rules are added to a problem. It is thereafter possible to solve multiple instances of the same problem with the executable program.

A.4.1 The problem definition

The definitions of all the entity attributes, together with the demands, make up the full definition of the problem. The problem definition is parsed into the code and in this way, the algorithm is customized for the problem being solved.

The user must specify the attributes to be tracked by each entity. Start time and end time are mandatory attributes, but all other attributes are customizable. Because these attributes are being specified at compile time, the specification includes code for tasks such as calculating new attribute values. The entities and their attributes can be listed as:

$$\begin{aligned}
\text{Shift} &:= (\mathit{shift\_start\_time}, \mathit{shift\_end\_time},\\
&\quad \mathit{shift\_custom\_att}_1, \ldots, \mathit{shift\_custom\_att}_n)\\
\text{Offstretch} &:= (\mathit{offstretch\_start\_time}, \mathit{offstretch\_end\_time},\\
&\quad \mathit{offstretch\_custom\_att}_1, \ldots, \mathit{offstretch\_custom\_att}_n)\\
\text{Onstretch} &:= (\mathit{onstretch\_start\_time}, \mathit{onstretch\_end\_time},\\
&\quad \mathit{onstretch\_custom\_att}_1, \ldots, \mathit{onstretch\_custom\_att}_n)\\
\text{Workstretch} &:= (\mathit{workstretch\_start\_time}, \mathit{workstretch\_end\_time},\\
&\quad \mathit{workstretch\_custom\_att}_1, \ldots, \mathit{workstretch\_custom\_att}_n)\\
\text{Rosterline} &:= (\mathit{rosterline\_start\_time}, \mathit{rosterline\_end\_time},\\
&\quad \mathit{rosterline\_custom\_att}_1, \ldots, \mathit{rosterline\_custom\_att}_n)
\end{aligned}$$

The definition of attributes of simple entities is straight forward and simply introduces the attribute with a name and a numeric type such as integer or floating point number.

The definition of attributes of composite entities requires more information as attribute values need to be calculated whenever new entities are formed. Recall that entities are always created by adding one entity to another parent entity. We generally describe this operation ‘accumulation’. The case where an object is created without a parent is called ‘initialization’. Thus, we have the following entity creation events:

Initialization:	shift(s)		$\rightarrow$ on-stretch
	on-stretch(o)	+ off-stretch(f)	$\rightarrow$ work-stretch
	work-stretch(w)		$\rightarrow$ roster-line
Accumulation:	on-stretch(o)	+ shift(s)	$\rightarrow$ on-stretch
	roster-line(r)	+ work-stretch(w)	$\rightarrow$ roster-line

In the attribute definitions for these composite entities, we have to specify the code used to calculate an attribute's value during these events. Dominance, costs and feasibility details must also be given. Thus, in addition to the attribute name and type, the following must be specified for these attributes:

- Feasibility type: Whether or not bounds apply on the feasible attribute values.
- Cost type: Any contribution that this attribute makes to the cost of the entity.
- Initialization code and Accumulation code: The code that runs to calculate the attribute's value when a new entity instance is created.
- Domination criterion: What it means for one entity's attribute value to be 'equal or worse than' another's.

All entities have start time and end time attributes specified in integer minutes from the start of the roster period. The definitions of these two attributes are shown below for on-stretches, and illustrate typical attribute definitions. Note that the initialization function of the on-stretch refers to the shift (s) from which it is initialized. Likewise, the accumulation function may use values of the parent on-stretch (o) and the new shift (s). The functions may also look up the value of any other attributes of these entities.

Attribute:	*onstretch_start_time*
Numeric Type:	Integer
Feasibility type:	All values feasible
Cost type:	No cost
Initialization function:	*onstretch_start_time* = *s.shift_start_time*
Accumulation function:	*onstretch_start_time* = *o.onstretch_start_time*
Domination Criterion:	Dominate on equal values

Attribute:	*onstretch_end_time*
Numeric Type:	Integer
Feasibility type:	All values feasible
Cost type:	No cost
Initialization function:	$onstretch\_end\_time = s.shift\_end\_time$
Accumulation function:	$onstretch\_end\_time = s.shift\_end\_time$
Domination Criterion:	Dominate on equal values

The start time and the end time of work-stretches and roster-lines are defined similarly.

In our framework, attributes are implemented as objects that have user-specified procedures for implementing feasibility checks, cost calculations, domination and initialization. For example, the *onstretch_start_time* attribute class definition might provide the following methods:

```
onstretch_start_time.IsFeasible() {return true}
onstretch_start_time.CalculateCost() {return 0}
onstretch_start_time.Initialize(Shift s) {return s.shift_start_time}
onstretch_start_time.Accumulate(Onstretch o, Shift s)
                                  {return o.onstretch_start_time}
onstretch_start_time.Dominates(Onstretch o)
                                  {return (value == o.shift_start_time)}
```

To simplify the task of defining attributes, the code framework provides many standard implementations for **IsFeasible()**, **CalculateCost()** and **Dominates()** that cover the most commonly occurring cases. These are implemented efficiently by using C++ templates.

Once the attributes have been defined, the code framework forms entity objects that contain the user-specified set of attributes for that entity. Entity objects also provide methods for feasibility checks, cost calculations, domination and initialization.

The code framework automatically generates these functions. For an on-stretch, for example, these include: *Onstretch.***Accumulate()** and *Onstretch.***Initialize()**, which accumulate/initialize all the attributes in an on-stretch; *Onstretch.***IsFeasible()** which returns `true` if and only if **IsFeasible()** is true for all the attributes; *Onstretch.***CalculateCost()**, which returns the sum of the costs of the attributes; and *Onstretch.***Dominates()** which is `true` (and hence allows domination) only if all the attributes allow domination.

A.4.2 Code generation

The standard way to implement a system such as we have described in C++ would be to build a class framework with a 'base' attribute classes that would then be specialized by the user using inheritance. To provide the entity functions such as **IsFeasible()** and **CalculateCost()**, the entity classes would then contain loops that iterated through all the attributes in the entity calling the appropriate attribute method. There are significant efficiency penalties associated with this approach. Firstly, a run time penalty is incurred every time an attribute call is made because (1) a function call must be made which has overhead, and (2), because the functions must be 'virtual', the exact attribute class to be called must be determined at run time (termed 'late binding'). Secondly, compile time optimizations such as loop unrolling and code in-lining cannot be performed.

The approach we have taken instead is implement the required loops at compile time by using the Boost Preprocessor Library of Karvonen and Mensonides (2001); see Dohn et al. (2010*b*) for full details of our implementation. At the conceptual level, the Boost library allows us to write code such as:

```
Onstretch.Initialize(Shift s) {
  for all on-stretch attributes do
      attribute.Initialize(s)
}
```

The Boost preprocessor unrolls this loop to give the following code which is then compiled.

```
Onstretch.Initialize(Shift s) {
    onstretch_start_time.Initialize(s)
    onstretch_end_time.Initialize(s)
    onstretch_custom_att_1.Initialize(s)
         ⋮
    onstretch_custom_att_n.Initialize(s)
}
```

Because each attribute is referred to explicitly in this expanded code, its type is available to the compiler, and so the correct **Initialize()** function is called directly without the run-time overhead of late binding. Furthermore, because functions such as **Initialize()** are typically very simple, they can be inlined by the compiler, thereby removing the need for a function call. As we discuss shortly, these efficiency gains lead to significantly reduced run times.

A.5 Example

To illustrate how the framework can be set up for a particular problem, an example is introduced below. The example is a nurse rostering problem from Middlemore Hospital in Auckland, New Zealand, which was also used by Smith (1995) and Engineer (2003). For clarity, a slight simplification of the problem is presented. The entities are defined below.

$$\begin{aligned}
\text{Shift} &:= (\mathit{shift\_start\_time}, \mathit{shift\_end\_time},\\
&\quad \mathit{shift\_type},\\
&\quad \mathit{shift\_paid\_hours},\\
&\quad \mathit{shift\_number\_of\_days\_on})\\
\text{Offstretch} &:= (\mathit{offstretch\_start\_time}, \mathit{offstretch\_end\_time},\\
&\quad \mathit{offstretch\_number\_of\_days\_off},\\
&\quad \mathit{offstretch\_number\_of\_weekends\_off},\\
&\quad \mathit{offstretch\_single\_day\_off})\\
\text{Onstretch} &:= (\mathit{onstretch\_start\_time}, \mathit{onstretch\_end\_time},\\
&\quad \mathit{onstretch\_paidhours},\\
&\quad \mathit{onstretch\_number\_of\_days\_on})\\
\text{Workstretch} &:= (\mathit{workstretch\_start\_time}, \mathit{workstretch\_end\_time},\\
&\quad \mathit{workstretch\_paidhours},\\
&\quad \mathit{workstretch\_number\_of\_days\_on},\\
&\quad \mathit{workstretch\_number\_of\_days\_off},\\
&\quad \mathit{workstretch\_number\_of\_single\_days\_off},\\
&\quad \mathit{workstretch\_feasible\_on\_off\_combination})\\
\text{Rosterline} &:= (\mathit{rosterline\_start\_time}, \mathit{rosterline\_end\_time},\\
&\quad \mathit{rosterline\_paidhours},\\
&\quad \mathit{rosterline\_number\_of\_days\_on},\\
&\quad \mathit{rosterline\_number\_of\_days\_off},\\
&\quad \mathit{rosterline\_number\_of\_single\_days\_off})
\end{aligned}$$

The definition of most of the attributes is straight forward. For example, the number of hours worked is accumulated in the attribute *paidhours* in all entities except in off-stretches, where it is not applicable. For on-stretches the attribute is defined as:

Attribute:	*onstretch_paidhours*
Numeric Type:	Integer
Feasibility type:	All values feasible
Cost type:	No cost
Initialization function:	$onstretch\_paidhours = s.shift\_paid\_hours$
Accumulation function:	$onstretch\_paidhours = o.onstretch\_paidhours + s.shift\_paid\_hours$
Domination Criterion:	Dominate on equal values

The definition of *workstretch_paidhours* is similar, where the initialization function becomes $workstretch\_paidhours = o.onstretch\_paidhours$. In *rosterline_paidhours* the value is accumulated over *workstretch_paidhours* and the feasibility type is changed to 'Bounded by upper and lower bound'. The actual values of the bounds are read as part of the data input and individually for each employee.

This problem includes a penalty discouraging long on-stretches. To implement this, the attribute *onstretch_number_of_days_on* has a piecewise linear cost function. The problem also specifies that 5 or more days on must be followed by at least 2 days off. This is modeled by the attribute *workstretch_feasible_on_off_combination* defined as:

Attribute:	*workstretch_feasible_on_off_combination*
Numeric Type:	Boolean
Feasibility type:	Must be `true`
Cost type:	No cost
Initialization function:	$workstretch\_feasible\_on\_off\_combination = (o.onstretch\_number\_of\_days\_on \leq 4)$ or $(f.offstretch\_number\_of\_days\_off \geq 2)$
Domination Criterion:	Ignore this for domination

Most of the other attributes have similarly simple definitions. These definitions are all specified in a single header file using a mixture of C++ code and Boost preprocessor commands.

The problem definition described above together with the actual data instance is all that is needed to solve a new problem. The problem definition is given in a modeling language type of description, where the problem definition is parsed into the code.

In addition to the problem definition, instance-specific data must be provided. This input data consists of four data sets. Two of these list the shifts and off-stretches with values of start time, end time and the customized attributes. Another set holds the demands that specify staff requirements in terms of combinations of overlapping shifts and staff skills. Finally, the fourth data set lists the employees along with their skills, preferences and employee-specific costs and bounds.

A.6 Computational results

To illustrate the capabilities of the algorithm, three real nurse rostering data instances are introduced, one from a hospital in New Zealand, and two from hospitals in Denmark. The three instances have different characteristics and it is therefore essential to have a versatile setup in order to be able to solve all three instances with the same approach. All three instances have a scheduling horizon of four weeks.

Instance A has 85 nurses which must be allocated shifts of 5 different types. This is the same instance as described in Section A.5 and Engineer (2003) also presents computational results for this instance. There are penalties for long on-stretches, split-weekends and for certain shift transitions. The roster includes a mix of full time and part time staff, with the exact number of hours to be worked in a roster-line being specified for each different nurse group. The nurses have skills and contractual agreements that limit each nurse to work a subset of all shifts and prohibit certain shift sequences. Demands are specified for various combinations of skills, and typically involve more than one shift type.

Instance B contains 28 nurses and 4 shift types. Several demands with varying skill requirements exist for each shift. 10 of the nurses are part time employees. The hours to be worked for the four weeks is specified with a small tolerance being allowed. A nurse cannot work two consecutive weekends and never more than one shift during a weekend. Furthermore, a sophisticated cost structure applies that distributes weekend shifts fairly and distributes off-days evenly over the weeks. This cost structure makes it very hard to find near-optimal solutions.

In Instance C, 40 nurses are scheduled on 18 shift types in two wards. Each nurse has an individually specified requirement on work hours per four weeks and a set of individual requests and preferences on shifts. Continuity is desired in one of the wards and a sequence of less than three successive shifts in the ward is penalized. Consecutive weekends on are undesired and no more than two weekends on duty in a month can be scheduled for each nurse.

To test the nested column generator using our compile-time customization approach, we embedded our new C++ column generator within the branch-and-cut-and-price framework of COIN-OR (Lougee-Heimer, 2003). To reduce the run times (and sacrifice guarantees of optimality), we followed the approach described by Engineer (2003) of allowing the user to artificially limit the number of entities generated in the subproblem, with the code then discarding poor entities when this limit was reached. We also used a partial depth first search referred to as *diving* in the branch-and-bound tree to find feasible solutions faster. Some tuning experiments were undertaken that indicated the best number of columns to add to the master from each column generation was 15.

Tests were run on a PC with a Dual Core AMD Opteron(tm) Processor 175 running 64 bit Linux with 2GB of RAM. The results of these are summarized in Table A.1. The runs detailed in the upper table are heuristic in that they were performed with artificial limits applied to the number of entities allowed during each column generation. The table shows LP and IP solution values, detailed timing and entity count information for the LP, branch and bound, and each of the different entity creation phases used by the nested generator. The total number of entities generated over each run are also shown, as well as how this number is reduced by dominance rules and feasibility tests and by applying the attribute bounds discussed in Section A.3.5. Note that the order in which this filtering is applied is different for the different entities. For reference, the true lower bound and the optimal solution from a test without entity limits are listed in the bottom of the table.

For Instances B and C, the root node lower bound is larger than the solution value of the best feasible solution. This is a consequence of the imposed entity limit, where columns with negative reduced cost may not be generated and hence the bounds may not be correct. On the other hand, the entity limits introduce a significant speed up, as evident from the run times as well as from the number of entities before and after discarding. Note that the algorithm terminates after finding a solution with value equal to or less than the lower bound. However, when bounds are heuristic, the solution found at this point is not guaranteed to be optimal.

The time distribution within the algorithm is clearly problem dependent. In one case (Instance C), the major part of the time is spent in the master problem. In another (Instance B), it is spent mainly in the subproblem and in the third (Instance A), time is divided equally between these. The distribution of time spent creating the different entity types within the pricing problem is again problem dependent and emphasizes the fact that each component of the column generator must be optimized to ensure high performance for all instances.

In all three instances, high-quality feasible solutions are found within 15 minutes. In Instance A and Instance C, the best solution found is optimal. The low values

	Instance A	Instance B	Instance C
Heuristic root node LP value	19.667	288.822	1.500
Heuristic first feasible IP solution value	23	296	21
Heuristic best feasible IP solution value	23	281	1
Seconds in root node	7.50	89.63	74.59
Seconds to find first IP feasible solution	23.95	180.77	562.99
Seconds to find best IP feasible solution	23.95	186.20	690.96
Total runtime (s)	26.91	192.56	722.04
- Solving LP	41.3%	10.2%	81.4%
- Branching	1.2%	0.2%	0.2%
- Overhead	13.1%	3.8%	5.2%
- Pricing	44.5%	85.8%	13.1%
- Setup	5.0%	1.4%	0.5%
- On-stretch	6.1%	0.7%	4.7%
- Work-stretch	25.7%	1.2%	7.0%
- Rosterline	7.7%	82.6%	0.9%
Tree size (number of nodes)	771	371	1,023
Max depth	383	185	329
Pricing problems solved	4,445	1,489	5,551
Columns generated	1,919	7,982	22,654
Onstretches Generated	1,941,902	3,823,761	167,578,440
Onstretches After Domination	1,656,646	2,789,875	150,308,919
Onstretches After Discard	1,656,646	2,365,988	49,509,176
Onstretches Feasible	1,422,949	2,298,844	49,509,176
Workstretches Generated	24,038,014	8,021,722	200,873,954
Workstretches Feasible	23,090,067	8,021,722	200,873,954
Workstretches After Bounding	7,978,248	7,682,419	153,047,740
Workstretches After Domination	5,888,045	3,016,510	146,601,429
Workstretches After Discard	5,888,045	3,016,510	3,478,809
Rosterlines Generated	27,358,806	848,767,690	51,165,962
Rosterlines After Bounding	8,499,618	318,240,021	31,683,310
Rosterlines After Domination	490,657	251,973,111	29,692,367
Rosterlines After Discard	490,657	12,501,839	1,150,622
True root LP value	19.667	234.000	1.000
Optimal IP solution value	23	235[*]	1
Seconds to find true root LP value	8.08	2,024.72	2,639.36
Seconds to find optimal IP solution	27.48	> 10 h	5,652.58

[*] For Instance B, it was not possible to prove optimality of this solution within 10 hours. However, the gap to the LP-lower bound is less than 0.5%.

Table A.1: Test results for the three data instances. Each instance has been solved twice. First, the column generator was run in heuristic mode (with limits applied to the number of entities created). Detailed statistics are given for these runs. For comparison, each problem is solved again to proven optimality (when possible).

for Instance A and C represent rosters with very few preferences being violated. In the latter case, only a single roster-line for one employee contains an undesired shift sequence, with all other preferences being met. In all three instances, all demand constraints are met without any violations.

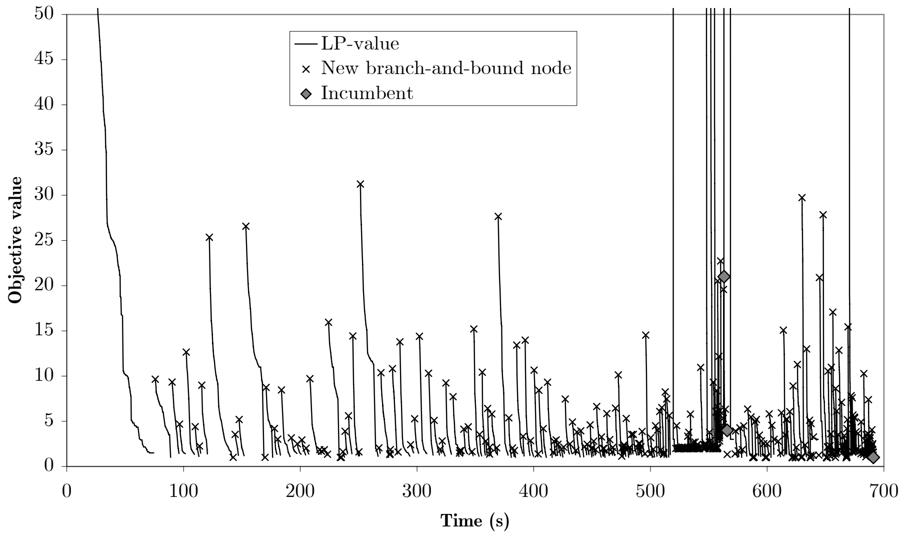


Figure A.7: Development of the LP values and integer incumbent solution over time when solving Instance C.

Figure A.7 shows the sequence of new LP and IP solution values generated over time when solving Instance C. The plot also indicates when a new node was selected in the branch and bound tree. The new node is often the child node of the previously solved node. In each node of the branch-and-bound tree, the LP-value is decreased by adding columns with negative reduced cost. Branching decisions remove columns and thereby increase the LP-value. Hence, the LP-value "jumps" when branching is applied. After 520 seconds, the objective values observed are slightly worse than before. This is a consequence of a tree diving strategy that favors depth-first search to quickly find feasible solutions. This strategy is successful, with the first feasible solution (with objective value 21) being found after 563 seconds. The subsequent search finds another feasible solution (with objective value 4) after 566 seconds and finally the optimal solution (with objective value 1) after 691 seconds. In column generation, there can be long sequences with no improvement in the objective value, because of degeneracy in the master problem. As explained in Section A.3.1, the solution process in each node is interrupted as soon as the LP value is equal to the lower bound of the root node. In our case, it is enough to remove the worst effects of degeneracy.

An important contribution of this work is the new compile-time customization approach and the benefit this gives over the more standard approach used in Engineer (2003) in which a single executable modifies its behavior based on problem-specific data loaded at run time. To quantity the speed differences between the two approaches, our new software was run on Instance A, and the dual vectors (and associated staff member) used for each column generate recorded. The column generator from Engineer (2003) was then run on Instance A using these 338 different dual values. The tests were run without entity limits. Detailed run times and entity counts were recorded for each entity generator. These showed that the number of on-stretches and work-stretches generated by the two systems are very similar. However, the new bounding scheme presented in Section A.3.5 reduced the number of roster-lines to 37% of those generated by the older system. After correcting for this, we found that over the 338 tests, speedup factors between 5 and 100 were observed, giving a weighted average of approximately 20. Clearly, the compile time approach is giving us a significant performance improvement.

A.7 Conclusions

We have successfully implemented a branch-and-price algorithm to solve the generalized rostering problem. The solution approach builds on a generic model and hence allows solution of problems with varying characteristics. From the literature and by looking at the rostering problems at hand, it was clear that a solution approach to the generalized rostering problem must be very flexible. At the same time, rostering problems are typically highly constrained and it is often a demanding task to even find feasible solutions. Therefore the solution approach must not only be flexible, but also very efficient.

To meet these requirements, we have modeled the problem as a generalized set partitioning problem and built a new branch-and-price algorithm to solve the problem. The pricing problem is solved in three stages which allows us to exploit the structure inherent in rostering problems, helping make the problem tractable in realistic settings. By explicitly embedding the problem definition in the program code, and compiling a separate executable for each new problem definition, we can ensure a high efficiency throughout the algorithm. Using this approach, it is possible to model all constraints seen in the 10 application of former projects (Bliddal and Tranberg, 2002; Nielsen, 2003; Engineer, 2003; Poulsen, 2004), as well as all commonly occurring constraints listed by Cheang et al. (2003) and Burke et al. (2009). By embedding the problem definition in the code, we can realize much of the efficiency of a purpose built implementation without the software development costs typically associated with such bespoke work. In a comparison to a former method, where the customization is included at run-time, a speed up of a factor 20 was observed. Furthermore, by running just the pre-processor, it is possible to view, compile and then debug the problem-specific code. This makes software development more intuitive as the actual problem is given explicitly in the code rather than embedded in complex data structures.

The value of the algorithm was illustrated for three different nurse rostering applications. The model captures all features of the realistic problems and provides high-quality solutions in less than 15 minutes for a scheduling horizon of four weeks. Heuristic components can be introduced in the future to allow for additional speedups.

Burke et al. (2004) state that:

> *The current state of the art is represented by interactive approaches which incorporate problem specific methods and heuristics (that are derived from specific knowledge of the problem and the required constraints) with powerful modern metaheuristics, constraint based approaches and other search methods.*

We believe that we have provided a viable alternative to these methods. Furthermore, our compile-time customization constitutes a contribution not only to rostering. This approach can be applied to a wide range of optimization problems, and is likely to deliver similar improvements in run time performance.

Acknowledgements

The first author was supported by the EliteForsk-grant of the Danish Ministry of Science, Technology and Innovation.

References

Al-Yakoob, S. M. and H. D. Sherali (2008). “A column generation approach for an employee scheduling problem with multiple shifts and work locations”. In: *Journal of the Operational Research Society* 59.1, pp. 34–43.

Bard, J. F. and H. W. Purnomo (2005*a*). "A column generation-based approach to solve the preference scheduling problem for nurses with downgrading". In: *Socio-Economic Planning Sciences* 39.3, pp. 193–213.

Bard, J. F. and H. W. Purnomo (2005*b*). "Preference scheduling for nurses using column generation". In: *European Journal of Operational Research* 164.2, pp. 510–534.

Beliën, J. and E. Demeulemeester (2006). "Scheduling trainees at a hospital department using a branch-and-price approach". In: *European Journal of Operational Research* 175.1, pp. 258–278.

Beliën, J. and E. Demeulemeester (2008). "A branch-and-price approach for integrating nurse and surgery scheduling". In: *European Journal of Operational Research* 189.3, pp. 652–668.

Beliën, J. and E. Demeulemeester (2007). "On the trade-off between staff-decomposed and activity-decomposed column generation for a staff scheduling problem". In: *Annals of Operations Research* 155.1, pp. 143–166.

Berghe, G. V. (2002). "An Advanced Model and Novel Meta-Heuristic Solution Methods to Personnel Scheduling in Healthcare". PhD thesis. School of Computer Science and Information Technology, University of Nottingham.

Bliddal, C. and O. Tranberg (2002). "Vagtplanlægning Med Constraint Programming". MA thesis. Informatics and Mathematical Modelling, Technical University of Denmark, Denmark.

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

Burke, E. K., T. Curtois, R. Qu, and G. Vanden-Berghe (2009). *Problem Model for Nurse Rostering Benchmark Instances*. Tech. rep. http://www.cs.nott.ac.uk/∼tec/NRP/papers/ANROM.pdf. ASAP, School of Computer Science, University of Nottingham, Jubilee Campus, Nottingham, UK.

Chabrier, A. (2006). "Vehicle Routing Problem with elementary shortest path based column generation". Ed. by Louis-Martin Rousseau Michel Gendreau Gilles Pesant. In: *Computers and Operations Research* 33.10, pp. 2972–2990.

Cheang, B., H. Li, A. Lim, and B. Rodrigues (2003). "Nurse rostering problems–a bibliographic survey". In: *European Journal of Operational Research* 151.3, pp. 447–460.

Curtois, T. (2010). *EURO Working Group on Automated Timetabling*. Homepage. http://www.asap.cs.nott.ac.uk/watt/resources/employee.html.

Desrochers, M. (1988). *An Algorithm for the Shortest Path Problem with Resource Constraints*. Tech. rep. Technical Report Les Cahiers du GERAD G-88-27, University of Montreal, Montreal.

Desrosiers, J. and M. E. Lübbecke (2005). "A Primer in Column Generation". In: *Column Generation*. Ed. by G. Desaulniers, J. Desrosiers, and M.M. Solomon. Springer, New York. Chap. 1, pp. 1–32.

Desrosiers, J., F. Soumis, and M. Desrochers (1984). "Routing with time windows by column generation". In: *Networks* 14.4, pp. 545–565.

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

Dumitrescu, I. and N. Boland (2003). "Improved preprocessing, labeling and scaling algorithms for the weight-constrained shortest path problem". In: *Networks* 42.3, pp. 135–153.

Eitzen, G., D. Panton, and G. Mills (2004). "Multi-Skilled Workforce Optimisation". Ed. by Houyuan Jiang, Mohan Krishnamoorthy, and David Sier. In: *Annals of Operations Research* 127.1-4, pp. 359–372.

Engineer, F. G. (2003). "A Solution Approach to Optimally Solve the Generalized Rostering Problem". MA thesis. Department of Engineering Science, University of Auckland, New Zealand.

Ernst, A. T., H. Jiang, M. Krishnamoorthy, B. Owens, and D. Sier (2004*a*). "An Annotated Bibliography of Personnel Scheduling and Rostering". Ed. by Houyuan Jiang, Mohan Krishnamoorthy, and David Sier. In: *Annals of Operations Research* 127.1-4, pp. 21–144.

Ernst, A. T., H. Jiang, M. Krishnamoorthy, and D. Sier (2004*b*). "Staff scheduling and rostering: A review of applications, methods and models". Ed. by E. Burke and S. Petrovic. In: *European Journal of Operational Research* 153.1, pp. 3–27.

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

Jaumard, B., F. Semet, and T. Vovor (1998). "A generalized linear programming model for nurse scheduling". In: *European Journal of Operational Research* 107.1, pp. 1–18.

Karvonen, V. and P. Mensonides (2001). *Preprocessor Metaprogramming*. C++ library. http://www.boost.org/ (Boost 1.36.0: 14/08/2008).

Kohl, N. and S. E. Karisch (2004). "Airline Crew Rostering: Problem Types, Modeling, and Optimization". Ed. by Houyuan Jiang, Mohan Krishnamoorthy, and David Sier. In: *Annals of Operations Research* 127.1-4, pp. 223–257.

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

Lübbecke, M. E. (2005). "Dual variable based fathoming in dynamic programs for column generation". Ed. by S. Martello and E. Pesch. In: *European Journal of Operational Research* 162.1, pp. 122–125.

Maenhout, B. and M. Vanhoucke (2008). *Branching Strategies in a Branch-and-Price Approach for a Multiple Objective Nurse Scheduling Problem*. Tech. rep. Faculty of Economics and Business Administration, Ghent University, Belgium.

Mason, A. J. and M. C. Smith (1998). “A Nested Column Generator for Solving Rostering Problems with Integer Programming”. In: *International Conference on Optimisation: Techniques and Applications.* Ed. by L. Caccetta, K. L. Teo, P. F. Siew, Y. H. Leung, L. S. Jennings, and V. Rehbock, pp. 827–834.

Nielsen, D. (2003). “A Broad Application Optimisation-Based Rostering Model”. PhD thesis. Department of Engineering Science, University of Auckland, New Zealand.

Poulsen, H. T. (2004). “Vagtplanlægning for Sygeplejersker - et Kombinatorisk Optimeringsproblem”. MA thesis. Datalogisk Institut, University of Copenhagen, Denmark.

Righini, G. and M. Salani (2006). “Symmetry helps: Bounded bi-directional dynamic programming for the elementary shortest path problem with resource constraints”. Ed. by Ulrich Faigle, Leo Liberti, Francesco Maffioli, and Stefan Pickl. In: *Discrete Optimization* 3.3, pp. 255–273.

Ryan, D. M. and B. Foster (1981). “An integer programming approach to scheduling”. Ed. by A. Wren. In: *Computer Scheduling of Public Transport. Urban Passenger Vehicle and Crew Scheduling. Proceedings of an International Workshop*, pp. 269–280.

Smith, M. C. (1995). “Optimal Nurse Scheduling Using Column Generation”. MA thesis. Department of Engineering Science, University of Auckland, New Zealand.

Appendix B

An Integrated Approach to the Ground Crew Rostering Problem with Work Patterns

Richard Lusby, Anders Dohn, Troels Martin Range, and Jesper Larsen

Submitted to: *Journal of the Operational Research Society* (2010).

An Integrated Approach to the Ground Crew Rostering Problem with Work Patterns[*]

Richard Lusby[1], Anders Dohn[1], Troels Martin Range[2] and Jesper Larsen[1]

This paper addresses the Ground Crew Rostering Problem with Work Patterns, an important manpower planning problem arising in the ground operations of airline companies. We present a cutting stock based integer programming formulation of the problem and describe a powerful decomposition approach, which utilizes column generation and variable fixing, to construct efficient rosters for a six month time horizon. The time horizon is divided into smaller blocks, where overlaps between the blocks ensure continuity. The proposed methodology is able to circumvent one step of the conventional roster construction process by generating rosters directly based on the estimated workload. We demonstrate that this approach has the additional advantage of being able to easily incorporate robustness in the roster. Computational results on real-life instances confirm the efficiency of the approach.

Keywords: Manpower Planning, Optimization, Cutting Stock Problem.

B.1 Introduction

In this paper, we consider the Ground Crew Rostering Problem with Work Patterns (GCRPWP) for a major European airline. Ground crew comprises all of the crew that an airline employs at an airport to take care of passengers and aircrafts in order to facilitate a smooth operation. This could be, for instance, customer service representatives or ramp service workers. The rostering of these workers is a complex, multi-stage planning process, which starts with the initial forecast of labour requirements and concludes with the construction of a roster that covers the anticipated workload as well as possible. The roster specifies what days of a prespecified time horizon each employee will work as well as the type of work they will do. At this airline, the roster is published before the

[*]Submitted to: *Journal of the Operational Research Society* (2010).
[1]Department of Management Engineering, Technical University of Denmark, Produktionstorvet, 2800 Kongens Lyngby, Denmark.
[2]Department of Business and Economics, University of Southern Denmark, Campusvej 55, 5230 Odense M, Denmark.

start of each season and states what each person will be doing for the next six months.

Rostering staff can be seen as the process of assigning an employee to a sequence of shifts. A shift refers to a block of work, typically around nine hours in duration and is associated with a specific task. Here, it is assumed that shifts have already been defined and the challenge is to generate an efficient roster for the employees while respecting the legislation and the staff agreements. The most important requirement, in this particular problem, is that all staff work the same *work pattern*. A work pattern specifies the number of consecutive days of work as well as the number of required days of rest in between. For instance, the airline uses a 6&3 pattern, which states that an employee will be assigned six days of work and then receive three days rest. The roster should typically cover the workload, while ensuring a certain degree of robustness. Uncovered work is allowed, but incurs a penalty. Robustness is incorporated by providing over coverage on the estimated workload.

In this paper, we propose a cutting stock based mixed integer programming (MIP) formulation of the problem. Initially, it is assumed that the required staffing level is specified for each shift. To solve the model, we decompose the six month time horizon into smaller, computationally tractable blocks. A procedure that combines column generation and variable fixing is developed to solve each block. The blocks are solved sequentially and consistency between the rosters of successive blocks is enforced by shift fixing in the overlaps of blocks. The initial model is then extended to generate rosters directly on the forecast workload, and demands by shift are made dispensable. The number of employees working any given shift is determined by the new model as a part of the solution and robustness is built into the solution. Finally, we test and compare the proposed methodology on several instances arising in practice.

The outline of the paper is as follows. Section B.2 presents a review of the research in this area. Section B.3 provides a more formal definition of the problem and presents the mathematical programming formulation. A discussion of the proposed optimization-based heuristic is given in Sections B.4 and B.5. In Section B.6, the initial model is modified to generate rosters directly from the workload, while Section B.7 explains how we incorporate robustness. Computational results on real-life instances are presented in Section B.8 and conclusions from this research are summarized in Section B.9.

B.2 Literature review

Crew rostering is a classical optimization problem. It is a very important problem as people are often both a critical and an expensive part of the operations. Utilizing the available manpower as effectively as possible can lead to significant potential savings. Furthermore, good crew planning ensures a high job satisfaction, which in turn results in higher productivity.

An improved productivity can be obtained by using computerized decision-support tools based on advanced optimization techniques. The development has been pioneered by the airline industry (see e.g. Barnhart et al., 2003). Nowadays, the rostering of pilots and cabin crew without such tools is unthinkable for all of the major airlines. In Ryan and Foster (1981), the authors estimate the annual savings of Air New Zealand to be around NZD 15 million, more than 6% of the annual estimated crew costs. Similarly, in Anbil et al. (1991) annual savings in excess of USD 20 million are reported. This is roughly 1% of the annual estimated crew costs. The underlying optimization techniques have since penetrated into other areas of manpower planning and rostering. For example, many train companies now use optimization methods for rostering drivers and conductors (see e.g. Abbink et al., 2005). Other prominent areas of rostering are call centres, protection and emergency services, venue management, retail, and civic services. A review can be found in Ernst et al. (2004*c*).

While there has been a large and continued focus on optimization within the rostering of pilots and cabin crew, the successful results have not lead airlines to thoroughly investigate the potential of applying similar methods to the rostering of ground crew. This is evidenced by the lack of research in this area. One of the few papers is by Dowling et al. (1997). The authors present a solution approach for rostering around 500 staff at a large international airport. The proposed algorithm is based on simulated annealing and rosters airline ground staff over a monthly planning period, where the objective is to minimize idle time. Other contributions include Brusco et al. (1995) and Chu (2007). Brusco et al. describe a manpower planning tool for United Airlines (UA). This tool produces tour schedules for which employees bid using a seniority-based system and is used by UA at over 100 airports. It is a two-phase approach; the first phase generates requirements for labour using a set cover formulation, while the second phase is a simulated annealing based metaheuristic that attempts to improve the tours found in the first phase. Chu proposes a goal programming approach to generate daily schedules for baggage handling at Hong Kong International Airport.

Despite not being a well studied problem itself, the GCRPWP does bare strong similarities to many other rostering problems arising in various contexts. In particular, nurse rostering and physician scheduling, both of which arise in the area of health care, are two problems which possess the strongest similarities. In nurse rostering, one must usually provide suitably qualified nurses to cover the workload demand based on the number of patients in the ward, while satisfying a wide range of local and national working regulations. Similarly, in physician scheduling, one must construct rosters for doctors at hospitals so each shift of every day is covered by exactly one physician. Although these problems can be modelled similarly, they are slightly more complicated than the GCRPWP. Firstly, the lengths of the on and off periods are typically not fixed, as is the case here, but should be within certain bounds. Furthermore, both problems attempt to satisfy as many individual staff requests as possible, while we attempt to construct anonymous rosters for groups of staff members. The most recent surveys on the nurse rostering problem are Cheang et al. (2003) and Burke et al. (2004), while a good overview on physician scheduling can be found in Gendreau et al. (2006*a*). In addition, some interesting questions on the lack of transition from academia to industry are raised in Kellogg and Walczak (2007).

The use of work patterns is, however, also not uncommon in rostering problems. For example, Alfares (2002) describes a particular rostering problem where a 14&7 work pattern is required. The author considers cyclic weekly demand and, besides minimizing the size of the labour force, attempts to use as few different patterns as possible. Both Vohra (1987) and Alfares (1997) consider day off scheduling assuming a 5&2 work pattern. Alfares (1997) presents a two-phase algorithm for finding the optimal allocation of patterns, while Vohra (1987) provides results on the minimum workforce size required. The three papers consider a somewhat simpler problem than the GCRPWP in that they are only concerned with determining an optimal allocation of days off. In the GCRPWP we must also include the subsequent step of shift allocation if an employee is assigned a particular day on.

One major difficulty with the GCRPWP is the length of the rostering horizon. Six months is far too large to solve in one model. We develop a decomposition approach that breaks the rostering horizon into blocks of manageable size and then solve a sequence of integrated optimization models to construct rosters that span the six month period. This approach can be seen as a form of the iterative sweeping method described in Eveborn and Rönnqvist (2004). Since excessive solution times are undesirable, to accelerate the solution times for each of the optimization models we implement a heuristic variable fixing routine when forcing integrality. This is a well-known approach (see e.g. Desaulniers et al., 2002 and Danna and Pape, 2005). Wäscher and Gau (1996) evaluate several integer fixing heuristics for the cutting stock problem.

When solving practical optimization problems, it is also essential that one includes some degree of robustness in the solution. That is, one should incorporate flexibility in the solution to guard against unexpected uncertainties in the input data. This is becoming an increasingly popular field of research as companies realize the potential savings by not having to re-optimize their schedule when something unforeseen occurs. The majority of recent contributions have appeared in the airline industry (see e.g. Clausen et al. (2010); Burke et al. (2010); Weide et al. (2010)). Here, robustness is incorporated by specifying a certain contingency of over coverage. This is to account for any unexpected increases and/or delays in the forecast workload, and can also compensate for absent employees.

The proposed solution approach has similarities with the set partitioning formulations usually found in the literature on scheduling. Possible solution methods include Mehrotra et al. (2000), where shift scheduling problems are solved by a branch-and-cut approach, and Eveborn and Rönnqvist (2004), where a branch-and-price approach forms the basis of a practical scheduling system. Branch-and-price has also been applied with success in many cabin crew rostering problems; see e.g. Day and Ryan (1997). This research contributes to the literature on crew rostering by providing a powerful decomposition method to the GCRPWP, which is capable of finding robust solutions, which are proven to be within a few percent of optimality.

B.3 The Ground Crew Rostering Problem with Work Patterns

In this section, we consider the GCRPWP in more detail. In particular, we provide a formal definition of the problem and present a cutting stock based integer programming formulation. To aid in the discussion, we begin by introducing the required terminology and notation.

The GCRPWP entails assigning a set of employees $\mathcal{E}$ (where $|\mathcal{E}| = n$) to a set of *shifts* $\mathcal{S}$ (indexed from $1, \ldots, S$). The required employee demand on each shift $s \in \mathcal{S}$ must be satisfied as closely as possible. Having too few employees on any given shift is termed *under coverage*, which is undesirable.

In this context, a shift refers to a block of work that has a given start time e_s, a given end time l_s, and a day $d \in \mathcal{D}$ on which the shift starts. Within a shift, there are breaks, where no work is carried out. It is important to account for breaks, when calculating the amount of work that can be done within a shift. The set $\mathcal{D}$ (indexed from $1, \ldots, D$) denotes the set of all days in the time horizon, while we denote the set of all shifts on day $d \in \mathcal{D}$ as $\mathcal{S}_d \subseteq \mathcal{S}$. The number of employees required for shift $s \in \mathcal{S}$ is given as q_s. It is assumed that each staff member can perform at most one shift on any given day. In constructing a sequence of shifts for any employee, one must respect several practical constraints. Typically, one must ensure that each employee has a certain minimum rest time between any pair of consecutive shifts and that no employee is assigned more than a certain number of consecutive night shifts. The classification of a shift as being a night shift depends on the shift start time. As stated earlier, it is also required that all employees work the same work pattern. This specifies the number of days an employee will work consecutively, (*on-stretch*), and the number of days of consecutive break, (*off-stretch*). During an off-stretch an employee can not be assigned any shifts. Here, we consider a 6&3 work pattern. That is, each employee must be assigned six consecutive days (i.e. six shifts) of work before being assigned a three day consecutive break. Hence, this is a work pattern of length nine days. The fixed nature of the work pattern ensures that all employees work, on average, the same number of days per week. A legal sequence of on and off stretches spanning the time horizon gives a *roster-line* for a particular employee or group of employees. The set of roster-lines for all employees together constitute the *roster*.

The use of work patterns limits the number of feasible roster-lines; however, the pattern is staggered across the employees to ensure an even distribution of off days. One can hence identify a set of *pattern groups* $\mathcal{G}$ based on a given work pattern. For example, an employee could start the time horizon on the first day of the 6&3 pattern and thus work the first six days of the time horizon, or the employee could start on the eighth day of the pattern and therefore have the first day of the time horizon off. Naturally, all combinations in between are also possible and this yields nine groups with different pattern offsets. A pattern group consists of the employees always working on the same days. Associated with each pattern group $g \in \mathcal{G}$ is an upper bound m_g on the number of employees in the pattern group. This can be used, for instance, to force consistency with a previously generated set of roster-lines.

In addition to the hard constraints, which must be respected, it is often desirable to satisfy several soft constraints when constructing rosters. A soft constraint is a constraint that one tries to satisfy if possible; however, it can be violated if necessary. If it is violated, the violation is minimized. An important soft constraint in the GCRPWP is that employees should receive the maximum number of hours off during their three day break. That is, the first shift of an on-stretch should be a shift starting late in the day, while the last shift should be one finishing early in the day. Inclusion of this soft constraint is described in Section B.4.2.

A key difference between the GCRPWP and many other rostering problems is that the aim is not to find individual roster-lines directly, but rather roster-lines that several employees may work. All staff are assumed to be able to work any shift and due to the 6&3 work pattern, it is impossible to take into consideration such individual preferences as particular weekends on or off. One can hence formally define the GCRPWP as follows: Given a set of employees, a set of shifts (each demanding a certain number of employees), and a pattern, find an allocation of employees to legal roster-lines such that the total cost of the roster is minimized.

The GCRPWP can be formulated as the following mixed integer program (MIP). In what follows, we denote the set of all legal roster-lines as $\mathcal{R}$. We introduce a general integer decision variable x_r for each roster-line $r \in \mathcal{R}$ that counts the number of times roster-line r is used in the solution. We also introduce the binary indicator variables a_{sr} and a_{gr}. The former indicates whether or not shift $s \in \mathcal{S}$ is contained in roster-line $r \in \mathcal{R}$, while the latter indicates whether or not roster-line $r \in \mathcal{R}$ belongs to pattern group $g \in \mathcal{G}$. Additionally, we define a decision variable u_s for each shift $s \in \mathcal{S}$, which indicates the level of under coverage on the shift. A unit of under cover on shift $s \in \mathcal{S}$ is assumed to cost $\check{c}_s$. Finally, we denote the cost of any legal roster-line $r \in \mathcal{R}$ as c_r. This cost reflects the penalties incurred in not satisfying soft constraints.

$$\min \sum_{r \in \mathcal{R}} c_r x_r + \sum_{s \in \mathcal{S}} \check{c}_s u_s, \tag{B.1}$$

s.t.

$$\sum_{r \in \mathcal{R}} a_{sr} x_r + u_s \geq q_s \qquad \forall s \in \mathcal{S}, \tag{B.2}$$

$$\sum_{r \in \mathcal{R}} a_{gr} x_r \leq m_g \qquad \forall g \in \mathcal{G}, \tag{B.3}$$

$$\sum_{r \in \mathcal{R}} x_r \leq n, \tag{B.4}$$

$$u_s \geq 0 \qquad \forall s \in \mathcal{S}, \tag{B.5}$$

$$x_r \in \mathbb{Z}_+ \qquad \forall r \in \mathcal{R}. \tag{B.6}$$

The objective function (B.1) minimizes the total cost of the roster-lines as well as the sum of the penalties incurred in not satisfying the demand of each shift. The first set of constraints (B.2) ensures that the total demand for any shift is satisfied, possibly through the use of the relevant under cover variable. Constraints (B.3) restrict the number of roster-lines of a particular pattern group to be no more than the maximum number permitted, while (B.4) is a constraint on the total number of staff. It prevents one from assigning more roster-lines than there are employees. Constraints (B.5) and (B.6) ensure that all decision variables are non-negative. In addition, all x_r variables are also required to be integer. One can observe that the above formulation possesses strong similarities to the well known cutting stock formulation (see Amor and Carvalho, 2005). While there are relatively few constraints ($|\mathcal{G}| + |\mathcal{S}| + 1$), there can potentially be an exponential number of variables. In the next section, we briefly introduce the column generation approach for solving problems of this nature, before describing, in detail, how it can be applied to the GCRPWP.

B.4 Column Generation

Column generation is a well known technique for solving large scale linear programming problems in which it is impossible to explicitly consider all of the variables in the problem. The approach requires one to decompose the original problem into two optimization problems, which are termed the *master* and the *pricing problem*, respectively.

B.4.1 The Master Problem

The master problem is a restricted version of the original problem, containing only a subset of the variables, while the pricing problem is an optimization problem that attempts to identify potential entering variables (columns) for the master problem. The fundamental idea of column generation is that since the majority of the variables in the original problem will be non-basic at an optimal solution, one need only consider, and generate, those variables that have the potential to improve the objective function. The objective function of the pricing problem is hence the reduced cost calculation for the non-basic variables given the dual variable values for the optimal master solution.

Column generation is an iterative procedure between the master and pricing problem. The master problem solves to optimality a restricted version of the original problem and the pricing problem, using the dual variables of the optimal master solution, implicitly prices all non-basic variables and finds the one with the most negative reduced cost. This variable is then added to the master problem. This procedure continues until the pricing problem can not identify a master variable with negative reduced cost. In which case, optimality of the original problem has been obtained. If one is solving a MIP, integrality can be achieved by embedding the LP column generation methodology in a branch-and-bound framework, termed branch-and-price. For an introduction to column generation, we refer to Desrosiers and Lübbecke (2005).

To apply column generation to the GCRPWP, we first relax the integrality requirements on the x_r variables. That is, constraints (B.6) are replaced with

$$x_r \in \mathbb{R}_+, \ \forall r \in \mathcal{R}. \tag{B.7}$$

The *relaxed master problem* can then be identified as model (B.1)-(B.5) and (B.7). The relaxed master problem with only a subset of the possible roster-lines from $\mathcal{R}$ is termed the *restricted master problem.* Using the dual vector of the optimal solution of the restricted master problem, the role of the pricing problem is to identify the non-basic roster-line with the most negative reduced cost. That is, one must find:

$$\min_{r \in \mathcal{R}} \left(c_r - \sum_{s \in \mathcal{S}} a_{sr} \pi_s - \sum_{g \in \mathcal{G}} a_{gr} \mu_g - \gamma \right), \tag{B.8}$$

where π_s is the dual variable value on the constraint of type (B.2) associated with shift $s \in \mathcal{S}$, μ_g is the dual variable value on the constraint of type (B.3) associated with pattern group $g \in \mathcal{G}$, and γ is the dual variable on Constraint (B.4). All constraints defining feasibility of a roster-line must be included in the pricing problem. For the GCRPWP, identifying the most negative reduced cost roster in the pricing problem entails solving a resource constrained shortest path problem.

In solving the GCRPWP, we must construct rosters that span a six month period. Since it is computationally intractable to consider such a long time horizon in one MIP, we present a decomposition approach that divides the six month time horizon into several shorter blocks, each of which has an associated master problem and a pricing problem. The blocks are solved in sequence where the partial roster of one block is used as input to the subsequent block. To allow any partial roster to be easily augmented with that of the successive block, the blocks are defined so that they have an overlapping time period equal to the number of days in the on-stretch. The size of the overlap is the smallest that ensures that each pattern group has at least one day off in the overlap or on the day immediately following the overlap. Days off are important in this context since they provide a starting point for the pricing problem in which all shift transitions are feasible. Proceeding from a day off allows one to easily enforce the continuation of particular patterns rather than individual roster-lines in the pricing problem of the subsequent block. Furthermore, providing an overlap allows the succeeding block to resolve part of the preceding block and correct for any bad choices made as a result of optimizing over a time horizon that is too short. Hence, our strategy within the overlap is to roll back to the last day off for each pattern, fix all shifts assigned up until this point, and resolve all later days in the optimization of the subsequent block. As a result, the pricing problems of all blocks (other than those of the first block) have a different time horizon.

	Days																		
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Group 1	1	1	1	1	1	1	0	0	0	1	1	1	1	1	1	0	0	0	1
Group 2	1	1	1	1	1	0	0	0	1	1	1	1	1	1	0	0	0	1	1
Group 3	1	1	1	1	0	0	0	1	1	1	1	1	1	0	0	0	1	1	1
Group 4	1	1	1	0	0	0	1	1	1	1	1	1	0	0	0	1	1	1	1
Group 5	1	1	0	0	0	1	1	1	1	1	1	0	0	0	1	1	1	1	1
Group 6	1	0	0	0	1	1	1	1	1	1	0	0	0	1	1	1	1	1	1
Group 7	0	0	0	1	1	1	1	1	1	0	0	0	1	1	1	1	1	1	0
Group 8	0	0	1	1	1	1	1	1	0	0	0	1	1	1	1	1	1	0	0
Group 9	0	1	1	1	1	1	1	0	0	0	1	1	1	1	1	1	0	0	0

Figure B.1: Block overlap and shift fixing

Figure B.1 further explains the block overlap and shift fixing concept. Let us suppose we are rostering a 6&3 pattern, that the first block runs from day 1 to day 18, and that the second block begins on day 13. The overlap contains days 13 to 18. Figure B.1 indicates for each day of the 19 day period, if a pattern is on (with a one) or if a pattern is off (with a zero). The (pattern, day) combinations marked with gray give the shifts that will be fixed in the second block, while all other days on within the overlap are resolved. For example, all shifts for roster-lines on pattern 3 are fixed on day 13, but resolved for days 17 and 18. Hence, the pricing problem for this pattern in the second block is defined from day 17 onwards only.

B.4.2 Pricing Problem

As described previously, the pricing problem determines whether any columns with negative reduced costs exist. If such columns exist, then these must be generated. In this paper, we will use a pricing problem for each group as it makes the identification of legal roster-lines easier. To identify legal roster-lines for a given group, we construct a directed acyclic graph, where the nodes represent possible activities and the arcs represent transitions between activities. An activity can be to work a shift on a certain day or to have the day off. With some additional constraints described below, the identification of a legal roster-line amounts to solving a resource constrained shortest path problem in an acyclic graph. A feasible roster-line must satisfy the following hard constraints:

1. no staff member can perform more than one shift on any day,

2. an employee must have at least 10 hours of rest between consecutive shifts,

3. given group $g \in \mathcal{G}$, an employee has to work the work pattern corresponding to that group,

4. no more than three consecutive night shifts are allowed.

Constraints 1-3 can be handled in the construction of the directed acyclic graph, whereas Constraint 4 must be enforced using a resource. In addition to the hard constraints, we also have the following soft constraints

5. the first shift on an on-stretch should be a late shift,

6. the last shift on an on-stretch should be an early shift.

The soft constraints are handled in the objective function of the pricing problem and will be discussed later.

Before describing the pricing algorithm in detail, we first expand the notation. We let $\mathcal{S}' = \mathcal{S} \cup \{S+1, \ldots, S+D\}$ be the index set of activities, where activity $S+d$ corresponds to having day d off. Hence, we let $\mathcal{S}'_d = \mathcal{S}_d \cup \{S+d\}$ be the possible activities on day d. To be able to distinguish between night shifts and the remaining shifts, we define $\mathcal{N} \subseteq \mathcal{S}$ as the index set of night shifts. We let $v = (v_0, \ldots, v_p)$ be a binary vector of length $(p+1)$ corresponding to the number of days in the work pattern. For example, v has length 9 when applying a 6&3 work pattern. Each entry in v indicates whether the day is a day on or a day off. The pattern $(1,1,1,1,1,1,0,0,0)$ states that the six first days is the on-stretch and the three last days is the off-stretch. Since we may need to use the pricing problem in different settings, e.g. for different day intervals and different work patterns, we construct a representation which is sufficiently general to accommodate the required settings.

The individual roster-lines are sequences of on-stretches and off-stretches. Each activity has a time interval of execution and this gives a natural ordering of activities, where some activities are successors of others. This ordering gives rise to an acyclic directed graph, which we refer to as the underlying graph. In the following, we denote ω as the origin and δ as the destination. For a given day interval $[d_1, d_2]$ we denote

$$\mathcal{V}'(d_1, d_2) = \bigcup_{d=d_1}^{d_2} \mathcal{S}'_d$$

as the index set of shift nodes joined with the index set of the day-off nodes for all days between d_1 and d_2. The set of vertices of the graph will then be $\mathcal{V}(d_1, d_2) = \{\omega, \delta\} \cup \mathcal{V}'(d_1, d_2)$.

We will refer to $\mathcal{A}(d_1, d_2)$ as the set of arcs in the graph. On each day, exactly one activity has to be carried out. This constitutes layers in the graph, i.e. one layer for each day d in the interval $[d_1, d_2]$. In the set of nodes $\mathcal{V}'(d_1, d_2)$ it is only possible to progress from one day to the next day, thus only arcs between nodes $i \in \mathcal{S}'_d$ and $j \in \mathcal{S}'_{d+1}$ for $d = d_1, \ldots, d_2 - 1$ are included. The origin ω only has arcs leaving it. For shift fixing, we need to be able to fix activities for days d_1 to day h where $h \leq d_2$. To make this possible, we include an arc from the origin to all nodes $i \in \mathcal{V}'(d_1, d_2)$. Later, we will eliminate the arcs which are not allowed from the origin. The destination node δ only has entering arcs. As we have to have exactly one activity each day, it is only possible to enter the destination from the last day in the interval. Hence, the only arcs (i, δ) that are allowed to enter the destination are those with $i \in \mathcal{S}'_{d_2}$. Note that we also eliminate any transition between two shifts that does not satisfy the 10-hour rule.

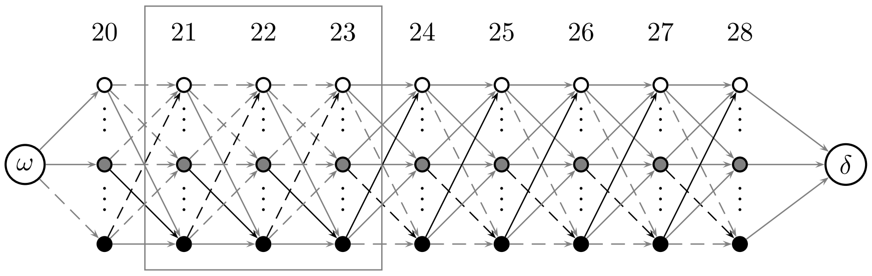


Figure B.2: Example of 6&3 pattern

In Figure B.2 we give an example of the underlying graph for the day interval $[20, 28]$. Each of the days has a column of nodes corresponding to the possible activities on that day. The black nodes are day-off nodes, the gray nodes are the night-shift nodes and the white nodes are the remaining shifts. The nodes are therefore partitioned horizontally. Each partition may have multiple nodes, of each kind, in each column, but for simplicity we have only shown one node. The transitions between activities are shown as arcs between the layers. The black arcs are those which are penalized due to soft constraint violations, whereas the gray arcs have a cost of zero. The difference between dashed arcs and filled arcs is explained later.

A path $P = (w_0, \dots, w_p)$ from the origin $w_0 = \omega$ to the destination $w_p = \delta$ can be translated directly into a roster-line (and vice versa) as the nodes in the path correspond to individual activities. Furthermore, any path will have at most one shift each day and will satisfy the 10-hour rule. We still need to satisfy the constraints on the specific patterns for each group and the requirement that no employee can have more than three consecutive night shifts.

One can easily apply a work pattern to the underlying graph. With any arc $(i, j) \in \mathcal{A}(d_1, d_2)$ we associate a binary value $u(i, j)$, which is equal to 1, if and only if, we allow the arc to be used in the solution. We put the value of $u(i, j) = 1$ for all arcs $(i, j) \in \mathcal{A}(d_1, d_2)$ unless it is stated otherwise. This value allows us to use the same underlying graph for several different setups of the pricing problem.

First, we use the $u(i, j)$ to apply the patterns. Suppose that we are given the pattern $v = (v_0, \dots, v_p)$ and wish to apply this as a work pattern. We assume that the pattern is applied from the first day index of $\mathcal{D}$, i.e. v_0 states whether day 0 should be a work day or not. In general, given a day $d \in \mathcal{D}$ we have that $v_{(d \mod (p+1))}$ states whether or not day d is a day on or a day off. Now, for each day $d \in [d_1, d_2]$ we have the following two cases:

1. if $v_{(d \mod (p+1))} = 1$ then day d is a day on and it should not be allowed to enter the day-off node $m + d$ for day d. Hence, for all arcs $(i, m + d) \in \mathcal{A}(d_1, d_2)$ we set $u(i, m + d) = 0$.

2. if $v_{(d \mod (p+1))} = 0$ then day d is a day off and it should not be allowed to enter any shift nodes $j \in \mathcal{S}_d$. Hence, for all arcs $(i, j) \in \mathcal{A}(d_1, d_2)$ with $j \in \mathcal{S}_d$ we set $u(i, j) = 0$.

Hence, we can modify the underlying graph to satisfy any pattern vector. Given that d is the first day which is not fixed, we put $u(\omega, j) = 0$ for all $j \notin \mathcal{S}'_d$. This ensures that we can only visit nodes in $\mathcal{S}'_d$ as the first node after the origin.

In Figure B.2, we have applied the work pattern $(v_0, v_1, \dots, v_8) = (1, 1, 1, 0, 0, 0, 1, 1, 1)$. The dashed arcs are those which have to be eliminated to accommodate the pattern. According to this pattern, day 20 will be a day on as $v_{(20 \mod 9)} = v_2 = 1$ and day 21 will be the first day off in the off stretch. The frame around the nodes for days 21-23 indicates that these days are days off. Note that we have not eliminated all unnecessary arcs, but only a sufficient subset of arcs to enforce the pattern. A more substantial elimination is possible, but in practice it does not have a significant impact on the running time of the algorithm.

The direct cost of a roster is based on the penalties given for violating the soft constraints. We assume that the soft constraints are constraints on individual transitions between shifts. Let c_{ij} be the penalty of using arc (i,j). Let λ_{ij} be the dual price of using arc (i,j). The reduced cost accumulated along path P_r is

$$\bar{c}(P_r) = \sum_{q=0}^{p-1} \left(c_{(w_q^r, w_{q+1}^r)} - \lambda_{(w_q, w_{q+1})} \right).$$

The objective of the pricing problem is given in problem (B.8), in which we have $\lambda_{ij} = \pi_j$ for all $(i,j) \in \mathcal{A}(d_1,d_2)$ with $j \in \mathcal{S}_d$ and $\lambda_{ij} = 0$ for all $(i,j) \in \mathcal{A}(d_1,d_2)$ with $j \notin \mathcal{S}$. Since all groups $g \in \mathcal{G}$ are independent, for all roster-lines $r \in \mathcal{R}$ we have that $a_{gr} = 1$ for exactly one group $g \in \mathcal{G}$ and $a_{gr} = 0$ for all other groups. Furthermore, as there are only a small number of groups, e.g. nine for the 6&3 work pattern, we may keep the group fixed and solve one problem for each group. For each group $g \in \mathcal{G}$ we have the following decision problem:

$$\min_{r \in \mathcal{R}: a_{gr}=1} \bar{c}(P_r) - \mu_g - \gamma < 0, \tag{B.9}$$

which checks whether or not a negative reduced cost roster-line exists for the group.

Once we have the underlying graph, we can try to identify feasible (ω, δ)-paths. Such a path will correspond to a roster satisfying the work pattern v. The GCRPWP has, however, an additional restriction for a roster-line, which cannot be handled directly by modifying the underlying graph, and for which we use the notion of resources and resource extension functions. We refer the reader to Desaulniers et al. (1998), Irnich and Desaulniers (2005), and Irnich (2008) for an introduction to resource extension functions. The requirement that no more than three consecutive night shifts is modelled by a resource. The intuition is that the resource is initialized at zero and is incremented by one each time a night shift is undertaken. When the accumulated resource is equal to three, then it should not be possible to transition to another night shift. As it is not the total number of night shifts which is bounded but the number of consecutive night shifts, we have to reinitialize the resource at zero whenever a non-night-shift activity is undertaken. The resource extension function has the property that given two paths P and P' both ending in node i with a resource consumption of T_i and T_i', respectively, such that $T_i \leq T_i'$, then for any extension of P', we can identify a similar extension of P with at most the same consumption as the extension of P'. Thus, we will consider P to be a better path than P' with respect to the resource.

To identify a cost-minimizing and resource-feasible path in the underlying graph, we turn to dynamic programming. This type of problem is often referred to as a shortest path problem with resource constraints. The fundamental idea is to construct paths by extending partial paths to all possible successor nodes and do this repeatedly until no path can be extended. If this is done carefully, we will end up with at least one cost-minimizing and resource feasible path. We briefly describe the dynamic programming procedure we use, but refer the reader to Irnich and Desaulniers (2005) for a review of dynamic programming algorithms for resource constrained shortest path problems. The structure of the pricing problem will allow us to solve the problem efficiently.

To each path P we associate a state (or a label) $\mathcal{L}(P) = (\bar{c}(P), T(P))$ which is the vector of accumulated reduced cost and the number of current consecutive night shifts. Given two paths P and P', both ending in node i, we would like to determine which one is the most promising. If the two states $\mathcal{L}(P) \neq \mathcal{L}(P')$ are distinct and, in addition, we have $\bar{c}(P) \leq \bar{c}(P')$ and $T(P) \leq T(P')$, then the path P is the most promising. We say that the path P dominates the path P' and write $\mathcal{L}(P) \prec \mathcal{L}(P')$. In this case, we call P the dominant path and P' the dominated path. When a dominated path is extended, the resulting path will also be dominated by an identical extension of the dominant path. Hence, we can eliminate the dominated path. For any node i, we let F_i be the set of all known states with paths ending in node i. Furthermore, we let $E_i = \{\mathcal{L}(P) \in F_i | \nexists \mathcal{L}(P') : \mathcal{L}(P') \prec \mathcal{L}(P)\}$ be the set of efficient states. It is sufficient to extend the paths having states in E_i as the paths with states in $F_i \setminus E_i$ will be dominated by at least one path in E_i. From the layers of the graph, the nodes have an inherent topological order. Hence, we need only to extend paths from each node once, when using the ordering $(n_1, \ldots, n_{|V|})$. Thus we have a natural iterative approach for the dynamic programming, where we iterate through the nodes given their topological ordering.

The acyclic resource constraint shortest path problem can be solved in pseudo-polynomial time (Desrochers and Soumis, 1988). The algorithm is pseudo-polynomial on the resource width, i.e. on the number of feasible values of the resources. As our pricing problem has only one resource and as that resource is bounded by the pattern length, the pricing problem can actually be solved in polynomial time for a given work pattern.

B.5 Enforcing Integrality of the Solution

As mentioned earlier, column generation is used to solve the relaxed master problem. However, to solve the GCRPWP, we need a solution to the original master problem and hence we reintroduce the integer requirement for variables x_r, where it is violated. The traditional approach to reintroduce integrality is by including the column generation procedure in a branch-and-price framework. In a standard MIP-model, variable branching is usually the branching method of choice. In variable branching the solution space is partitioned into two disjoint subspaces (branches), constructed by splitting the value set for a single variable with a current fractional value $x_b = x_b^*$. In the left branch, the variable is bounded downwards, i.e. $x_b \leq \lfloor x_b^* \rfloor$, and in the right branch it is bounded upwards, i.e. $x_b \geq \lceil x_b^* \rceil$. Unfortunately, it is hard to transfer this approach directly to a column generation context. In column generation, the property that keeps us from regenerating existing columns is the fact that any variable in an optimal basic solution has a reduced cost of zero and all existing non-basic variables have non-negative reduced costs. This is not true for variables with an upper bound. These may have negative reduced cost, even if they are in the basis. The variable bound may instead be enforced by an additional constraint in the master problem. The dual of the new constraint would be reflected in the reduced cost of the variable, and the variable would therefore be non-negative in an optimal solution. However, in the pricing problem, it is not trivial and usually highly inefficient to deal with dual costs for specific variables.

As variable branching is not well suited for column generation, another approach is usually taken, namely the use of constraint branching. For set partitioning problems, the approach introduced by Ryan and Foster (1981) is widely used. Unfortunately, it does not carry over to the generalized set covering problem, which we are considering here. Desaulniers (2009) proposes a four layered branching strategy to the split delivery vehicle routing problem, where the master problem is similar to that of the GCRPWP. An assumption for the completeness of the branching strategy is that there is only one, so called, split customer. In the split delivering vehicle routing problem this assumption always holds, but we do not have the corresponding property in the GCRPWP. The branching strategy may be applied, but as it is not complete, there may be fractional solutions where no branching candidate exists. For a complete constraint branching strategy, we consider another related problem, namely the cutting stock problem. Amor and Carvalho (2005) describe a branching strategy for a model similar to the master problem presented here. Branching is applied to aggregated arc flows. To get a complete branching scheme, nodes of the pricing problem may need to be split into several nodes.

Rostering problems contain a high level of degeneracy and as a consequence, it is often possible to find many different optimal solutions. Indeed, initial tests on the GCRPWP showed that this was the case, and the fractionality of solutions did usually only decrease after introducing a very large number of branching constraints. As described above, it is theoretically possible to find the optimal solution for any instance of the GCRPWP by exploring the full branch-and-bound tree. However, as small run times are desired here, instead we introduce a greedy approach to achieve integrality.

From a fractional solution, the idea is to iteratively apply variable fixing to the variables in the optimal LP-solution. As described above, in column generation, it is computationally hard to bound variables with an upper bound. Therefore, we introduce solely lower bounds on the variables. The bounding will in most cases have an identical effect to that of true variable fixing.

In the following, we describe the variable bounding scheme. Consider an optimal solution to the relaxed master problem, x^*. If $x_r^* \in \mathbb{Z}_+$, $\forall r \in \mathcal{R}$ then x^* is also a solution to the original formulation and the algorithm terminates. Otherwise, we look at the fractional part of the variable values, $f_r^* = x_r^* - \lfloor x_r^* \rfloor$. Given a prespecified threshold, τ (with $0 < \tau < 1$), we impose the following bounds:

$$x_r \geq \lceil x_r^* \rceil, \; \forall r \in \mathcal{R} : f_r^* \geq \tau. \tag{B.10}$$

If $f_r^* < \tau$ for all $r \in \mathcal{R}$, we instead impose the bound on the variable with the largest fractional part:

$$\{r \in \mathcal{R} : f_r^* \geq \tau\} = \emptyset \Rightarrow x_r \geq \lceil x_r^* \rceil, \; r = \operatorname*{argmax}_r f_r^*. \tag{B.11}$$

As we can introduce an additional bound for any fractional solution and as the value of any variable in an optimal solution of the GCRPWP is finite, this approach eventually gives a feasible integer solution, assuming that m_g is integer. The approach may be seen as a special case of variable branching, where the left branch is never explored.

B.6 Rostering Directly on the Forecast Workload

In Section B.3, the mathematical model of the GCRPWP was introduced. The model assumes that the employee demand has been defined for each shift, i.e. q_s is defined for any shift, $s \in \mathcal{S}_d$. Determining the value of q_s is an optimization problem in itself, but we have so far assumed that it is predetermined, usually by an experienced manual planner.

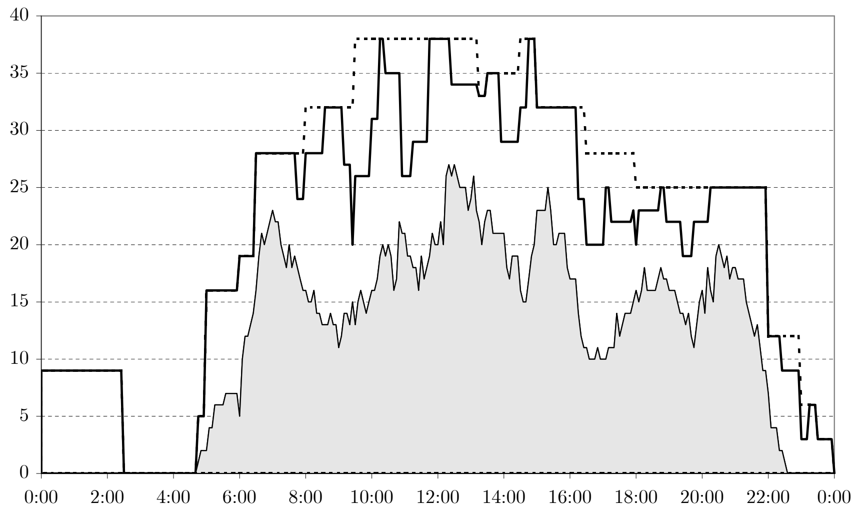


Figure B.3: A workload graph with a suggested shift cover.

Figure B.3 shows a workload graph over one day. The gray area is the forecast workload discretized in 5 minute intervals. The dashed bold line shows the suggested shift cover. As shifts contain breaks, the actual cover is sometimes lower than the shift cover. The actual cover is depicted with the full bold line in the figure. For each time interval, the cover has been found by summing the demands of all shifts that overlap with that time interval.

In the following, we introduce a model, where the roster is constructed to directly cover the forecast workload. As a consequence, we are going to disregard the values of q_s and the process of defining shift demands becomes superfluous. Figure B.4 shows how one step of the usual rostering workflow is circumvented with this approach.

We introduce a discretization of time and refer to an individual time interval in the set of intervals as $t \in \mathcal{T}$. The set $\mathcal{S}_t \subseteq \mathcal{S}$ contains all shifts that overlap with time interval t. The forecast workload of period t is denoted w_t. $\check{c}_t$ refers to the cost of under coverage in interval t. The mathematical model becomes:

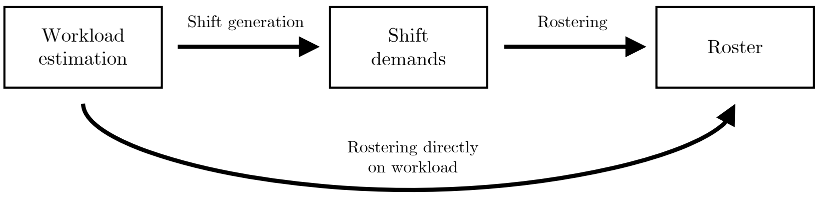


Figure B.4: The workflow in rostering, where the intermediate step of creating shift demands is made superfluous by rostering directly on the workload estimation.

$$\min \sum_{r\in\mathcal{R}} c_r x_r + \sum_{t\in\mathcal{T}} \check{c}_t u_t, \tag{B.12}$$

s.t.

$$\sum_{s\in\mathcal{S}_t}\sum_{r\in\mathcal{R}} a_{sr} x_r + u_t \geq w_t \qquad \forall t \in \mathcal{T}, \tag{B.13}$$

$$\sum_{r\in\mathcal{R}} a_{gr} x_r \leq m_g \qquad \forall g \in \mathcal{G}, \tag{B.14}$$

$$\sum_{r\in\mathcal{R}} x_r \leq n, \tag{B.15}$$

$$u_t \geq 0 \qquad \forall t \in \mathcal{T}, \tag{B.16}$$

$$x_r \in \mathbb{Z}_+ \qquad \forall r \in \mathcal{R}. \tag{B.17}$$

The model is similar to that of Section B.3. The objective (B.12) sums the cost of under coverage for all intervals. For each interval, the workload is either fully covered or the corresponding under coverage is registered in u_t (B.13). Constraints (B.14)-(B.17) correspond directly to Constraints (B.3)-(B.6).

An issue with this model is the number of constraints of the form (B.13), as the set $\mathcal{T}$ may be very large. To alleviate this problem, we aggregate the time intervals. In the following, we describe how to do this without losing any information in the model.

The idea is to aggregate all time intervals for which the cover is always equal. The covers of two separate time intervals are equal if they overlap with an identical set of shifts. Let $\mathcal{T}_1, \ldots, \mathcal{T}_q$ refer to a partitioning of time intervals in q partitions, where all time intervals in a set $\mathcal{T}_i$ have the same shift cover. The partitioning is made so that $\mathcal{T}_1 \cup \ldots \cup \mathcal{T}_q = \mathcal{T}$ and for any two sets $\mathcal{T}_i$ and $\mathcal{T}_j$:

$i \neq j \Rightarrow \mathcal{T}_i \cap \mathcal{T}_j = \emptyset$. Any set can hold only consecutive elements. As explained in Section B.3, each shift contains a break and the cover of a shift does therefore not contain a set of consecutive time intervals. In this work, we assume that each shift contains only one break, starting at time e^b_s ending at l^b_s. The model is easily extended to consider multiple breaks.

The partitioning into these sets is straight forward. Let the set $\mathcal{T}^p = \{t^p_1, t^p_2, \dots, t^p_q, t^p_{q+1}\} = \bigcup_{s \in \mathcal{S}} \{e_s, l_s, e^b_s, l^b_s\}$ contain all split times in sorted order. We define $\mathcal{T}_i = t^p_i, \dots, t^p_{i+1} - 1, i = 1, \dots, q$. From the partitioning of $\mathcal{T}$ define a new set of time intervals $\Theta = \theta_1, \dots, \theta_q$, where the start times and the end times of the new time intervals are the split times of $\mathcal{T}^p$. The definition of $\mathcal{S}_t$ easily transfers to $\mathcal{S}_\theta$ for $\theta \in \Theta$. However, the workload in a time intervals θ_i is not necessarily constant and hence the under cover is not as easily defined as in the previous model.

Let $w_{\theta_i} = \max_{t \in \mathcal{T}_i} w_t$ and introduce the decision variables $u_{\theta_i j}, i = 1, \dots, q, j = 1, \dots, w_{\theta_i}$ with $0 \leq u_{\theta_i j} \leq 1$. $\sum_{j=1}^{w_{\theta_i}} u_{\theta_i j}$ denotes the under coverage for time interval θ_i. By defining the under coverage as a sum of variables, we are able to introduce a piecewise linear cost function for under coverage. The cost of each piece of the function is defined by the number of original time intervals, for which the cover is insufficient. The cost of $u_{\theta_i j}$ is $\check{c}_{\theta_i j}$ and is calculated as:

$$\check{c}_{\theta_i j} = \sum_{t \in \mathcal{T}_i : w_t > w_{\theta_i} - j} \check{c}_t. \tag{B.18}$$

$\check{c}_{\theta_i j}$ sums the cost of all original time periods, which will not be fully covered if the interval θ_i has under coverage of j or more.

Figure B.5 shows a workload graph zoomed in on one interval, θ', covering the time from 15:00 to 16:00. The gray area is the workload. The curve above the workload area with a shape similar to the workload represents robustness, which is described in Section B.7. We disregard this right now. In the example, $w_{\theta'} = 25$. As an under coverage of 1 (relative to $w_{\theta'}$) only introduces under coverage in interval 5: $\check{c}_{\theta' 1} = \check{c}_5$. Following the same argument, $\check{c}_{\theta' 2} = \check{c}_5$. Under coverage of 3 or 4 introduces under coverage in more intervals and therefore: $\check{c}_{\theta' 3} = \check{c}_{\theta' 4} = \check{c}_1 + \check{c}_2 + \check{c}_3 + \check{c}_4 + \check{c}_5 + \check{c}_6$. The cost coefficients for further under coverage are calculated similarly. Hence, the aggregated model becomes:

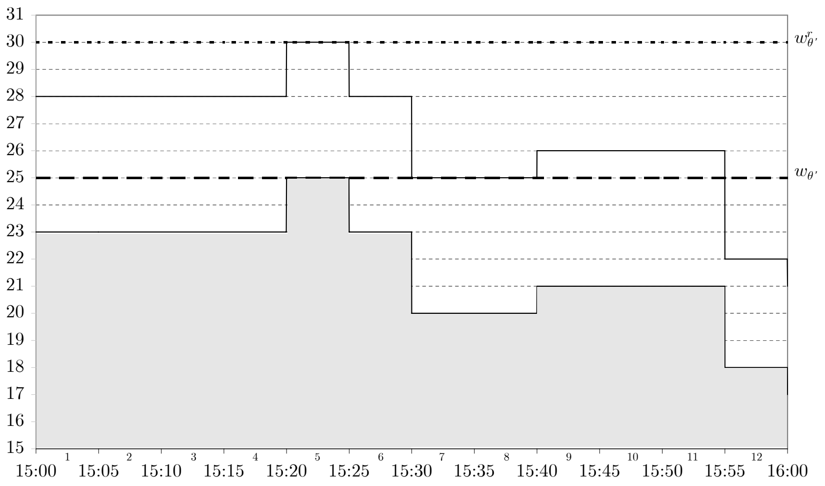


Figure B.5: Example of cost calculation for aggregated time intervals.

$$\min \sum_{r\in\mathcal{R}} c_r x_r + \sum_{\theta\in\Theta}\sum_{j=1}^{w_\theta} \check{c}_{\theta j} u_{\theta j}, \tag{B.19}$$

s.t.

$$\sum_{s\in\mathcal{S}_\theta}\sum_{r\in\mathcal{R}} a_{sr} x_r + \sum_{j=1}^{w_\theta} u_{\theta j} \geq w_\theta \qquad \forall \theta\in\Theta, \tag{B.20}$$

$$\sum_{r\in\mathcal{R}} a_{gr} x_r \leq m_g \qquad \forall g\in\mathcal{G}, \tag{B.21}$$

$$\sum_{r\in\mathcal{R}} x_r \leq n, \tag{B.22}$$

$$0 \leq u_{\theta j} \leq 1 \qquad \forall \theta\in\Theta, \forall j = 1,\dots,w_\theta, \tag{B.23}$$

$$x_r \in \mathbb{Z}_+ \qquad \forall r\in\mathcal{R}. \tag{B.24}$$

The constraints of the model correspond directly to Constraints (B.12)-(B.17) with the described aggregation. $\check{c}_{\theta j}$ is increasing over j and therefore, in an optimal solution, we have that $u_{\theta 1} \geq u_{\theta 2} \geq \ldots \geq u_{\theta w_\theta}$, as intended. At most one $u_{\theta j}$ is fractional. If w_θ is integer, all $u_{\theta j}$ are binary.

In practice, we want to limit the number of variables as much as possible. $\check{c}_{\theta j}$ is often unchanged for a sequence of values of j. In this case, we may remove all but one of the associated variables by increasing the upper bound of the remaining variable, accordingly.

B.7 Robustness

In the previous section, we assumed that there was no preference between rosters that cover the forecast workload equally well. In practice, the actual workload on a given day is not going to match the forecast workload exactly, and we therefore introduce robustness measures to create a roster, which deals well with small changes in workload.

To be able to handle a larger workload than anticipated, we add a certain percentage to the original forecast workload. We refer to this contingency as r^c, e.g. $r^c = 0.2$ for a 20% contingency. Furthermore, we expect some tasks to be delayed. This will result in a delayed workload. We define r^d as the number of minutes of slippage that need to be accounted for, e.g. $r^d = 15$. We want to cover both cases as well as possible.

Let $\check{c}_t^r$ be the cost per unit of not covering the workload with the added contingency or of not covering the slipped workload, whatever is more demanding. The objective of the disaggregated model (B.12) is changed and two additional constraints are added:

$$\min \sum_{r \in \mathcal{R}} c_r x_r + \sum_{t \in \mathcal{T}} (\check{c}_t u_t + \check{c}_t^r u_t^r), \tag{B.25}$$

$$\sum_{s \in \mathcal{S}_t} \sum_{r \in \mathcal{R}} a_{sr} x_r + u_t + u_t^r \geq \lceil (1 + r^c) w_t \rceil \qquad \forall t \in \mathcal{T}, \tag{B.26}$$

$$\sum_{s \in \mathcal{S}_t} \sum_{r \in \mathcal{R}} a_{sr} x_r + u_t + u_t^r \geq w_{t-r^d} \qquad \forall t \in \mathcal{T}. \tag{B.27}$$

A new set of variables, u_t^r, has been introduced to correctly penalize inadequate robustness. The right hand side of Constraints (B.26) is converted to an integer for simplicity. This means that under coverage is always integer if the workload estimations are.

The changes to the model carry over to the aggregated model easily. Each time interval may now contribute to the cost of the aggregation with either $\check{c}_t$ or $\check{c}_t^r$. The right hand sides of Constraints (B.20) hold the necessary coverage for no penalty to apply, including penalties from robustness. Hence, we define: $w^r_{\theta_i} = \max_{t\in\mathcal{T}_i}(\max\{(1+r^c)w_t, w_{t-r^d}\})$. We also introduce the subset $\mathcal{T}^r_{ij} = \{t \in \mathcal{T}_i : \lceil(1+r^c)w_t\rceil > w^r_{\theta_i} - j \vee w_{t-r^d} > w^r_{\theta_i} - j\}$. The right hand side of (B.20) is replaced by $w^r_{\theta_i}$ and the coefficients of the objective function become $(i = 1,\ldots,q, j = 1,\ldots,w^r_{\theta_i})$:

$$\check{c}^r_{\theta_i j} = \sum_{t\in\mathcal{T}_i : w_t > w^r_{\theta_i} - j} \check{c}_t + \sum_{t\in\mathcal{T}^r_{ij}} \check{c}^r_t. \tag{B.28}$$

As a consequence, the right hand sides of (B.20) are increased, but the costs are decreased correspondingly. For $\check{c}^r_t = 0$ the total cost is equal to the cost of the model without robustness. Assuming that $\check{c}^r_t \le \check{c}_t$, in an optimal solution we still have $u_{\theta 1} \ge u_{\theta 2} \ge \ldots \ge u_{\theta w^r_\theta}$ with no fractional $u_{\theta j}$ for integer w^r_θ.

As an example, refer again to Figure B.5. Under coverage is now relative to $w^r_{\theta'}$. Again, an under coverage of 1 or 2 (relative to $w^r_{\theta'}$) only introduces under coverage in interval 5 and therefore: $\check{c}_{\theta'1} = \check{c}_{\theta'2} = \check{c}^r_5$. The subsequent coefficients are calculated as: $\check{c}_{\theta'3} = \check{c}_{\theta'4} = \check{c}^r_1 + \check{c}^r_2 + \check{c}^r_3 + \check{c}^r_4 + \check{c}^r_5 + \check{c}^r_6$, $\check{c}_{\theta'5} = \check{c}^r_1 + \check{c}^r_2 + \check{c}^r_3 + \check{c}^r_4 + \check{c}^r_5 + \check{c}^r_6 + \check{c}^r_9 + \check{c}^r_{10} + \check{c}^r_{11}$. The coefficient of an under coverage of six includes the original costs as well: $\check{c}_{\theta'6} = \check{c}^r_1 + \ldots + \check{c}^r_{11} + \check{c}_5$. The calculations for the remaining coefficients are similar. The final model with time interval aggregation and robustness becomes:

$$\min \sum_{r\in\mathcal{R}} c_r x_r + \sum_{\theta\in\Theta}\sum_{j=1}^{w^r_\theta} \check{c}^r_{\theta j} u_{\theta j}, \tag{B.29}$$

s.t.

$$\sum_{s\in\mathcal{S}_\theta}\sum_{r\in\mathcal{R}} a_{sr}x_r + \sum_{j=1}^{w^r_\theta} u_{\theta j} \ge w^r_\theta \qquad \forall\theta\in\Theta, \tag{B.30}$$

$$\sum_{r\in\mathcal{R}} a_{gr}x_r \le m_g \qquad \forall g\in\mathcal{G}, \tag{B.31}$$

$$\sum_{r\in\mathcal{R}} x_r \le n, \tag{B.32}$$

$$0 \le u_{\theta j} \le 1 \qquad \forall\theta\in\Theta, \forall j = 1,\ldots,w^r_\theta, \tag{B.33}$$

$$x_r \in \mathbb{Z}_+ \qquad \forall r\in\mathcal{R}. \tag{B.34}$$

B.8 Experimental results

In this section, we present the results obtained for the proposed methodology on three real-life instances supplied by the airline. Due to its superiority from a robustness modelling perspective, we only test Model (B.29)-(B.34). The instances, denoted W07, S08, and S10 below, each have a rostering horizon of 189 days and contain 65, 95, and 139 staff members, respectively. All instances have 11 different shifts, three of which are night shifts. To incorporate flexibility with respect to breaks, three copies of each shift are created and differ only in the break time of the shift. The workload demand is cyclic with a period of one week and all staff are assumed to be working a 6&3 pattern.

Since this is a somewhat small test set, an additional ten artificial instances have been constructed and used in the analysis of the algorithm's performance. All artificial instances are based on the real-life instances and are, in particular, an attempt to stress test the approach given more dramatic workload demands. That is, the artificial instances are identical in structure to the real-life instances in terms of the number of shifts and rostering horizon; however, each has a different workload demand and number of staff. These instances are referred to as A01-A10 below. The entire algorithm has been written in the C++ programming language and utilizes the commercial solver Cplex 10.0 with default parameters to solve the master problem. All computational experiments have been performed on a 64 bit Linux operating system equipped with a dual core AMD 2.2 GHz processor and 2GB of RAM.

We begin with an analysis of how the model and solution approach perform over a 63 day time horizon. Before considering a longer time horizon, we want to ascertain the effect on solution quality of decomposing the rostering horizon into shorter, more tractable, overlapping blocks. Also the heuristic shift fixing and branching routines described in Sections B.4 and B.5 will compromise solution quality. The question is, to what degree. In the 63 day time horizon, we are able to calculate the optimal solution to the LP relaxation of the non-decomposed problem and thus provide a lower bound on the value of the decomposed integer solution obtained. Furthermore, as 63 is the lowest common multiple of the pattern length and the period of the workload demand, if the obtained rosters are *wrappable*, then one can simply copy the rosters to any rostering horizon that is a multiple of 63 days. A wrappable roster is one in which it is possible for all staff on a particular pattern group to transition from their last shift (on day 63) to one of the shifts worked by the pattern group on the first day. One cannot guarantee this to always be the case, since such shift transitions are not taken into consideration in our approach.

To test our decomposition approach the 63 day horizon is divided into an initial

Instance	LP^I	IP^I	LP^D	IP^D	LP^N	G^D	G^N	t^D_{IP}(s)	t^N_{LP}(s)
A01	59.18	59.18	39.90	39.90	39.90	0.00%	0.00%	110.67	614.94
A02	375.18	377.89	251.43	254.14	250.36	1.08%	1.51%	805.84	2516.2
A03	365.20	366.96	241.55	243.31	242.17	0.73%	0.47%	692.48	4783.25
A04	740.31	740.51	495.93	496.12	495.97	0.04%	0.03%	236.87	1316.6
A05	327.78	328.17	218.43	218.82	218.57	0.18%	0.11%	1122.08	3041.76
A06	7182.80	7183.84	4799.55	4800.59	4799.94	0.02%	0.01%	711.48	2933.6
A07	0.55	0.55	0.35	0.35	0.35	0.00%	0.00%	27.29	204.02
A08	1630.88	1631.00	1099.57	1099.69	1099.60	0.01%	0.01%	130.69	546.72
A09	232.15	237.60	150.41	155.86	152.07	3.62%	2.49%	2079.08	6378.47
A10	4595.07	4597.57	3171.40	3173.90	3172.17	0.08%	0.05%	713.56	2211.54
S08	0.00	0.00	0.00	0.00	0.00	0.00%	0.00%	19.05	45.48
S10	306.68	306.83	217.46	217.61	217.48	0.07%	0.06%	147.92	965.84
W07	0.85	0.85	0.57	0.57	0.57	0.00%	0.00%	49.24	305.71

Table B.1: Results for a 63 Day Rostering Horizon

block of 23 days and 5 additional blocks with a length of 14 days. Each additional block contains 8 new days and 6 days of the preceding block (that will be resolved). For this discretization one has many possibilities; we tried various values, and the above parameters appeared to work well. In all experiments the cost of one hour of uncovered work is assumed to be 10 units, while an hour of uncovered robustness costs 1 unit. Given that we are dealing with a cyclic, weekly workload demand and that staff are working a fixed 6&3 work pattern, a preallocation step that equalizes the number of staff working each pattern group is performed. That is, the algorithm does not determine how many staff members will work each pattern group.

Table B.1 provides the results of the initial experiments. For each instance, the table gives the *inflated* LP and IP objective values (LP^I and IP^I) as well as the *deflated* LP and IP objective values (LP^D and IP^D). The inflated LP and IP values are simply the sum of the respective LP and IP objective values found in each of the blocks. This is an inaccurate indication of the total cost since the cost contribution from each block overlap is counted twice. IP^D is the true cost of the 63 day roster. This value is obtained by correctly adjusting the costs incurred in the overlaps. LP^D, on the other hand, is obtained by reducing the LP^I by the difference between the IP^I and IP^D. Due to the heuristic shift fixing in the overlaps one cannot obtain an optimal decomposed LP solution. The purpose of the LP^D is to provide a lower bound when the non-decomposed bound (LP^N) cannot be obtained (i.e. for longer time horizons). Table B.1 also reports the time taken to solve the decomposed model (t^D_{IP}) as well as the time taken to solve the LP for the non-decomposed model (t^N_{LP}), both of which are in seconds. The percentage gap between the deflated LP and IP objective values (G^D) and the optimality gap between the decomposed IP objective value and the non-decomposed LP objective values (G^N) are also given.

One can observe that the algorithm performs very efficiently with small opti-

Instance	LP^I	IP^I	LP^D	IP^D	$3IP^D_{63}$	G^D	$t^D_{IP}(s)$
A01	198.76	198.76	120.19	120.19	119.7	0.00%	249.19
A02	1265.19	1274.04	751.73	760.58	762.42	1.18%	1581.17
A03	1213.80	1220.55	724.89	731.64	729.93	0.93%	1040.34
A04	2485.13	2485.69	1488.10	1488.66	1488.36	0.04%	555.58
A05	1096.74	1098.43	656.11	657.80	656.46	0.26%	1920.91
A06	23936.35	23941.99	14405.78	14411.42	14401.77	0.04%	798.84
A07	1.78	1.78	1.05	1.05	1.05	0.00%	65.30
A08	5470.04	5470.24	3310.39	3310.59	3299.07	0.01%	251.30
A09	768.96	782.49	453.08	466.60	467.58	2.98%	2753.54
A10	15686.69	15695.55	9520.05	9528.92	9521.70	0.09%	945.74
S08	3.30	3.30	3.30	3.30	0.00	0.00%	54.32
S10	1047.70	1048.26	648.05	648.61	652.83	0.09%	322.95
W07	2.87	2.87	1.70	1.70	1.71	0.00%	121.58

Table B.2: Results for a 189 Day Rostering Horizon

mality gaps between the best found integer solution using the decomposition approach and the optimal LP objective value for the non-decomposed model. Excluding instances A02, A03, and A09, all optimality gaps are less than 0.20%. The fact that $LP^D \approx LP^N$ indicates that the larger part of the gap is an integrality gap, caused by the relaxation of the integrality property for x_r. Hence, we conclude that the block structure does not reduce solution quality significantly. It is very encouraging to see that the block decomposition produces close to optimal integer solutions much faster than it can find the optimal LP objective value of the non-decomposed model. Furthermore, LP^D appears to be a slightly pessimistic approximation of LP^N as it is smaller in all but one instance (A02). However, both gaps, in general, are small enough that one can be confident of the superior performance of the algorithm. Finally, the results suggest that the real-life instances can be solved extremely efficiently, while it is just the artificial instances that create some difficulties.

Table B.2 gives the results of similar experiments in which the rostering horizon is increased from 63 days to 189 days. The results are very similar to those of Table B.1, with a small percentage gap between the IP^D and the LP^D as well as acceptable running times, particularly for the real-life instances. One can also observe that the best found solutions are approximately a factor three more than those found for the 63 day experiments ($3IP^D_{63}$). As was mentioned earlier, there is no guarantee that a solution with the latter objective value is even feasible. In some cases, i.e. A02, A05, A09, S10, and W07, the value IP^D is at least as good as $3IP^D_{63}$. This is due to the unpredictable behaviour of the heuristic shift fixing and the branching routines in the block decomposition.

A particularly interesting graph is given in Figure B.6 which illustrates the cost incurred on each day of the rostering horizon for S10. It is clear that the cost has a cyclic behaviour. A closer inspection shows that the costs are highly dependent on the weekday, which is not surprising, as the workload estimation

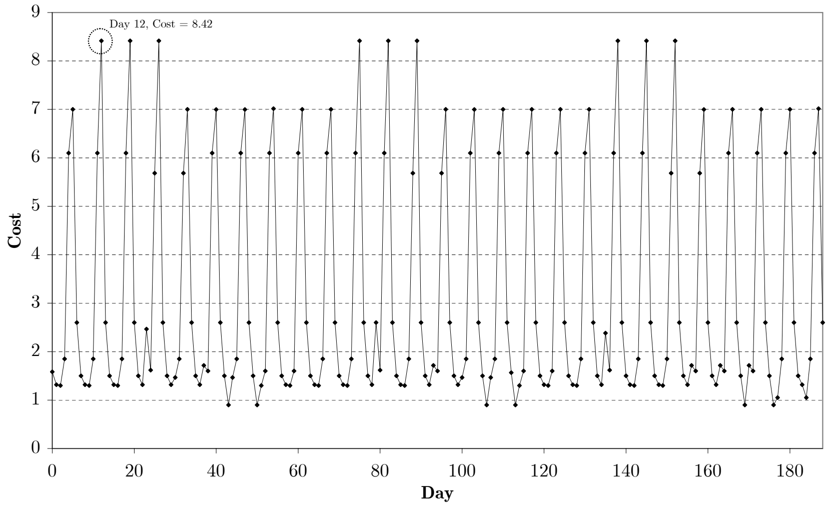


Figure B.6: The cost incurred on each day of the rostering horizon for S10.

is by weekday, but not over weeks. The workload graph for a day of the horizon with highest cost, Day 12, is shown in Figure B.7. Day 12 is a Friday and all the larger costs of the horizon are observed on Fridays. It is clear that even on the worst days, the cover is fairly good and the major contribution to the cost is from the robustness measure. S10 is the realistic instance, where it is, by far, the most difficult to find a satisfactory cover. As the figure illustrates, even the worst day here has an acceptable cover. We conclude that the proposed method is very well suited for solving the problems at hand.

As can be seen from Table B.2, the worst cover is obtained in instance A06. To sketch the worst case scenario, Figure B.8 shows the workload graph of the most costly day of A06. Indeed, the amount of uncovered work is severe, but it is observed that this not due to a bad distribution of available manpower. The assigned manpower covers the estimated workload tightly, but there are simply too few employees to cover all the work. Also, this particular instance has a very jagged workload estimation, which makes it difficult to cover the highest peeks. The artificial instances were introduced to stress test the algorithm and to show what happens, when very hard instances are encountered. The figure illustrates that A06 is well suited for this purpose.

Table B.3 provides a breakdown of the total cost (IP) into total roster-line costs (RC) and total coverage costs (CC). The RC component indicates how many late to early sequences are not satisfied in the chosen roster-lines, while CC

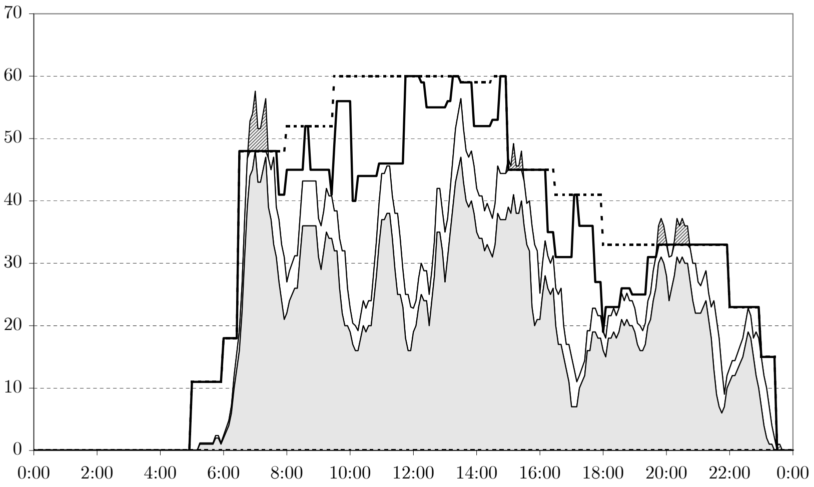


Figure B.7: Workload graph for the most costly day of S10 with the cover of the found solution.

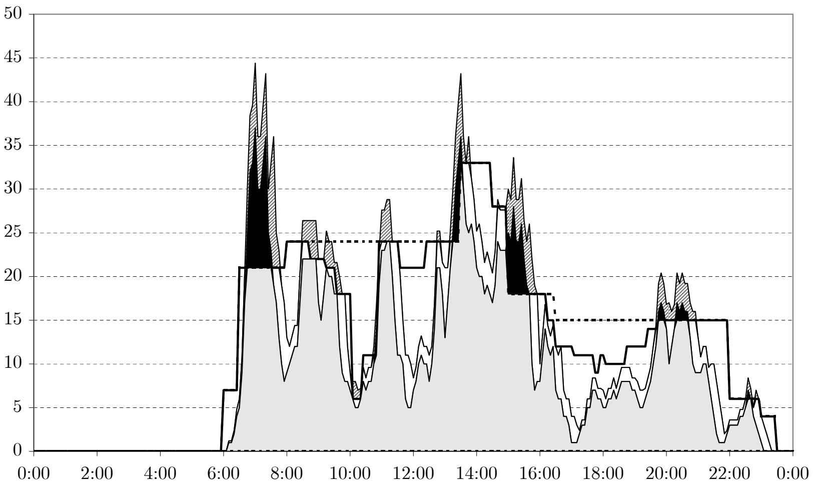


Figure B.8: Workload graph for the most costly day of A06 with the cover of the found solution.

Instance	IP	RC	CC	UWL	UR	IE
A01	120.19	0.00	120.19	0.00	4.45	59.28%
A02	760.58	17.00	743.58	0.53	22.24	65.12%
A03	731.64	0.00	731.64	0.10	26.10	61.52%
A04	1488.66	6.00	1482.66	1.23	42.61	62.11%
A05	657.80	0.00	657.80	0.02	24.16	65.18%
A06	14411.42	593.00	13818.43	37.92	132.59	59.08%
A07	1.05	0.00	1.05	0.00	0.04	50.53%
A08	3310.59	373.00	2937.59	6.60	42.80	56.90%
A09	466.60	0.00	466.60	0.00	17.28	67.77%
A10	9528.92	916.00	8612.92	18.73	131.70	62.12%
S08	3.30	2.00	1.30	0.00	0.05	45.98%
S10	648.61	60.00	588.61	0.03	21.50	55.30%
W07	1.70	0.00	1.70	0.00	0.06	57.07%

Table B.3: Solution Statistics for 189 Day Rosters

states the cost incurred from uncovered workload and uncovered robustness. In Table B.3, we also give both the number of uncovered workload hours on average per week (UWL) and the number of uncovered hours of robustness on average per week (UR). Here one can see that uncovered hours of robustness is the main component of the coverage cost in most cases. It is not surprising to see that the instances with the largest number of uncovered workload hours (A06, A08, and A10) also have the largest RC cost contribution. Here the model is simply attempting to cover the workload as well as possible, often disregarding the late to early sequence preference for the roster-line. Finally, IE gives the *Implied Efficiency* of the obtained roster. Implied efficiency is the percentage of time that people at work are actually working.

To provide an indication of how the solutions to S08, S10, and W07 in Table B.2 compare to the airline's solutions, Table B.4 makes a comparison of the average number of uncovered workload hours per week, the average number of uncovered hours of robustness, and the implied efficiency of each of the rosters. In Table B.4, column headings with an A superscript denote the airline's value. For all instances, we perform much better, significantly improving the robustness of the roster. It is not possible to improve implied efficiency without reducing the staffing level. Instances S08$'$, S10$'$, and W07$'$ are identical to S08, S10, and W07 in which the staffing level has been reduced by 10-12%. Here we see that we can provide a similar coverage and more robustness than the airline, while at the same time improving efficiency by around 7-11%. Although instance S10 has 0.33 uncovered workload hours per week on average, this equates to around 20 minutes per week and can be considered negligible. It should also be mentioned that in increasing the efficiency there is only a very slight increase in the RC for instance S10.

Instance	UWL^A	UWL	UR^A	UR	RC	IE^A	IE
S08	4.42	0.00	12.52	0.05	0.00	46.10	45.98
S10	0.00	0.03	147.69	21.50	60.00	55.29	55.30
W07	0.00	0.00	41.28	0.06	0.00	57.14	57.07
S08′	4.42	0.00	12.52	0.01	0.00	46.10	51.52
S10′	0.00	0.33	147.69	36.40	63.00	55.29	59.57
W07′	0.00	0.00	41.28	2.56	0.00	57.14	61.60

Table B.4: Comparison with Airline's Solutions

B.9 Conclusion

In this paper, we have considered the GCRPWP arising at a major European airline. We have proposed a cutting stock based integer programming formulation of the problem, which is not only able to circumvent one step of the roster construction process but which can also accurately incorporate the necessary robustness measures. A powerful decomposition approach utilizing column generation and variable fixing is developed to solve a sequence of integrated optimization problems. This is combined with a shift fixing routine to ensure the roster-lines obtained for each of the smaller problems can be pieced together to construct a roster for the entire six month planning horizon. Computational results on three real-life instances and 10 artificial instances confirm the efficiency of the proposed methodology. Not only do we find better solutions than those implemented by the airline, particularly from a robustness perspective, but we have also shown that more robust solutions can be obtained even if staffing levels are reduced by 10-12%.

References

Abbink, E., M. Fischetti, L. Kroon, G. Timmer, and M. Vromans (2005). "Reinventing crew scheduling at Netherlands railways". In: *Interfaces* 35.5, pp. 393–401.

Alfares, H. K. (1997). "An efficient two-phase algorithm for cyclic days-off scheduling". In: *Computers and Operations Research* 25.11, pp. 913–923.

Alfares, H. K. (2002). "Optimum Workforce Scheduling Under the (14,21) Days-Off Timetable". In: *Journal of Applied Mathematics and Decision Sciences* 63.3, pp. 191 –199.

Amor, H. B. and J. V. de Carvalho (2005). "Cutting Stock Problems". In: *Column Generation*. Ed. by G. Desaulniers, J. Desrosiers, and M. Solomon. Springer. Chap. 5, pp. 131 –161.

Anbil, R., E. Gelman, B. Patty, and R. Tanga (1991). "Recent advances in crew-pairing optimization at American Airlines". In: *Interfaces* 21.1, pp. 62 –74.

Barnhart, C., A. M. Cohn, E. L. Johnson, D. Klabjan, G. L. Nemhauser, and P. H. Vance (2003). "Airline Crew Scheduling". In: *Handbook of Transportation Science.* Ed. by Randolph W. Hall. Kluwer Academic Publishers.

Brusco, M. J., L. W. Jacobs, R. J. Bongiorno, D. V. Lyons, and B. Tang (1995). "Improving Personnel Scheduling at Airline Stations". In: *Operations Research* 43.5, 741–751 and 172029.

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

Burke, E. K., P. De Causmaecker, G. De Maere, J. Mulder, M. Paelinck, and G. Vanden Berghe (2010). "A multi-objective approach for robust airline scheduling". Ed. by Jesper Larsen Jens Clausen Allan Larsen. In: *Computers and Operations Research* 37.5, pp. 822–832.

Cheang, B., H. Li, A. Lim, and B. Rodrigues (2003). "Nurse rostering problems–a bibliographic survey". In: *European Journal of Operational Research* 151.3, pp. 447–460.

Chu, S. C. K. (2007). "Generating, scheduling and rostering of shift crew-duties: Applications at the Hong Kong International Airport". In: *European Journal of Operational Research* 177, pp. 1764 –1778.

Clausen, J., A. Larsen, J. Larsen, and N. J. Rezanova (2010). "Disruption management in the airline industry-Concepts, models and methods". Ed. by Jesper Larsen Jens Clausen Allan Larsen. In: *Computers and Operations Research* 37.5, pp. 809–821.

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 Jacques Desrosiers Guy Desaulniers and Marius M. Solomon. Springer. Chap. 4, pp. 99–129.

Day, P. R. and D. M. Ryan (1997). "Flight attendant rostering for short-haul airline operations." In: *Operations Research* 45.5, pp. 649 –661.

Desaulniers, G., J. Desrosiers, I. Ioachim, M. Solomon, F. Soumis, and D. Villeneuve (1998). "A Unified Framework for Deterministic Time Constrained Vehicle Routing and Crew Scheduling Problems". In: *Fleet Management and Logistics.* Ed. by T. Crainic and G. Laporte. Kluwer Academic Publishers. Chap. 3, pp. 57 –94.

Desaulniers, G., J. Desrosiers, and M. M. Solomon (2002). "Accelerating Strategies in Column Generation Methods for Vehicle Routing and Crew Scheduling Problems". In: *Essays and Surveys in Metaheuristics.* Ed. by C.C. Riveiro and P. Hansen. Kluwer, Norwell, pp. 309–324.

Desaulniers, G. (2009). "Branch-and-Price-and-Cut for the Split-Delivery Vehicle Routing Problem with Time Windows". In: *Operations Research* To appear.

Desrochers, M. and F. Soumis (1988). “A Reoptimization Algorithm for the Shortest Path Problem with Time Windows”. In: *European Journal of Operational Research* 35.2, pp. 242–254.

Desrosiers, J. and M. E. Lübbecke (2005). “A Primer in Column Generation”. In: *Column Generation.* Ed. by G. Desaulniers, J. Desrosiers, and M.M. Solomon. Springer, New York. Chap. 1, pp. 1–32.

Dowling, D., M. Krishnamoorthy, H. Mackenzie, and D. Sier (1997). “Staff rostering at a large international airport”. In: *Annals of Operations Research* 72, pp. 125 –147.

Ernst, A., H. Jiang, M. Krishnamoorthy, and D. Sier (2004*c*). “Staff Scheduling and rostering: A review of applicaitons, methods, and models”. In: *European Journal of Operations Research* 153, pp. 3 –27.

Eveborn, P. and M. Rönnqvist (2004). “Scheduler - A System for Staff Planning”. Ed. by Houyuan Jiang, Mohan Krishnamoorthy, and David Sier. In: *Annals of Operations Research* 128.1-4, pp. 21–45.

Gendreau, M., J. Ferland, B. Gendron, N. Hail, B. Jaumard, S. Lapierre, G. Pesant, and P. Soriano (2006*a*). “Physician Scheduling in Emergency Rooms”. In: *Practice and Theory of Automated Timetabling VI. 6th International Conference, PATAT.* Springer-Verlag.

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

Irnich, S. (2008). “Resource extension functions: properties, inversion, and generalization to segments”. In: *OR Spectrum* 30.1, pp. 113–148.

Kellogg, D. L. and S. Walczak (2007). “Nurse Scheduling: From Academia to Implementation or Not?” In: *Interfaces* 37.4, pp. 355 –369.

Mehrotra, A., K. Murphy, and M. Trick (2000). “Optimal shift scheduling: a branch-and-price approach”. In: *Naval Research Logistics* 47.3, pp. 185–200.

Ryan, D. M. and B. Foster (1981). “An integer programming approach to scheduling”. Ed. by A. Wren. In: *Computer Scheduling of Public Transport. Urban Passenger Vehicle and Crew Scheduling. Proceedings of an International Workshop*, pp. 269–280.

Vohra, R. V. (1987). “The cost of consecutivity in the (5, 7) cyclic staffng problem”. In: *IIE Transactions* 29, pp. 942 –950.

Wäscher, G. and T. Gau (1996). “Heuristics for the integer one-dimensional cutting stock problem: a computational study”. In: *OR Spektrum* 18.3, pp. 131–144.

Weide, O., D. Ryan, and M. Ehrgott (2010). “An iterative approach to robust and integrated aircraft routing and crew scheduling”. Ed. by Jesper Larsen Jens Clausen Allan Larsen. In: *Computers and Operations Research* 37.5, pp. 833–844.

Appendix C

The Manpower Allocation Problem with Time Windows and Job-Teaming Constraints: A Branch-and-Price Approach

Anders Dohn, Esben Kolind, and Jens Clausen

Published in: *Computers and Operations Research* (2009) 36.4, pp. 1145-1157.

The Manpower Allocation Problem with Time Windows and Job-Teaming Constraints: A Branch-and-Price Approach[*]

Anders Dohn[1], Esben Kolind[1], and Jens Clausen[1]

In this paper, we consider the Manpower Allocation Problem with Time Windows, Job-Teaming Constraints and a limited number of teams (m-MAPTWTC). Given a set of teams and a set of tasks, the problem is to assign to each team a sequential order of tasks to maximize the total number of assigned tasks. Both teams and tasks may be restricted by time windows outside which operation is not possible. Some tasks require cooperation between teams, and all teams cooperating must initiate execution simultaneously. We present an Integer Programming model for the problem, which is decomposed using Dantzig-Wolfe decomposition. The problem is solved by column generation in a Branch-and-Price framework. Simultaneous execution of tasks is enforced by the branching scheme. To test the efficiency of the proposed algorithm, 12 realistic test instances are introduced. The algorithm is able to find the optimal solution in 11 of the test instances. The main contribution of this article is the addition of synchronization between teams in an exact optimization context.

Keywords: Manpower allocation; Crew scheduling; Vehicle routing with time windows; Synchronization; Simultaneous execution; Branch-and-Price; Branching rules; Column generation; Decomposition; Set covering; Integer programming.

C.1 Introduction and Problem Description

The Manpower Allocation Problem with Time Windows, Job-Teaming Constraints and a limited number of teams (m-MAPTWTC) is the problem of assigning m teams to a number of tasks, where both teams and tasks may be restricted by time windows outside which operation is not possible. Tasks may

[*]Published in: *Computers and Operations Research* (2009) 36.4, pp. 1145-1157.

[1]Informatics and Mathematical Modelling, Technical University of Denmark, Building 321, Richard Petersens Plads, 2800 Lyngby, Denmark.

require several individual teams to cooperate. Due to the limited number of teams, some tasks may have to be left unassigned. The objective is to maximize the number of assigned tasks.

The problem arises in various contexts where cooperation between teams / workers, possibly with different skills, is required to solve tasks. An example is the home care sector, where the personnel travel between the homes of the patients who may demand collaborative work (e.g. for lifting). The problem also occurs in hospitals where a number of doctors and nurses are needed for surgery and the composition of staff may vary for different tasks. Another example is in the allocation of technicians to service jobs, where a combination of technicians with individual skills is needed to solve each task.

This study focuses on the scheduling of ground handling tasks in some of Europe's major airports. Between arrival and the subsequent departure of an aircraft, numerous jobs including baggage handling and cleaning must be performed. Typically, specialized handling companies take on the jobs and assign crews of workers with different skills. Any daily work plan must comply with the time windows of tasks, the working hours of the staff, the skill requirements of tasks, and union regulations. It may be necessary to have several teams cooperating on one task in order to complete it within the time window. The workload has to be divided equally among the cooperating teams. Furthermore, all teams involved must initiate work on the task simultaneously (synchronized cooperation), as only one of the team leaders is appointed as responsible supervisor. In the remainder of this paper, a team is a fixed group of workers, whereas when referring to job-teaming or cooperation, we refer to a temporary constellation of teams joined together for a specific task. In the airport setting, all tasks require exactly one skill each.

MAPTWTC has previously been treated by Lim et al. (2004) and Li et al. (2005) in a metaheuristic approach. They study an example originating from the Port of Singapore, where the main objective is to minimize the number of workers required to carry out all tasks, rather than carrying out the maximum number of tasks with a given workforce. Both papers describe secondary objectives as well.

Our problem is closely related to the Vehicle Routing Problem with Time Windows (VRPTW) which has been studied extensively in the literature.

The Synchronized Vehicle Dispatching Problem (SVDP) as presented by Rousseau et al. (2003) is a dynamic vehicle routing problem similar to MAPTWTC. In SVDP, the visits of the vehicles may require additional assistance from other vehicles or special teams, and hence the vehicles and the special teams have to be synchronized. A number of benchmark problems are solved by a constraint programming based greedy procedure with post-optimization using local search.

The Vehicle Routing Problem with Split Deliveries (VRPSD) allows a customer to be visited by several vehicles, each fulfilling some of the demand. The problem was introduced by Dror and Trudeau (1989). See Lee et al. (2006) for an overview of the literature. Frizzell and Giffin (1995) were the first to include the time window extension in the split delivery problem (VRPTWSD). They solve the problem heuristically. A tabu search for VRPTWSD is developed by Ho and Haugland (2004).

Lau et al. (2003) formulate the vehicle routing problem with time windows and a limited number of vehicles (m-VRPTW) and solve it using a tabu search approach. See Lim and Zhang (2005) and Li et al. (2004) for other heuristic approaches to the same problem.

The most promising recent results for exact solution of VRPTW use column generation. Ioachim et al. (1999) describe a routing problem with synchronization constraints and use column generation to solve this problem. The synchronization constraints are modeled in the master problem with the consequence that a large number of columns with a small variation in departure time are generated.

Boussier et al. (2007) describe a Branch-and-Price algorithm for solving m-VRPTW and report promising results. The work is a continuation of preliminary work by Gueguen (1999). In this work, Gueguen also describes an exact approach to VRPTWSD. Gendreau et al. (2006*b*) consider the VRPTWSD as well and introduce a new set covering model for this problem. Properties of the model are studied, and a column generation based solution method is presented. With the method, they are able to solve a number of smaller instances to optimality.

Column generation for the pure VRPTW was initiated by Desrochers et al. (1992). They solve the pricing problem as a *Shortest Path Problem with Time Windows (SPPTW)*. Their approach proved very successful and was further applied and developed by Kohl (1995), Kohl et al. (1999), Larsen (1999), Cook and Rich (1999), Kallehauge et al. (2001), Righini and Salani (2006), Irnich and Villeneuve (2006).

Recently, Feillet et al. (2004) suggested solving the pricing problem as an *Elementary Shortest Path Problem with Time Windows (ESPPTW)* building on the ideas of Beasley and Christofides (1989). Chabrier (2006), Danna and Pape (2005), and Jepsen et al. (2006) have extended the ideas and achieved very promising results.

Finally, we turn the attention to recent work by Bredström and Rönnqvist which is described in a discussion paper (Bredström and Rönnqvist, 2007) extending the work of an earlier discussion paper (Bredström and Rönnqvist, 2006). The problem considered is similar to MAPTWTC and is dealing with an application in home care. The problem is solved using a Branch-and-Price setup and the conclusions of the paper correspond nicely with the findings that we present in this paper.

The remainder of this paper is structured as follows. In Section C.2, we present an Integer Programming (IP) formulation of m-MAPTWTC. In Section C.3, the formulation is decomposed into a master problem and a pricing problem using Dantzig-Wolfe decomposition. This decomposition allows us to solve the problem using column generation in a Branch-and-Price framework. In Section C.4, the necessary branching rules are described. This includes branching to enforce integrality as well as synchronized cooperation on tasks. The computational results on a number of real-life problems are presented in Section C.5. Finally, in Section C.6 we conclude on our work and discuss possible areas for future research.

C.2 Problem Definitions and Formulation

C.2.1 IP Formulation of m-MAPTWTC

Consider a set C of n tasks and a workforce of inhomogeneous teams V. All shifts begin at a service center, referred to as location 0. The set of tasks together with the service center is denoted N. For each task $i \in C$ a time window is defined as $[a_i, b_i]$ where a_i and b_i are the earliest and the latest starting times for task i, respectively. r_i is the number of teams required to carry out task i (Task i is divided into r_i *split tasks*). Each team $k \in V$ has a time window $[e_k, f_k]$, where the team starts at the service center at time e_k and must return no later than f_k. Between each pair of tasks (i, j), we associate a time t_{ij} which contains the travel time from i to j and the service time at task i. Further, g_{ik} is a binary parameter defining whether team k has the required qualifications for task i ($g_{ik} = 1$) or not ($g_{ik} = 0$).

We assume that a_i, b_i, e_k, and f_k are non-negative integers and that each t_{ij} is a positive integer. We also assume that the triangular inequality is satisfied for t_{ij}.

To solve the problem, two sets of decision variables have to be defined:

x_{ijk} is binary with $x_{ijk} = \begin{cases} 1, & \text{if team } k \text{ goes directly from task } i \text{ to task } j. \\ 0, & \text{otherwise} \end{cases}$

s_i is an integer variable and defines the start time of task i.

m-MAPTWTC can be formulated mathematically as:

$$\max \sum_{k\in V}\sum_{i\in C}\sum_{j\in N} x_{ijk} \tag{C.1}$$

$$\sum_{k\in V}\sum_{j\in N} x_{ijk} \leq r_i \qquad \forall i \in C \tag{C.2}$$

$$x_{ijk} \leq g_{ik} \qquad \forall i \in C, \forall j \in C, \forall k \in V \tag{C.3}$$

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

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

$$e_k + t_{0j} - M(1 - x_{0jk}) \leq s_j \qquad \forall j \in C, \forall k \in V \tag{C.6}$$

$$s_i + t_{i0} - M(1 - x_{i0k}) \leq f_k \qquad \forall i \in C, \forall k \in V \tag{C.7}$$

$$s_i + t_{ij} - M(1 - x_{ijk}) \leq s_j \qquad \forall i \in C, \forall j \in C, \forall k \in V \tag{C.8}$$

$$a_i \leq s_i \leq b_i \qquad \forall i \in C \tag{C.9}$$

$$x_{ijk} \in \{0, 1\} \qquad \forall i \in N, \forall j \in N, \forall k \in V \tag{C.10}$$

$$s_i \in \mathbb{Z}^+ \cup \{0\} \qquad \forall i \in C \tag{C.11}$$

The objective (C.1) is to maximize the number of assigned tasks. A task is counted multiple times if split between teams ($r_i \geq 2$). The constraints (C.2) guarantee that each task is assigned the right number of teams or possibly less, if some of its split tasks are left unassigned. Only teams with the required skill can be assigned to a specific task (C.3). Furthermore, we have to ensure that all shifts start in the service center (C.4). Constraints (C.5) ensure that no shifts are segmented. Any task visited by a team must be left again. The next four constraints deal with the time windows. First, we ensure that a team can only be assigned to a task during their working hours (C.6)-(C.7). Next, we check if the time needed for traveling between tasks is available (C.8). If a customer i is not visited, the large scalar M makes the corresponding constraints non-binding. Constraints (C.9) enforce the task time windows. Finally, constraints

(C.10)-(C.11) are the integrality constraints. The introduction of a service start time removes the need for sub-tour elimination constraints, since each customer can only be serviced once during the scheduling horizon because t_{ij} is positive.

C.2.2 Relations to Vehicle Routing

As mentioned earlier, m-MAPTWTC is closely related to VRPTW. Consider the teams as vehicles driving from one customer to another as they in m-MAPTWTC move from one task to another. The service that the teams deliver is an amount of their time, unlike the vehicles that deliver goods which have taken up a part of the total volume. Hence, in that sense m-MAPTWTC is uncapacitated. Except for the binding between teams inflicted by the possibility of cooperation on tasks, the problem is similar to the Uncapacitated Vehicle Routing Problem with Time Windows and a limited number of vehicles (m-VRPTW).

Column generation has proven a successful technique for exact solution of VRPTW and as m-MAPTWTC is also NP-hard (see Li et al., 2005) the solution procedure in this article is built on the principles of column generation in a Branch-and-Price framework.

C.3 Decomposition

We present the Dantzig-Wolfe decomposition (Dantzig and Wolfe, 1960) of m-MAPTWTC. First, we introduce the notion of a *path*. A feasible path is defined as a shift starting and ending at the service center, obeying time windows and skill requirements, but disregarding the constraints dealing with interaction between shifts. By this definition the feasibility of a path can be determined without further knowledge about other paths. We define $\mathcal{P}_k$ as the set of all feasible paths for team $k \in V$. Let the set T_i be the set of all possible start times for task i. Each path is defined by the tasks it visits and the time of initiation of each task. Let $\hat{a}_{ik}^{pt} = 1$ if task i is initiated at time t on path p for team k and $\hat{a}_{ik}^{pt} = 0$ otherwise.

C.3.1 Master Problem

In the *integer master problem* we solve the problem of optimally choosing one feasible path for each team, maximizing the total number of assigned tasks. In the original formulation, the equations (C.3)-(C.9) are used to ensure feasibility of paths. In the master problem, the set $\mathcal{P}_k$ is used to guarantee this feasibility. The use of only one s_i for each task had the effect that cooperating teams would initiate work simultaneously. In the master problem this is enforced by a new binary decision variable γ_i^t. Ioachim et al. (1999) and van den Akker et al. (2006) describe versions of the master problem model, where the s_i variables are introduced directly in the master problem. This, however, introduces non-binary coefficients in the master problem, and that is usually a feature that leads to highly fractional solutions when solving the LP-relaxation.

Now, the integer programming master problem is formulated as below, where λ_k^p are binary variables, which for each vehicle k are used to select a path p from $\mathcal{P}_k$. γ_i^t is a binary variable deciding if task i is initiated at time t. Any feasible solution to the master problem is a feasible solution to the original formulation.

$$\max \sum_{k\in V}\sum_{i\in N}\sum_{p\in\mathcal{P}_k}\sum_{t\in T_i} \hat{a}_{ik}^{pt}\lambda_k^p \tag{C.12}$$

$$\sum_{k\in V}\sum_{p\in\mathcal{P}_k} \hat{a}_{ik}^{pt}\lambda_k^p \leq r_i\gamma_i^t \qquad \forall i\in C, \forall t\in T_i \tag{C.13}$$

$$\sum_{t\in T_i}\gamma_i^t = 1 \qquad \forall i\in C \tag{C.14}$$

$$\sum_{p\in\mathcal{P}_k}\lambda_k^p = 1 \qquad \forall k\in V \tag{C.15}$$

$$\lambda_k^p \in \{0,1\} \qquad \forall k\in V, \forall p\in\mathcal{P}_k \tag{C.16}$$

$$\gamma_i^t \in \{0,1\} \qquad \forall i\in C, \forall t\in T_i \tag{C.17}$$

The objective still is to maximize the number of assigned tasks (C.12). (C.13) has two effects. For each team it ensures that a path can only be selected if all tasks in the path comply with their respective time of initiation. Further, it ensures that each task is not assigned more teams than requested. In (C.14) we force all tasks to have only one time of initiation, and (C.15) guarantees that all teams have exactly one path assigned to them.

To apply column generation, the integrality constraints are relaxed to allow solution of the master problem by a standard linear solver. Unfortunately, the γ_i^t-variables lose all significance when LP-relaxed. Consider the LP-relaxed problem, i.e. (C.12)-(C.15) with the relaxed constraints $0 \leq \lambda_k^p \leq 1, \forall k \in V, \forall p \in \mathcal{P}_k$ and $0 \leq \gamma_i^t \leq 1, \forall i \in C, \forall t \in T_i$. The LP-problem is a relaxation of the following problem:

$$\max \sum_{k\in V} \sum_{i\in N} \sum_{p\in\mathcal{P}_k} \sum_{t\in T_i} \hat{a}_{ik}^{pt} \lambda_k^p \tag{C.18}$$

$$\sum_{k\in V} \sum_{p\in\mathcal{P}_k} \sum_{\forall t\in T_i} \hat{a}_{ik}^{pt} \lambda_k^p \leq r_i \qquad \forall i \in C \tag{C.19}$$

$$\sum_{p\in\mathcal{P}_k} \lambda_k^p = 1 \qquad \forall k \in V \tag{C.20}$$

$$0 \leq \lambda_k^p \leq 1 \qquad \forall k \in V, \forall p \in \mathcal{P}_k \tag{C.21}$$

Proof. According to Wolsey (1998): A problem (PR) $z^R = \max\{f(x) : x \in T \subseteq \mathbb{R}^n\}$ is a relaxation of (P) $z = \max\{c(x) : x \in X \subseteq \mathbb{R}^n\}$ if:

1. $X \subseteq T$
2. $f(x) \geq c(x), \qquad \forall x \in X$

Take any feasible solution λ' to (C.18)-(C.21). Set each γ'^t_i equal to the portion of paths where time t is used for task i:

$$\gamma'^t_i = \sum_{k\in V} \sum_{p\in\mathcal{P}_k} \hat{a}_{ik}^{pt} \lambda_k^p \,/ \sum_{k\in V} \sum_{p\in\mathcal{P}_k} \sum_{t'\in T_i} \hat{a}_{ik}^{pt'} \lambda_k^p$$

Using (C.19), (C.13) is satisfied since:

$$\begin{aligned}
\forall i \in C, \forall t \in T_i : r_i \gamma'^t_i &= r_i \sum_{k\in V} \sum_{p\in\mathcal{P}_k} \hat{a}_{ik}^{pt} \lambda_k^p / \sum_{k\in V} \sum_{p\in\mathcal{P}_k} \sum_{t'\in T_i} \hat{a}_{ik}^{pt'} \lambda_k^p \\
&= \sum_{k\in V} \sum_{p\in\mathcal{P}_k} \hat{a}_{ik}^{pt} \lambda_k^p \, (r_i / \sum_{k\in V} \sum_{p\in\mathcal{P}_k} \sum_{t'\in T_i} \hat{a}_{ik}^{pt'} \lambda_k^p) \\
&\geq \sum_{k\in V} \sum_{p\in\mathcal{P}_k} \hat{a}_{ik}^{pt} \lambda_k^p
\end{aligned}$$

γ'^t_i obviously satisfies (C.14) and (C.15) is identical to (C.20). So for each solution to (C.18)-(C.21) there is a corresponding solution to the LP-relaxation of (C.12)-(C.17). Since the objective functions (C.12) and (C.18) are identical, the projection of the LP-relaxation of (C.12)-(C.17) onto the λ-subspace is a relaxation of (C.18)-(C.21). $\square$

Hence, instead of using the model directly, we relax the constraint on synchronized cooperation by using the model (C.18)-(C.21). We define $a_{ik}^{p} = \sum_{\forall t \in T_i} \hat{a}_{ik}^{pt}, \forall i \in C, \forall k \in V, \forall p \in \mathcal{P}_k$, where $a_{ik}^{p} = 1$ if task i is in path p for vehicle k and $a_{ik}^{p} = 0$ otherwise. At the same time, we choose to change from a maximization problem to a minimization problem by introducing δ_i as the number of unassigned split tasks of task i. This is our *relaxed master problem.* Finally, to decrease the size of the problem, a set of promising paths $\mathcal{P}'_k$ $(\subseteq \mathcal{P}_k)$ is used instead of $\mathcal{P}_k$. In a column generation context $\mathcal{P}'_k$ contains all paths generated for team k in the pricing problem so far. We arrive at the *restricted master problem (RMP)*:

$$\min \sum_{i \in C} \delta_i \tag{C.22}$$

$$\delta_i + \sum_{k \in V} \sum_{p \in \mathcal{P}'_k} a_{ik}^{p} \lambda_k^p \geq r_i \qquad \forall i \in C \tag{C.23}$$

$$\sum_{p \in \mathcal{P}'_k} \lambda_k^p = 1 \qquad \forall k \in V \tag{C.24}$$

$$\lambda_k^p \geq 0 \qquad \forall k \in V, \forall p \in \mathcal{P}'_k \tag{C.25}$$

$$\delta_i \geq 0 \qquad \forall i \in C \tag{C.26}$$

The sum of δ_i over all tasks is minimized (C.22). (C.19) is changed to a greater-than inequality constraint, penalizing inadequate assignment to a task by adding δ_i (C.23). This change allows tasks to be done more times than required, which is useful in a column generation setting, where an existing column may enter the solution basis, and we do not have to generate a new, almost identical column containing a subset of the tasks. As a consequence, the estimates of the final dual variables improve (see Kallehauge et al., 2005). The new master problem has the form of a generalized set-covering problem.

On the downside, any solution may now contain *overcovering*, i.e. we may have tasks which are assigned to more teams than requested. However, in the new formulation, overcovering can be removed without altering the objective value by unassigning the superfluous number of teams for each task. The modified solution is still feasible and the overcovering can hence easily be removed from an optimal solution.

If the master problem contains no columns representing paths from the outset of the column generation procedure, the problem will be infeasible due to the team constraints (C.24). Therefore, we add an *empty path* λ_k^0 ($a_{ik}^0 = 0, \forall i \in C$) for each team to ensure feasibility whether regular paths are present or not. An empty path can only be part of an optimal solution if the presence of the team can not decrease the number of unassigned tasks. This will be the case if manpower is available in abundance or the skills or working hours of the team do not match those of the tasks.

The solution to the restricted master problem may not be integer. In addition, we have relaxed the constraint on synchronization of tasks. Both of these properties must be enforced by a branching scheme.

The solution to the restricted master problem is not guaranteed to be optimal either, since only a small subset of feasible paths is considered. For each primal solution λ to the restricted master problem we obtain a dual solution $[\pi, \tau]$, where π and τ are the dual variables of constraints (C.23) and (C.24) respectively. In column generation, the dual solution is used in the pricing problem to ensure the generation of columns leading to an improvement of the solution to the master problem.

C.3.2 Pricing problem

The *pricing problem* specifies all the requirements of a feasible path. The objective is to find the path with the lowest possible reduced cost. In m-MAPTWTC with inhomogeneous teams as described above, we obtain $m = |V|$ separate pricing problems. Each pricing problem is an *Elementary Shortest Path Problem with Time Windows (ESPPTW)*. The binary variable x_{ij} is defined as $x_{ij} = 1$ if the team goes directly from task i to task j and $x_{ij} = 0$ otherwise. For a team $k' \in V$ the pricing problem is formulated as:

$$\min \sum_{i \in C_{k'}} \sum_{j \in C_{k'}} -\pi_i x_{ij} - \tau_{k'} \tag{C.27}$$

$$\sum_{j \in N_{k'}} x_{0j} = 1 \tag{C.28}$$

$$\sum_{i \in N_{k'}} x_{ih} - \sum_{j \in N_{k'}} x_{hj} = 0 \qquad \forall h \in N_{k'} \tag{C.29}$$

$$e_{k'} + t_{0j} - M(1 - x_{0j}) \leq s_j \qquad \forall j \in C_{k'} \tag{C.30}$$

$$s_i + t_{i0} - M(1 - x_{i0}) \leq f_{k'} \qquad \forall i \in C_{k'} \tag{C.31}$$

$$s_i + t_{ij} - M(1 - x_{ij}) \leq s_j \qquad \forall i \in C_{k'}, \forall j \in C_{k'} \tag{C.32}$$

$$a_i \leq s_i \leq b_i \qquad \forall i \in C_{k'} \tag{C.33}$$

$$x_{ij} \in \{0, 1\} \qquad \forall i \in N_{k'}, \forall j \in N_{k'} \tag{C.34}$$

$$s_i \in \mathbb{Z}^+ \cup \{0\} \qquad \forall i \in C_{k'} \tag{C.35}$$

The constraints match the constraints of the original formulation except for the relation between vehicles (C.2). The skill requirements are respected by fixing $x_{ij} = 0$ for all tasks where $g_{ik} = 0$ and hence excluding those tasks from the sets: $C_{k'}$ and $N_{k'}$.

The pricing problem can be interpreted as a graph problem. Consider a graph $G(N_G, E_G, c, t)$, where the nodes N_G are all tasks plus the service center and E_G is the set of edges connecting all nodes. With each edge $e \in E_G$ is associated a travel time $t_e = t_{ij}$ and a cost $c_e = c_{ij} = -\pi_i$, where i and j are the two nodes connected by e. To simplify, the service center is usually split into two vertices: a start vertex 0 and an end vertex $n + 1$. The objective is to find a path in G from 0 to $n + 1$ with a minimum sum of edge costs that does not violate any time windows.

Solution methods to the Shortest Path Problem with Time Windows have been studied extensively in the literature and successful algorithms for solving SPPTW have been built on the concept of dynamic algorithms. We solve the elementary version of the problem (ESPPTW), where no cycles are allowed. Dror (1994) proves that the problem is NP-hard in the strong sense and thus no pseudo-polynomial algorithms are likely to exist. We use a label setting algorithm built on the ideas of Chabrier (2006) and Jepsen et al. (2006). The authors of both papers have recently succeeded in solving previously unsolved VRPTW benchmarking instances (from the Solomon Test-sets of Solomon, 1987) by ESPPTW-based column generation. Furthermore, Feillet et al. (2005), Feillet et al. (2004) address the Vehicle Routing Problem with Profits (similar to the Vehicle Routing Problem with a limited number of vehicles) and state that solving the elementary shortest path problem as opposed to the relaxed version is essential to obtain good bounds.

We will not go into the details of the label setting algorithm, since the problem is almost identical to the pricing problem of VRPTW. We have a shortest path problem where all arc costs out of a node are identical and hence can be moved to the node. The pricing problems are first solved in a heuristic label setting approach and if no columns can be added, we switch to the exact label setting algorithm.

C.3.3 Linking the Pricing Problem to the Master Problem

Team Priorities

As described earlier, each team has its separate pricing problem. This introduces the challenge of choosing the pricing problem in each iteration that is most likely to return usable columns. Initially, we implement a round-robin style mechanism, where each team is picked in turn. If a whole round is completed without at least one pricing problem returning a path with negative reduced cost, optimality is proven for the relaxed master problem.

Typically, some teams have less tight schedules than others and good columns are generated earlier in the process. We introduce another scheme to utilize this feature. We associate each team with a *team priority,* which is set equal to the reduced cost of the latest returned column. If no column was returned for team k, the team priority is set to a positive number higher than all other priority values to ensure that all other teams are treated before considering team k again.

By using team priorities, the teams which have recently shown the biggest improvements are treated first. Notice, that in some iterations we may not find the column with minimum reduced cost as it may be associated with a different team. However, when terminating the column generation, optimality is guaranteed in the same way as for the simple round-robin scheme.

Store Last Solved Pricing Problem

Having a number of separate teams with different skills and scheduling horizons means that the pricing problems of some teams do not change for many iterations. In the extreme case, we sometimes see master problems which are actually separable, i.e. the assignment of tasks to one team has no way of altering the dual variables for the pricing problem of another team. In these cases we may solve the exact same pricing problem repeatedly. To avoid this, we save the last solved pricing problem for each team, if it did not return any columns with negative reduced cost. If it did return such a column, there is no point in saving the problem as the dual variables will now have changed.

Prior to solving a pricing problem, it is checked whether any circumstances have changed since last time. These circumstances include dual variables and relevant branching decisions.

C.4 Branching

C.4.1 Branching to get integral solutions

Various branching strategies for VRPTW have been proposed. See Kallehauge et al. (2005) for a more thorough review of branching strategies for VRPTW. In the MAPTWTC setting, a 0-1 branching on an original flow variable x_{ijk} (proposed independently by Halse, 1992 and Desrochers et al., 1992) is equivalent to forcing team k to do (banning team k from doing, respectively) task j immediately after task i. The branching is enforced by removing illegal columns in the master problem in each child node and removing illegal arcs in the network formulation of the pricing problem for team k. In VRPTW, another possibility is to perform a 0-1 branching on $\sum_k x_{ijk}$ thus imposing the above constraint on all teams simultaneously. However, since the teams are inhomogeneous due to different qualifications and work hours and since tasks i and j may need several teams to cooperate, the branching rule is no longer a 0-1 branching and the advantage of keeping just one identical pricing problem for all teams is obviously lost.

Instead, we focus on a 0-1 branching scheme based on $\sum_j x_{ijk}$ which simply implies that team k is either forced to or banned from an assignment to task i. Unlike the two strategies above, there is no need to keep track of the status of individual arcs in the pricing problems of the child nodes. The node corresponding to task i is either removed from the network (along with *all* arcs incident to it) or given a very low (negative) cost to ensure its inclusion in any optimal solution to the pricing problem.

C.4.2 Synchronized Cooperation using branching

Consider an optimal solution to the relaxed master problem, fractional or integral, and let s_i^p be the point in time where execution of task i begins on path p (if i is not a part of p, s_i^p is irrelevant). The solution violates the synchronized cooperation constraint for some task i if there exist positive variables $\lambda_{k_1}^{p_1}$ and $\lambda_{k_2}^{p_2}$ associated with the two paths p_1 and p_2 ($p_1 \neq p_2$), both containing i where

$$s_i^{p_1} \neq s_i^{p_2}$$

If the solution is fractional, the teams k_1 and k_2 may be identical. In this case, the team can be perceived as cooperating with itself.

Define $s_i^* = \lceil (s_i^{p_1} + s_i^{p_2})/2 \rceil$ as the *split time*. Now, split the problem into two new branches and define new time windows for task i as

$$a_i' \leq s_i \leq s_i^* - 1 \qquad \text{and} \qquad s_i^* \leq s_i \leq b_i'$$

respectively, where $a_i' \leq s_i \leq b_i'$ was the time window of task i in the current branch. Existing columns not satisfying the new time windows are removed from the corresponding child nodes and new columns generated must also respect the updated time window. In this way, the current solution is cut off in both branches and the new subspaces are disjoint. Since time has been discretized the branching strategy is guaranteed to be complete.

The idea behind this branching scheme is to restrict the number of points in time, where the execution of task i can begin. If the limited time window makes it inconvenient for the teams to complete task i, the lower bound will increase and the branch is likely to be pruned at an early stage. On the other hand, if the limited time window contains an optimal point in time for the execution of task i, it may be necessary to continue the time window branching until a singleton interval is reached. The time is discretized into a finite number of steps (minutes), and hence this will always be possible. However, since the label setting algorithm for the pricing problem aims at placing tasks as early as possible (see Desrochers et al., 1992), the actual number of different positions in time for any task is rather small. In fact, as the time windows are reduced, the tasks are more and more likely to be placed at the very beginning of their time window. This property greatly reduces the number of branching steps needed.

Using time window branching, the solution will eventually become feasible with respect to the synchronized cooperation constraint. It is not guaranteed to be integral, though, and it may therefore be necessary to apply the regular $\sum_j x_{ijk}$ branching scheme, branching on a combination of a task and a team. As both schemes have a finite number of branching candidates, the solution algorithm will terminate when they are used in combination. In general, when none of the feasibility criteria (integrality and synchronized cooperation) are fulfilled, we have a choice of branching scheme.

Our algorithm has been set to use time window branching whenever applicable. The restricted time windows reduce flexibility in the column generation which, in turn, limits the possibilities of combining fractional columns when solving the master problem. Thus, time window branching is also expected to have a positive influence on the integrality of the solution as observed by Gélinas et al. (1995) for VRPTW. This property has also been observed in practice when testing the algorithm, hence the choice of prioritizing time window branching.

We now focus on how good branching candidates are selected for branching. Let P_i be the set of all paths p including task i with $\lambda_k^p > 0$ in the current solution to the restricted master problem. If

$$s_i^{p_1} \neq s_i^{p_2}$$

for any two paths $p_1, p_2 \in P_i$, task i is stored in the set C' of possible candidates. We determine the split time as

$$s_i^* = \left\lceil \frac{\min_{p \in P_i} (s_i^p) + \max_{p \in P_i} (s_i^p)}{2} \right\rceil, \forall i \in C'$$

When ranking the branching candidates, we prefer candidates that provide a balanced search tree. That is, the paths in P_i should be divided equally into the two child nodes when weighted according to the variable values λ_k^p. Define

$$S_i = \sum_{k \in V, p \in P_i} \lambda_k^p, \forall i \in C'$$

as the sum of all positive variables containing i and let

$$S_i^< = \sum_{k \in V, p \in P_i | s_i^p < s_i^*} \lambda_k^p, \forall i \in C'$$

be the same sum restricted to the variables where task i is executed before the split time. The branching candidate i^* is now determined by

$$i^* = \arg\min_{i \in C'} \left| \frac{S_i^<}{S_i} - 0.5 \right|$$

C.5 Computational Results

The Branch-and-Price algorithm has been implemented in the Branch-and-Cut-and-Price framework of COIN-OR (Lougee-Heimer, 2003, Coin, 2006) and tests have been run on 2.7 GHz AMD processors with 2 GB RAM. The implementation has been tuned to the problems at hand and parameter settings have been made on the basis of these problems. The algorithm is set to do strong branching (Achterberg et al., 2005) with 25 branching candidates and adds up to 10 columns with negative reduced cost per pricing problem.

The test data sets originate from real-life situations faced by ground handling companies in two of Europe's major airports. This gives rise to four different problem types, since the two airports each produce problems of two distinctive types. Each type is represented by three problem instances, each spanning approximately one 24-hour day, thus, a total of 12 test instances are available.

Generally, the four problem types can be summarized as (In brackets: The total number of tasks after splitting into requested split tasks):

Type A Small instances, Airport 1. 12-13 teams and 80 (120) tasks

Type B Medium instances, Airport 2. 27 teams and 90 (150) tasks.

Type C Small instances, Airport 2. 15 teams and 90 (110) tasks.

Type D Large instances, Airport 1. 19-20 teams and 270 (300) tasks.

The problem instance **A.1** and its optimal solution is illustrated in Figure C.1. The figure depicts the distribution of tasks over the day and the skill requirements for these. The execution time of tasks and the length of their time windows are similar in the other problem types. In our problem instances, each team must be given a predefined number of breaks during their day and within certain time windows. Breaks are treated as regular tasks, with the exceptions that they can only be assigned to the related team, and they cannot be left unassigned in a feasible solution.

The individual schedules of the teams are captured in the 13 boxes, which clearly show the start and end time of each shift. Each task is represented by

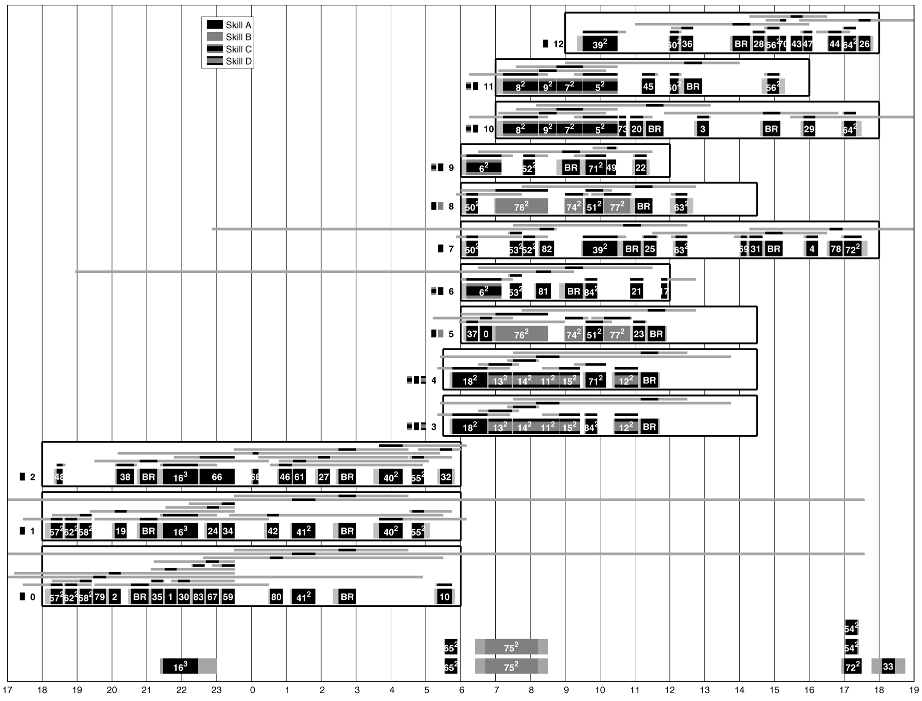


Figure C.1: Problem instance **A.1** and its optimal solution.

one or more small boxes labeled with the task ID (Breaks have ID: "BR"). The superscript denotes the number of teams that the task must be split between. This number therefore corresponds to the total number of boxes labeled with the task ID of this task. Above each task is a thin box depicting the time window of the task. Furthermore, each task has a color pattern revealing its skill requirement. Each team has between one and three skills, identified by the small squares to the left of the team ID. To assign a task to a team, the color pattern of the task must match that of one of these small squares.

To illustrate how to read the figure, we go through the work plan of team 9. The first task carried out is task 6 which requires skill C. The task is scheduled from 6:10 to 7:10 and hence the time window of the task is respected, since execution cannot start before 6 o'clock and must be finished by 7:30. The task is solved in collaboration with team 6. The light gray box in front of the task gives the required travel time. Next, the team takes care of task 52 (requires skill A), this time cooperating with team 7. After this, team 9 is given their daily break. Subsequently, they will carry out 71, 49, and 22, where task 49 and task 22 are dealt with by team 9 alone.

	A.1	**A.2**	**A.3**	**B.1**	**B.2**	**B.3**	**C.1**	**C.2**	**C.3**	**D.1**	**D.2**	**D.3**
Unassigned split tasks	9	*7	1	0	3	5	*3	*6	*10	*29	24	*31
Lower Bound$^{\otimes}$	9	6	1	0	3	5	2	4	9	27	24	30
Time (s)	133	OM	2663	120	172	97	OM	OM	OM	TO	2719	TO
- LP (%)	15	46	20	10	10	11	29	9	34	2	5	3
- Branching (%)	68	7	70	82	82	78	34	81	32	5	10	4
- Pricing Problem (%)	4	8	2	1	2	2	4	4	9	93	83	91
- Overhead (%)	13	39	8	7	6	9	33	6	25	0	2	2
Tree size	605	42435	3207	537	597	507	188623	87843	69637	4961	487	2741
Max. depth	160	162	168	264	291	253	122	166	204	219	235	228
# Pricing Problems	13292	$3 \cdot 10^6$	107320	15554	17240	14813	$3 \cdot 10^6$	$2 \cdot 10^6$	$2 \cdot 10^6$	379799	20728	247634
# Vars added	12268	$2 \cdot 10^6$	109810	4074	5223	4321	$2 \cdot 10^6$	$1 \cdot 10^6$	$1 \cdot 10^6$	231209	16659	204614

Table C.1: Results of the Branch-and-Price algorithm with no initial solution. OM = Out-of-Memory was encountered. TO = The Time-Out limit of 10 hours was reached.
* The solution given is the best feasible solution found.
$^{\otimes}$ Lower Bound (more details in Table C.3).

In Table C.1 the results from the 12 datasets are given. From the table we conclude the following. 6 of the 12 datasets were solved to optimality within one hour. The remaining 6 instances are split in two cases: one case for the small and medium-sized problems (**Type A-C**) and one case for the large instances (**Type D**). For the unsolved problems of **Type A-C** we see an explosion in the size of the branching tree. In these cases the time-out limit is never reached, since we run out of memory before time out. The reported results for these instances have been recorded after 2 hours, which in these cases is just before the memory limit is reached. For **Type D** the results indicate that the generation of columns is now in itself a time-consuming task and time-out is encountered with a relatively small tree-size.

The branching trees from the above test have been built without a good initial solution. For each of the unfinished problems, we restart the algorithm with an initial solution, namely the best feasible solution of Table C.1. The results of the new test are displayed in Table C.2.

It is interesting that most of these instances are now solved to optimality within seconds. It clearly indicates that inexpedient branching decisions were made in the first run and more reliable branching is possible when promising columns exist initially. Another observation is that solving **C.1** under default settings leads to another out-of-memory failure, whereas changing the settings slightly gives an optimal solution within one second. This is another indication of the importance of making the right branching decisions and the consequence of not doing so. It has been tested that the settings giving a fast solution in this case are not superior in general.

Systematic exploitation of these features is outside the scope of this article.

	A.1	A.2	A.3	B.1	B.2	B.3	C.1	C.2	C.3	D.1	D.2	D.3
Unassigned split tasks	9	7	1	0	3	5	$^{\times}$3	4	9	*29	24	31
Lower Bound$^{\otimes}$	9	6	1	0	3	5	2	4	9	27	24	30
Time (s)		0.84					0.80	36	0.97	TO		235
- LP (%)		33					25	21	17	0		5
- Branching (%)		5					8	25	8	0		0
- Pricing Problem (%)		18					6	14	8	100		95
- Overhead (%)		44					61	40	67	0		0
Tree size		11					19	981	59	447		9
Max. depth		3					5	46	28	40		4
# Pricing Problems		530					561	32921	1358	42284		6415
# Vars added		785					758	16406	475	37212		6104

Table C.2: Results of the Branch-and-Price algorithm with initial solution from the test of Table C.1.
TO = The Time-Out limit of 10 hours was reached.
* The solution given is the best feasible solution found.
$^{\times}$ After OM on the first run, the pricing problem solver was in this case changed to not create heuristic columns.
$^{\otimes}$ Lower Bound (more details in Table C.3).

Automatic restart of the branching procedure could be implemented fairly easy. Beck et al. (2006) describes a more sophisticated approach, where a number of promising solutions are saved and the tree search is restarted from one of these solutions, when the search seems to be stuck. A similar methodology may prove to be very efficient in our case. To achieve even faster results, a variety of acceleration strategies should be investigated. Look to Danna and Pape (2005) for more on this topic.

	A.1	A.2	A.3	B.1	B.2	B.3	C.1	C.2	C.3	D.1	D.2	D.3
Unassigned split tasks	9	6	1	0	3	5	2	4	9	27	24	30
Time (s)	0.96	1.10	1.37	0.64	0.77	0.80	1.18	1.86	1.65	75	413	2196
- LP (%)	16	8	15	6	4	3	19	25	10	17	10	2
- Branching (%)	7	0	0	0	0	1	39	24	49	17	8	1
- Pricing Problem (%)	45	74	69	19	22	30	9	11	17	62	81	97
- Overhead (%)	32	18	16	75	74	67	33	40	24	4	1	0
Tree size	3	3	1	1	3	3	11	21	19	35	83	97
Max. depth	1	1	0	0	1	1	5	8	9	17	41	22
# Pricing Problems	163	93	291	103	80	81	288	481	367	4450	8783	9811
# Vars added	407	350	663	309	222	212	586	683	435	4489	7773	14111

Table C.3: Results of the Branch-and-Price algorithm with no constraint on synchronized coordination.
All solution values can be used as lower bounds on the original formulation.

To reveal the complexity added by the synchronized cooperation requirement, we also show results for a version of the problem where no branching on time windows is done (Table C.3). This means that cooperation is no longer synchronized, but we are able to reach optimal solutions faster. Since the latter is a relaxation of the original problem, we are able to use the solution values as lower bounds on our problem.

Solution times of Table C.3 should be compared to the times of Table C.1 and reveal that solving the relaxed problem evidently is much faster and optimal solutions are found in all cases. The running times for the small and medium problems are up to 2 seconds, where one of the large problem instances uses around 37 minutes.

It is conspicuous that all the optimal solutions found in Table C.1 are equal to the lower bound found in Table C.3. The lower bound found by the unsynchronized model is naturally closely related to the lower bound found in the root node of the branching tree of the problems in Table C.1 and these results stress how important a good lower bound is.

C.6 Conclusion and future work

The Manpower Allocation Problem with Time Windows, Job-Teaming Constraints and a limited number of teams is successfully solved to optimality using a Branch-and-Price approach. By relaxing the synchronization constraint and using Dantzig-Wolfe decomposition, the problem is divided into a generalized set covering master problem and an elementary shortest path pricing problem. Applying branching rules to enforce integrality as well as synchronized execution of divided tasks enables us to arrive at optimal solutions in half of the test instances. Running a second round of the optimization, initiated from the best solution found in round one, uncovers the optimal solution to all but one of the 12 test instances. The test instances are all full-size realistic problems originating from scheduling problems of ground handling tasks in major airports. Synchronization between teams in an exact optimization context has not previously been treated in the literature. We have successfully integrated the extra requirements into the solution procedure and the results are promising.

Future work could aim at creating a structured approach to utilize the effect of restarting the branching mechanism. By simply restarting the algorithm once, we see a remarkable increase in the number of solvable problems, and an extended strategy may shorten solution time significantly and it may further increase the chance of finding optimal solutions. Other acceleration strategies are likely to reveal improved results as well.

Acknowledgements

We thank the anonymous referees for their constructive feedback and useful references.

References

Achterberg, T., T. Koch, and A. Martin (2005). “Branching Rules Revisited”. In: *Operations Research Letters* 33.1, pp. 42–54.

Beasley, J. E. and N. Christofides (1989). “An Algorithm for the Resource Constrained Shortest Path Problem”. In: *Networks* 19, pp. 379–394.

Beck, J., P. Prosser, and E. Selensky (2006). “A case study of mutual routing-scheduling reformulation”. In: *Journal of Scheduling* 9.5, pp. 469–491.

Boussier, S., D. Feillet, and M. Gendreau (2007). “An exact algorithm for team orienteering problems”. In: *4OR* 5.3, pp. 211–230.

Bredström, D. and M. Rönnqvist (2006). *Combined Vehicle Routing and Scheduling with Temporal Precedence and Synchronization Constraints.* Discussion Paper 2006/18. Norway: Norwegian School of Economics, Business Administration (NHH) - Department of Finance, and Management Science.

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

Chabrier, A. (2006). “Vehicle Routing Problem with elementary shortest path based column generation”. Ed. by Louis-Martin Rousseau Michel Gendreau Gilles Pesant. In: *Computers and Operations Research* 33.10, pp. 2972–2990.

Coin (2006). *COmputational INfrastructure for Operations Research (COIN-OR).* http://www.coin-or.org/.

Cook, W. and J. L. Rich (1999). *A Parallel Cutting-Plane Algorithm for the Vehicle Routing Problem with Time Windows.* Tech. rep. Rice Uni, Houston, TX, USA.

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 Jacques Desrosiers Guy Desaulniers and Marius M. Solomon. Springer. Chap. 4, pp. 99–129.

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

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

Dror, M. (1994). “Note on the Complexity of the Shortest Path Models for Column Generation in VRPTW”. In: *Operations Research* 42.5, pp. 977–978.

Dror, M. and P. Trudeau (1989). “Savings by Split Delivery Routing”. In: *Transportation Science* 23.2, pp. 141–149.

Feillet, D., P. Dejax, and M. Gendreau (2005). “Traveling Salesman Problems with Profits”. In: *Transportation Science* 39.2, pp. 188–205.

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”. In: *Networks* 44.3, pp. 216–229.

Frizzell, P. W. and J. W. Giffin (1995). “The Split Delivery Vehicle Scheduling Problem with Time Windows and Grid Network Distances”. In: *Computers and Operations Research* 22.6, pp. 655–667.

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

Gendreau, M., P. Dejax, D. Feillet, and C. Gueguen (2006*b*). *Vehicle Routing with Time Windows and Split Deliveries.* Tech. rep. 851. Laboratoire d'Informatique d'Avignon.

Gueguen, C. (1999). “Méthodes de Résolution Exacte Pour Les Problèmes de Tournées de Véhicules (Exact Methods for Solving Vehicle Routing Problems)”. PhD thesis. Laboratoire Productique Logistique, École Centrale Paris.

Halse, K. (1992). “Modeling and Solving Complex Vehicle Routing Problems”. PhD thesis. Technical University of Denmark.

Ho, S. C. and D. Haugland (2004). “A Tabu Search Heuristic for the Vehicle Routing Problem with Time Windows and Split Deliveries”. In: *Computers and Operations Research* 31.12, pp. 1947–1964.

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

Irnich, S. and D. Villeneuve (2006). “The Shortest Path Problem with Resource Constraints and k-cycle Elimination for $k \geq 3$”. In: *INFORMS Journal on Computing* 18.3.

Jepsen, M., B. Petersen, S. Spoorendonk, and D. Pisinger (2006). *A Non-Robust Branch-and-Cut-and-Price Algorithm for the Vehicle Routing Problem with Time Windows.* Tech. rep. Department of Computer Science, University of Copenhagen, Denmark.

Kallehauge, B., J. Larsen, and O. B. G. Madsen (2001). *Lagrangean Duality Applied on Vehicle Routing with Time Windows - Experimental Results.* Tech. rep. IMM, Technical University of Denmark, Copenhagen, Denmark.

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

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

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”. In: *Transportation Science* 33.1, pp. 101–116.

Larsen, J. (1999). “Parallelization of the Vehicle Routing Problem With Time Windows”. PhD thesis. Informatics and Mathematical Modelling, Technical University of Denmark, DTU, Denmark.

Lau, H. C., M. Sim, and K. M. Teo (2003). “Vehicle Routing Problem with Time Windows and a Limited Number of Vehicles”. In: *European Journal of Operational Research* 148, pp. 559–569.

Lee, C.-G., M. A. Epelman, C. C. W. III, and Y. A. Bozer (2006). “A Shortest Path Approach to the Multiple-Vehicle Routing Problem with Split Pick-Ups”. In: *Transportation Research Part B* 40, pp. 265–284.

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

Li, Z., S. Guo, F. Wang, and A. Lim (2004). “Improved GRASP with Tabu Search for Vehicle Routing with Both Time Window and Limited Number of Vehicles”. In: *Innovations in Applied Artificial Intelligence.* 17th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems, IEA/AIE 2004, pp. 552–561.

Lim, A. and X. Zhang (2005). “A Two-Stage Heuristic for the Vehicle Routing Problem with Time Windows and a Limited Number of Vehicles”. In: *Proceedings of the Proceedings of the 38th Annual Hawaii International Conference on System Sciences (HICSS’05) - Track 3 - Volume 03.* IEEE Computer Society, pp. 82c–82c.

Lim, A., B. Rodrigues, and L. Song (2004). “Manpower allocation with time windows”. In: *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”. In: *IBM Journal of Research and Development* 47.1, pp. 57–66.

Righini, G. and M. Salani (2006). “Symmetry helps: Bounded bi-directional dynamic programming for the elementary shortest path problem with resource constraints”. Ed. by Ulrich Faigle, Leo Liberti, Francesco Maffioli, and Stefan Pickl. In: *Discrete Optimization* 3.3, pp. 255–273.

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.

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

van den Akker, J., J. Hoogeveen, and J. van Kempen (2006). “Parallel machine scheduling through column generation: Minimax objective functions”. In: *Lecture Notes in Computer Science* 4168, pp. 648–659.

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

Appendix D

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

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

Submitted to: *European Journal of Operational Research* (2010).

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 caretakers 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 caretakers 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

*Submitted to: *European Journal of Operational Research* (2010).

[1]Department of Management Engineering, Technical University of Denmark, Produktionstorvet, 2800 Kongens Lyngby, Denmark.

D.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 caretakers in a way that maximises the service level, possibly even at a reduced cost.

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 caretakers.

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 caretakers. From the long-term plan, a specific schedule is created on a daily basis. In the daily problem, caretakers are assigned to visits. A route is built for each caretaker, respecting the competence requirements and time window of each visit and working hours of the caretaker.

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 caretakers), as this establishes confidence with the citizen. 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 caretakers.

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 caretaker 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 caretakers 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 caretakers 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 caretakers 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 caretakers. 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.

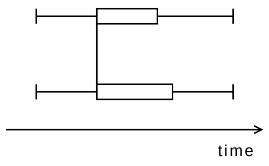
Related to the HCCSP is the Manpower Allocation Problem with Time Windows (MAPTW). A demanded number of servicemen must be allocated to each location within the time windows. Primarily the number of used servicemen must be minimised, and secondarily the used travel time. The jobs have different locations, 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. (2009*c*), 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. (2009*d*). 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.

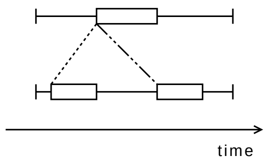
The remainder of the paper is organised as follows. In Section D.2, we present an IP formulation of HCCSP. In Section D.3, we introduce a decomposed version of this formulation. In Section D.4, we present the specialised branching scheme used. Section D.5 introduces clustering of visits and other methods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

D.2 Problem formulation

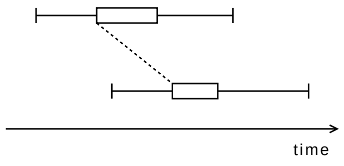
The set of caretakers is denoted $\mathcal{K}$, and the set of visits at the citizens is denoted $\mathcal{C} = \{1, \ldots, n-1\}$. For each visit $i \in \mathcal{C}$ a time window is defined as $[\alpha_i, \beta_i]$, where $\alpha_i \geq 0$ and $\beta_i \geq 0$ specify the earliest respectively latest 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 caretaker $k \in \mathcal{K}$, and we define $\mathcal{N}^k = \mathcal{C} \cup \{0^k, n^k\}$ as the set of all potential visits for caretaker k. The duty length for each caretaker $k \in \mathcal{K}$ is given by the time window $[\alpha_{0^k}, \beta_{0^k}] = [\alpha_{n^k}, \beta_{n^k}]$, i.e. caretaker $k \in \mathcal{K}$ 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^k_{ij}. The parameter depends on $k \in \mathcal{K}$ as the caretakers use different means of transportation. If it is not possible to travel directly between visits i and j for caretaker k, then $s^k_{ij} = \infty$. We define $s^k_{ii} = \infty$, $\forall k \in \mathcal{K}, \forall i \in \mathcal{N}^k$. The parameter s^k_{ij} includes the duration (service time) of visit i. Travelling between any two visits i and j gives rise to the costs c^k_{ij} dependent on the caretaker $k \in \mathcal{K}$. For any combination of $i \in \mathcal{C}$ and $k \in \mathcal{K}$ the parameter $\rho^k_i = 1$ if k can be assigned to visit i, $\rho^k_i = 0$ otherwise. Also, for any combination of $i \in \mathcal{C}$ and $k \in \mathcal{K}$ the preference parameter $\delta^k_i \in \mathbb{R}$ gives the cost for letting caretaker k handle visit i. A negative cost means that we would like caretaker k to handle visit i, whereas a positive cost means that we would



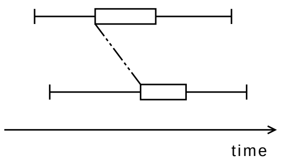
(a) Synchronisation.



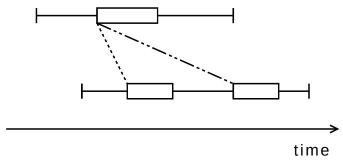
(b) Overlap.



(c) Minimum difference.



(d) Maximum difference.



(e) Min+max difference.

Figure D.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.

prefer not to let caretaker 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 D.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 D.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 D.1(a), where two visits are required to start at 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. (2009*d*), Table D.1 shows how to model all the temporal dependencies of Figure D.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	diff_{min}	N/A
(d)	Maximum difference	N/A	$-\text{diff}_{\text{max}}$
(e)	Minimum+maximum difference	diff_{min}	$-\text{diff}_{\text{max}}$

Table D.1: Values for p_{ij} for the five temporal dependencies of Figure D.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.

D.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 $t_i^k \in \mathbb{Z}_+$ and the binary coverage variables y_i. $x_{ij}^k = 1$ if caretaker $k \in \mathcal{K}$ 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 caretaker $k \in \mathcal{K}$ starts handling visit $i \in \mathcal{N}^k$. $t_i^k = 0$ if caretaker k is not assigned to visit i. $y_i = 1$ if visit $i \in \mathcal{C}$ is not covered by any caretaker 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).

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

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

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

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

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

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

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

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

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

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

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

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

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

The objective (D.1) is multi-criteria. Often, minimising uncovered visits (the third term) is prioritised over maximising caretaker-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 (D.2) ensure that each visit is covered exactly once or left uncovered, and caretakers can only handle allowed visits (D.3). Constraints (D.4)–(D.6) ensure that the caretakers begin at the start visit, end at the end visit, and that routes are not segmented. Constraints (D.7) and (D.8) control that time windows are respected. Furthermore, Constraints (D.7) set $t_i^k = 0$ when k does not visit i. Travelling distances are respected due to Constraints (D.9). Constraints (D.10) are the generalised precedence constraints. The y-variable terms ensure that generalised precedence constraints are respected even when visits are cancelled. Finally, Constraints (D.11)–(D.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 caretaker 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 caretaker 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.

D.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 caretaker 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 caretaker. Given a set of caretakers $\mathcal{K}$, each caretaker must choose a schedule from the set $\mathcal{R}^k$, which is the set of potential schedules for caretaker k. Together, the schedules must cover as many visits as possible from the set $\mathcal{C}$.

A feasible schedule r for a caretaker $k \in \mathcal{K}$ is defined as a route starting at 0^k and ending at n^k and respecting all constraints in the IP formulation from Section D.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 caretaker $k \in \mathcal{K}$ 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 caretaker k and $a_{ir}^k = 0$ otherwise. Moreover, t_{ir}^k is the start time of visit i in schedule r for caretaker k, if i is included in r for k. If i is not included in r for k, $t_{ir}^k = 0$.

D.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 caretaker $k \in \mathcal{K}$, 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.

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

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

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

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

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

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

The total costs of the schedules plus the costs of leaving visits uncovered are minimised (D.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 (D.15) ensure that all visits are either included in exactly one schedule or considered uncovered. One schedule must be assigned to each caretaker (D.16), and the generalised precedence constraints must be respected (D.17). Again, the Λ-variable terms in Constraints (D.17) ensure that precedence constraints are respected even for uncovered visits. Integrality of the decision variables is ensured by Constraints (D.18) and (D.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 (D.17) are relaxed, as we thereby have no constraints interconnecting the starting times in the schedules of different caretakers. This gives a simpler pricing problem, which will be explained further in Section D.3.2. The two relaxed constraints will be handled in the branching.

As the number of feasible schedules for each caretaker is enormous, the set $\mathcal{R}^k$ of schedules for caretaker $k \in \mathcal{K}$ 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 (D.15) and (D.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.

D.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 caretakers 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 (1994). The pricing problem for a caretaker $k \in \mathcal{K}$ 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.

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

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

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

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

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

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

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

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

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

Here, we have introduced the decision variables x_{ij} and t_i which are the same as in (D.1)–(D.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 (D.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 van den Akker et al. (2006) and in Dohn et al. (2009*d*) 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 van den 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. (2009*d*) show that in many cases it is as good to relax the generalised precedence constraints, as to keep them in the master problem.

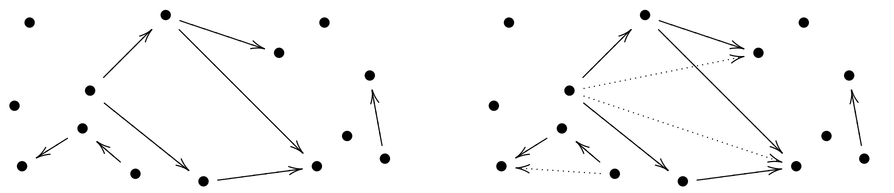
We solve the pricing problem with a labelling algorithm built on ideas from Chabrier (2006) and Feillet et al. (2004).

D.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.

D.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 D.2(a). From the existing generalised precedence constraints, additional *derived generalised prece-*


(a) Example of a temporal dependency graph.

(b) The same graph expanded with derived arcs.

Figure D.2: A temporal dependency graph.

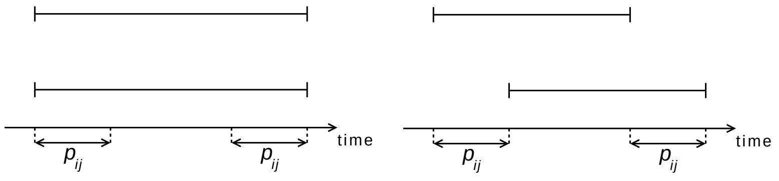
dence constraints can be found. In every subgraph with three or more nodes, we look for triples $i, j, k \in \mathcal{C}$ where $(i, j), (j, k) \in \mathcal{P}$ with $i \neq j, j \neq k, k \neq i$. If $(i, k) \notin \mathcal{P}$, then the pair is added to $\mathcal{P}$ with $p_{ik} = p_{ij} + p_{jk}$. 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 D.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 D.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).

D.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



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

Figure D.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).

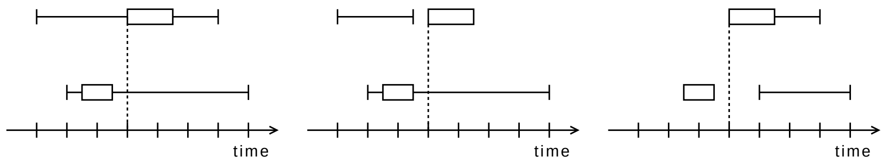
which

$$t_{i,r_1}^{k_1} + p_{ij} \nleq t_{j,r_2}^{k_2} \ .$$

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 $[\alpha_i, t_{\text{split}} - 1]$, where t_{split} is the split time. 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 $t_{j,r_2}^{k_2} - p_{ij} + 1 \leq t_{\text{split}} \leq t_{i,r_1}^{k_1}$. 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. (2009*d*), 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 D.4.

D.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_{ir}^k for a given caretaker $k \in \mathcal{K}$, 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 D.5. For clarity, we only show ones of the constraint matrix and introduce $m = |\mathcal{K}|$ and $\bar{n} = n - 1$.



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

Figure D.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.

$$
\begin{array}{c|ccc|c|ccc|ccc|}
 & \lambda_1^{k_1} & \cdots & \lambda_{|\mathcal{R}'^{k_1}|}^{k_1} & \cdots & \lambda_1^{k_m} & \cdots & \lambda_{|\mathcal{R}'^{k_m}|}^{k_m} & \Lambda_1 & \cdots & \Lambda_{\bar{n}} \\
\hline
1 & & & & & & & & 1 & & \\
\vdots & & A^{k_1} & & \cdots & & A^{k_m} & & & \ddots & \\
\bar{n} & & & & & & & & & & 1 \\
\hline
k_1 & 1 & \cdots & 1 & & & & & & & \\
\vdots & & & & \ddots & & & & & & \\
k_m & & & & & 1 & \cdots & 1 & & & \\
\hline
\end{array}
$$

Figure D.5: Constraint matrix for the RMP.

We observe that the RMP has strong integer properties due to the generalised upper bound constraints (D.16) for each caretaker, see e.g. Rezanova and Ryan (2010) for further details and references. That is, for all caretakers $k \in \mathcal{K}$, their submatrix of the constraint matrix is perfect. Consequently, fractionality in the LP solutions will never appear within one caretaker's block of schedules. Any fractions in the RMP can therefore only occur between blocks of columns, belonging to different caretakers. Hence, if the LP solution is fractional, it is because two or more caretakers compete for one or more visits in their schedules. Let $i \in \mathcal{C}$ denote a visit for which caretaker $k \in \mathcal{K}$ is competing with one or more other caretakers. Since the visit can only be handled by one caretaker, 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-caretaker pair (i, k) of a visit $i \in \mathcal{C}$ and a caretaker $k \in \mathcal{K}$ 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. This can be shown by a contradiction argument, see Dohn and Kolind (2006).

Whenever the sum S_i^k is fractional for two or more visit-caretaker 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 caretaker 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 $(i^*, k^*) = \arg\min_{(i,k) \in \mathcal{C} \times \mathcal{K} : 0 < S_i^k < 1} \left| S_i^k - \frac{1}{2} \right|$.

D.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 caretaker-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 caretaker $k \in \mathcal{K}$ 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 D.2.1), a caretaker k will only handle visits i where δ_i^k is favourable. Therefore, we have devised three ways of clustering visits for a caretaker according to the preference parameter δ_i^k, thereby effectively reducing the network sizes for each caretaker. 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 caretaker. 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 caretaker 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 D.4.3. If the caretakers 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 caretaker are ordered non-decreasingly. Hereafter, the scheme iterates over the caretakers in a round-robin fashion and puts the most profitable visit in the caretaker's cluster (if it is not already put in another caretaker's cluster). Suppose visit j is already in $\bar{\mathcal{C}}^k$ for caretaker 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 $-s_{ji}^k < p_{ij} \wedge -s_{ij}^k < p_{ji}$.

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.

D.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.

D.5.2 Removal of idle arcs

We will here present another method to reduce the network sizes. The time where the caretaker is neither visiting a citizen nor travelling is called *idle time.* This is time where the caretaker 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 $\phi_{ij}^k = \alpha_j - (\beta_i + s_{ij}^k)$ 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.

D.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 caretakers and visits and original travelling times. The new generalised precedence constraint sets are based on the five types of temporal dependencies from Figure D.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. (2009*d*). Characteristics for these test instances can be seen in Table D.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 caretakers, 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 caretakers in that instance have the same duty hours. Characteristics for these test instances can also be seen in Table D.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 D.2: Characteristics for the test instances. $|\mathcal{K}|$ is number of caretakers, $|\mathcal{C}|$ is number of visits and $|\mathcal{P}|$ is number of generalised precedence constraints.

D.7 Computational results

The aim in this section is to compare the different visit clustering techniques presented in Section D.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, caretaker-visit preferences, and total travel costs. The weights of the objective function are set as suggested in Section D.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 caretakers 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 caretaker 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 D.3-D.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 caretaker k where $\delta_i^k < 0$ for all visits i in the cluster, as described in Section D.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

	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
	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,ll1]	N/A	100	87	70	106	105	104	48	45	8	5	5	4	N/A
[hh,ll1]-td0-A	N/A	100	246	99	322	1395	1384	1872	1770	77	35	66	36	N/A
[hh,ll1]-td0-B	N/A	100	44	102	43	540	1362	160	1235	30	23	30	18	N/A
[hh,ll1]-td0-C	N/A	100	45	72	88	266	645	168	971	63	41	56	42	N/A
[hh,ll1]-td1-A	N/A	100	100	74	74	138	155	110	235	47	45	43	38	N/A
[hh,ll1]-td1-B	N/A	100	78	89	75	161	126	123	133	40	34	38	33	N/A
[hh,ll1]-td1-C	N/A	100	89	97	109	101	123	89	104	56	37	48	29	N/A
[hh,ll1]-td2-A	N/A	100	97	89	80	29	29	39	36	5	5	5	6	N/A
[hh,ll1]-td2-B	N/A	100	2788	216	253	121	124	100	91	15	15	15	13	N/A
[hh,ll1]-td2-C	N/A	100	115	107	848	211	173	140	133	81	52	75	58	N/A
[hh,ll1]-td3-A	N/A	100	43	91	90	510	440	258	247	5	4	6	5	N/A
[hh,ll1]-td3-B	N/A	100	58	201	177	15	17	17	16	3	2	2	1	N/A
[hh,ll1]-td3-C	N/A	100	104	104	104	11	8	8	5	1	1	2	1	N/A
[hh,ll1]-td4-A	N/A	100	61	50	176	103	106	71	64	18	11	19	12	N/A
[hh,ll1]-td4-B	N/A	100	207	67	119	351	351	106	90	8	5	11	7	N/A
[hh,ll1]-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 D.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 D.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.

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 D.3-D.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 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 caretakers. 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 caretaker'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 D.3-D.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 caretaker'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 D.5 shows detailed statistics for individual test runs. The test runs shown here are chosen, because they are representative for the numbers from

Tables D.3-D.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 D.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 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 caretakers. 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.

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

D.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 caretakers.

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 caretakers, 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 caretakers 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 caretakers, and an approach could be inspired by what Rezanova and Ryan (2010) do for train driver rescheduling.

References

Begur, S., D. Miller, and J. Weaver (1997). “An integrated spatial DSS for scheduling and routing home-health-care nurses”. In: *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”. Ed. by Louis-Martin Rousseau Michel Gendreau Gilles Pesant. In: *Computers and Operations Research* 33.10, pp. 2866–2890.

Bredström, D. and M. Rönnqvist (2007). *A branch and price algorithm for the combined vehicle routing and scheduling problem with synchronization constraints.* Tech. rep. 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”. In: *European Journal of Operational Research* 191.1, pp. 19–31.

Chabrier, A. (2006). “Vehicle Routing Problem with elementary shortest path based column generation”. Ed. by Louis-Martin Rousseau Michel Gendreau Gilles Pesant. In: *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 Paolo Toth and Daniele Vigo. Society for Industrial and Applied Mathematics. Chap. 7, pp. 176–213.

Dohn, A. and E. Kolind (2006). “A Practical Branch and Price Approach to the Crew Scheduling Problem with Time Windows”. MA thesis. Informatics and Mathematical Modelling, Technical University of Denmark.

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

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

Dror, M. (1994). “Note on the Complexity of the Shortest Path Models for Column Generation in VRPTW”. In: *Operations Research* 42.5, pp. 977–978.

Eveborn, P., P. Flisberg, and M. Rönnqvist (2006). “Laps Care—an operational system for staff planning of home care”. Ed. by J. Krarup L. Sakalauskas. In: *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”. In: *Networks* 44.3, pp. 216–229.

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

Garey, M. R. and D. S. 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”. In: *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”. In: *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”. In: *Column Generation*. Ed. by Guy Desaulniers, Jacques Desrosiers, and Marius M. Solomon. GERAD 25th anniversary series. New York: Springer. Chap. 3, pp. 67–98.

Lessel, C. R. (2007). “Ruteplanlægning i hjemmeplejen”. MA thesis. 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”. In: *Naval Research Logistics* 52.4, pp. 302–311.

Lim, A., B. Rodrigues, and L. Song (2004). “Manpower allocation with time windows”. In: *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”. In: *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”. Ed. by Jesper Larsen Jens Clausen Allan Larsen. In: *Computers and Operations Research* 37.5, pp. 845–856.

Ryan, D. M. and B. Foster (1981). “An integer programming approach to scheduling”. Ed. by A. Wren. In: *Computer Scheduling of Public Transport. Urban Passenger Vehicle and Crew Scheduling. Proceedings of an International Workshop*, pp. 269–280.

Thomsen, K. (2006). “Optimization on Home Care”. MA thesis. Informatics and Mathematical Modelling, Technical University of Denmark.

van den Akker, J., J. Hoogeveen, and J. van Kempen (2006). “Parallel machine scheduling through column generation: Minimax objective functions”. In: *Lecture Notes in Computer Science* 4168, pp. 648–659.

Appendix E

The Vehicle Routing Problem with Time Windows and Temporal Dependencies

Anders Dohn, Matias Sevel Rasmussen, and Jesper Larsen

Conditionally accepted for publication in: *Networks* (2010).

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.

[*]Conditionally accepted for publication in: *Networks* (2010).

[1]Department of Management Engineering, Technical University of Denmark, Produktionstorvet, 2800 Kongens Lyngby, Denmark.

E.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, 1960) of the mathematical model using column generation in a branch-and-cut-and-price framework. The method was first proposed by Desrochers et al. (1992). 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. (2009*b*) 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. (2008*c*) 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 van den 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 E.2, we present two valid compact formulations of VRPTWTD. Possible decompositions of the compact formulations are presented and compared in Section E.3. For the decomposed models, a tailored branching method is required, which is described in Section E.4. A set of test instances are introduced in Section E.5 and the test results for these are found in Section E.6. Finally, we conclude on our findings and discuss possible areas for future research in Section E.7.

E.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.

E.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 $s_{n+1,k} = \beta_0, \forall k \in \mathcal{V}$.

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

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

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

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

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

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

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

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

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

The objective is to minimize the total cost of all edges traveled (E.1). All customers must be visited by exactly one vehicle (E.2) and the route for each vehicle must respect the capacity of that vehicle (E.3). (E.4) and (E.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 (E.5). If a vehicle is set to travel between two customers, there has to be enough time between the two visits (E.7). Finally, we need to make sure that all time windows are respected (E.8). (E.8) also ensure that $s_{ik} = 0$ when vehicle k does not visit customer i. (E.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.

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

Constraint (E.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 E.1.

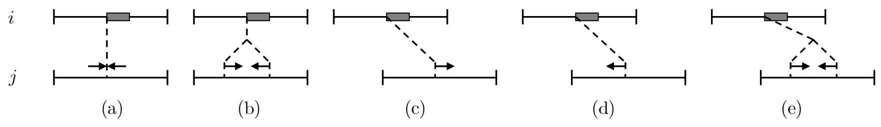


Figure E.1: Five kinds of temporal dependencies that are often encountered in practice. Each of the five subfigures shows the time windows of two customers i and j with a temporal dependency between them. Assuming some start time for customer i, the dashed line together with the arrows give the corresponding 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 E.1 using constraints (E.10). The correct values for δ_{ij} and δ_{ji} are listed in Table E.1.

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

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

E.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 $\mathcal{T}^{\delta}_{tij} = \{\alpha_0, \ldots, \min\{\beta_0, t + \delta_{ij} - 1\}\}$.

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

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

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

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

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

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

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

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

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

$$x_{ijkt}\in\{0,1\} \qquad \forall i,j\in\mathcal{N}, \forall k\in\mathcal{V}, \forall t\in\mathcal{T} \tag{E.20}$$

Constraints (E.11)–(E.16) are similar to Constraints (E.1)–(E.6), where we now sum over the time index as well. Constraints (E.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 (E.18). Constraints (E.18) are the equivalent of Constraints (E.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 (E.19) enforce the time windows and (E.20) are the integrality constraints.

E.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 (E.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.

E.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 (E.1)-(E.10).

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

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

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

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

The corresponding subproblem is that of generating negative reduced cost routes for the master problem (E.21)–(E.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 (E.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 (E.12) and (E.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 (E.11)-(E.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:

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

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

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

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

The obvious problem with the time-indexed restricted master problem (E.25)–(E.28) is the number of constraints of type (E.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 (E.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. (2009*b*) 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*.

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

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

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

Limited time-indexed formulation

In the time-indexed formulation (E.25)-(E.28), it is possible to include only a subset of Constraints (E.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 (E.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 E.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 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 (E.25)–(E.28) and prove that the solution is also feasible for Constraints (E.21)–(E.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 (E.27) with $t = \alpha_0$ is:

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

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

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

$$\Downarrow$$

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

$$\Downarrow$$

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

Summing Constraints (E.26), (E.35), and (E.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 (E.25)–(E.28), we have for $(i, j) \in \Delta$:

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

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

(E.25)–(E.28): $s_i^r = \sum_{t \in \mathcal{T}} t a_{it}^r$. The result in (E.40) proves that any feasible solution of (E.25)–(E.28) also respects (E.23). (E.22) is trivially respected as $a_i^r = \sum_{t' \in T} a_{it'}^r$, and hence (E.21)–(E.24) is a relaxation of (E.25)–(E.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 (E.21)–(E.24). In model (E.25)–(E.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 (E.21)–(E.24) but not in (E.25)–(E.28). This is verified by inspecting (E.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 (E.27) for $i = 1, j = 2, t = 1$:

$$\sum_{r \in \mathcal{R}'} (a_{11}^r \lambda_r + a_{12}^r \lambda_r + a_{13}^r \lambda_r + a_{14}^r \lambda_r) + \sum_{r \in \mathcal{R}'} (a_{21}^r \lambda_r + a_{22}^r \lambda_r) = 1 + 0.5 \nleq 1$$

$\square$

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.

E.3.2 Identifying violated cuts

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

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

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 $\sum_{r\in\mathcal{R}'}\sum_{t'=t,\dots,\beta_0} a^r_{it'}\lambda_r$ is non-increasing for increasing t. Correspondingly, the sum $\sum_{r\in\mathcal{R}'}\sum_{t'=\alpha_0,\dots,\min\{\beta_0,t+\delta_{ij}-1\}} a^r_{jt'}\lambda_r$ is non-decreasing. Constraints (E.27) are never violated for $t = \alpha_0$ as such violations are prevented by preprocessing the time windows, see Section E.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.

E.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 (E.3)–(E.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:

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

$\pi_i, i \in \mathcal{N}$, are the dual variables of Constraints (E.30) of the VRPTW master problem. Dror (1994) 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. (1992) 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 (E.22), π_i, and additionally the dual variables of Constraints (E.23), σ_{ij}, and the objective function becomes:

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

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:

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

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

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

where π_i are the dual variables of Constraints (E.26) and ρ_{ijt} are the non-positive dual variables of Constraints (E.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 (E.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 (E.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 (E.43b) or (E.43c). (E.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 (E.43b) is obtained by the smallest possible value of t as $\sum\limits_{t'=\alpha_0,\ldots,t} \rho_{ijt'}$ 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 (E.43c) will decrease as t is increased (for given i,j) and only when terms with $\rho_{jit'} < 0$ are excluded. The value of (E.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.

E.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. (2009*b*), 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.

E.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 E.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.

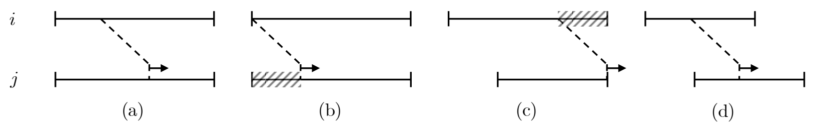


Figure E.2: Time window reductions. (a) The original time windows. (b) Using the generalized precedence constraint, the first part of the time window of customer j is removed. (c) In a similar way, the last part of the time window of customer i is removed. (d) The time windows after the reduction.

E.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 E.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 E.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 for-

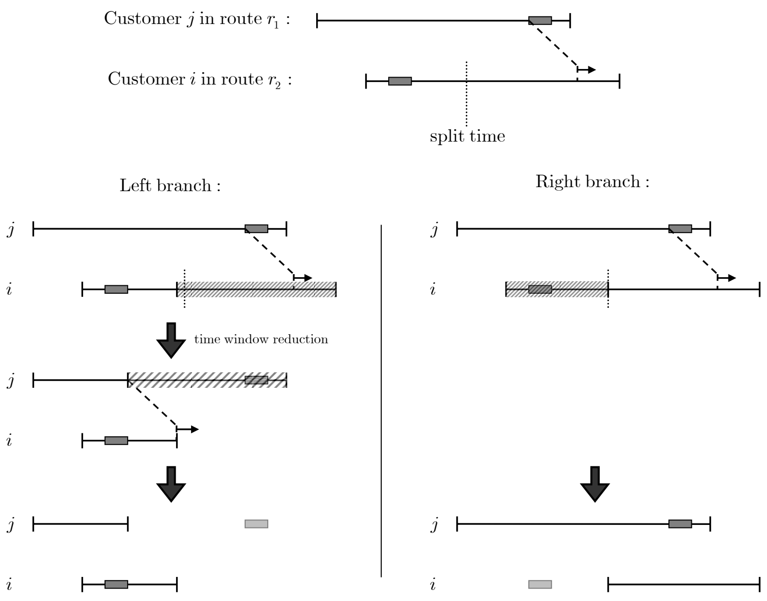


Figure E.3: Branching to avoid a violation of a precedence constraint.

merly 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 E.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

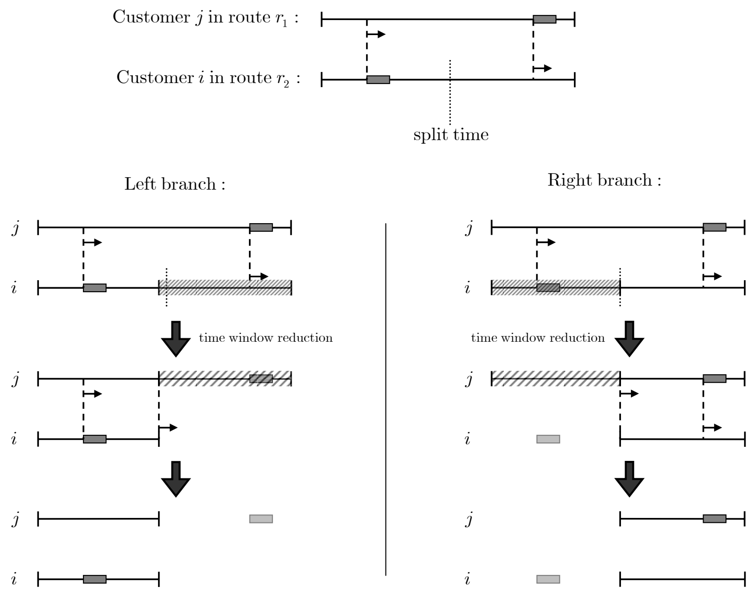


Figure E.4: Branching on a generalized precedence constraint of a synchronization constraint.

subject. When using strong branching (see e.g. Achterberg et al., 2005) a few candidates are chosen for further probing. In any case, we need to specify a 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 E.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 E.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

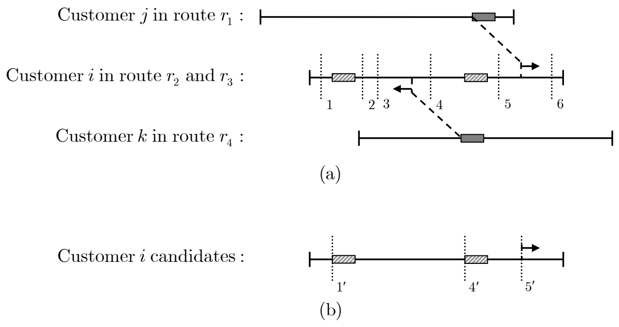

Customer j in route r_1 :
Customer i in route r_2 and r_3 :
Customer k in route r_4 :
1
2
3
4
5
6
(a)
Customer i candidates :
1′
4′
5′
(b)

Figure E.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 E.2: Effect of the branching candidates of Figure E.5.

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

In Figure E.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 E.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 E.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 schedule 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 (1992).

E.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 (1987). 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 E.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. diff_{min} and 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 E.1.

6. Reduce time windows as explained in Section E.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.

E.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

		Time-indexed formulation (all cuts)		Time-indexed formulation (limited)		Relaxed formulation	
	Instances in total	Solved in the root	Solved before timeout	Solved in the root	Solved before timeout	Solved in the root	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 E.3: Overview of the test results for instances with 25 customers. Time out is one hour.

experiments of this paper do not include the direct formulation.

To give an idea of the overall performance of the remaining three approaches, the test results are summarized in Table E.3. As described in Section E.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 E.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 E.4, the results are similar to those of Table E.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.

	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 E.4: Overview of the test results for instances with 50 customers. Time out is one hour.

In the remainder of this section, we have chosen to focus on two of the 25 customer instance sets, namely instances with only synchronization relations and a set with a random mix of the five temporal dependencies of Table E.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.

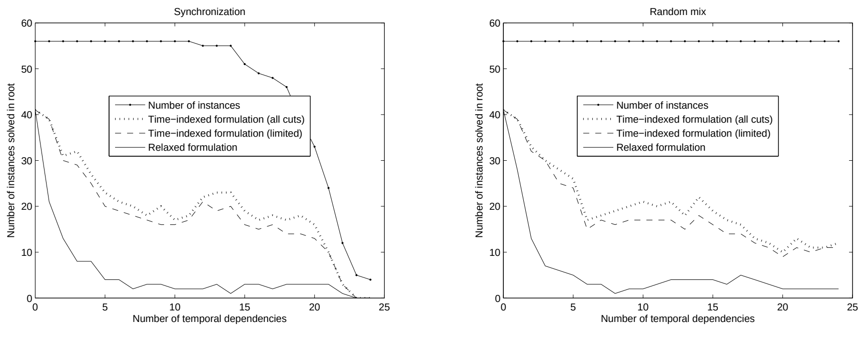


Figure E.6: Number of instances solved at the root node of the branch-and-bound tree.

In Figure E.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 E.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 E.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.

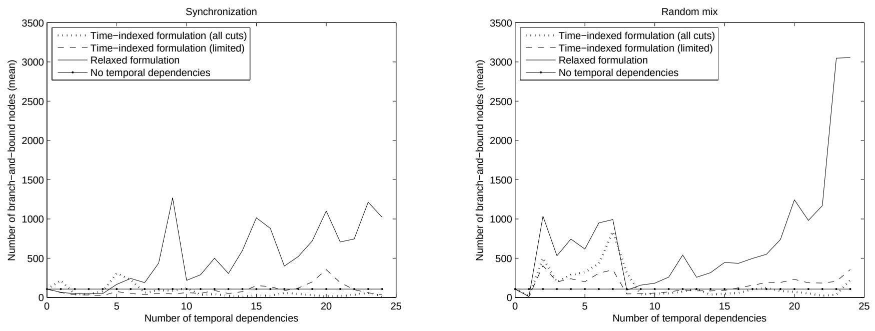


Figure E.7: Number of nodes in the branch-and-bound tree.

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 E.8.

Looking at Figure E.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

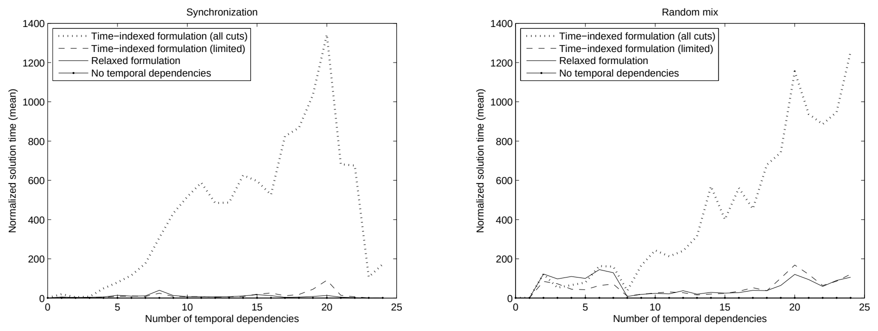


Figure E.8: Normalized solution time (mean).

approach is faster than the relaxed formulation and vice versa. The results are summarized in Figure E.9. Looking at the instances individually, the limited time-indexed approach seems a little better.

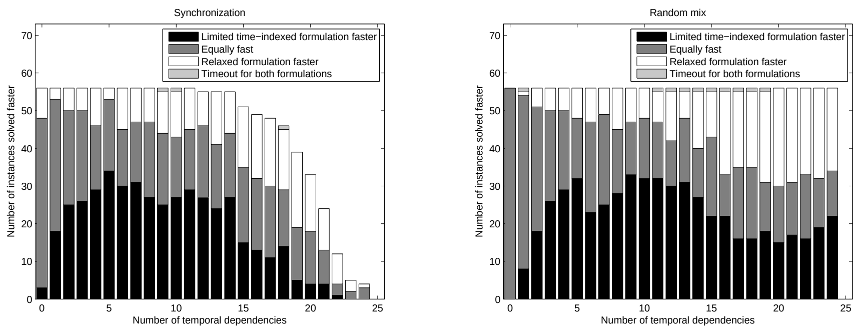


Figure E.9: Number of tests where one of the approaches is faster than the other. The two approaches are considered equally fast if they are within 20% of each other.

Finally, we look at the distribution of time spent in the algorithm. This is illustrated in Figure E.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 E.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%.

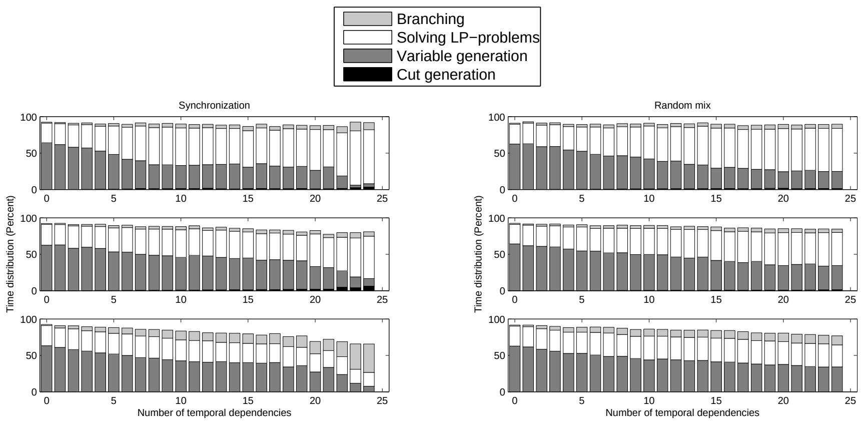


Figure E.10: Distribution of solution time for the time-indexed model (top), the limited time-indexed model (middle), and the relaxed model (bottom).

From Figure E.10, we observe that for the time-indexed formulation, the portion of time spent in the LP-solver increases as problems with more temporal dependencies 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.

E.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 (2005). "Branching Rules Revisited". In: *Operations Research Letters* 33.1, pp. 42–54.

Bélanger, N., G. Desaulniers, F. Soumis, and J. Desrosiers (2006). "Periodic airline fleet assignment with time windows, spacing constraints, and time dependent revenues". In: *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." In: *INFORMS Journal on Computing* 20.1, p. 133.

Bredström, D. and M. Rönnqvist (2007). *A branch and price algorithm for the combined vehicle routing and scheduling problem with synchronization constraints.* Tech. rep. 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". In: *European Journal of Operational Research* 191.1, pp. 19–31.

Chabrier, A. (2006). "Vehicle Routing Problem with elementary shortest path based column generation". Ed. by Louis-Martin Rousseau Michel Gendreau Gilles Pesant. In: *Computers and Operations Research* 33.10, pp. 2972–2990.

Christiansen, M. and B. Nygreen (2005). "Robust Inventory Ship Routing by Column Generation". In: Desaulniers G., Desrosiers J., Solomon M.M.: Column Generation, Springer, New York. 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 Jacques Desrosiers Guy Desaulniers and Marius M. Solomon. Springer. Chap. 4, pp. 99–129.

Dantzig, G. B. and P. Wolfe (1960). "Decomposition Principle for Linear Programs". In: *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”. In: *Transportation Science* 42.3, p. 387.

Desrochers, M., J. Desrosiers, and M. Solomon (1992). “A new optimization algorithm for the vehicle routing problem with time windows”. In: *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”. In: *Computers and Operations Research* 35.9, pp. 3034–3048.

Dohn, A., M. S. Rasmussen, T. Justesen, and J. Larsen (2008*c*). “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 (2009*b*). “The manpower allocation problem with time windows and job-teaming constraints: A branch-and-price approach”. In: *Computers and Operations Research* 36.4, pp. 1145–1157.

Dror, M. (1994). “Note on the Complexity of the Shortest Path Models for Column Generation in VRPTW”. In: *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”. In: *Networks* 44.3, pp. 216–229.

Fügenschuh, A. (2006). “The vehicle routing problem with coupled time windows”. In: *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”. In: *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”. In: *Networks* 31.3, pp. 193–204.

Ioachim, I., J. Desrosiers, F. Soumis, and N. Bélanger (1999). “Fleet assignment and routing with schedule synchronization constraints”. In: *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”. In: *Operations Research* 56.2, pp. 497–511.

Justesen, T. and M. S. Rasmussen (2008). “The Home Care Crew Scheduling Problem”. MA thesis. Department of Informatics and Mathematical Modeling, Technical University of Denmark.

Kallehauge, B., J. Larsen, O. B. Madsen, and M. Solomon (2005). “The Vehicle Routing Problem with Time Windows”. In: *Column Generation*. Ed. by Guy Desaulniers, Jacques Desrosiers, and Marius 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”. In: *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”. In: *Transportation Science* 33.1, pp. 101–116.

Lesaint, D., N. Azarmi, R. Laithwaite, and P. Walker (1998). “Engineering Dynamic Scheduler for Work Manager”. In: *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”. In: *Naval Research Logistics* 52.4, pp. 302–311.

Lim, A., B. Rodrigues, and L. Song (2004). “Manpower allocation with time windows”. In: *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”. In: *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”. In: *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. (1992). “The Solution of Massive Generalized Set Partitioning Problems in Aircrew Rostering”. In: *Journal of the Operational Research Society* 43.5, pp. 459–467.

Savelsbergh, M. (1985). “Local search in routing problems with time windows”. In: *Annals of Operations Research* 4.1-4, pp. 285–305.

Solomon, M. M. (1987). “Algorithms for the Vehicle Routing and Scheduling Problems with Time Window Constraints”. In: *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”. In: *INFORMS Journal on Computing* 12.2, pp. 111–124.

van den Akker, J., J. Hoogeveen, and J. van Kempen (2006). “Parallel machine scheduling through column generation: Minimax objective functions”. In: *Lecture Notes in Computer Science* 4168, pp. 648–659.

Appendix F

Optimizing the Slab Yard Planning and Crane Scheduling Problem using a Two-Stage Heuristic

Anders Dohn and Jens Clausen

Published in:
International Journal of Production Research (2010) 48.15, pp. 4585-4608.

Optimizing the Slab Yard Planning and Crane Scheduling Problem using a Two-Stage Heuristic*

Anders Dohn[1] and Jens Clausen[1]

In this paper, we present the Slab Yard Planning and Crane Scheduling Problem. The problem has its origin in steel production facilities with a large throughput. A slab yard is used as a buffer for slabs that are needed in the upcoming production. Slabs are transported by cranes and the problem considered here is concerned with the generation of schedules for these cranes. The problem is decomposed and modeled in two parts, namely a planning problem and a scheduling problem. In the planning problem, a set of crane operations is created to take the yard from its current state to a desired goal state. In the scheduling problem, an exact schedule for the cranes is generated, where each operation is assigned to a crane and is given a specific time of initiation. For both models, a thorough description of the modeling details is given along with a specification of objective criteria. Preliminary tests are run on a generic setup with simulated data. The test results are very promising. The production delays are reduced significantly in the new solutions compared to the corresponding delays observed in a simulation of manual planning.

Keywords: large-scale scheduling; stacker crane problem; crane/hoist scheduling; generalized precedence constraints; schedule visualization; simulation.

F.1 Introduction

The Slab Yard Planning and Crane Scheduling Problem is a complex optimization problem, combining planning and scheduling in an effort to generate feasible schedules for a number of interacting cranes. The problem instances originate from real-world data. Costs and constraints have been defined in cooperation with the industry. The industrial problem instances are of a large size and

*Published in: *International Journal of Production Research* (2010) 48.15, pp. 4585-4608.
[1]Department of Management Engineering, Technical University of Denmark, Richard Petersens Plads, 2800 Kongens Lyngby, Denmark.

therefore it is important to create a solution method that can make superior heuristic choices in little time.

The problem here is from a steel hot rolling mill. A large number of slabs arrive by train at a slab yard, where they are stored until transported to the hot rolling mill by a roller table. The slabs need to be transported from the train to the yard and later from the yard to the roller table in the correct order and at specific points in time. Each slab has distinct properties, so we need to consider each slab as being unique.

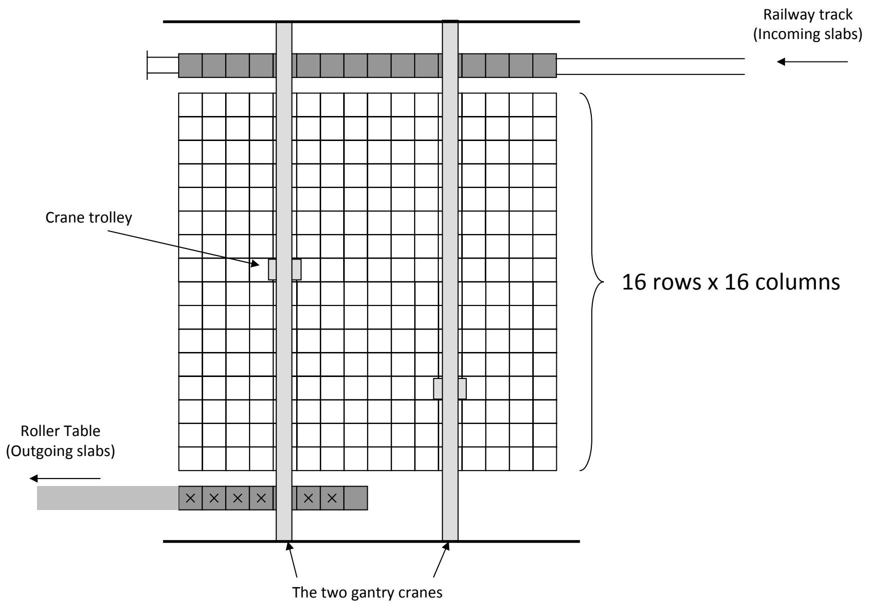


Figure F.1: Overview of the slab yard.

Figure F.1 shows an overview of the slab yard. The two gantry cranes are used to move slabs from one stack to another. As seen in the figure, both the train and the roller table can be modeled as special sets of temporary stacks. The generic slab yard under consideration in this paper consists of 16 $\times$ 16 stacks where each stack is a number of slabs on top of each other. Each crane can only carry one slab at a time and therefore only the top slab of a stack can be moved. The cranes operate in two directions. Horizontally, they run on a pair of shared tracks and can therefore never pass each other in this direction. Vertically, they operate by a trolley attached to the crane, which can move freely from top to bottom.

The problem is approached in a two-stage planning/scheduling conception. The planning problem of the yard is of an abstract nature. Desired locations for slabs are stated without specifying times of slab movement or crane allocations. For a pre-specified time horizon, a desired end state is formulated, i.e. the end positions of the slabs in the yard are determined. We also generate the operations that need to be carried out in order to arrive at this state. The aim of the crane scheduling problem is to concretize the decisions of the planning problem. Operations are allocated to cranes and all operations are sequenced and positioned in time. The final scheduling solution is directly applicable in practice.

The Slab Yard Planning and Crane Scheduling Problem is considered in a similar context by Hansen (2003). The problem is from a shipyard where ships are constructed by welding together steel plates. The plates are stored in stacks in a plate storage facility and are moved by two gantry cranes sharing tracks. A simulator and a control system are developed and implemented in a system to be used as decision support for the crane operators. Another similar problem is presented by Gambardella et al. (2001) (based on the work in Zaffalon et al., 1998) where containers are transported by cranes in a container terminal.

An immediate advantage of the two-stage approach is that the planning problem and the scheduling problem individually have received considerable attention in the literature.

The literature relating to The Slab Yard Planning Problem is mainly in other areas of slab yard planning and in container stacking. Tang et al. (2002) describe a steel rolling mill where slabs need to be transported from a slab yard according to a scheduled rolling sequence. The article builds on the initial work by the same authors (Tang et al., 2001). The layout of the slab yard is different from ours and the cranes are located so that they never collide. Another difference is that for each batch, several candidates exist among the slabs, and therefore the objective is to minimize the cost by choosing the right slabs among the candidates. Singh et al. (2004) address the same problem and solve it using an improved Parallel Genetic Algorithm. König et al. (2007) investigate a similar problem from storage planning of steel slabs in integrated steel production. The problem formulation in the article is kept at a general level to make the model versatile. The stacking problem is considered alone, thereby disregarding the crane schedules. The authors present a greedy construction heuristic and by a linear programming relaxation of a mixed integer formulation of the problem, they are able to quantify the quality of their solutions.

A problem in container stacking with many similarities to the slab stacking problem is described by Dekker et al. (2006). A significant difference is that the maximum height of container stacks is 3, whereas the corresponding number in slab stacks is usually considerably larger than this. A number of stacking policies are investigated by means of simulation and in this sense, the work of Dekker et al. resembles the work by Hansen (2003) in a container stacking context. Kim and Bae (1998) describe a container stacking problem where a current yard layout is to be reorganized to a new specific layout. The problem is to convert the current bay layout to the desired new layout by moving the fewest possible containers. The problem is decomposed into three sub-problems, namely a bay-matching, a move-planning, and a task-sequencing stage, where the latter two are similar to the two stages that we introduce for The Slab Yard Planning and Crane Scheduling Problem. Kim et al. (2000) consider a similar container stacking problem. See Steenken et al. (2004) for a recent review of literature on container stacking.

The Crane Scheduling problem considered here is an example of a Stacker Crane Problem (Frederickson et al., 1978) with time windows and multiple cranes. Parallel crane/hoist scheduling has been thoroughly treated in production of electronics, especially in printed circuit board production. In circuit board production, the hoists are used to move products between tanks, where the products are given various chemical treatments. Leung and Zhang (2003) introduce the first mixed-integer programming formulation for finding optimal cyclic schedules for printed circuit board lines with multiple hoists on a shared track, where the processing sequence may be different from the location sequence of the tanks. The solution method itself is not transferable, but several of the elements in the modeling phase are very relevant to the Crane Scheduling problem of the slab yard. These include the formulation of collision avoidance constraints. Collision avoidance constraints are also described in a dynamic hoist scheduling problem by Lamothe et al. (1996) and in a fixed sequence production by Che and Chu (2004) and Leung et al. (2004).

As it becomes apparent in the following sections, in the present scheduling problem we are able to abstract from the practical context of the problem and consider the problem as a parallel scheduling problem with sequence-dependent setup times. Zhu and Wilhelm (2006) present a recent literature review for this type of scheduling problem.

This article is arranged as follows. In Section F.2 the problem is described and the most important properties are extracted. The solution method is divided into two stages, first solving a planning problem and subsequently a scheduling problem. The two problems and their models are described in Section F.3 and Section F.4, respectively. The solution method itself is described in Section F.5 and test results are presented in Section F.6. Finally, conclusions and areas for future work are outlined in Section F.7.

F.2 Problem Description

The slab yard is modeled as a large set of slab stacks. As was shown in Figure F.1, train wagons and the roller table are also modeled as special stacks. The train is only at the yard for a certain amount of time and hence all slabs must be moved away from the wagons within a specific time window. In principle, we have access to the roller table in multiple positions as shown on Figure F.1 (shown as 8 stacks wide). The order of the slabs on the roller table is essential and to ensure that the sequence of slabs leaving the roller table follows the order in which they were brought there, we only allow the cranes to bring slabs to the right-most of the roller table stacks. Further, as there is room for at most 8 slabs on the roller table, we have to wait whenever the roller table is full. As time goes, the slabs are removed from the roller table in a first-in, first-out manner. For each slab to be moved from the yard in a near future, we have the production time, *Aim Leave Time* (*ALT*). By looking forward 8 slabs in the sequence, we know when there will be free room for a new slab on the roller table, and this gives us the *Earliest Leave Time* (*ELT*). Hence, we have a time window for moving the specific slab to the roller table. Slabs which are not a part of the immediately following production instead have an *Estimated Leave Time* (*EST*).

Each move consists of lift time, transportation time, and drop time. We assume that the cranes move at constant velocity. Transportation time is equal to the maximum of the vertical and the horizontal transportation time, as the cranes are able to move horizontally at the same time as the crane trolley is moving in a vertical direction.

F.2.1 Objective

The objective of the schedule is to minimize maximum tardiness (delay). The reason is as follows. Take all slabs leaving the slab yard within the scheduling horizon. Whenever a slab is not moved to the roller table before its Aim Leave Time, it causes a delay in the production. The production is not immediately able to catch up on this delay and therefore subsequent slabs are needed later in the production than we initially anticipated. Production is further delayed, only if subsequent slabs are delayed even more. Hence, the most delayed slab determines the quality of the solution.

A feasible schedule consists of a sequence of *operations* with crane allocation and time specification, i.e.: Crane X picks up slab Y (at its current location) at time T and moves it to position Z. Naturally, none of the operations are allowed to conflict with other operations, neither within the schedule of one crane nor between the two cranes.

F.2.2 A Simple Example

Before describing the details of the decomposition, we introduce an illustrative example to clarify the concepts and ideas that are introduced in the following sections.

Example *[A Simple Example]*

An overview of a small yard is shown in Figure F.2. The example is similar to the one shown in Figure F.1, only significantly smaller.

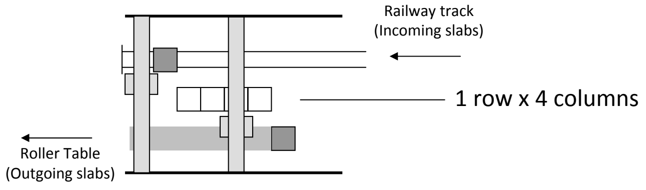


Figure F.2: Overview of a very simple slab yard used in the example.

In this example, we have a scheduling horizon of $[0, 22]$. *A side view of the initial yard state is shown in Figure F.3. Note that the vertical dimension of Figure F.2 is not visible in the figure. However, in the figure, we are able to illustrate the exact composition of each stack. The yard consists of a single arrival stack,* T_{ar1}*, four stacks in the main yard,* $T_1, ..., T_4$*, and one exit stack,* T_{exit}*. In the yard are 14 slabs,* $S_1, ..., S_{14}$*.*

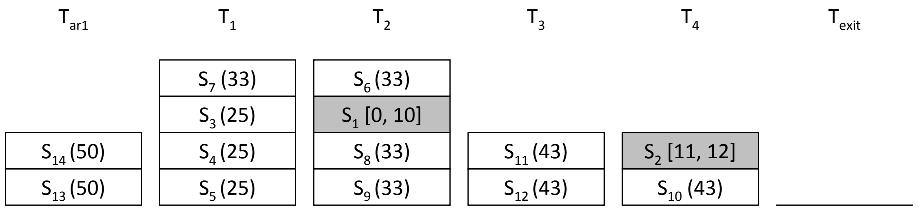


Figure F.3: Slab Yard Crane Scheduling Problem: Side-view of the slab yard of the example. Gray slabs are slabs that must leave the yard during the scheduling horizon. Leaving slabs (gray): [ELT, ALT]. Non-leaving slabs (white): (EST)

F.3 The Slab Yard Planning Problem

In the planning stage of the algorithm, we generate a plan that takes us from the current state of the yard to a final state for the horizon. In the final state, all slabs with a deadline within the horizon are brought to the roller table. At the same time, the plan should leave the yard in the best possible condition for subsequent planning periods.

To arrive at a feasible and superior plan within reasonable computational time, the idea is to relax a number of the original constraints in the planning stage. Whatever is relaxed here is fixed in the scheduling stage so that the final solution is always fully descriptive.

F.3.1 Operations

In the planning stage, we are going to consider a solution as defined by a number of successive operations. An operation contains the following information:

Slab	The slab to be transported.
Destination	The stack where the slab is put on top.
Priority	How important is it to include this operation in the final schedule.

A solution to the planning problem consists of a sequence of operations. Many operations are related directly to slabs which are moved to the roller table. Such operations are compulsory and hence have a priority of ∞. As is described in Section F.4.3, some operations are optional, however, and the priorities give an ordering of their importance.

For a planning solution to be feasible, we require the following:

- All slabs with a deadline within the scheduling horizon are transported to the exit stack in the correct order.
- All incoming slabs (i.e. slabs in the train wagon stacks) must be moved to permanent stacks.
- All operations must be valid in the sequence. Only slabs on top of a stack may be moved and only to stacks where the maximum stack height has not been reached.

The two first criteria are easy to verify, when we know the set of incoming slabs and the set of outgoing slabs. The third criterion can be verified by updating a yard state as the sequence of operations is processed.

F.3.2 Assessment Criteria

To assess the quality of a plan, we introduce a number of objectives. The following properties characterize a good solution. The two first are directly concerned with the plan, where the two last evaluate the end state of the yard.

- The number of operations is low.
- Operations do not span too far vertically. Even though the operations are not allocated to cranes yet, we would like a solution to accommodate such an allocation. Operations are faster and have less risk of conflicting when they span over as little vertical space as possible.
- Slabs that are to leave the yard soon (but after the current horizon) are close to the exit stack.
- The number of *false positions* is low. Slabs in false positions are slabs that are in the way of other slabs below them. A false position in our context is a stochastic term, as many slabs only have an estimated leave time. We introduce probabilities to approximate the number of false positions in the non-deterministic context. The leave time is represented by a Gaussian distribution. The probability of one slab leaving before another is found by inspecting the cumulative distribution of the difference between the two distributions. The probability of a false position is calculated as the sum over all slabs in the stack, where correlation between the distributions is also taken into account. Details are found in (Dohn and Clausen, 2008*c*).

The criteria stated above are quantified suitably for each individual real-world application.

Example ***[continued]***

A planning solution for the example is seen in Figure F.4. The solution consists of a sequence of operations. The end state of the storage is fully determined by the planning solution and is shown in Figure F.5.

o_1: ($S_6 \rightarrow T_3$)	o_2: ($S_1 \rightarrow T_{exit}$)	o_3: ($S_2 \rightarrow T_{exit}$)	o_4: ($S_{14} \rightarrow T_2$)	o_5: ($S_{13} \rightarrow T_2$)

Figure F.4: A solution to the planning problem of the example.

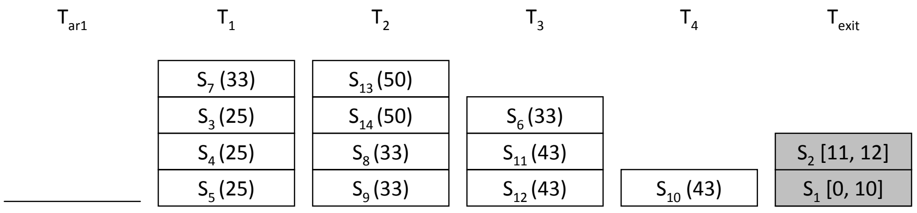


Figure F.5: End state for the solution of Figure F.4.

F.4 The Crane Scheduling Problem

From a solution to the planning problem, the goal is now to generate a complete and feasible schedule. First, the ordering of operations is relaxed to allow for parallel execution of operations. Most operations are locally independent of each other. These independencies are detected and only meaningful precedence constraints are kept for the scheduler. The crane scheduling problem is similar to a traditional parallel scheduling problem. We have a number of operations that we need to allocate to two cranes (machines). Between operations there are several temporal constraints. The anti-collision constraint is an important temporal constraint added by the fact that we have two cranes in operation. As the crane operation times are of a stochastic nature, we also need to introduce buffers. The buffers ensure that no crane collision occurs, even with disturbances in operation time. For major disturbances, the scheduling problem and possibly the whole planning may have to be resolved.

F.4.1 Precedence Relations

To ensure that the end state of the schedule is identical with the end state of the planning solution, we establish a number of precedence relations. Using the planning sequence as a starting point, we ensure that, whenever relevant, the order of the operations in the schedule stays the same as in the plan. There are four cases where reordering operations may change the state of the storage and may therefore cause direct or indirect infeasibility of the solution. In these four cases we do not allow reordering of the operations. See Figure F.6 for a graphical illustration of the four cases.

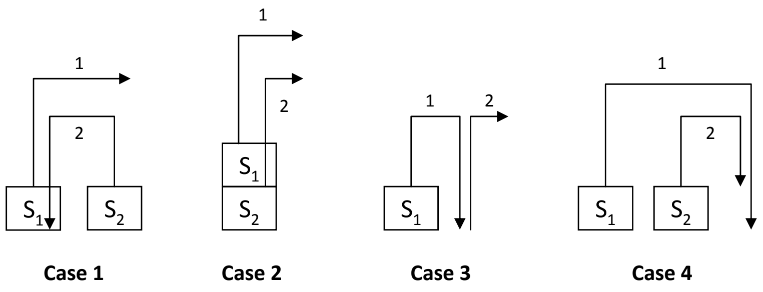


Figure F.6: Graphical description of the state preserving precedences.

Case 1 Moving slab S_2 to a stack from which slab S_1 was moved away from earlier. If the order of these two operations is changed, S_2 is going to block S_1 and the solution becomes infeasible.

Case 2 Moving slab S_1 and then slab S_2, where S_1 is on top of S_2. Again, changing the order of the two operations leads to infeasibility.

Case 3 Moving the same slab twice. If the order of such two moves is changed, the final destination of the slab may change. If the slab is moved again at a later time the final destination, however, remains unchanged.

Case 4 Two slabs S_1 and S_2 are moved to the same stack. If the order is changed it may lead to infeasibility later. If the two slabs are not moved later, the end state is altered, but the solution remains feasible.

Example *[continued]*

Going back to the example, we are now able to determine the precedence relations of the plan. Using the four cases depicted in Figure F.6 we arrive at the precedences of Figure F.7.

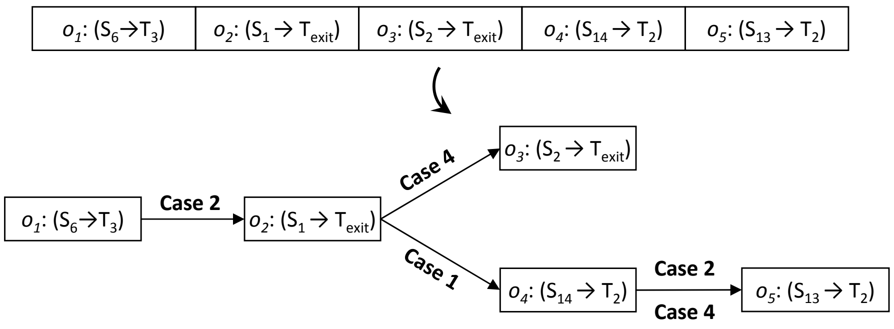


Figure F.7: Precedence relations for the plan of Figure F.4.

F.4.2 Temporal Constraints

The precedence constraints described in the previous section ensure that the end state of a parallel schedule is the same as the corresponding sequential plan. We still need to introduce temporal constraints to create a schedule that is feasible with respect to the individual movement of a crane and to create a schedule which is collision free.

For two operations i and j, we have four positions that have to be considered and where temporal constraints may have to be added correspondingly. The four positions are:

T_i^{orig}	Origin stack of operation i
T_i^{dest}	Destination stack of operation i
T_j^{orig}	Origin stack of operation j
T_j^{dest}	Destination stack of operation j

In the following, we say that i is before j, if i enters and leaves the conflict zone between the two operations, before j. When two operations are allocated to the same crane, the crane needs to complete the first operation before initiating the next. In this case, if i is before j, this means that operation i is completed before operation j is initiated. However, if the operations are allocated to different cranes, they may have a small conflict zone. Hence, even if operation i is before operation j, this does not necessarily mean that it is neither initiated first nor completed first. It only means that it will be the first of the two moves in any of their conflict positions. If two operations have no conflict zone, it is irrelevant whether i is considered to be before j or vice versa.

In the following, we calculate the required gap between two operations i and j, when i is before j. The gap is defined as the amount of time required from initiation of operation i to initiation of operation j. There are three different types of gaps depending on the crane allocation of operations i and j.

g^s_{ij}	Required gap when i and j are allocated to the same crane (s).
g^l_{ij}	Required gap when i is allocated to the left crane (l) and j to the right crane.
g^r_{ij}	Required gap when i is allocated to the right crane (r) and j to the left crane.

The following generalized precedence constraint is imposed: $t_i + g_{ij} \leq t_j$, where g_{ij} represents g^s_{ij}, g^l_{ij} or g^r_{ij} according to the situation. To calculate the gaps between operations, we need to introduce a number of parameters:

p_i	time required to pick up slab of operation i.
q_i	time required to drop off slab of operation i.
$m_{T_xT_y}$	time required to move a laden crane from stack T_x to stack T_y.
$e_{T_xT_y}$	time required to move an empty crane from stack T_x to stack T_y.
b	buffer time required between two cranes.

We assume that $m_{T_xT_y}$ and $e_{T_xT_y}$ are linear in the distance traveled. Both measures are independent of the crane involved. In the following we will use the assumption that the two cranes move at the same speed. We will also assume that a crane cannot move faster when laden than when it is empty.

Precedence relations are included in the generalized precedence constraints, so the values of g^s_{ij}, g^l_{ij} and g^r_{ij} hold all the information we need with respect to precedence constraints. If precedence relations disallow the execution of operation i before operation j, we set: $g^s_{ij} = g^l_{ij} = g^r_{ij} = \infty$.

When two operations are allocated to the same crane, we need to make sure that there is sufficient time to finish the first operation and to move to the start position of the second operation. Consequently, we get:

$$g^s_{ij} = p_i + m_{T^{orig}_i T^{dest}_i} + q_i + e_{T^{dest}_i T^{orig}_j}$$

See Figure F.8 for a visualization of this. We use Time-Way diagrams that are frequently used when depicting solutions of crane/hoist scheduling problems, especially in printed circuit board production (see e.g. Liu and Jiang, 2005). The horizontal and vertical axes in the diagram represent the time and crane positions, respectively. Solid lines indicate that the crane is processing an operation, whereas dashed lines indicate that the crane is either waiting or moving to the start position of the next operation.

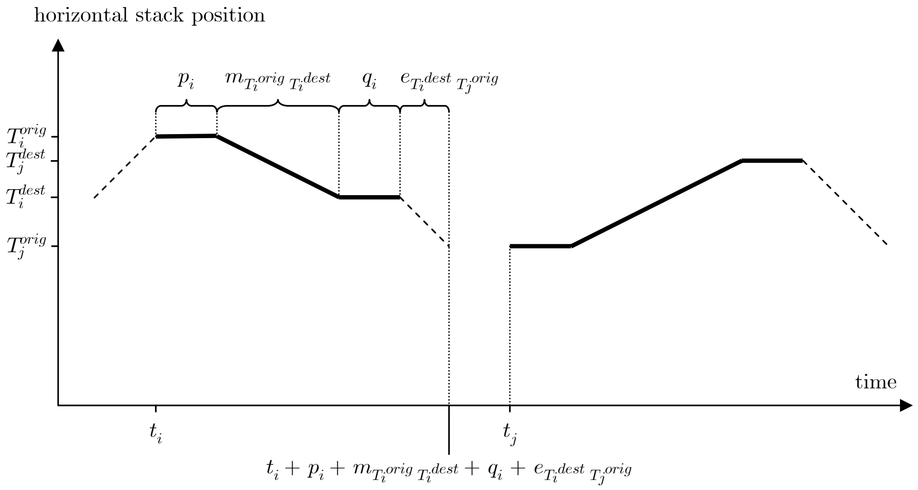

horizontal stack position
p_i
$m_{T_i^{orig}\,T_i^{dest}}$
q_i
$e_{T_i^{dest}\,T_j^{orig}}$
T_i^{orig}
T_j^{dest}
T_i^{dest}
T_j^{orig}
time
t_i
t_j
$t_i + p_i + m_{T_i^{orig}\,T_i^{dest}} + q_i + e_{T_i^{dest}\,T_j^{orig}}$

Figure F.8: Time-Way diagram visualizing the calculation of gap size between two operations executed sequentially by the same crane.

When two operations are allocated to two separate cranes, we need to make sure that the cranes never collide. Further, as we are dealing with a highly stochastic system, we introduce the concept of a buffer. The buffer denotes the amount of time we require between two cranes traversing the same position. By introducing a buffer we establish a certain degree of stability in the schedule. If one of the cranes is delayed by an amount of time less than the buffer size the schedule is still guaranteed to be feasible. The buffer size is set so that infeasibility only occurs in rare cases. In the following, a violation of the prespecified buffer size is considered to be a collision. In the Time-Way diagrams, the buffer is illustrated by a shaded area.

In Table F.1 we describe how to calculate g_{ij}^l. For g_{ij}^l, the left crane is allocated to operation i and the right crane to operation j. In the event of conflict between the two operations, operation i enters and leaves the conflict zone before operation j. There are five different cases to be considered. These are shown in Table F.1 and in Figure F.9 - Figure F.13. ($l2$) and ($l3$) may both apply at the same time and in that case g_{ij}^l is equal to the larger of the two values. The comparison of two stacks is done with respect to their horizontal

	Precondition	Gap
$(l1)$	$T_j^{orig} \le T_i^{dest}$	$g_{ij}^l = p_i + m_{T_i^{orig} T_i^{dest}} + q_i + e_{T_i^{dest} T_j^{orig}} + b$
$(l2)$	$T_j^{dest} \le T_i^{dest} < T_j^{orig}$	$g_{ij}^l \ge p_i + m_{T_i^{orig} T_i^{dest}} + q_i + b - (p_j + m_{T_j^{orig} T_i^{dest}})$
$(l3)$	$T_i^{dest} < T_j^{orig} \le T_i^{orig}$	$g_{ij}^l \ge p_i + m_{T_i^{orig} T_j^{orig}} + b$
$(l4)$	$T_i^{dest} < T_j^{dest} \le T_i^{orig} < T_j^{orig}$	$g_{ij}^l = p_i + m_{T_i^{orig} T_j^{dest}} + b - (p_j + m_{T_j^{orig} T_j^{dest}})$
$(l5)$	Otherwise	$g_{ij}^l = -\infty$

Table F.1: Calculation of the required gap between two operations. Operation i is allocated to the left crane and operation j to the right crane. In conflict, i is moved before j.

position, e.g. $T_i^{orig} < T_j^{orig}$ means the origin stack of operation i is to the left of origin stack of operation j.

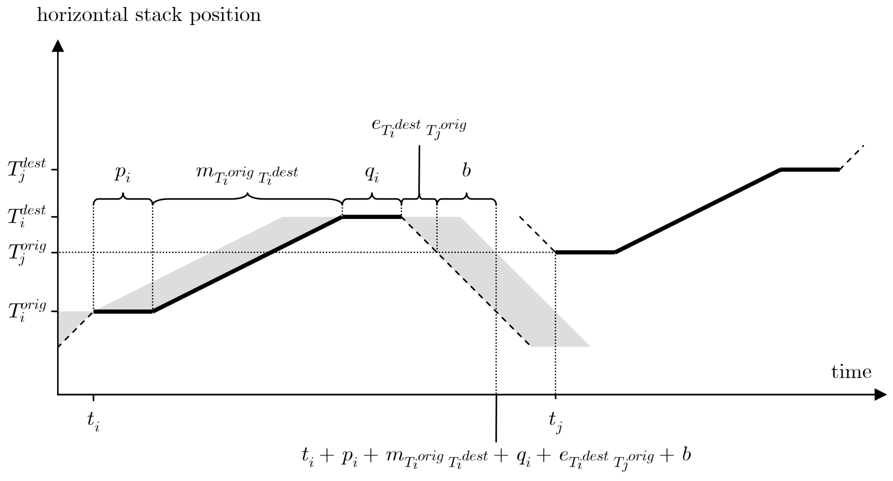

horizontal stack position
T_j^{dest}
T_i^{dest}
T_j^{orig}
T_i^{orig}
p_i
$m_{T_i^{orig} T_i^{dest}}$
$e_{T_i^{dest} T_j^{orig}}$
q_i
b
time
t_i
t_j
$t_i + p_i + m_{T_i^{orig} T_i^{dest}} + q_i + e_{T_i^{dest} T_j^{orig}} + b$

Figure F.9: Time-Way diagram visualizing the calculation of gap size between two operations in case $(l1)$, where $T_j^{orig} \le T_i^{dest}$. The no-collision requirement in this case becomes: $t_i + p_i + m_{T_i^{orig} T_i^{dest}} + q_i + e_{T_i^{dest} T_j^{orig}} + b \le t_j$.

It is clear from each of the five figures (Figure F.9 - Figure F.13) why a violation of the constraint introduces a collision. These five cases are sufficient for avoiding all possible collisions. The proof is found in the technical report (Dohn and Clausen, 2008c).

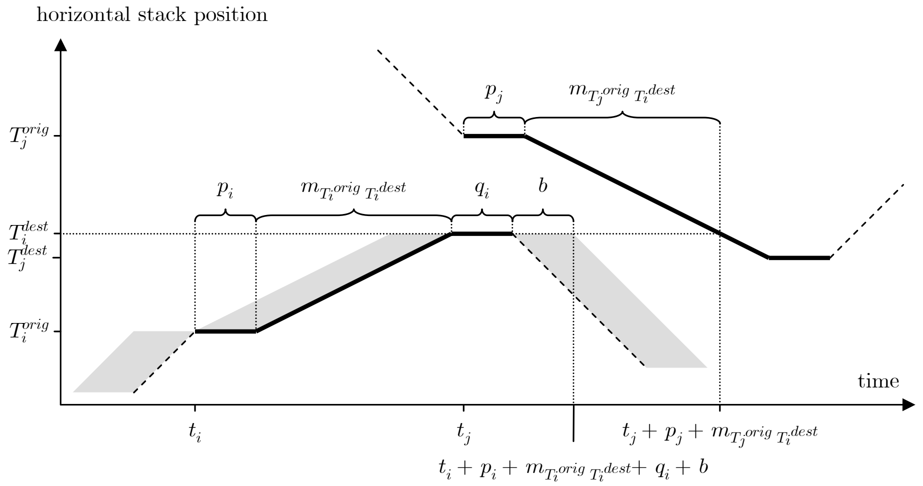

horizontal stack position
T_j^{orig}
T_i^{dest}
T_j^{dest}
T_i^{orig}
p_j
$m_{T_j^{orig}\,T_i^{dest}}$
p_i
$m_{T_i^{orig}\,T_i^{dest}}$
q_i
b
time
t_i
t_j
$t_j + p_j + m_{T_j^{orig}\,T_i^{dest}}$
$t_i + p_i + m_{T_i^{orig}\,T_i^{dest}} + q_i + b$

Figure F.10: Time-Way diagram visualizing the calculation of gap size between two operations in case ($l2$), where $T_j^{dest} \leq T_i^{dest} < T_j^{orig}$. The no-collision requirement in this case becomes: $t_i + p_i + m_{T_i^{orig} T_i^{dest}} + q_i + b \leq t_j + p_j + m_{T_j^{orig} T_i^{dest}}$.

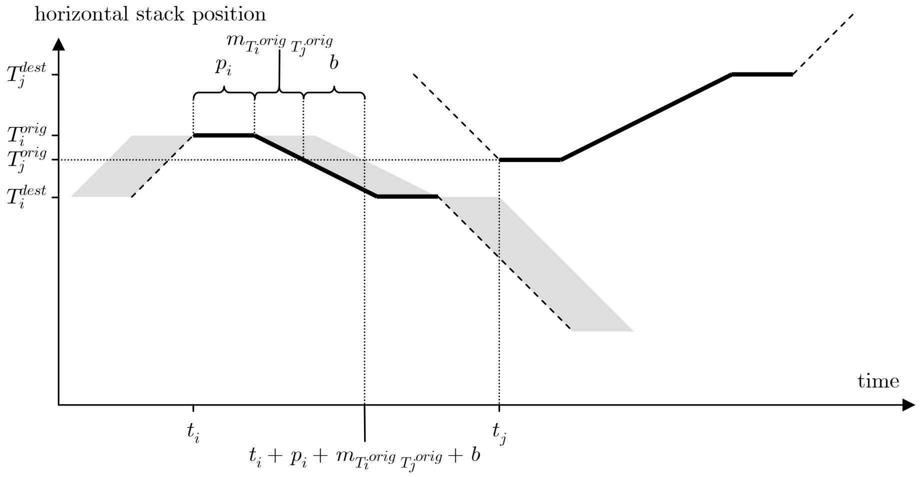

horizontal stack position
$m_{T_i^{orig}\,T_j^{orig}}$
p_i
b
T_j^{dest}
T_i^{orig}
T_j^{orig}
T_i^{dest}
time
t_i
t_j
$t_i + p_i + m_{T_i^{orig}\,T_j^{orig}} + b$

Figure F.11: Time-Way diagram visualizing the calculation of gap size between two operations in case ($l3$), where $T_i^{dest} < T_j^{orig} \leq T_i^{orig}$. The no-collision requirement in this case becomes: $t_i + p_i + m_{T_i^{orig} T_j^{orig}} + b \leq t_j$.

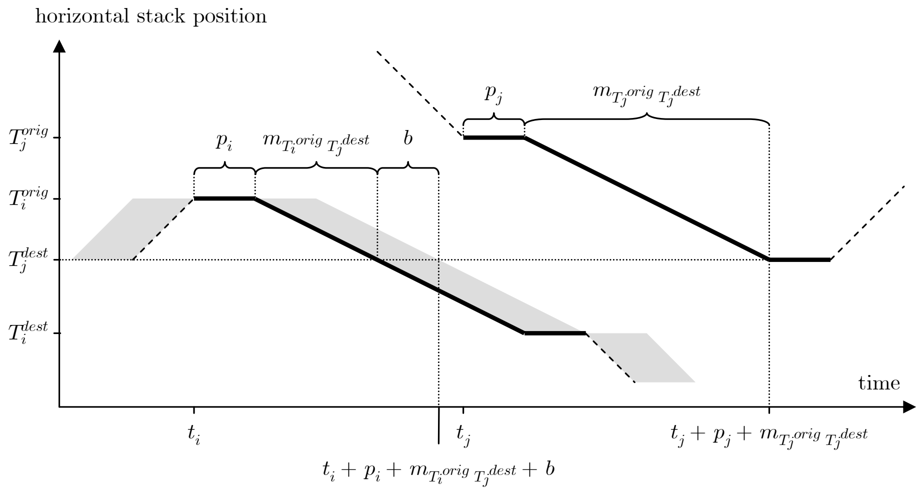

horizontal stack position
T_j^{orig}
T_i^{orig}
T_j^{dest}
T_i^{dest}
p_i
$m_{T_i^{orig}T_j^{dest}}$
b
p_j
$m_{T_j^{orig}T_j^{dest}}$
time
t_i
t_j
$t_j + p_j + m_{T_j^{orig}T_j^{dest}}$
$t_i + p_i + m_{T_i^{orig}T_j^{dest}} + b$

Figure F.12: Time-Way diagram visualizing the calculation of gap size between two operations in case $(l4)$, where $T_i^{dest} < T_j^{dest} \leq T_i^{orig} < T_j^{orig}$. The no-collision requirement in this case becomes: $t_i + p_i + m_{T_i^{orig}T_j^{orig}} + b \leq t_j + p_j + m_{T_j^{orig}T_j^{dest}}$.

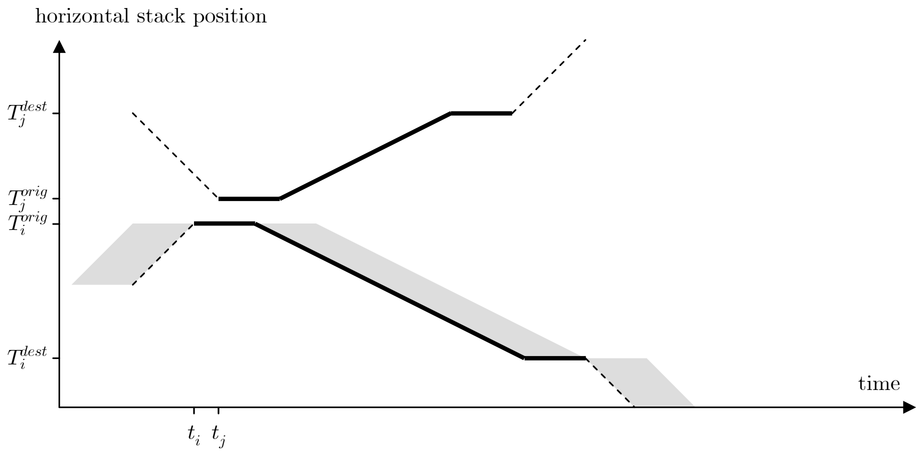


Figure F.13: Time-Way diagram visualizing the calculation of gap size between two operations in case $(l5)$, where there are no direct temporal relations between operation i and operation j.

	Precondition	Gap
$(r1)$	$T_j^{orig} \geq T_i^{dest}$	$g_{ij}^r = p_i + m_{T_i^{orig} T_i^{dest}} + q_i + e_{T_i^{dest} T_j^{orig}} + b$
$(r2)$	$T_j^{dest} \geq T_i^{dest} > T_j^{orig}$	$g_{ij}^r \geq p_i + m_{T_i^{orig} T_i^{dest}} + q_i + b - (p_j + m_{T_j^{orig} T_i^{dest}})$
$(r3)$	$T_i^{dest} > T_j^{orig} \geq T_i^{orig}$	$g_{ij}^r \geq p_i + m_{T_i^{orig} T_j^{orig}} + b$
$(r4)$	$T_i^{dest} > T_j^{dest} \geq T_i^{orig} > T_j^{orig}$	$g_{ij}^r = p_i + m_{T_i^{orig} T_j^{dest}} + b - (p_j + m_{T_j^{orig} T_j^{dest}})$
$(r5)$	Otherwise	$g_{ij}^r = -\infty$

Table F.2: Calculation of the required gap between two operations. Operation i is allocated to the right crane and operation j to the left crane. In conflict, i is moved before j.

In Table F.2, we show how to calculate g_{ij}^r. The calculations are analogous to the ones of g_{ij}^l. Operation i is now allocated to the right crane and j to the left crane. Again, in case of any conflict between the two operations, operation i is before operation j. All coordinates are mirrored, which does not affect any of the movement times and hence the calculations are very similar.

Figure F.14 illustrates how $(r1)$ is closely related to $(l1)$. The only difference is the precondition, which is mirrored.

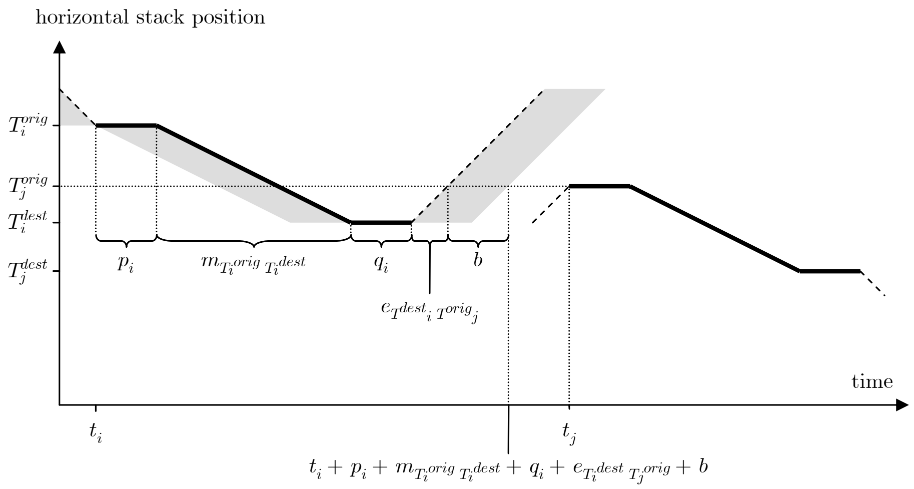

horizontal stack position
T_i^{orig}
T_j^{orig}
T_i^{dest}
T_j^{dest}
p_i
$m_{T_i^{orig} T_i^{dest}}$
q_i
b
$e_{T^{dest}_i T^{orig}_j}$
time
t_i
t_j
$t_i + p_i + m_{T_i^{orig} T_i^{dest}} + q_i + e_{T_i^{dest} T_j^{orig}} + b$

Figure F.14: The situation of Figure F.9 mirrored vertically. Operation i is now allocated to the right crane.

g^s_{ij}	a_1	a_2	a_3	a_4	a_5
a_1	–	4	4	6	6
a_2	∞	–	6	10	10
a_3	∞	∞	–	8	8
a_4	∞	∞	6	–	6
a_5	∞	∞	6	∞	–

g^l_{ij}	a_1	a_2	a_3	a_4	a_5
a_1	–	5	$-\infty$	7	7
a_2	∞	–	7	11	11
a_3	∞	∞	–	9	9
a_4	∞	∞	$-\infty$	–	7
a_5	∞	∞	$-\infty$	∞	–

g^r_{ij}	a_1	a_2	a_3	a_4	a_5
a_1	–	2	5	-1	-1
a_2	∞	–	4	-1	-1
a_3	∞	∞	–	$-\infty$	$-\infty$
a_4	∞	∞	7	–	2
a_5	∞	∞	7	∞	–

Table F.3: Coefficients of generalized precedence constraints for the example.

Example ***[continued]***

With these definitions, we can illustrate how to calculate the coefficients for the generalized precedence constraints of Example 1. We have the three sets of coefficients: g^s_{ij}, g^l_{ij}, *and* g^r_{ij} *represented by the three matrices of Table F.3. First, we use the precedence constraints of Figure F.7 to fill in the* ∞*-values. This includes the entailed precedence constraints (e.g.* $a_1 \to a_2 \wedge a_2 \to a_3 \Rightarrow a_1 \to a_3$*). In this example, we have for all operations:* $p_i = 1, q_i = 1$, *and* $b = 1$. $m_{T_xT_y}$ *and* $e_{T_xT_y}$ *are equal and are set to the horizontal distance between the two stacks, cf. Figure F.3 (e.g.* $m_{T_1T_{exit}} = 4$*). Three examples of the calculations for the matrices are shown below (*$g^r_{a_1a_2}$ *is calculated from* $(r2)$*+*$(r3)$ *and* $g^r_{a_1a_4}$ *is calculated from* $(r4)$*).*

$$
\begin{aligned}
g^s_{a_2a_3} &= p_{a_2} + m_{T^b_{a_2}T^e_{a_2}} + q_{a_2} + e_{T^e_{a_2}T^b_{a_3}} = 1+3+1+1 = 6\\
g^r_{a_1a_2} &= \max\{p_{a_1} + m_{T^b_{a_1}T^e_{a_1}} + q_{a_1} + b - (p_{a_2} + m_{T^b_{a_2}T^e_{a_1}}), p_{a_1} + m_{T^b_{a_1}T^b_{a_2}} + b\}\\
&= \max\{1+1+1+1-(1+1), 1+0+1\} = 2\\
g^r_{a_1a_4} &= p_{a_1} + m_{T^b_{a_1}T^e_{a_4}} + b - (p_{a_4} + m_{T^b_{a_4}T^e_{a_4}}) = 1+0+1-(1+2) = -1
\end{aligned}
$$

F.4.3 Operation Priority

So far we have assumed that all operations had to be included in the final schedule. However we may deviate slightly from this strategy. We introduce the concept of *optional operations*. We may add a number of optional operations to the end of the plan. The purpose of adding these operations is to enhance the final state of the slab yard, so that the risk of delays in future scheduling is reduced.

Example ***[continued]***

In the example, we could e.g. add to the solution of Figure F.4 the operation ($S_7 \rightarrow T_4$) as an optional operation. This would give a final yard state with fewer false positions.

To quantify the importance of the individual optional operations we use the *operation priority*. A high priority means that we prefer the inclusion of this operation to other optional operations with lower priority. In this work, the priority is calculated as follows. We consider the two possible end states that will be entailed by respectively including or excluding the operation from the plan. For both states it is possible to calculate the number of false positions in the yard. As described in Section F.3.2, the number of false positions, in our case, is a stochastic measure. The difference between the two sums, i.e. the possible gain in number of false positions, is used as the operation priority. We would never include an operation which increases the number of false positions.

F.4.4 Objective Function

As was described in Section F.2.1, the objective function is to minimize the maximum tardiness of the schedule. At the same time, a good schedule includes many optional operations. The sum of the priorities of the optional operations included is used to evaluate this criterion. The individual priorities of operations are determined by the planning module, as described in the previous section. The objective function is two-layered so that minimization of maximum tardiness is always prioritized over the second objective. However, we still require all operations with priority ∞ (compulsory operations) to be in the schedule.

F.4.5 Generic Formulation of the Crane Scheduling Problem

We are now able to abstract fully from the real-world context and introduce an explicit formulation of the Crane Scheduling Problem as a parallel scheduling problem with generalized precedence constraints, non-zero release times, and sequence-dependent setup time. Using the three-field notation of Graham et al. (1979) extended by Brucker et al. (1999) and Allahverdi et al. (2008) we denote the problem $R2|temp, r_j, s_{ijm}|T_{max}$.

Sets:

$\mathcal{O}$	The set of operations.
$\mathcal{C} = \{C^l, C^r\}$	The two cranes, left crane and right crane respectively.

Decision variables:

$x_i \in \mathbb{B}$	$i \in \mathcal{O}$	1 if operation i is included in the schedule, 0 otherwise.
$t_i \in \mathbb{Z}$	$i \in \mathcal{O}$	Start time of operation i.
$c_i \in \mathcal{C}$	$i \in \mathcal{O}$	The crane allocation of operation i.
$y_{ij} \in \mathbb{B}$	$i \in \mathcal{O}, j \in \mathcal{O}$	1 if the operation i is considered to be before operation j, 0 otherwise.
$\tau_i \in \mathbb{Z}$	$i \in \mathcal{O}$	Tardiness of operation i.

Parameters:

$g^s_{ij} \in \mathbb{Z}$	$i \in \mathcal{O}, j \in \mathcal{O}$	The required gap between operations i and j when allocated to the same crane and i is before j.
$g^l_{ij} \in \mathbb{Z}$	$i \in \mathcal{O}, j \in \mathcal{O}$	The required gap between operations i and j when allocated respectively to the left crane and the right crane and i is first in a conflict.
$g^r_{ij} \in \mathbb{Z}$	$i \in \mathcal{O}, j \in \mathcal{O}$	The required gap between operations i and j when allocated respectively to the right crane and the left crane and i is first in a conflict.
r_i	$i \in \mathcal{O}$	Release time of operation i.
d_i	$i \in \mathcal{O}$	Due date of operation i.
p_i	$i \in \mathcal{O}$	Priority (weight) of operation i.
t^{max}_i	$i \in \mathcal{O}$	Deadline of operation i.

The Constraint Programming Model:

$$\min \max \tau_a \text{ and secondly } \max \sum_{i \in A} p_i x_i \tag{F.1}$$

$$\tau_i = \max\{0, t_i - d_i\} \qquad \forall i \in \mathcal{O} \tag{F.2}$$

$$x_i = 1 \wedge x_j = 1 \Rightarrow y_{ij} = 1 \vee y_{ji} = 1 \qquad \forall i \in \mathcal{O}, \forall j \in \mathcal{O}, i \neq j \tag{F.3}$$

$$t_i \geq r_i \qquad \forall i \in \mathcal{O} \tag{F.4}$$

$$t_i \leq t_i^{max} \qquad \forall i \in \mathcal{O} \tag{F.5}$$

$$y_{ij} = 1 \wedge c_i = C^l \wedge c_j = C^l \Rightarrow t_i + g_{ij}^s \leq t_j \qquad \forall i \in \mathcal{O}, \forall j \in \mathcal{O} \tag{F.6}$$

$$y_{ij} = 1 \wedge c_i = C^r \wedge c_j = C^r \Rightarrow t_i + g_{ij}^s \leq t_j \qquad \forall i \in \mathcal{O}, \forall j \in \mathcal{O} \tag{F.7}$$

$$y_{ij} = 1 \wedge c_i = C^l \wedge c_j = C^r \Rightarrow t_i + g_{ij}^l \leq t_j \qquad \forall i \in \mathcal{O}, \forall j \in \mathcal{O} \tag{F.8}$$

$$y_{ij} = 1 \wedge c_i = C^r \wedge c_j = C^l \Rightarrow t_i + g_{ij}^r \leq t_j \qquad \forall i \in \mathcal{O}, \forall j \in \mathcal{O} \tag{F.9}$$

$$p_i = \infty \Rightarrow x_i = 1 \qquad \forall i \in \mathcal{O} \tag{F.10}$$

$$x_i \in \mathbb{B}, t_i \in \mathbb{Z}, c_i \in \mathcal{C}, y_{ij} \in \mathbb{B}, \tau_i \in \mathbb{Z} \qquad \forall i \in \mathcal{O}, \forall j \in \mathcal{O} \tag{F.11}$$

The model (F.1)-(F.11) captures all the problem properties that have been described in this section. (F.1) is the objective function, which has two criteria. (F.2) sets the tardiness of each operation. (F.3) ensures that if both operations are included in the schedule, then their internal precedence constraints must be respected either in one direction or the other. Operations cannot be started before their release date (F.4) and must be scheduled within the horizon (F.5). (F.6)-(F.9) connect the decision on crane allocation with the correct precedence constraints. Finally, (F.10) makes sure that all compulsory operations are included in the schedule, and (F.11) gives the domains of the decision variables.

The parameter p_i is calculated by the planning module, as explained previously. r_i and d_i are calculated as $r_i = ELT_i - dur_i$ and $d_i = ALT_i - dur_i$, where dur_i is the duration of an operation, i.e. $dur_i = p_i + m_{T_i^{orig} T_i^{dest}} + q_i$. g_{ij}^s, g_{ij}^l, and g_{ij}^r are calculated as described in Section F.4.2.

Example ***[continued]***

We consider the example again. We have the 5 operations of Figure F.4 which make up the set of operations $\mathcal{O} = \{o_1, o_2, o_3, o_4, o_5\}$. We have the precedence coefficients from Table F.3. The duration of each operation and subsequently the release date and due date of each operation is calculated in the table. In this *example, the scheduling horizon is 22:*

Operation (i)	dur_i	r_i	d_i	t_i^{max}	p_i
o_1	3	0	19	19	∞
o_2	5	0	5	17	∞
o_3	3	8	9	19	∞
o_4	4	0	18	18	∞
o_5	4	0	18	18	∞

An optimal solution is (Solution 1):

Operation (i)	x_i	t_i	c_i	y_{io_1}	y_{io_2}	y_{io_3}	y_{io_4}	y_{io_5}	τ_i
o_1	1	0	C^r	–	1	1	1	1	0
o_2	1	2	C^l	0	–	1	1	1	0
o_3	1	9	C^r	0	0	–	1	1	0
o_4	1	12	C^l	0	0	0	–	1	0
o_5	1	18	C^l	0	0	0	0	–	0

Another solution is (Solution 2):

Operation (i)	x_i	t_i	c_i	y_{io_1}	y_{io_2}	y_{io_3}	y_{io_4}	y_{io_5}	τ_i
o_1	1	0	C^r	–	1	1	1	1	0
o_2	1	4	C^r	0	–	1	1	1	0
o_3	1	10	C^r	0	0	–	1	1	1
o_4	1	3	C^l	0	0	0	–	1	0
o_5	1	9	C^l	0	0	0	0	–	0

Solution 1 of the example is visualized in a Gantt chart in Figure F.15 (top). The Gantt chart does not depict the value of neither y_{ij}-variables nor τ_i-variables.

Solution 2 is illustrated in Figure F.15 (bottom). The solution is more compact and may actually look more attractive. However, the due date of o_3 is violated and therefore this solution is worse than Solution 1.

The nature of the problem makes the transitive closure valid for all choices of ordering of conflicting operations, i.e. if operation i is before j (with respect to conflicts) and j is before k then we may assume that i is before k ($y_{ij} = 1 \wedge y_{jk} = 1 \Rightarrow y_{ik} = 1$). We also find this property in lists and therefore we can use a list to represent all sequencing decisions. If, further, we state the crane allocation of each operation, c_i, and if we assume that all operations are scheduled at the earliest possible time according to the given sequence and the crane allocations, then the list representation is sufficient to explicitly represent

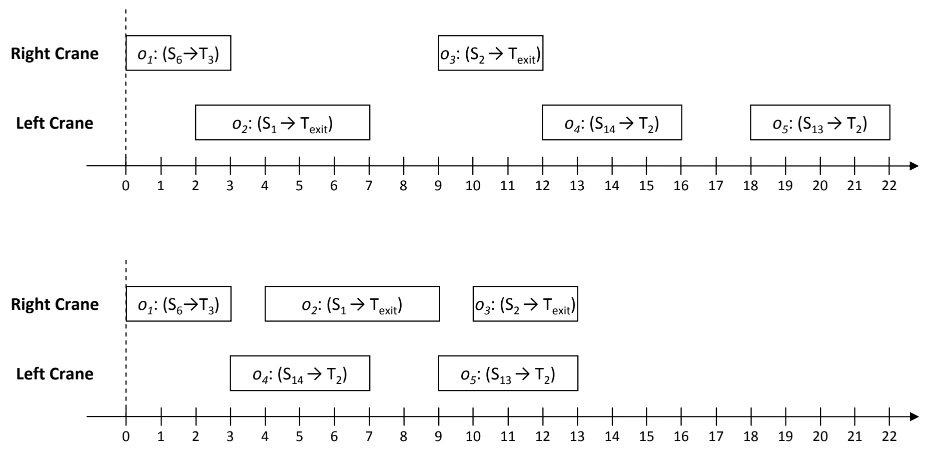


Figure F.15: Gantt charts of Solution 1 (top) and Solution 2 (bottom).

the solution. The earliest possible times are found in polynomial time by running through the list. For every operation i the generalized precedence constraints to all preceding operations are checked and the most limiting of those determine the starting time of operation i. We adapt the graphical representation from the planning solutions but add information on the crane allocation. We still lack information on t_i and τ_i and therefore the objective function of the solution is not immediately available, but it can be calculated by running through the list. The two solutions from before are represented as seen on Figure F.16.

o_1: ($S_6 \rightarrow T_3$) R	o_2: ($S_1 \rightarrow T_{exit}$) L	o_3: ($S_2 \rightarrow T_{exit}$) R	o_4: ($S_{14} \rightarrow T_2$) L	o_5: ($S_{13} \rightarrow T_2$) L

o_1: ($S_6 \rightarrow T_3$) R	o_2: ($S_1 \rightarrow T_{exit}$) R	o_3: ($S_2 \rightarrow T_{exit}$) R	o_4: ($S_{14} \rightarrow T_2$) L	o_5: ($S_{13} \rightarrow T_2$) L

Figure F.16: List representations of Solution 1 (top) and Solution 2 (bottom).

The advantage of this representation is clear from Figure F.16. The only difference between the two solutions is the change in crane allocation of operation o_2. All other variable changes (that were observed on Figure F.15) can be interpreted as consequences of this variable change. Another nice feature of the list representation is that any permutation that respects all precedence relations is also feasible with respect to (F.2)-(F.4) + (F.6)-(F.11). Only the scheduling horizon is possibly violated.

Another way of visualizing a scheduling solution graphically is by using Time-Way diagrams as introduced in Section F.4.2. Solution 1 and Solution 2 are depicted in Figure F.17.

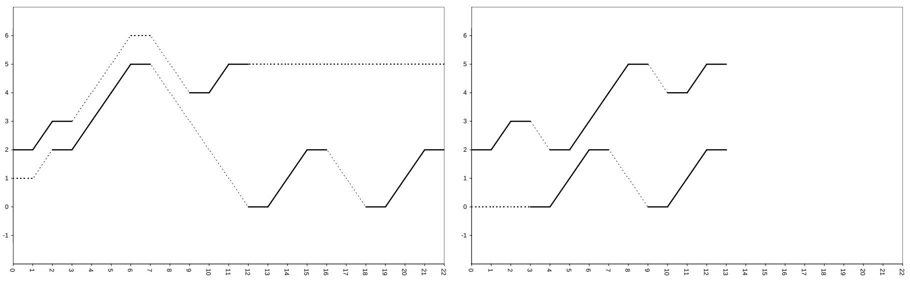


Figure F.17: Time-Way diagrams of Solution 1 (left) and Solution 2 (right).

F.5 Solution Method

A solution method has been implemented based on the presented model. In the following, we present two greedy methods, one for the planning problem and one for the scheduling problem. The two methods are straight-forward in their implementation and more sophisticated methods will probably enhance performance. However, the simple methods are still able to generate good results, and so we will use them to assess the value of the model.

F.5.1 Planning

The planning method will provide a plan as described in Section F.3. When the final schedule is created in the second stage of the method, all precedence constraints are respected, so the sequence of operations that we specify in the planning solution fully determines the state of the yard. As a consequence, we are able to update the yard state as the operations are added to the plan. For any partial plan, we have a current yard state. When we refer to the location of slabs, it is with respect to the current state of the yard.

The implemented method is divided into three steps:

- Generation of operations for slabs that must leave the yard during the scheduling horizon (*exit slabs*).
- Generation of operations for incoming slabs (*arrival slabs*).
- Generation of optional operations.

In this solution method, we treat the three steps separately, one by one, in a state space exploration.

First, we generate a list of operations for the slabs that must leave the yard during the scheduling horizon. We already have their Aim Leave Time and hence we have a predetermined ordering of these operations. The slabs may not be on top of their stacks, so we may also need to generate *reshuffle* operations and move the slabs on top to other stacks. For reshuffle operations we must specify a destination stack. The destination stack is chosen from a number of criteria. First, we disallow movement to stacks still containing exit-slabs. Moving a slab to such a stack will trigger another reshuffle operation later, where the same slab has to be reshuffled again. This should be avoided if possible. Further, when choosing a destination stack, we look for stacks within a short horizontal range. This limits the duration of the operation and at the same time decreases the risk of crane collision involving this operation. We also look for stacks, where the slab has a small chance of being in a new false position (and hence in need of another reshuffle in a future plan).

When all exit operations have been generated, we proceed with the arrival operations. For each slab on the railway wagons, we generate an operation that will bring the slab to the yard. When choosing a destination for these slabs it is particularly important to keep the sum of false position probabilities low. All arrival operations are sequenced after the exit operations. This does not necessarily mean that they are also scheduled later than all arrival operations. The reason for sequencing the operations in this way in the planning solution is that all stacks involved in both exit and arrival operations will have the exit operation executed first, which is obviously a desirable feature. To introduce flexibility in the scheduler, we try to select destination stacks that do not have any outgoing exit operations. The order of arrival operations is partially predetermined. We have to move the slabs from top to bottom from the stacks on the railway wagons. However, we have a choice between the arrival stacks.

Finally, we generate a number of operations that are not mandatory for the feasibility of schedules, but that will increase the quality of the solution by reducing the total false position probability and may also move slabs with an upcoming due date closer to the exit stack. The optional operations are always added at the end of the plan to ensure that the remaining plan is feasible, even if some of the optional operations are not included in the schedule.

F.5.2 Scheduling

Given a planning solution, we need to schedule the operations on the two cranes. The generic formulation of the problem is given in Section F.4.5. In the following we describe a greedy heuristic on which the current implementation is based.

The heuristic is very simple. We process the operations in the order given in the planning solution. For each operation, the earliest possible time of initiation is calculated for both cranes. The operation is allocated to the crane that is able to initiate first. As we have release times for operations, there may still be some waiting time from the preceding operation to the current one. Therefore, we check if we are able to squeeze in any of the unscheduled operations. The operations with high priority are preferred to the others. When squeezing in operations like this, we need to make sure that all precedence constraints are respected.

The heuristic is greedy and may therefore make decisions, which are not advantageous in the end. This issue could be addressed by the implementation of a local search procedure to enhance the results of a greedy construction heuristic. As a starting point, a steepest descent algorithm would probably increase quality significantly. Adding metaheuristic features to such a search will enhance the solution even further. Preliminary test results from a metaheuristic show promising results.

F.6 Test results

To evaluate the quality of the solutions, we generate a reference solution that represents the solution obtainable by manual planning for each instance. We try to imitate the behavior of the cranes when they are under the control of the individual crane operator. The operators work on an ad-hoc basis. We expect them to deal with exit slabs as we approach their deadlines and reshuffles are carried out when needed. More specifically, we equip the crane operators with a two hour foresight. Slabs that are to leave within this period will not be blocked by new slabs. If a crane has free time in between moves, it will use

	Two-Stage				Manual			
Slabs per day	Avg moves p. slab	Avg move dur.	Avg deadline violation	Max deadline violation	Avg moves p. slab	Avg move dur.	Avg deadline violation	Max deadline violation
400	2.37	43.91	0.01	1.66	2.61	46.55	0.24	11.78
450	2.39	43.76	0.02	1.35	2.60	46.28	0.09	8.52
500	2.37	43.84	0.02	1.86	2.61	46.68	5.53	61.13
550	2.38	43.80	0.02	1.86	2.59	46.50	2.94	74.64
600	2.41	43.84	1.08	10.54	2.61	46.62	26.69	234.82
650	2.39	43.91	0.40	7.28	2.58	46.63	27.30	253.55
700	2.38	43.91	0.38	7.51	2.59	46.60	139.64	543.17
750	2.38	43.90	0.72	16.50	2.68	46.61	878.07	1883.29
800	2.39	44.00	2.59	44.61	2.90	46.80	3092.32	4840.96
850	2.40	43.92	1.68	43.33	3.14	46.75	5202.69	7874.27
900	2.41	43.88	13.72	135.50	3.30	46.45	7930.29	13108.21
950	2.39	43.95	47.02	270.68	3.34	46.04	9204.57	15641.77
1000	2.44	43.93	118.06	490.39	3.43	45.72	10704.42	18626.53

Table F.4: Comparison of test results from the Two-Stage method and simulation of manual planning. Each value is an average over 10 identical runs. Deadline violations and durations are measured in seconds.

the time to move slabs from the train wagons to the yard. In the schedules described earlier, we needed a buffer between cranes to make the schedules more robust. To the advantage of this simulation, the buffer is disregarded in this part, as we are not really creating a schedule. Rather, we are simulating manual planning/scheduling, and hence the operations are to be interpreted as happening in real time and not as a pre-made schedule to be followed. Again, a more detailed description is found in the technical report (Dohn and Clausen, 2008*c*).

By comparing the solutions of the method presented in this paper to the solutions of such a simulation, we are able to assess the value of the proposed method. In the following we run a number of simulations. The average yard throughput is fixed in each of the test instances. The throughput is increased in the hard test instances to check the effect on the quality of the schedules. The simulations are kept as close as possible to the real world conditions. The details of the data simulation and the settings for simulation of manual planning are in the technical report (Dohn and Clausen, 2008*c*).

For the simulations, we assume that the requested throughput of the yard for each day is randomly drawn from a Gaussian distribution. In the same way, we assume that the production time and through time (i.e. storage time in the yard) for each slab are also drawn from Gaussian distributions.

From Table F.4 it is clear that the proposed method provides significantly better results than the simulation of manual planning. In the table we have shown four performance measures. For each method, the first column gives the average number of times a slab is moved before it leaves the yard. As slabs in our setup are never transferred directly from train wagons to the exit belt, the minimum number of moves of each slab is 2. This measure illustrates how well the moves are planned, i.e. a low number indicates that the slabs are seldom in the way of others. From the tests, we see a significant difference between the two methods, especially for the harder problems, where the Two-Stage algorithm on average uses approximately one move less per slab. Also the duration of each move is of interest and is shown in the second column. The difference between the two methods is not remarkable, even though the Two-Stage algorithm is a few seconds faster in all cases. The duration seems stable over the set of test instances.

The two last columns report on deadline violations. The rightmost of the columns gives the maximum deadline violation, which is, as stated earlier, the main objective considered in this work. The first of the two columns reports on the average violations. This is interesting if we assume that the following production is able to catch up on the delays we may have caused. A low average deadline violation is equivalent to a low sum of violations, which is another objective often used for scheduling problems in the literature. For both objectives, we see that the Two-Stage algorithm clearly outperforms the other. The manual planning has severe problems in the hard instances, where the results of the Two-Stage method are still satisfactory. The figures for manual simulation may seem very large, but it is noted that the numbers can only be used for comparison with other similar tests. As soon as a method is unable to keep up with the rate at which slabs enter the yard, it will lead to larger and larger violations as we let the simulation run. In each run of these tests, the two methods naturally span over the same production plan.

Both methods are able to produce results in less than a second. Such computation times are insignificant in these settings and are therefore not compared here.

Figure F.18 illustrates a schedule created by the Two-Stage algorithm.

F.7 Conclusions

The Slab Yard Planning and Crane Scheduling Problem has been modeled in a novel way that facilitates a beneficial, and at the same time transparent op-

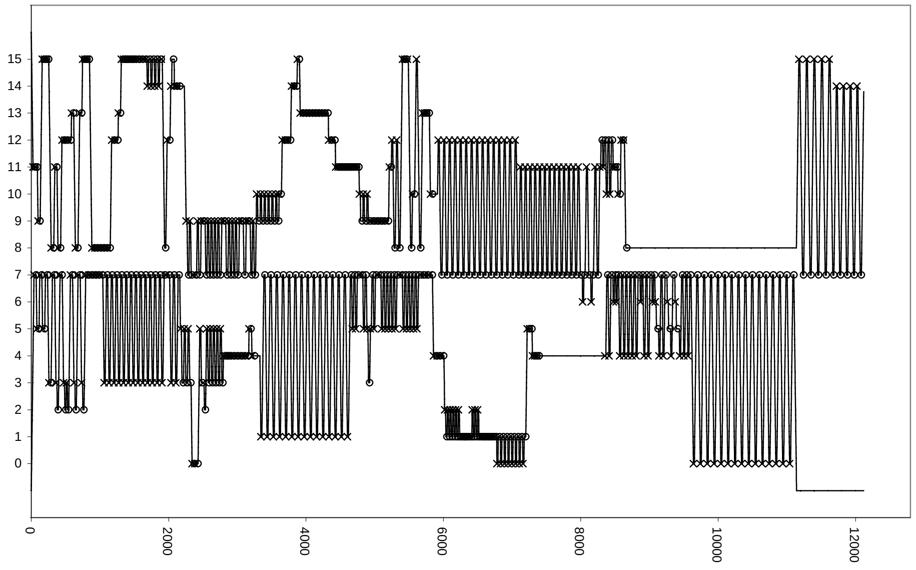


Figure F.18: Schedule created by the Two-Stage algorithm. The two curves describe the horizontal positions of the two cranes over time. Each curve contains a number of pick-ups (×) and drop-offs (∘).

timization. The model is generic enough to capture several variations of the problem. The solution methods adapt to variations of the problem, correspondingly.

From the test results it is clear that the model facilitates an algorithm that is capable of providing solutions superior to those achievable by manual planning. The tests are, however, preliminary and based on simulations which rely on a number of assumptions.

We have introduced a model that, by splitting The Slab Yard Planning and Crane Scheduling Problem into two stages, facilitates a solution procedure that is clear in the formulation of objectives and is able to generate superior schedules by addressing the problem at two different abstraction levels.

Future work should be aimed at real-world applications. So far, the experimental conclusions are based on simulations and artificially generated data. In a practical application it is possible to tailor the algorithms to fit the exact properties of that particular problem. In this paper, we have made sure not to take advantage of structures in the problem data, as such structures may not transfer to variations of the problem. Therefore, in a practical application, it may be possible to utilize problem-specific knowledge in the creation of the planning and the scheduling method, and get even better results. On the other hand, practical problems may also introduce new challenges. Either way, the model presented in this paper will be a valuable starting point for exhaustive purpose-built practical models.

References

Allahverdi, A., C. Ng, T. Cheng, and M. Kovalyov (2008). "A survey of scheduling problems with setup times or costs". In: *European Journal of Operational Research* 187.3, pp. 985–1032.

Brucker, P., A. Drexl, R. Mohring, K. Neumann, and E. Pesch (1999). "Resource-constrained project scheduling: Notation, classification, models, and methods". In: *European Journal of Operational Research* 112.1, pp. 3–41.

Che, A. and C. Chu (2004). "Single-track multi-hoist scheduling problem: a collision-free resolution based on a branch-and-bound approach". In: *International Journal of Production Research* 42.12, pp. 2435–2456.

Dekker, R., P. Voogd, and E. Asperen (2006). "Advanced methods for container stacking". In: *OR Spectrum - Quantitative Approaches in Management* 28.4, p. 563.

Dohn, A. and J. Clausen (2008*c*). *Optimizing the Slab Yard Planning and Crane Scheduling Problem Using a Two-Stage Approach (Technical Report)*. Tech. rep. Technical University of Denmark.

Frederickson, G., M. Hecht, and C. Kim (1978). "Approximation algorithms for some routing problems". In: *SIAM Journal on Computing* 7.2, pp. 178–93.

Gambardella, L., M. Mastrolilli, A. Rizzoli, and M. Zaffalon (2001). "An optimization methodology for intermodal terminal management". In: *Journal of Intelligent Manufacturing* 12.5-6, p. 521.

Graham, R., E. Lawler, J. Lenstra, and A. Rinnooy Kan (1979). "Optimization and approximation in deterministic sequencing and scheduling: a survey". In: *Discrete Optimisation* 5, pp. 287–326.

Hansen, J. (2003). "Industrialised application of combinatorial optimization". PhD thesis. Richard Petersens Plads, Building 321, DK-2800 Kgs. Lyngby: Informatics and Mathematical Modelling, Technical University of Denmark, DTU.

Kim, K. and J. Bae (1998). "Re-marshaling export containers in port container terminals". Ed. by M. Sadek Eid, Hamid Parsaie, Mahmoud Younis, and Soha Eid Moussa. In: *Computers and Industrial Engineering* 35.3-4, pp. 655–658.

Kim, K., Y. Park, and K.-R. Ryu (2000). “Deriving decision rules to locate export containers in container yards”. In: *European Journal of Operational Research* 124.1, pp. 89–101.

König, F., M. Lübbecke, R. Möhring, G. Schäfer, and I. Spenke (2007). “Solutions to real-world instances of PSPACE-complete stacking”. Ed. by L. Arge, M. Hoffmann, M. Hoffmann, E. Welzl, and E. Welzl. In: *Algorithms - ESA 2007. Proceedings 15th European Symposium. (Lecture Notes in Computer Science vol. 4698)*, pp. 729–40.

Lamothe, J., C. Thierry, and J. Delmas (1996). “A multihoist model for the real time hoist scheduling problem”. In: *Symposium on Discrete Events and Manufacturing Systems. CESA’96 IMACS Multiconference. Computational Engineering in Systems Applications*, pp. 461–6.

Leung, J. and G. Zhang (2003). “Optimal cyclic scheduling for printed circuit board production lines with multiple hoists and general processing sequence”. In: *IEEE Transactions on Robotics and Automation* 19.3, pp. 480–484.

Leung, J., G. Zhang, X. Yang, R. Mak, and K. Lam (2004). “Optimal Cyclic Multi-Hoist Scheduling: A Mixed Integer Programming Approach”. In: *Operations Research* 52.6, pp. 965–976.

Liu, J. and Y. Jiang (2005). “An efficient optimal solution to the two-hoist no-wait cyclic scheduling problem”. In: *Operations Research* 53.2, pp. 313–27.

Singh, K., Srinivas, and M. Tiwari (2004). “Modelling the slab stack shuffling problem in developing steel rolling schedules and its solution using improved Parallel Genetic Algorithms”. In: *International Journal of Production Economics* 91.2, pp. 135–147.

Steenken, D., S. Voß, and R. Stahlbock (2004). “Container terminal operation and operations research - a classification and literature review”. In: *OR Spectrum* 26.1, pp. 3–49.

Tang, L., J. Liu, A. Rong, and Z. Yang (2001). “An Effective Heuristic Algorithm to Minimise Stack Shuffles in Selecting Steel Slabs from the Slab Yard for Heating and Rolling”. In: *Journal of the Operational Research Society* 52.10, pp. 1091–1097.

Tang, L., J. Liu, A. Rong, and Z. Yang (2002). “Modelling and a genetic algorithm solution for the slab stack shuffling problem when implementing steel rolling schedules”. In: *International Journal of Production Research* 40.7, pp. 1583–95.

Zaffalon, M., A. Rizzoli, L. Gambardella, and M. Mastroiilli (1998). “Resource allocation and scheduling of operations in an intermodal terminal”. In: *Simulation Technology: Science and Art. 10th European Simulation Symposium 1998. ESS’98*, pp. 520–7.

Zhu, X. and W. Wilhelm (2006). “Scheduling and lot sizing with sequence-dependent setup: a literature review”. In: *IIE Transactions* 38.11, pp. 987–1007.

In a modern society, manpower is both a scarce and an expensive resource. Skilled personnel is usually in high demand and accounts for a significant part of total expenses in many companies. In order to minimize costs and overstaffing, to maximize the utilization of available staff, and to ensure a high level of satisfaction among the employees, sophisticated scheduling methods are required. This thesis has its background in operations research, which among other things, is concerned with the development of advanced scheduling methods and with a structured approach to optimization of complex planning problems.
The thesis contains six individual scientific papers along with a summary of the most important contributions and conclusions. A number of industrial applications in rostering and task scheduling are presented with emphasis on generalized rostering and task scheduling with temporal dependencies between tasks. The applications exist within various contexts in health care, the aviation industry, transportation, and production. Important contributions include the development of a versatile approach to generalized rostering, building on an idea of compile-time customization. Several extensions of regular rostering problems are presented. For task scheduling, a general modeling of temporal dependencies is introduced and included in a well established solution methodology, termed column generation. Column generation is an iterative exact solution method based on the theory of linear programming. The approach is applied to several practical problems with promising results. Lastly, a novel approach to crane scheduling with superior results is presented.

ISBN 978-87-90855-84-0

DTU Management Engineering
Department of Management Engineering
Technical University of Denmark

Produktionstorvet
Building 424
DK-2800 Kongens Lyngby
Denmark
Tel. +45 45 25 48 00
Fax +45 45 93 34 35

www.man.dtu.dk