Models and Algorithms for Ground Staff Scheduling on Airports

Von der Fakultät für Mathematik, Informatik und Naturwissenschaften der
Rheinisch-Westfälischen Technischen Hochschule Aachen
zur Erlangung des akademischen Grades eines Doktors der Naturwissenschaften
genehmigte Dissertation

von
Diplom-Informatiker Jörg Herbers
aus
Haselünne

Berichter: Universitätsprofessor Dr. Drs. h.c. Hans-Jürgen Zimmermann
Universitätsprofessor Dr. rer. nat. Juraj Hromkovič
Universitätsprofessor Dr. rer. nat. Peter Rossmanith

Tag der mündlichen Prüfung: 14. März 2005

Abstract

The planning of airport ground staff gives rise to a number of challenging optimisation problems. Ground handling workloads are naturally represented as work tasks, e.g. for baggage unloading or passenger check-in. These workloads must be covered by appropriate employees. Staff scheduling is usually carried out in several stages: In *demand planning*, workloads are aggregated and analysed, in *shift planning*, appropriate shift duties are generated, and *rostering* consists in generating lines of duty for the workers. These phases are strongly interrelated, and different optimisation problems have to be solved at each stage.

Workforce scheduling models have traditionally built upon aggregate labour requirements given in discrete time periods. However, the literature does not describe any models or algorithms for the generation of appropriate workload representations. Additionally, it will not always be sufficient to cover coarse-grained abstractions of workloads. If information on flights as well as passenger and load figures are sufficiently exact, we will rather be interested in directly covering individual work tasks. Furthermore, shift scheduling and rostering approaches have regularly taken special assumptions or investigated simplified problems, limiting their practical applicability.

In this work, we tackle optimisation problems at different planning stages. We show how in the presence of movable tasks, we can obtain a suitable demand curve representation of workloads, using a levelling procedure which combines aspects from vehicle routing and resource levelling. Furthermore, we devise two algorithms for task-level shift planning which relates to vehicle routing and shift scheduling models. The first method is an improvement procedure, building upon the results of a construction phase and dealing with a complex shift planning setting. The second algorithm focuses on a subclass of task-level shift planning and is able to solve many problems to proven optimality. Finally, we design an algorithm for complex cyclic rostering on the basis of aggregate workloads. The approach builds upon a novel model for representing flexible breaks and solves the shift scheduling and rostering stage simultaneously.

Models and algorithms proposed in this thesis are more integrated and tackle more complex settings than previous approaches. We employ modern constraint programming and integer programming solution techniques, including column generation and branch-and-price. For the novel optimisation problems treated in this work, we provide complexity results. All algorithms are evaluated on complex large-scale test cases from the practice of airlines, airports and ground handling companies.

Zusammenfassung

Die Planung von Bodenpersonal an Flughäfen beinhaltet eine Reihe anspruchsvoller Optimierungsprobleme. Das Arbeitsaufkommen für Abfertigungsdienste wird typischerweise in Form von Arbeitsaufträgen dargestellt, z.B. für die Gepäckentladung oder für Check-in-Dienste. Dieses Arbeitspensum muss durch geeignete Mitarbeiter abgedeckt werden. Die Planung wird üblicherweise stufig durchgeführt: In der *Bedarfsplanung* wird das Arbeitsaufkommen aggregiert und analysiert, in der *Schichtplanung* werden geeignete Schichtdienste generiert, und in der *Dienstplanung* werden Dienstpläne für die Mitarbeiter erstellt. Die einzelnen Phasen sind dabei eng verzahnt, und auf jeder Stufe müssen verschiedene Optimierungsprobleme gelöst werden.

Personalplanungsmodelle bauen traditionell auf aggregierten Bedarfszahlen auf, die in diskreten Zeitschritten angegeben werden. Für die tatsächliche Generierung einer solchen Bedarfskurvenrepräsentation sind in der Literatur allerdings keine Modelle oder Algorithmen beschrieben worden. Darüber hinaus ist eine Planung auf Basis grober Bedarfszahlen nicht immer ausreichend. Wenn hinreichend genaue Informationen über abzufertigende Flüge und Passagier-/Gepäckzahlen zur Verfügung stehen, ist man vielmehr daran interessiert, die einzelnen Arbeitsaufträge zu verplanen. Schicht- und Dienstplanungsansätze in der Literatur gehen zudem durchgehend von speziellen Annahmen aus oder behandeln vereinfachte Probleme, was ihre praktische Anwendbarkeit einschränkt.

In dieser Arbeit werden Optimierungsprobleme für verschiedene Planungsschritte gelöst. Es wird gezeigt, wie eine geeignete Bedarfskurvendarstellung unter Berücksichtigung verschieblicher Aufträge generiert werden kann, indem Elemente des Vehicle Routing und des Resource Levelling kombiniert werden. Darüber hinaus werden zwei Algorithmen für die auftragsbasierte Schichtplanung entwickelt, die auf Modellen des Vehicle Routing und des Shift Scheduling aufbauen. Die erste Methode ist ein Verbesserungsverfahren, das auf den Ergebnissen einer Konstruktionsheuristik basiert und ein komplexes Schichtplanungsproblem behandelt. Der zweite Algorithmus bezieht sich auf eine Teilproblemklasse und löst viele praktische Probleminstanzen beweisbar optimal. Schließlich wird ein Algorithmus für die Erstellung komplexer Schichträder auf Basis einer aggregierten Bedarfsdarstellung konzipiert. Der Ansatz baut auf einem Modell zur impliziten Darstellung flexibler Pausen auf und löst das Shift Scheduling- und Dienstplanungsproblem simultan.

Die Modelle und Algorithmen in dieser Arbeit sind stärker integriert und berücksichtigen komplexere Nebenbedingungen als frühere Beiträge. Moderne Techniken des Constraint Programming und der ganzzahligen Programmierung (einschließlich Spaltengenerierungs- und Branch-and-Price-Ansätzen) werden eingesetzt. Für die vorgestellten neuartigen Optimierungsprobleme werden Komplexitätsuntersuchungen durchgeführt. Alle Algorithmen werden auf großen, komplexen Testfällen aus der Praxis von Fluglinien, Flughäfen und Bodenverkehrsgesellschaften evaluiert.

Acknowledgements

This work has been carried out at the airport division of INFORM Institut für Operations Research und Management GmbH. Many people have contributed to this thesis in one way or the other.

First, I would like to thank my doctoral advisors. Prof. Dr. Drs. h.c. Hans-Jürgen Zimmermann initially made this work possible, and I am very grateful for his valuable support and feedback. Prof. Dr. Juraj Hromkovič was so kind to act as referee even if this entailed an additional burden with his new position at ETH Zürich. I would like to express my gratitude for his support and comments on earlier drafts of this work. Thanks also to Prof. Dr. Peter Rossmanith for acting as an additional co-referee.

Special thanks go to Adrian Weiler for leading me to this challenging field of research and for his interest in my work, and to Thomas Schmidt and Uschi Schulte-Sasse for their encouragement. I am particularly indebted to Werner Siemes who leads the planning development group at INFORM's airport division. Without his support, this work would not have been possible.

Torsten Fahle provided me with numerous comments and suggestions, and I am very grateful for the fruitful discussions. His extensive proof-reading down to the smallest indices has considerably contributed to the quality of this thesis. Special thanks also to Ulrich Dorndorf and Tuomo Takkula for discussions, suggestions and comments on earlier versions of this work. Christian, Daniel, Melanie and Wolfgang deserve my gratitude for further corrections and suggestions.

Christof, Dimitra, Julia, Simone and Tom were not only colleagues, but have become real friends, and their support has meant a lot to me. Thanks also to Andrea, Alex, Bernd, Frank, Holger, Melanie, Michael, Olli, Sunil, Clementine and Fich for the pleasant working environment, the motivating atmosphere and the guidance with the optimisation problems and the planning system. Christian was an inspiring partner for lunchtime discussions during my short return to student life.

The staff at Dash Optimization was very ambitious in providing me with suggestions for tuning the cyclic roster algorithm.

My sincere gratitude goes to my parents who made this possible, for supporting and encouraging me during the years. Special thanks also to my brothers Jens and Ulf for their support.

Last but not least, special thanks to Katrin for her patience, love and encouragement throughout all ups and downs of my work.

Contents

List of Figures

List of Tables

1. Introduction

The best scientist is open to experience
and begins with romance
– the idea that anything is possible.
— Ray Bradbury

1.1. Airport Ground Handling

Today’s airlines and airports are facing an enormous cost pressure. Several crises have adversely affected the business, including September 11th, war in Iraq, SARS and the world economic situation. Insolvencies of major carriers reflect economical problems which are partly self-inflicted and partly due to external factors. Increasing oil prices are a new threat to the recovery of the airline business. In spite of this, business has recently regained growth rates of earlier years. In the first three quarters of 2004, worldwide air transport passenger kilometres have increased by 17.7% and freight tonne kilometres by 14.1% over the preceding year [IATA, 2004a]. Low cost carriers and new markets in Asia are the main drivers of growth. By 2010, the International Air Transport Association (IATA) expects 2.2 billion passengers per annum which is 600 million more than in 2003 [IATA, 2004b].

In view of new competitors and the growth of the air transport market, airlines try to cut their costs and increase competitivity. Beneath fuel costs and crew personnel, ground handling activities represent a major cost factor [Brusco et al., 1995]. Handling tasks can be distinguished by planeside (ramp) operations and passenger services [Ashford et al., 1997]. Tasks on the ramp include baggage handling, aircraft cleaning, refuelling, water services, bus transportation, cargo and catering, load planning and control, traffic control, towing and de-icing. Passenger services mainly refer to check-in, boarding, ticketing, help desks, sales reservation offices and backoffice activities. Beyond these basic tasks, aircraft maintenance and security checks must be considered. At their base airports, airlines usually accomplish these tasks by their own personnel. Alternatively, airports and ground handling companies offer ground handling services to airlines. With increasing deregulation, international handling companies with worldwide subsidiaries see their market shares growing.

Clearly, efficient planning of staff is crucial in controlling costs. Due to the size of airport operations, even small improvements translate into large savings. As an example, Holloran and Byrn [1986] reported savings of approximately six million dollars by the introduction of an automated staff scheduling application at United Airlines. Further eight million dollars per annum have been saved by the introduction of an improved system in 1994 [Brusco et al., 1995].

Ground staff scheduling is a very complex task. Planning frequently involves several hundred employees, several thousand work tasks per week and multitudes of constraints. Airports often work on a continuous basis with operations on 24 hours, seven days a week. Workforce demands are subject to high variations at different times of the day. Airlines often use the hub-and-spoke principle, meaning that connecting flights from smaller airports are bundled at larger stations in order to minimise passenger transfer times for long-haul connections. This practice as well as commuter activity of business people result in three, four or even more pronounced peak times of high workload within each day (see Fig. 1.1) [Ashford et al., 1997].

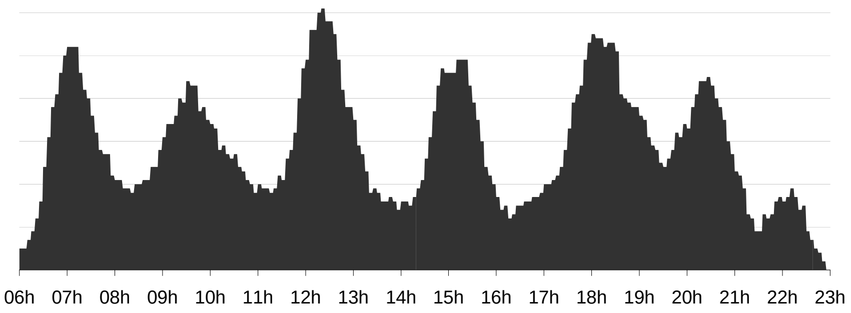


Figure 1.1.: Typical workloads at an airport over the day.

Manual scheduling is a tedious and error-prone task, and it frequently takes planners several hours or even days to prepare staff plans by hand [Dowling et al., 1997] [Yu, 1998]. Processes are difficult to overview by planners, and different resources are tightly coupled [Ashford et al., 1997, p. 185] [Yu, 1998, p. xiv]. At large airports, it is normally impossible to handle staff without computer support. All airlines, airports and ground handling companies nowadays make use of scheduling systems. In the literature, several airline-specific systems have been reported, e.g. at Air Canada [Nobert and Roy, 1998], Alitalia [Felici and Gentile, 2004], Pan Am [Schindler and Semmel, 1993] and United Airlines [Holloran and Byrn, 1986] [Brusco et al., 1995].

Much progress has been made since the development of the first ground staff planning systems. Scheduling software is nowadays required to flexibly adapt to changing work rules, problem characteristics and sizes of operations [Dowling et al., 1997]. Clearly, the development of advanced planning systems is itself a costly investment. Many airlines, airports and ground handling companies therefore resort to generic software packages, including advanced optimisation components and powerful graphical user interfaces, see e.g. Khoong and Lau [1992], Khoong et al. [1994] and Dowling et al. [1997].

This work deals with offline aspects of ground staff planning, i.e. the estimation of workloads and the generation of staff plans several days or weeks before operations. This means that workloads for the planning period are calculated from a fixed flight schedule, forecasted passenger and baggage figures and service agreements between airlines and ground handling departments or companies. We will not deal with real-time aspects, i.e. issues like short-term schedule changes or interruptions due to flight delays or illnesses will not be considered.

The optimisation models and algorithms developed in this work are generic and apply to a wide range of ground handling activities. However, some services may not conform to the general framework and may therefore not be covered (e.g. cargo handling, catering and bus transportation). As different ground handling activities (like baggage or check-in services) still differ considerably, we aim at conceiving algorithms which are robust on a wide range of relevant planning scenarios. However, we will also show how problem-specific features can sometimes be exploited by special solution approaches.

This work has been carried out as a part of an appointment of the author to INFORM Institut für Operations Research und Management GmbH, Aachen, Germany. INFORM markets a comprehensive suite of airport applications named GroundStar which is in use at major customers all over the world. Airport users e.g. include Berlin, Frankfurt and Moscow, ground handling companies comprise Aviapartner, DNATA, ServisAir GlobeGround and Swissport, and airline customers are e.g. Air Canada, Air France, British Airways, Emirates, Lufthansa and SAS. All optimisation algorithms described in this work are part of the staff planning application of GroundStar.

In the following, we will give a short overview of planning processes and objectives in airport ground staff planning. We will point out how workforce scheduling is treated in the literature and

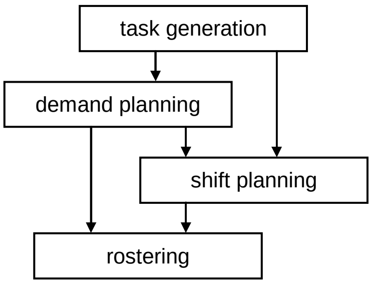


Figure 1.2.: Planning stages in airport ground staff planning.

show the relationship of airport scheduling problems with vehicle routing. Finally, we will give a survey on contributions of this thesis and describe its structure.

1.2. Planning Processes and Objectives

Planning of ground staff involves several stages (Fig. 1.2). First, we generate work tasks based on the flight schedule for the planning period, forecasted load figures and service agreements. These work tasks represent the input of a demand planning phase in which temporal demands are analysed in their temporal evolution and with regard to peak times. In a second stage, workloads are covered by suitable shift duties. Finally, rosters are built, specifying shifts and days off for the workforce at hand. Roster generation takes either original workloads or given shift plans as input. Virtually all airport services adhere to this general scheme.

Planning can take place on different levels. In *operative planning*, we aim at actual staff plans for the day of operations. If operative planning is carried out few days or weeks ahead of time, information on flights is generally quite accurate. However, rosters are sometimes prepared for a whole flight season in advance, meaning that planning is typically based on a model week with typical flight events and aircraft loads. *Tactical planning* e.g. involves calculating prices for different services, analysing work rules as a preparation for union negotiations, and what-if scenarios when acquiring new customers. On a *strategical level*, managers will e.g. be interested in cost calculations when extending business to new airports. Tactical and strategical plannings are usually based on model weeks of typical workloads.

In the following, we describe the planning processes in more detail and give annotations on the significance of planning stages under different conditions.

1.2.1. Task Generation

Figure 1.3 shows the basic scheme of task generation, see also Dowling et al. [1997]. A first input is the *flight schedule*, covering flight events that are relevant to the ground handling company or department. Each flight is made up of an *arrival* at the airport under consideration, a *departure* and the *groundtime* in between. Alternatively, flights may consist of an *arrival leg* or *departure leg* only, e.g. if the aircraft stays at the airport for a longer period of time. Flight schedules often contain positioning information, e.g. with regard to aircraft stands, arrival gates and check-in counters.

Task generation often depends on *passenger and baggage* information. While exact load information may be available if planning takes place shortly before operations, averaging or other statistical methods can be used to generate reliable load information ahead of time.

Engagement standards represent task generation rules matching on specific flight events, e.g.

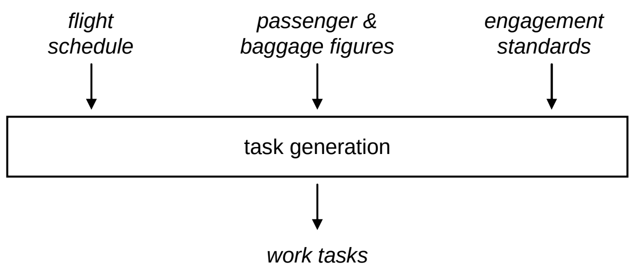


Figure 1.3.: Task generation.

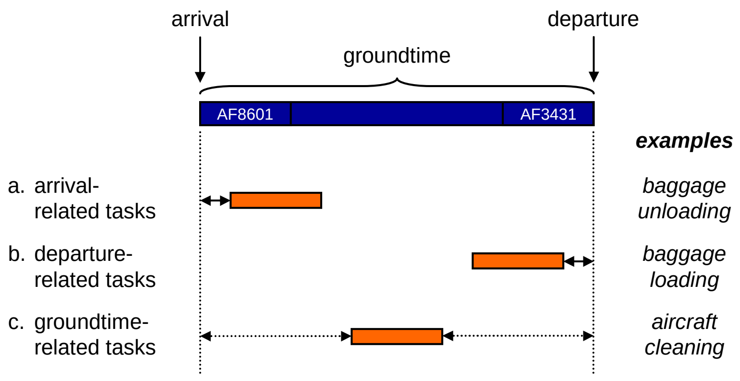


Figure 1.4.: Work tasks relating to aircraft arrivals, departures and groundtimes.

by airline, aircraft type, routings, intervals of baggage loads etc., see also Alvarez-Valdez et al. [1999], Sørensen and Clausen [2002]. As an example, an engagement standard may define the generation of eight cleaning tasks of 45 minutes each for a Boeing 747-400, to be carried out within the groundtime of the aircraft.

Especially tasks on the apron will usually pertain to *single flight events*, i.e. they can be attributed to either arrival, departure or the link of both legs. Times at which tasks must be carried out either refer to the arrival, departure or the groundtime of the aircraft (Fig. 1.4). As an example, an arrival flight may ask for a number of baggage unloading tasks, starting five minutes after touchdown and whose quantity and durations depend on baggage figures. In contrast, the above cabin cleaning tasks would refer to the groundtime of the aircraft.

Alternatively, work tasks can be generated for *multiple flights* in common. A typical example is check-in personnel which often serves groups of flights, e.g. StarAlliance Economy passengers. Workloads then result from the numbers of passengers to be served by a group of flights. *Arrival profiles* specify how many passengers are expected to arrive in intervals relative to the departure time. *Queueing models* are then employed to calculate counter and personnel demands, using information on typical *service times* per passenger and acceptable lengths of *passenger queues* [Schindler and Semmel, 1993] [Brusco et al., 1995]. Workloads can finally be broken down into work tasks for check-in personnel. Alternatively to the aforementioned check-in example, tasks may pertain to several flights if e.g. baggage unloading at adjacent aircraft stands can be done by the same staff. Task start times and lengths will then refer to the respective flight events.

A third class of work tasks does not make any reference to flight events. As an example, the number of staff at ticket and information counters does not depend on specific flights. Work tasks will then be specified by absolute start times and durations corresponding to the opening hours of the respective service.

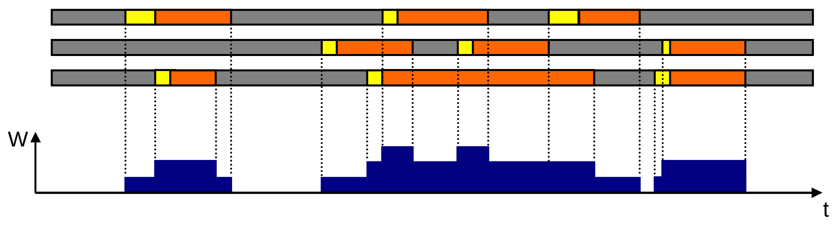


Figure 1.5.: Tour planning and demand curve.

Regardless of the mode of task generation, the resulting work tasks have a number of properties. As mentioned before, tasks have fixed lengths and either start at fixed times (e.g. baggage unloading tasks) or within given time windows (e.g. cabin cleaning tasks). They must be carried out at given positions, e.g. at an aircraft position on the apron or at given check-in counters. Tasks may require workers to have certain qualifications, e.g. for handling a specific aircraft or language skills for check-in and boarding. Cleaning tasks must often be carried out by teams of workers [Stern and Hersh, 1980].

Task generation is the most basic step in airport staff planning. Throughout this work, we will assume that work tasks for a given department are readily available.

1.2.2. Demand Planning

Workloads are usually visualised as a *demand curve*, i.e. a histogram of parallel tasks at each time of the planning horizon. As each work task requires one employee, this representation allows for an easy analysis of workforce requirements. Times of high workloads are easily analysed, and planners are enabled to assess suitable shift duties covering more than one demand peak. Demand curves are also a common abstraction of workloads to be covered in the subsequent shift planning phase; workloads are then usually discretised in steps of 15 or 30 minutes, see e.g. Brusco et al. [1995].

Simply superposing tasks is generally not sufficient for generating demand curves. On the one hand, it would not be clear how to fix movable tasks. Furthermore, start time decisions for different tasks are generally interdependent since a worker cannot cover more than one task at a time. Especially at peak times, tasks will block each other. On the other hand, we must consider that workers must travel between different work task locations. If we imagine distances between locations on the apron, such travel times may considerably contribute to workloads.

A model covering tasks by tours is therefore more appropriate, see Fig. 1.5 (work tasks are orange, travel times are displayed in yellow). Each tour can be interpreted as a sequence of tasks that must be carried out by a vehicle (e.g. a push-back tractor or baggage vehicle) or by one or several workers in turn. Superposing tasks in tours allows for an incorporation of mutual interdependencies between tasks and a consideration of travel times (lower part of Fig. 1.5). Clearly, minimising the number of tours corresponds to solving a resource investment problem, i.e. we determine e.g. the minimum number of required vehicles [Brucker et al., 1999].

Especially in staff planning, we will be interested in avoiding unnecessary demand peaks. This can be achieved by placing movable tasks at times of low workload and by avoiding travel times. In general, a smoother demand curve will also provide a better basis for demand-based shift planning as described in the following section. The associated problem of smoothing labour requirements will be called the *workload levelling problem*.

In the literature, authors have consistently assumed that work tasks are fixed in time and ignored travel times between different locations, see e.g. Schindler and Semmel [1993], Brusco et al. [1995] and Dowling et al. [1997]. Only Nobert and Roy [1998] deal with the levelling of air cargo workloads. The levelling of labour requirements that are given by movable tasks at different locations has not yet been described in the literature.

1.2.3. Shift Planning

Shift planning amounts to covering workloads by suitable and cost-efficient shift duties. On airports, employees who are assigned the shift duties are usually not known in the planning phase, i.e. shift planning is *anonymous planning*. The alternative replanning of given shifts or rosters with given staff information is less difficult and will not be part of this work. Anonymous planning e.g. implies that qualification requirements can only be considered on an aggregate level.

Shift types are a basic notion in ground staff planning. Each shift type defines a valid starting and ending time for a shift duty and prescribes one or several meal or relief breaks, see e.g. Koop [1988]. A given shift type may be valid on some or all days of the week. Shift type costs reflect actual labour costs or penalties for undesirable shifts, e.g. night shifts. Each shift used to cover labour demands thus incurs costs as given by its shift type. Airport shift planning usually involves between 10 and 600 shift types. Due to strongly varying demands throughout the day, workloads can be covered more efficiently if the shift model allows for much scheduling flexibility [Bailey and Field, 1985] [Brusco, 1998].

Shift types and break rules result from state and federal laws, union agreements, company policies or practical considerations [Buffa et al., 1976] [Aykin, 2000]. In the European Union, the working time directive 93/104/EC of 23 November 1993 with amendments 2000/34/EC, 2000/79/EC and 2002/15/EC [EU, 1993] [EU, 2000a] [EU, 2000b] [EU, 2002] gives a framework for admissible shift models. Especially relevant to the ground handling sector is the Council Directive 2000/79/EC of 27 November 2000 [EU, 2000a], concerning the organisation of working time of mobile workers in civil aviation.

The workforce of airlines, airports and ground handling companies is usually made up of full-time and part-time employees. The base workload is then covered by full-time staff with typical shift lengths of between eight and eleven hours and up to three meal and relief breaks [Bechtold, 1988]. Planners often try to use full-time shifts which cover at least two peak periods within the day. In contrast to full-time staff, the use of part-time employees is often restricted by union regulations. However, part-time shifts of typically four or five hours length are known to be important for flexibly covering labour requirements [Holloran and Byrn, 1986] [Schindler and Semmel, 1993].

When generating a shift plan for given workloads, different overlapping shift types compete in covering workloads. If an airport operates 24 hours, 7 days a week, shift types not only overlap within the day, but also spill over between days. Shift planning problems for such continuous operations are therefore more complex [Tien and Kamiyama, 1982].

Shift planning can either be based on workloads as given by a demand curve, or it can be directly based on work tasks [Ernst et al., 2004]. *Demand-level shift planning* aims at covering workforce requirements per time period (e.g. 15 minutes) by a cost-minimal set of shifts (Fig. 1.6). This view of shift planning is well-established in the practice of planning on airports as well as in other service industries, see e.g. Tien and Kamiyama [1982], Holloran and Byrn [1986], Schindler and Semmel [1993] and Brusco et al. [1995]. The assignment of actual work tasks may then be deferred to shortly before operations [Dowling et al., 1997]. Demand-level shift planning is appropriate when workloads are not exactly known, e.g. due to missing information on load data or flights to be handled. Clearly, aggregate planning also reduces problem complexity and can be a precondition for tractable optimisation models. Additionally, demand-based models often allow for additional scheduling flexibility. As an example, planners usually do not fully cover workloads in peak times. Such *understaffing* is easily accounted for in histogram-based models [Thompson, 1993].

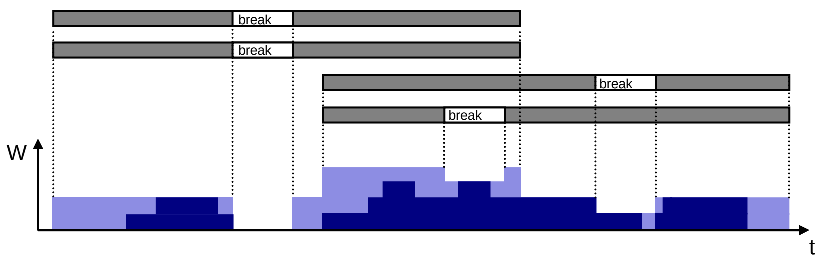


Figure 1.6.: Demand-level shift planning.

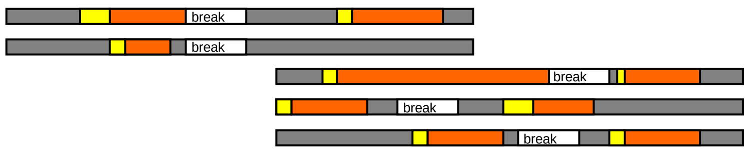


Figure 1.7.: Task-level shift planning.

In contrast, *task-level shift planning* (see Fig. 1.7) is more detailed and therefore more complicated. In task-level planning, shifts represent tours of tasks, starting and ending at a depot and respecting travel times between the tasks. Each task must start within its time window, and tours must fit into the boundaries imposed by shift types and lunch and relief breaks. Additionally, we may have to respect qualification constraints, e.g. limiting special skill requirements to few shifts. Tasks for check-in services or ticket counter personnel often span over the whole day and must be splitted in order to be assigned to shifts. Furthermore, tasks and shifts for teams may have to be placed in parallel.

Note that in demand-level planning, we assume task start times and travel times to be realised as in the demand planning phase. Furthermore, qualification requirements cannot be represented in a single demand curve. Decisions for shift types (and consequently, shift start and end times) do not take into account that tasks may not be splittable at the boundaries of shifts. However, if tasks are little movable, travel times are moderate, qualification requirements are rather homogeneous and tasks are splittable, shift planning based on a demand curve will provide a good approximation to task-level scheduling. In practice, planning scenarios often come close to these requirements.

The decision for demand-based or task-based models depends on a number of factors. If the quality of workload information is high and planners require high-resolution schedules, shift plans should make reference to single work tasks. If the planning is carried out weeks before actual operations, demand-based models will offer sufficient degrees of detail. In this thesis, we argue that demand-based and task-based models can both be appropriate in different situations, and we will cover both types of models.

It is worth mentioning that task generation and demand planning are not always independent from shift planning [Henderson and Mason, 1998]. If demands e.g. result from passenger arrivals and queueing models, moving demands to later time periods may render the coverage of workloads easier [Mason et al., 1998]. In general however, task generation can be separated from shift planning without major repercussions. Throughout this work, we will assume that tasks can be generated without any reference to the shift level.

	Mon	Tue	Wed	Thu	Fri	Sat	Sun
employee 1	05:00 13:30	05:00 13:30	05:00 13:30	05:00 13:30	05:00 13:30	–	–
employee 2	14:15 23:15	14:15 23:15	14:15 23:15	–	–	11:30 20:00	11:30 20:00
employee 3	–	–	08:00 17:30	08:00 17:30	08:00 17:30	09:00 18:30	09:00 18:30

Figure 1.8.: Weekly roster for three employees.

1.2.4. Rostering

The rostering problem consists in sequencing shift duties and days off into patterns for individual employees or groups of employees, cf. Fig. 1.8. Union and legal regulations (e.g. the aforementioned EU directives) and company policies impose a number of constraints. The most frequent work rules include

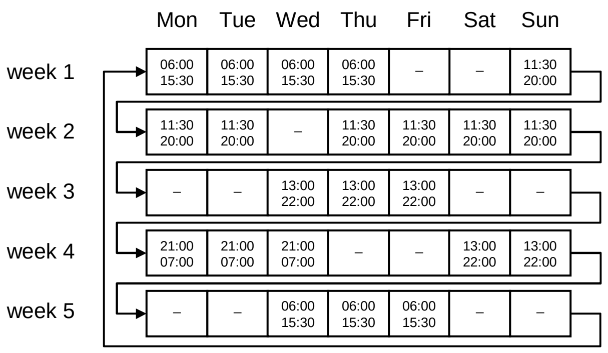
- minimum and maximum bounds on the number of consecutive days on and days off,
- minimum rest times and start time differences between consecutive shifts,
- minimum, maximum and average working hours per week and
- the number of weekends off per period,

see e.g. Bechtold and Showalter [1987], Lau [1994] and Dowling et al. [1997].

Different kinds of rosters are used in practice. Figure 1.8 shows *individual rosters* with a separate schedule for each employee. If concrete employees for each roster line are known (*named* rostering), we can take employee availabilities, preferences and qualifications into account [Thompson, 1996b]. In airport practice, rosters are usually planned on an *anonymous* basis. Schedules are then assigned to employees in a subsequent stage. In North America, anonymous rosters are often published for *bidding*, i.e. employees choose roster lines in decreasing order of seniority, see e.g. Holloran and Byrn [1986] and Schindler and Semmel [1993].

Alternatively, rosters can be *cyclic* (equivalently, *rotating*) and apply to a group of workers (Fig. 1.9). One employee will then start on each roster week, i.e. the length of the roster is equal to the number of employees. When finishing the first week, employee 1 switches to week 2, employee 2 to week 3 etc. while the worker on the last roster week starts over with the first week. Cyclic rosters are always anonymous; only the employee group for the whole roster may be known. Rotating shift patterns are usually planned for a whole flight season in advance. Clearly, they are less flexible as they do not adapt to fluctuating demands and absenteeism [Warner, 1976]. However, cyclic rosters provide a maximum degree of fairness since all employees work on the same pattern. Furthermore, employees can easily foresee their shift duties on given days in the future. Rotating rosters are often used to cover the bottomline demand by full-time workers. Holidays, short-term absences and changes in workloads are considered only later, e.g. by shift swaps and part-time staff [Dowling et al., 1997].

Rostering often builds upon given shifts determined in a preceding shift planning phase. If however significant restrictions are imposed on the roster level, it may not be possible to create efficient rosters upon given shift demands, and it will be more appropriate to integrate shift planning and rostering. In airport practice, individual rosters are often planned manually or with strong user interaction on the basis of given shift plans. Similarly, cyclic rostering is still little automated and usually amounts to applying small changes to rosters of the preceding flight season.



	Mon	Tue	Wed	Thu	Fri	Sat	Sun
week 1	06:00 15:30	06:00 15:30	06:00 15:30	06:00 15:30	–	–	11:30 20:00
week 2	11:30 20:00	11:30 20:00	–	11:30 20:00	11:30 20:00	11:30 20:00	11:30 20:00
week 3	–	–	13:00 22:00	13:00 22:00	13:00 22:00	–	–
week 4	21:00 07:00	21:00 07:00	21:00 07:00	–	–	13:00 22:00	13:00 22:00
week 5	–	–	06:00 15:30	06:00 15:30	06:00 15:30	–	–

Figure 1.9.: Cyclic roster for five employees.

Like shift planning, rostering can have different objectives. On an operative level, planners will be interested in efficient rosters for the workforce at hand. Tactical and strategical planning will typically involve what-if scenarios and may e.g. assess necessary staff sizes or evaluate the suitability of different shift duties, see e.g. Holloran and Byrn [1986], Dowling et al. [1997]. Rostering models usually include penalty terms for inconvenient characteristics of schedules, e.g. with regard to undesirable shift sequences or days off [Koop, 1988] [Jaumard et al., 1998].

1.3. Related Work on Workforce Scheduling

Since the first proposition of an integer programming model for shift scheduling by Dantzig [1954], workforce scheduling has received considerable attention in the literature. Shift scheduling and rostering problems are tackled in many different service industries, including airports and airlines, railways, telephone companies, hospitals, emergency services, toll collection and banks. With only few exceptions, all publications aim at covering aggregated workloads given on an appropriate discretisation level (often 30 minutes), i.e. models are demand-based with regard to the above characterisation. Overviews of applications and models can be found in Baker [1976], Tien and Kamiyama [1982], Bechtold et al. [1991] and more recently in Ernst et al. [2004].

Staff scheduling problems are usually categorised into *shift*, *day-off*, and *tour scheduling* models [Baker, 1976]. *Shift scheduling* is the problem of specifying, for a given day, the starting times and durations of shifts assigned to employees, see Dantzig [1954]. Furthermore, it may include the problem of positioning meal and relief breaks within each shift [Bechtold and Jacobs, 1990].

Day-off scheduling involves the placement of days off into a cyclic or non-cyclic roster [Tibrewala et al., 1972]. Typically, target shift numbers per day are given, implying on which weekdays days off should ideally be placed. Besides demand coverage constraints, typical restrictions include bounds on the number of consecutive days on and the placement of day-off stretches and weekends off.

If only one shift type is used, day-off scheduling is equivalent to the overall task of building a valid roster. In the multiple-shift case, we additionally face a *shift assignment* problem that consists in attributing shifts to the gaps left after day-off scheduling [Bennett and Potts, 1968]. In shift assignment, we must usually respect minimum rest times between shifts and maximum offsets between start times on consecutive days on [Jacobs and Brusco, 1996] [Lau, 1996b].

Many scientists have adopted sequential approaches for the solution of the overall rostering problem: First, target shift numbers for different shift types are determined, then days off are scheduled and finally, shifts are assigned to the remaining positions in the roster, see e.g. Bailey and Field [1985], Balakrishnan and Wong [1990], Khoong and Lau [1992] and Lau [1996a].

However, we have already pointed out that the single problems are interdependent, and it may not be possible at all to find a valid roster for the shifts determined in the first step. Authors have therefore called for integrated formulations subsumed under the notion *tour scheduling* [Baker, 1976]. Tour scheduling thus involves the simultaneous determination of shifts and day-off positions in a roster subject to demand coverage constraints.

A number of publications have dealt with the generation of cyclic rosters, e.g. Bennett and Potts [1968], Laporte et al. [1980], Balakrishnan and Wong [1990] and Mason [1999]. To the knowledge of the author, all models in the literature have built upon demand figures per shift type, i.e. there have not been any integrated approaches for simultaneous shift scheduling and cyclic rostering.

Many scientists have used integer programming or linear programming based rounding to solve scheduling and rostering problems [Ernst et al., 2004]. In Chapter 2, we will given an overview of relevant workforce scheduling literature with a focus on models and solution techniques.

1.4. Relationship to Vehicle Routing and Scheduling

The levelling problem for workforce demands (Section 1.2.2) is based on placing work tasks in tours such that time window and travel time constraints are taken into account. We will initially minimise the number of tours which are required to cover the tasks. In a second phase, we will take levelling effects for movable tasks into account. The basic setting amounts to a *vehicle routing problem with time windows*, covering “customers” (work tasks) by a set of “vehicle routes” (tours). Similar objectives as in workload levelling have been considered in the literature on resource levelling [Brucker et al., 1999].

Similarly, task-level shift planning consists in designing tours of tasks such that each tour fits into a shift duty. Additionally to time window and travel time constraints, we must respect break and shift type constraints. Task-level shift planning can therefore be interpreted as an integration of vehicle routing and shift scheduling. Additionally, we have to deal with splittable (*preemptive*) tasks, crews and qualification constraints. Khoong et al. [1994] and Rekik et al. [2003] have called for an integration of workforce scheduling with vehicle routing, but no actual work has been done up to now. Related work includes the article by Bailey et al. [1995] who rudimentally integrated project scheduling with shift scheduling and by Berman et al. [1997] who integrated shift scheduling with workflow optimisation in a mail-processing facility. From a vehicle routing perspective, some scientists have imposed tour length restrictions to deal with maximum durations of driver duties [Toth and Vigo, 2001a] [Campbell and Savelsbergh, 2004].

Vehicle routing has been a very active field of research over several decades. It subsumes a class of problems in which a number of customers (here: tasks) must be served by a set of vehicles (here: workers, shifts), using an appropriate network for their movements [Toth and Vigo, 2001a]. Tours start and end at one or several depots. Note that in task-level shift planning, depots not only have a spacial character, but we can also interprete shift start and end times as depots which impose temporal restrictions. Different objectives are considered in vehicle routing, e.g. minimising routing costs or the number of vehicles [Bodin and Golden, 1981]. Clearly, this is similar to minimising shift costs in task-level scheduling. Furthermore, qualification restrictions in shift planning can be interpreted as a special case of vehicle capacity restrictions.

While the *vehicle routing problem with time windows* (VRPTW) usually involves vehicle loads and capacity restrictions [Cordeau et al., 2001a], VRPTWs without capacity restrictions are usually denoted as *multiple travelling salesman problems with time windows* (m-TSPTW) [Desrosiers et al., 1993]. If the start times of customer visits are fixed, we deal with *vehicle scheduling problems* (VSP) or *capacitated vehicle scheduling problems* (CVSP) [Bodin and Golden, 1981] [Desrosiers et al., 1993].

Dantzig and Ramser [1959] were the first to propose a mathematical programming formulation for a real-world vehicle routing application. Clarke and Wright [1964] improved upon the Dantzig-Ramser approach and proposed their classical savings algorithm. Overviews of vehicle routing models and algorithms are given by Bodin et al. [1983], Christofides [1985] and in the excellent article of Desrosiers et al. [1993]. Further in-depth treatments can be found in the books of Golden and Assad [1988] and more recently in Toth and Vigo [2001b]. Complexity results on different vehicle routing and scheduling problems are summarised by Lenstra and Kan [1981].

A large variety of solution techniques have been applied to vehicle routing and scheduling problems, including heuristics (e.g. Solomon [1987]), local search (e.g. Potvin and Rousseau [1995]) and metaheuristics (e.g. tabu search [Gendreau et al., 1994] [Potvin et al., 1996], evolutionary programming [Potvin and Bengio, 1996]), integer programming (e.g. Desrosiers et al. [1984], Desrochers et al. [1992]), constraint programming (e.g. Kilby et al. [2000]) as well as hybrid integer/constraint programming approaches (e.g. Rousseau et al. [2002]). Nearly ten years ago, Laporte and Osman [1995] have already collected 500 references in the context of vehicle routing and scheduling. We will therefore restrict ourselves to an overview of constraint-based approaches in Chapter 3.

1.5. Contributions of the Dissertation

The goal of this work is to develop models and efficient algorithms for different stages of airport ground staff planning. The focus will be on models which are more integrated than former approaches. This is in accordance with a general tendency toward more integrated optimisation scheduling models in the literature, see e.g. Cordeau et al. [2001c], Cordeau et al. [2001b] and Sørensen and Clausen [2002]. Clearly, models which simultaneously deal with several planning stages potentially allow for realising additional savings.

However, we will not restrict our attention to task-based models, but argue that depending on the context, demand-level models may also be appropriate for shift planning and rostering. All models and algorithms will refer to anonymous scheduling, i.e. we do not explicitly assign shifts or rosters to employees. Depending on the context, we will deal with operational, tactical, or strategical planning scenarios with given or unknown staff sizes. In rostering, we will allow for planning with understaffing. Models and algorithms will be designed for potentially continuous 24×7 operations which clearly does not exclude exploiting the discontinuity of particular problem instances, e.g. due to night flying restrictions on some airports.

We will use state-of-the-art solution techniques to tackle these scheduling problems. Integer programming (IP) and constraint programming (CP) have received considerable attention in recent years and have proven to be powerful solution techniques for real-world problems. Integer programming has been the predominant solution techniques for demand-level shift scheduling and rostering models. Column generation is often used to decompose problems involving huge numbers of decision variables. While integer programming is a powerful solution technique for obtaining optimal or near-optimal solutions, constraint programming is particularly appropriate for very complex settings and non-linear objective functions or constraints. Throughout this thesis, both techniques will be used in different contexts.

We will start by conceiving a *levelling procedure* for demand planning involving movable tasks. While the basic setting is a vehicle routing problem with time windows, the goal of workload levelling can be compared to resource levelling problems in project scheduling. Workload levelling is a novel problem that has not yet been dealt with in the literature. It will be shown that the problem is $\mathcal{NP}$-hard. Based on the outcome of a tour-minimisation algorithm, we will conceive an constraint-based local improvement algorithm aiming at avoiding unnecessary demand peaks. For the search for improving solution, we will make use of a novel variant of a restricted tree traversal strategy.

Similarly, a constraint model will provide the basis for a local improvement algorithm for *complex task-level shift planning*. Integrating shift scheduling with vehicle routing, task-level shift planning is an optimisation problem which has not yet been treated in the literature. We will show how a large number of real-world constraints can be represented in a constraint programming model, including preemptive tasks, crew and qualification restrictions. Using large neighbourhoods, the local improvement algorithm will provide a remedy for deficiencies of a preceding heuristic construction algorithm.

In a further part of this thesis, we will show how *task-level shift planning* can be made amenable to *optimal solution* techniques by focusing on an important subclass. We will show that even a very basic shift planning setting is $\mathcal{NP}$-complete in the strong sense. For the problem considered, we will develop a column generation formulation by Dantzig-Wolfe decomposition. In conjunction with a preliminary problem decomposition technique, the resulting *branch-and-price algorithm* is able to solve a considerable number of real-world airport planning problems to optimality in a reasonable amount of time.

Turning to demand-based models, we will give an illustrative introduction to *implicit modelling* of *flexible break placement*. We will extend an existing model for the representation of flexible breaks, overcoming an important limitation that has not yet been explicitly dealt with in the literature.

The resulting model will provide the basis for a *branch-and-price algorithm* for *integrated cyclic rostering*. In contrast to previous models, our formulation will directly build upon workloads per demand period and integrate the shift planning and rostering stage. The setting includes a multitude of constraints which are relevant in ground staff rostering and considerably extends previous approaches. We will devise a novel IP formulation that is solved by multiple column generation and branch-and-price, yielding near-optimal solutions on real-world test cases.

All modelling approaches and algorithms will be motivated by literature reviews and introductions to solution techniques throughout the work.

All algorithms have been realised as a part of a commercial tool for generic ground staff planning. They are in use at many airlines, airports and ground handling companies or are about to be implemented at customer sites. The algorithms contribute to customer satisfaction, realise additional savings and allow for more flexibility in the scheduling of ground staff.

Some of the techniques in this work build upon solutions of heuristic algorithms which are not described in detail here. The immediate context of the algorithms in the planning system is illustrated in Fig. 1.10; contributions of this thesis are shaded.

Clearly, implementing the algorithms as a part of a commercial planning system has influenced design decisions. High-quality solutions are an important criterion, but robustness on a wide range of different planning scenarios was deemed at least as important. Computing times play a role especially when performing what-if analyses. Solution techniques which offer a tradeoff between running times and solution quality clearly provide an advantage. All algorithms were tested on real-world planning scenarios representing a wide range of different airport services. Test cases therefore incorporate quite different characteristics, e.g. with regard to task movability, skill requirements and scheduling flexibility. Problems often have a very large scale and frequently involve considerable numbers of constraints. The evaluation of the different algorithms will essentially be based on the same test cases, enabling for a recognition of common problem characteristics and comparisons between different solution techniques.

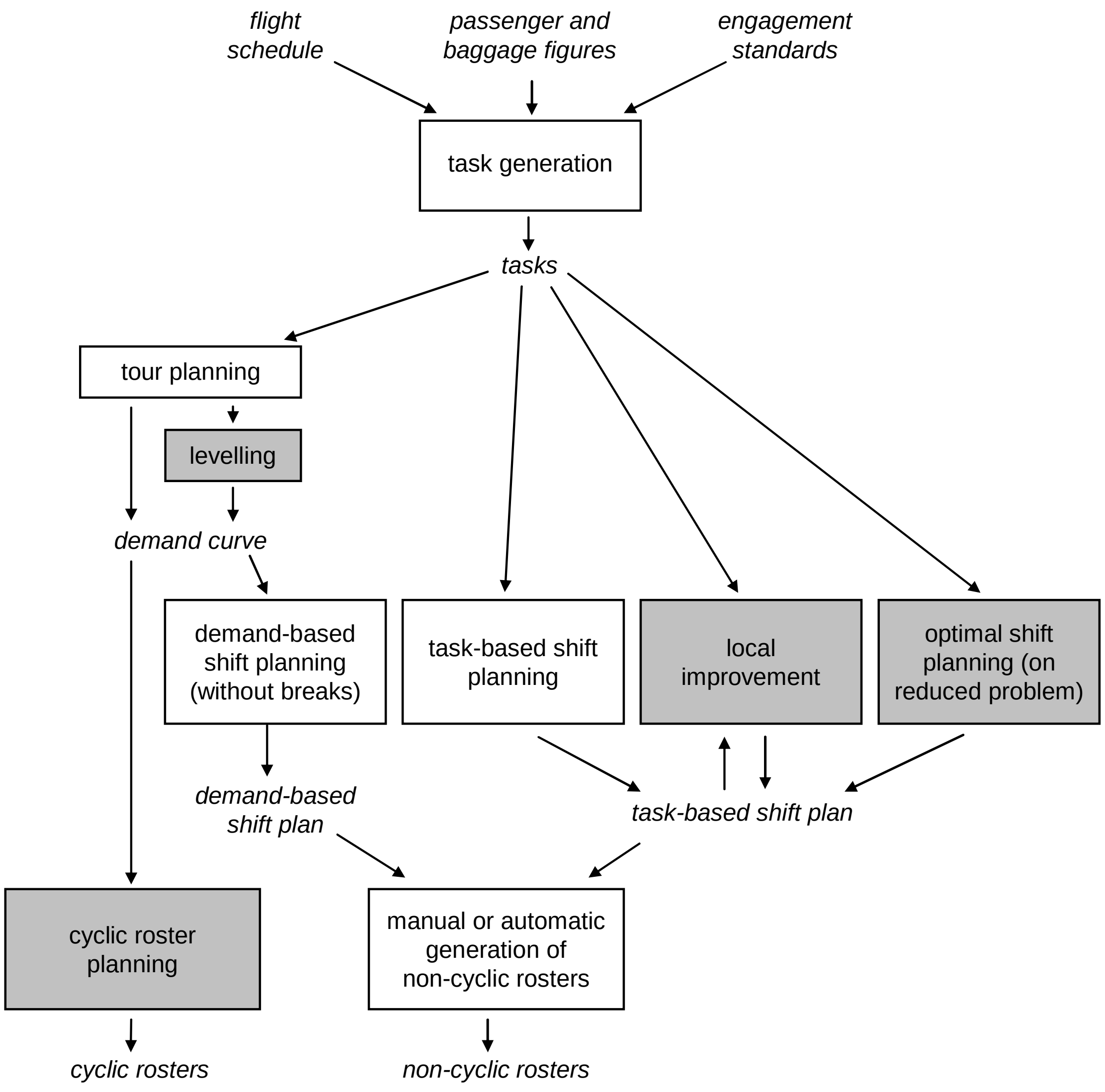


Figure 1.10.: Overview of contributed algorithms.

1.6. Structure of the Document

After having given an introduction to airport ground staff scheduling in this chapter, we first give a comprehensive overview of workforce scheduling literature in Chapter 2. A special focus is put on models and algorithms which are relevant to airport operations. We will furthermore draw analogies of general workforce scheduling to crew scheduling at airlines and public mass transport companies.

In Chapter 3, we will introduce constraint propagation techniques for the solution of constraint satisfaction and optimisation problems. Local search methods in constraint programming as well as modelling alternatives for CP-based vehicle routing will be reviewed, laying the ground for the two subsequent chapters.

Chapter 4 describes the constraint-based local improvement method for workload levelling in demand planning and proves the $\mathcal{NP}$-hardness of the problem. Different preprocessing techniques will be described to prepare the results of a preceding construction heuristic for levelling.

Complex task-level shift planning is the subject of Chapter 5. We will introduce the general problem and develop a mathematical model. A constraint programming model for the problem will be described, and it will be shown how problem-specific lower bounds can be derived. The resulting local search algorithms is demonstrated to be efficient on a large variety of test cases, providing a remedy for the flaws of a preceding construction heuristic.

Chapter 6 again has an introductory character. Dantzig-Wolfe decomposition principle and the resulting column generation and branch-and-price techniques will be described. Furthermore, we will point out the relationship to Lagrangian relaxation and give an overview of recent developments.

In Chapter 7, we will apply Dantzig-Wolfe decomposition to a subclass of the task-level shift planning problem described in Chapter 5 and solve the resulting model by branch-and-price. Additionally, we will prove that even a very restricted shift planning problem is $\mathcal{NP}$-hard in the strong sense.

While Chapter 2 already comprises a review of implicit models in IP-based scheduling, Chapter 8 gives an illustrative introduction to implicit modelling of break placement flexibility in demand-level scheduling. We will generalise an implicit model from the literature, overcoming limitations of previous approaches.

The resulting model provides the basis for the integrated cyclic rostering algorithm of Chapter 9. Reviewing different modelling alternatives and potentials for implicit models, we will develop a new model for the creation of complex cyclic rosters and devise adapted column generation algorithms and branching schemes.

Chapter 10 will give a summary of contributions and an outlook on future developments and research.

Chapters will partly build upon preceding explanations. As far as possible, we have used a common mathematical notation throughout the thesis. An overview of symbols and abbreviations can be found in Appendices A and B. Appendix C gives an overview of workforce scheduling publications.

2. An Annotated Bibliography on Workforce Scheduling

When the weight of the paperwork
equals the weight of the plane,
the plane will fly.
— Donald Douglas

2.1. Introduction

In Chapter 1, we have introduced basic notions of workforce scheduling. According to Baker [1976], workforce scheduling problems can be categorised as either *shift scheduling*, *day-off scheduling* or *tour scheduling* approaches. Shift scheduling deals with the determination of appropriate shift duties within a day in order to cover workforce requirements given on a suitable discretisation level, e.g. 15 or 30 minutes. Clearly, shift scheduling is trivial if shifts do not overlap [Baker, 1976] [Mabert and Watts, 1982] [Emmons and Fuh, 1997]. Day-off scheduling involves the determination of positions for days off in a cyclic or non-cyclic roster. If only one shift type is involved, day-off scheduling is equivalent to the overall task of building rosters. In the multiple-shift case, we need an additional *shift assignment* phase in order to match given shifts into the remaining roster gaps. Tour scheduling integrates these tasks by determining daily shifts and day-off positions simultaneously.

In several publications, it has been pointed out that integrated formulations can be important in order to avoid excessive labour costs [Buffa et al., 1976] [Easton and Rossin, 1991], see also the comparative evaluations of McGinnis et al. [1978] and Bechtold and Showalter [1987]. However, it should be noted that there is no need for an integrated formulation if there are few restrictions on valid shift transitions from one day to the other or if only one shift type is used [Holloran and Byrn, 1986]. For an overview of publications on different model types (shift, day-off and tour scheduling), the reader is referred to Appendix C.

Some publications have assumed the size of the workforce to be given externally, see e.g. Mabert and Raedels [1977], Bechtold [1988] and Thompson [1996b]. Other researchers initially solve a *workforce allocation* problem of determining the minimum staff size for the given workloads, see Panton and Ryan [1999] and the combinatorial approaches described in Section 2.3. However, a majority of publications has treated the determination of necessary workforce sizes as an objective in shift, day-off or tour scheduling, see e.g. Henderson and Berry [1976], Morris and Showalter [1983], Jarrah et al. [1994] and Brusco and Jacobs [2000].

Planning is often based on cyclic demands, representing exemplary workloads of a model day or week. Demands are then assumed to wrap around, i.e. the first demand period is assumed to follow the last scheduling interval [Çezik and Günlük, 2002]. Analogously, employee availability is frequently assumed to be cyclic with employees working on a repeating pattern [Morris and Showalter, 1983] [Jacobs and Brusco, 1996] [Brusco and Jacobs, 1998a]. As described in Section 1.2.4, cyclicity can also refer to the roster itself, meaning that the roster consists of a sequence of several weeks on which workers rotate (*cyclic rosters*) [Baker, 1976] [Mason, 1999].

Tour scheduling generally refers to the non-cyclic case (even if demands and employee availabilities can be cyclic), while publications on cyclic roster generation are more scarce. All cyclic roster approaches in the literature are non-integrative, i.e. they build upon demand figures per shift type.

As mentioned in Section 1.2.3, we can further distinguish *named* and *anonymous* scheduling approaches [Tien and Kamiyama, 1982] [Khoong et al., 1994]. Named scheduling implies that individual employees are explicitly assigned to shifts and tours, meaning that information on skills, availabilities and preferences can be taken into account , see e.g. Love and Hoey [1990]. In contrast, employee attribution is not part of anonymous scheduling. While named scheduling is e.g. frequent in nurse scheduling and airline crew rostering (see Sections 2.6 and 2.11), airport planners usually employ anonymous schedules. Clearly, cyclic rosters are always anonymous. Appendix C gives an overview of literature on named and anonymous scheduling.

Tour scheduling scenarios are frequently categorised as either *continuous* or *discontinuous*. Organisations working on a continuous basis are confronted with labour requirements on 24 hours a day, 7 days a week. Shifts for continuous operations usually overlap from one day to the other. In contrast, discontinuous models involve less-than-24-h operations. Discontinuity introduces a special structure into mathematical programming formulations of scheduling problems, meaning that such problems are often easier to solve [Jarrah et al., 1994] [Brusco and Jacobs, 1995] [Brusco and Johns, 1996].

Scheduling flexibility also varies with other factors. The workforce may be made up of only full-time employees (e.g. Morris and Showalter [1983], Bechtold et al. [1991], Brusco and Jacobs [1993b]) or a mix of full-time and part-time workers (e.g. Bennett and Potts [1968], Easton and Rossin [1991], Brusco [1998]). While in some organisations, it may be possible to start shifts at any time of the day within a given discretisation, other companies may only be able to use a small number of different shift types [Thompson, 1995] [Rekik et al., 2003]. Furthermore, one or several lunch and relief breaks may have to be considered at fixed times within the shifts or within given time windows [Aykin, 2000].

In tour scheduling, shifts on consecutive days may be constrained to equal start times and durations (see e.g. Laporte et al. [1980], Li et al. [1991]) while in other settings, start times are allowed to vary freely (e.g. Bailey [1985]) or within given bands (e.g. Jacobs and Brusco [1996]). Work rules may constrain days off to be taken consecutively or allow for single days off [Bechtold, 1981]. If shift durations are identical and the number of shifts per week is given, the number of working hours is fixed (see e.g. Glover and McMillan [1986], Thompson [1993] and Çezik et al. [2001]). In other problems, average working hours per week are explicitly taken into account, see e.g. Mason and Smith [1998].

Different degrees of flexibility are usually due to legal or union regulations. Additionally, special assumptions can be taken for computational reasons [Laporte et al., 1980] [Li et al., 1991]. However, Jacobs and Bechtold [1993] have empirically shown that higher degrees of scheduling flexibility often allow for a better labour utilisation.

Shift, day-off and tour scheduling can be represented by a generalised set covering formulation which is originally due to Dantzig [1954], see also Morris and Showalter [1983] and Bailey and Field [1985]:

$$\min \sum_{j \in J} c_j X_j \tag{2.1}$$

subject to

$$\begin{aligned} \sum_{j \in J} a_{ij} X_j &\geq d_i && \forall\, i \in I && (2.2) \\ X_j &\geq 0 \text{ and integer} && \forall\, j \in J && (2.3) \end{aligned}$$

In the context of shift scheduling, coefficients and decision variables have the following meanings:

X_j	number of employees assigned to shift j;
c_j	cost of shift j;
d_i	number of employees required to work in the i'th time period;
I	set of time periods in a day;
J	set of daily shift types;
a_{ij}	$= 1$ if time period i is a work period in daily shift type j (0 otherwise).

For day-off scheduling, parameters and variables are interpreted as follows:

X_j	number of employees assigned to day-off pattern j;
c_j	cost of day-off pattern j;
d_i	number of employees required on the i'th day of the week;
I	set of operating days per week;
J	set of day-off patterns to be considered;
a_{ij}	$= 1$ if day i is a work day in the day-off pattern j (0 otherwise).

In tour scheduling, variables and columns in the constraint matrix represent tours of shifts:

X_j	number of employees assigned to tour j;
c_j	cost of tour j;
d_i	number of employees required to work in the i'th time period;
I	number of time periods to be scheduled over the week;
J	number of tour types to be considered;
a_{ij}	$= 1$ if time period i is a work period in tour j (0 otherwise).

Most models for shift, day-off and tour scheduling implicitly or explicitly rely on this model, see e.g. Keith [1979], Morris and Showalter [1983], Brusco and Jacobs [1993a] and Aykin [2000]. As will become clear in the sequel, many authors have used integer programming (IP) approaches or linear programming based rounding heuristics to solve this problem, see also Ernst et al. [2004]. Several authors have noted that the aforementioned linear programming (LP) formulation generally exhibits advantageous integer properties (see e.g. Bailey [1985], Bailey and Field [1985]). Furthermore, the LP relaxation is known to be very tight and therefore lends itself well for integer solution methods by branch-and-bound [Henderson and Berry, 1976] [Morris and Showalter, 1983].

The following overview of workforce scheduling publications focuses more on models and solution techniques than previous surveys (like Ernst et al. [2004]). Furthermore, it concentrates on publications with relevance to airport ground staff planning. The presentation will mainly be structured by the aforementioned problem classes. Two modelling features have received attention on their own right and will be described separately: the modelling of flexible break placement as well as working subset approaches, aiming at reducing the number of used shift types. Furthermore, nurse scheduling is described as a special case of named scheduling. Complexity results in workforce scheduling problems are summarised in a separate section. At the end of the chapter, we will give annotations on similarities of general workforce scheduling with airline crew rostering and driver scheduling in public mass transport. Within each section, publications will be cited in chronological order if grouping contributions by affiliation is not more appropriate.

2.2. Shift Scheduling Approaches

Dantzig [1954] was the first to propose the above set covering formulation for shift scheduling of toll booth staff. He noted that using the simplex method often naturally yields integer solutions and proposed to round up fractional values in order to obtain feasible integer solutions. Furthermore, Dantzig called for integrating breaks into the formulation by setting the corresponding matrix coefficients to 0. Bennett and Potts [1968] used this explicit representation of breaks for shift scheduling of toll collectors with a given ratio of full-time and part-time employees. Aiming at minimising excess labour, the set covering model was solved by a multiple-step integer programming procedure.

Planning of telephone operator duties was a popular domain of application in the 1970's. Segal [1972] proposed a network flow model, using the out-of-kilter method. In his approach, labour demands are incorporated in lower bounds on so-called forward arcs while backward arcs represent feasible shifts. Breaks are scheduled in a second phase, possibly resulting in labour shortages which are added to the demands of the original problem. Buffa et al. [1976] described how to forecast telephone operator workloads, using historical data and a queueing model. Their heuristic shift scheduling algorithm allows for understaffing and overstaffing and used local estimations of deviations between operator availability and forecasted demands. Shifts are then heuristically assigned to operators, taking shift and day-off preferences into account. Henderson and Berry [1976] proposed an LP approach for telephone operator scheduling, using different rounding heuristics and local search. The formulation uses a *working subset* of shift types which are heuristically selected in a first step. In another LP approach to telephone operator scheduling, Keith [1979] allowed for two-level understaffing and overstaffing which is penalised in the objective function. He proposed to round fractional solutions to the nearest integer, subsequently applying several improvement steps.

Gaballa and Pearce [1979] used queueing models to forecast weekly workloads in sales reservation offices at Qantas. Daily shift scheduling problems are solved by integer programming, incorporating breaks by a distinct set of variables. Weekly rosters are built in a second step, again using integer programming. In another airline applications, Stern and Hersh [1980] described the scheduling of aircraft cleaning crews, using non-overlapping shift types and flexible breaks. In their setting, workloads for each job at an aircraft are given as man-hours to be worked by a crew of workers. The application leads to a special shift scheduling model to be decomposed and solved by integer programming.

In an interesting paper, Bartholdi et al. [1980] analysed a class of cyclic staffing problems, covering shift and day-off scheduling settings. Cyclic staffing problems lead to set covering formulations in which the columns (rows) of the constraint matrix contain cyclically consecutive ones. The authors showed how to solve this class of problems efficiently either by variable transformation and a network flow formulation or by linear programming and cumulative rounding. These results were generalised by Bartholdi III [1981] who presented another optimal LP-based round-off algorithm for column (row) circular matrices. For problems with non-consecutive ones in the constraint matrix, a quality bound was given in terms of the number of blocks of ones. Also based on this work, Karp and Orlin [1981] described the transformation of cyclic staffing problems into parametric shortest path problems and presented a polynomial-time solution algorithm. Vohra [1988] proposed a heuristic algorithm for cyclic problems allowing for more than one block of consecutive ones when blocks of ones have equal lengths. The quality of the heuristic depends on demand deviations from the maximum workload, ensuring optimality only if all requirements are equal.

In the context of shift scheduling with fixed breaks, Bailey and Field [1985] stressed the importance of using different shift lengths to cover workloads efficiently. The LP-based rounding algorithm of Bartholdi III [1981] was shown to be superior to the heuristic approach of McGinnis et al. [1978] proposed in the context of tour scheduling (see Section 2.5).

Love and Hoey [1990] described the development of a system for named shift planning in fast-food restaurants, taking employee availabilities into account and trying to balance skill levels between different working areas. The problem is decomposed into a set covering formulation for shift scheduling and an employee assignment phase, both of which are solved by network flow algorithms.

Shift scheduling for full-time and part-time staff at Pan Am was described by Schindler and Semmel [1993]. The model was solved by integer programming, considering lunch breaks in a second phase.

Thompson [1996b] developed a model for named shift scheduling with limited employee availabilities. A special set covering model with sets of shifts for each employee and multi-level penalties for understaffing and overstaffing was presented and solved by simulated annealing (SA). Experimentally evaluating different local search operators, the SA approach was shown to yield solutions which are only slightly worse than optimal IP solutions.

Berman et al. [1997] integrated the scheduling of shifts and workflow of a mail processing facility in an integer programming model. On the personnel scheduling level, full-time and part-time employees as well as breaks and qualifications were taken into account. Workflow was subjected to time window and capacity constraints.

In the airport cargo handling application of Nobert and Roy [1998], workloads are given as kilos of freight to be processed in given time intervals. The authors integrated the levelling of these workloads with shift scheduling into an integer programming formulation. Using test data from Air Canada, the levelling approach was shown to be superior to linear or no levelling.

Sørensen and Clausen [2002] experimentally investigated the effects of decentralising ground staff planning for British Airways operations at London-Heathrow airport. Their model integrates a so-called zoning problem with the allocation of aircraft stands and cyclic shift scheduling. While the solution approach uses simulated annealing on the zoning and stand level, the shift scheduling problems employ only limited numbers of shifts and can be solved heuristically. It was shown that increasing the number of zones and decentralising planning processes can lead to considerable excess workforce and a decline in the quality of stand allocations. By application of historical flight delay data, the robustness of stand allocations and staff plans was evaluated.

Musliu et al. [2004] described a weekly shift scheduling problem, avoiding an excessive use of different shift types over the week. The authors proposed a tabu search algorithm which exploits problem-specific knowledge. The algorithm is part of a commercial scheduling system.

2.3. Day-Off Scheduling Approaches

As mentioned before, day-off scheduling consists in determining the placement of days off in cyclic or non-cyclic rosters. Mabert and Raedels [1977] formulated a day-off scheduling problem for part-time bank tellers as a set covering model. In their formulation, workloads of different branches of the bank are aggregated, the resulting problem solved by integer programming and subsequently disaggregated. An alternative heuristic procedure was shown to be computationally advantageous, but inferior with regard to solution quality.

Based on this work, Bechtold [1988] presented a day-off scheduling model and heuristic algorithms for different goal sets, limits on the number of consecutive days on and employees working either at a single or at multiple locations. The algorithms were shown to outperform the alternative heuristic method of Mabert and Raedels [1977].

Most publications on day-off scheduling are more restrictive with regard to the given demand figures and the work rules considered. Many authors have proposed sequential approaches, starting by determining lower bounds on the workforce size. Afterwards, days off are assigned according to given rules, resulting in polynomial time algorithms for solutions using the minimum workforce size. Emmons and Burns [1991] call this the *combinatorial approach*. While early publications refer to only one shift type, the combinatorial approach has later been extended to multiple shifts. Since demand figures are given per shift type, these models are not integrated as in tour scheduling. Demands and employee availabilities are generally assumed to be cyclic.

In the setting of Tibrewala et al. [1972], variable demand figures are given for a single shift type over a week. The authors demonstrated how to arrive at a minimum workforce size, allowing for one or two days off per week. In the cyclic model of Baker [1974], variable shift demand figures are to be covered exactly by a mix of full-time staff with two consecutive days on per week and part-time staff available to work single shifts. Baker showed that a proposed tableau algorithm was always able to determine a solution using the minimum number of part-time staff.

Brownell and Lowerre [1976] studied necessary workforce sizes when demands for a single shift type are equal on all weekdays and on weekend days. With employees working five days each week, closed-form expressions for the minimum workforce size as well as optimal algorithms for different work policies (e.g. consecutive days off, every other weekend off) were presented. Lowerre [1977] refined these results, analysing the lengths of workstretches (number of consecutive days on) and adding two further policies for cyclic rosters. Baker and Magazine [1977] basically treated the same setting with only cyclic demands, but relaxed the condition of Brownell and Lowerre [1976] of weekend demands to be lower than weekday requirements. Burns [1978] allowed for a maximum of six days on as well as variable demands with every other weekend off in a cyclic setting.

Baker et al. [1979] treated the case of equal demands on all days with employees working five days each week and a maximum of six consecutive days on. Concentrating on an A-out-of-B weekends off constraint, they studied the consequences of different work policies on the workforce size. Bechtold [1981] allowed for arbitrary demands and presented algorithms for one or two days off per week. He showed that his algorithm is easier and more efficient than the comparable algorithm of Tibrewala et al. [1972]. However, it was noted that the latter algorithm yields better distributions of surplus labour while his own approach turned out to better disperse slack than the round-off procedure of Bartholdi et al. [1980]. In cyclic rostering, Emmons [1985] treated the case of a minimum workforce which is equal on weekdays and on weekend days, granting two days off per week, A out of B weekends off and workstretches of between two and four days. Burns and Carter [1985] extended their earlier studies on constructive algorithms by considering variable demands, five workdays per week, maximum workstretches of six days and A out of B weekends off in non-cyclic scheduling.

Workers of different skill rankings were considered by Emmons and Burns [1991], assuming that workforce requirements for each class of workers are constant throughout all days of the week. Employees were granted A out of B weekends off. Their method first determines necessary workforce sizes, then distributes weekends off and other days off and finally assign shifts, minimising the use of highly-qualified staff. A last single-shift model for non-cyclic scheduling was presented by Emmons and Fuh [1997]. Labour requirements were assumed to be constant throughout weekdays and weekend days. Two types of part-time workers were considered, the one entailing lower costs than full-time employees and a more expensive but unlimited pool of part-time staff.

Burns and Koop [1987] were the first to extend previous combinatorial approaches to a multiple shift case with three non-overlapping shifts and equal demands on weekdays and lower but equal demands on weekends. Shift changes within workstretches were disallowed and A out of B weekends off granted. Based on closed-form expressions for the minimum workforce size, their algorithm builds rosters on the basis of so-called *modules*, i.e. predetermined patterns of one or several weeks. The slots of these modules are then filled by appropriate shift duties. Koop [1988] studied another cyclic roster setting with multiple shifts and shift change restrictions. It was shown how to determine lower bounds on the workforce size. The author proposed a network model for cyclic rostering and formulated conditions for the existence of a circulation. Additionally, some special cases of shift change constraints were considered.

Hung considered a similar multiple-shift case with equal demands on weekdays and weekend days for each shift [Hung, 1993] [Hung, 1994a] [Hung, 1994b]. As in other combinatorial approaches, closed-form expressions for the minimum workforce size as well as constructive optimal algorithms were given for three [Hung, 1993], four [Hung, 1994a] and alternating three/four [Hung, 1994b] shifts per week. These approaches were extended by Burns and Narasimhan [1999], providing control over maximum lengths of workstretches. Optimal workforce sizes were determined, days off allocated and finally shifts assigned without considering restrictions on shift successions.

All combinatorial approaches are restricted to special cases and usually do not generalise to different work policies. Emmons and Burns [1991] note that "an integer programming formulation can more easily incorporate additional constraints, and be extended to more general models." However, combinatorial models have often incorporated constraints which have not yet been represented in full detail in more general solution approaches.

2.4. Shift Assignment Approaches

Shift assignment is concerned with the attribution of shifts of given types to the gaps left after day-off scheduling. Clearly, it only applies to sequential approaches which decompose the rostering problem into a day-off and shift assignment phase. Two of the aforementioned publications already comprise a shift assignment part, namely Emmons and Burns [1991] and Burns and Narasimhan [1999].

Van den Berg and Panton [1994] investigated two shift assignment problems for cyclic rosters. One problem constrains shifts on consecutive days to have equal types while the other allows for forward-rotating assignments, i.e. shifts rotate from earlier to later start times. Different existence conditions for such assignments were presented and subproblems solved by heuristics and network flow algorithms. Lau investigated a similar problem denoted as *changing shift assignment problem*, prescribing valid shift transitions by a matrix. While the overall problem was shown to be $\mathcal{NP}$-hard [Lau, 1996b], greedy and network flow approaches were proposed for some easier subclasses in Lau [1994], Lau [1996a] and Lau [1996b].

In a named scheduling model, Jackson et al. [1997] compared different approaches for assigning given shifts with qualification requirements to employees. Greedy and random greedy algorithms resulted in satisfactory results while different iterative improvement approaches like tabu search did not compare favourably.

2.5. Tour Scheduling Approaches

Ritzman et al. [1976] were the first to propose a simultaneous approach for shift and day-off scheduling in an application to a mail sorting facility. Integrating the planning of workforce and workflow between different operations, full-time and part-time staff was scheduled by dedicated decision rules and simulation.

Several publications have pointed out the importance of integrated models. McGinnis et al. [1978] showed an integrated heuristic algorithm to outperform a sequential approach, consisting in shift scheduling, day-off scheduling and shift assignment. In an integer programming approach to discontinuous tour scheduling, Bailey [1985] showed that considerable savings can be obtained from an integrated formulation. Bechtold and Showalter [1987] demonstrated the superiority of a heuristic tour scheduling algorithm over an LP-based two-phase approach.

A linear programming rounding procedure for continuous tour scheduling was proposed by Morris and Showalter [1983]. In their solution method, fractional solutions are first rounded down before restoring feasibility by successively adding tours. The algorithm was shown to yield a maximum deviation of 1% with regard to the LP relaxation and to compare favourably to the rounding heuristics of Henderson and Berry [1976] and Bartholdi III [1981].

Glover et al. [1984] presented a model for discontinuous named tour scheduling, taking preferences into account and incorporating full-time and part-time employees. As a precursor of tabu search, a local search algorithm was described, allowing for steps into the infeasibility region of the search space. A similar approach was proposed by Glover and McMillan [1986] for named tour scheduling with restricted employee availabilities, full-time and part-time staff, different skill levels, flexible breaks and restrictions on the number of weekly working hours. The approach was tested successfully on shift scheduling problems arising in fast-food restaurants.

Holloran and Byrn [1986] described the development of a scheduling system for United Airlines sales reservation offices and passenger services. Their approaches uses queueing models for the generation of typical weekly workloads. These are covered by monthly tours with equal shifts in each workstretch, using the LP-based rounding scheme of Henderson and Berry [1976]. Separate modules are used to reduce the number of employed shift start times and to ensure contiguity between monthly work schedules.

Further application-oriented papers were contributed by Andrews and Parsons [1989] and Taylor and Huxley [1989]. Andrews and Parsons [1989] described the evaluation of commercial scheduling systems for telephone operators at a mail-order sales company, including demand forecasting models and tour scheduling algorithms. Taylor and Huxley [1989] described the realisation of a scheduling system for the San Francisco Police Department. A primal-dual integer heuristic was developed to solve a tour scheduling problem with relatively strict restrictions on the distribution of shifts and days off.

Bechtold et al. [1991] compared different tour scheduling algorithms, including the LP-based rounding approaches of Henderson and Berry [1976], Keith [1979] and Morris and Showalter [1983] as well as the heuristic algorithms of Buffa et al. [1976], McGinnis et al. [1978] and Bechtold and Showalter [1987], using a tour scheduling setting which is amenable to all of these solution techniques. On a set of artificial test problems, the technique of Bechtold and Showalter [1987] turned out to be competitive with LP-based approaches while the rounding heuristics of Keith [1979] and Morris and Showalter [1983] outperformed alternative solution techniques.

Easton and Rossin [1991] described a column generation algorithm for tour scheduling. Only tours of nonzero values in the LP relaxation were subjected to an integer programming formulation. The authors showed that this method often finds optimal overall solutions and outperforms LP-based rounding or improvement algorithms.

In an application to a lockbox department of a bank, Li et al. [1991] interpreted constraint coefficients of a set covering formulation as employee productivities. For a discontinuous tour scheduling formulation for full-time and part-time employees, they proposed LP-based rounding based on the methods of Bartholdi III [1981] and Showalter and Mabert [1988].

Loucks and Jacobs [1991] presented a named tour scheduling problem arising in a fast-food restaurant, involving employee availabilities, skills and task assignments. Their formulation is based on goal programming, aiming at minimising overstaffing and deviations from targeted work hours for each employee. However, a goal programming solution method was deemed too costly, and construction and improvement heuristics were used instead.

In two publications, Brusco and Jacobs proposed simulated annealing approaches for tour scheduling scenarios involving flexible breaks. Brusco and Jacobs [1993b] applied SA to discontinuous tour scheduling and allowed for start-time float within the tours. In contrast, the SA algorithm of Brusco and Jacobs [1993a] refers to continuous tour scheduling with equal start times on all days and five consecutive days on per tour. Both approaches build upon an initial heuristic solution and use a neighbourhood which consists in dropping and rebuilding tours. Brusco and Jacobs [1993b] showed that simulated annealing is faster than LP-based methods while obtaining solutions which are within a maximum deviation of 1.95% from the LP relaxations. Brusco and

Jacobs [1993a] showed that SA outperforms the LP-based rounding techniques of Keith [1979] and Morris and Showalter [1983].

A number of publications is based on these simulated annealing algorithms, including Brusco and Jacobs [1995], Brusco and Johns [1995], Brusco et al. [1995], Brusco and Jacobs [1998b] and the shift scheduling approach of Thompson [1996b] described above. Brusco and Jacobs [1995] conducted experiments on a tour scheduling formulation for a mixed full-time/part-time workforce. They showed that applying a discontinuous formulation to continuous problems entails substantial excess labour costs if the use of part-time staff is limited. Brusco and Johns [1995] showed how to treat the even dispersion of surplus labour as a secondary goal in the tour scheduling model of Easton and Rossin [1991], using the above SA algorithm and the LP-based rounding approach of Morris and Showalter [1983].

The development of a PC-based personnel scheduling system for full-time and part-time workers at United Airlines was described in Brusco et al. [1995]. In this application, the above simulated annealing algorithm builds upon initial tour schedules generated by the system of Holloran and Byrn [1986]. Major savings were reported using the new system. The authors incorporated an LP-based technique for reducing the number of shift starting times which was described by Brusco and Jacobs [1998b] (cf. Section 2.9). Equally based on data of United Airlines operations, Brusco and Jacobs [1998a] showed how the number of columns in the IP formulation of a continuous tour scheduling problem can be reduced by some simple observations.

Jacobs and Bechtold [1993] conducted a comprehensive study on the efficiency of tour scheduling with different degrees of flexibility, using the integer programming formulation of Bailey [1985] and the break model of Bechtold and Jacobs [1990] (described in Section 2.8). On a set of different requirement patterns, they found out that flexibility with regard to tour lengths and break placement are major sources of schedule efficiency. While shift type flexibility (start times and lengths) had considerable effects, start-time float within the tours and day-off flexibility were only minor factors. Furthermore, it was noted that a different extent of labour requirements throughout the day has considerable impact on roster efficiency. While the amplitudes of demands are further important factors, mean requirements were not found out to be significant.

In a simple tour scheduling environment, Thompson [1993] conducted a simulation analysis on the consequences of different requirement representations. Alternatively to "at-least" staffing levels, a service operation was modelled by target staffing models, allowing for staff shortages and overcoverages. Target staffing levels as well as stepwise linear costs of understaffing and overstaffing were calculated from different customer arrival patterns, service times and customer dissatisfaction costs. Using the LP-based rounding techniques of Henderson and Berry [1976] and Keith [1979], Thompson [1993] concluded that more exact representations of demands and the admittance of supply shortage results in lower overall costs.

Bechtold and Brusco [1994a] proposed a sequential approach for discontinuous tour scheduling of full-time and part-time staff which is amenable to the solution by a personal computer. For the solution of the underlying shift scheduling problems, they combined the algorithm of Showalter and Mabert [1988] with an improvement phase by Henderson and Berry [1976], favouring the use of few shift start times across the days. The procedures of Bechtold [1981] and Bechtold [1988] were subsequently used for day-off scheduling. In comparison to the method of Easton and Rossin [1991], the approach was shown to yield slightly worse results on easy test problems while results were better on larger scenarios and required less computing time.

Inspired by Burns and Carter [1985] and Bechtold and Jacobs [1990], Jarrah et al. [1994] combined seven daily shift scheduling formulations into an integer programming model for discontinuous tour scheduling, using specialised branching rules on aggregate features. The effectiveness of the approach was demonstrated on test cases of the U.S. Postal Service.

Bailey et al. [1995] proposed an integrated integer programming model for project and tour scheduling. Only little information was given on modelling details and solution methods.

Brusco and Johns [1996] exploited the special structure of discontinuous tour scheduling problems. Their method is based on successive fixations of tour variables in an LP formulation, starting with tours whose shifts start in the first or last period of the days. On settings with only full-time workers, the authors showed that this approach nearly always finds optimal solutions and outperforms the techniques of Keith [1979] and Morris and Showalter [1983]. Using the reduction technique of Easton and Rossin [1991] for experiments on a mixed workforce, the algorithm turned out to perform consistently better than the method of Bechtold and Brusco [1994b].

A system for monthly ground staff rostering at a large airport was described by Dowling et al. [1997]. Allocating actual work tasks as late as one day before operations, a simulated annealing algorithm was used to solve the tour scheduling problem on the demand-curve level.

Brusco [1998] proposed to use the Gomory dual all-integer cutting plane to solve a discontinuous tour scheduling problem involving full-time and part-time employees. On scheduling environments with different degrees of flexibility, the approach was shown to be superior to a commercial branch-and-bound code for integer programming.

In a call centre application, Henderson and Mason [1998] considered the determination of labour requirements as an integral part of scheduling. They proposed to repeatedly iterate between shift or tour scheduling by integer programming and the simulation of customer arrivals by a queueing model, generating new demand constraints for the scheduling model.

Alvarez-Valdez et al. [1999] developed a tabu search algorithm for continuous tour scheduling of a mixed workforce at a Spanish aircraft fuelling company. In their contribution, schedules refer to one week, but respect constraints with regard to the preceding week as well as global work balances. The proposed algorithm first only determines shift classes for each day. Schedules are then assigned to employees and appropriate shifts filled in. The method was compared to a commercial integer programming package at equally restricted runtimes. However, the tabu search approach still yielded solutions exceeding IP results by 2% and more.

Mason and Nielsen [1999] described a named tour scheduling algorithm. The procedure is part of a commercial software package which has been implemented in call centres, at an airport Customs authority and for nurse rostering. Their model involves qualifications and staff preferences and was solved by heuristic decomposition and column generation.

2.6. Nurse Scheduling Approaches

The scheduling of nurses and physicians has received particular attention in the literature. While this application area is closely related to other domains of workforce scheduling, models usually bear some special characteristics. Nurse scheduling generally belongs to the class of named scheduling models, incorporating preferences and availabilities, see e.g. Warner [1976] and Dowsland [1998]. Furthermore, several researchers have considered nurses of different skill classes [Arthur and Ravindran, 1981] [Jaumard et al., 1998]. Requirements are usually given per shift type, i.e. nurse scheduling refers to the sequential or simultaneous determination of day-off and shift positions. Most publications assume three non-overlapping shift types, namely morning, evening and night shift [Arthur and Ravindran, 1981]. Breaks are not considered, but models may include the assignment of nurses to functions, see Schaerf and Meisels [1999]. Solution methods sometimes only aim at finding feasible solutions [Berrada et al., 1996] [Jaumard et al., 1998] [Schaerf and Meisels, 1999]. Alternatively, it may be known that requirements cannot be met at all [Warner, 1976], or nurse preferences are considered more important than meeting demands [Arthur and Ravindran, 1981]. As most nurse scheduling models are feasibility-driven, solution methods are often different from general workforce scheduling and e.g. include constraint programming and tabu search. The class of problems arising in nurse scheduling is sometimes subsumed under the term *employee timetabling problems* [Meisels et al., 1997] [Schaerf and Meisels, 1999]. We will shortly survey the relevant literature; two further publications on nurse scheduling will be described in the subsequent section on cyclic rostering.

Miller et al. [1976] presented a local search scheme for nurse scheduling with only one shift type. Their algorithm aims at meeting minimum and preferred staffing levels as well as nurse preferences. The method was implemented in a number of hospitals in the U.S. and Canada, yielding results which were superior to the schedules used by the hospitals before. In the model of Warner [1976], nurses were allowed to specify their preferences on score cards, distributing penalty points for inconvenient assignments. Different skill classes were considered. A multiple-choice integer programming formulation was used to generate 14-day schedules for each nurse with costs reflecting preferences. The optimisation approach was implemented at the University of Michigan Hospital.

Arthur and Ravindran [1981] described a sequential approach for nurse scheduling. While weekends off were granted according to nurse preferences, other days off were scheduled by a goal programming approach. Shifts of three skill classes were then assigned separately, again taking preferences into account. Musa and Saxena [1984] used a goal programming formulation for day-off scheduling of nurses. Goals included the coverage of workloads as well as meeting weekend-off preferences and weekly work hours. A special solution technique, inspired by the additive algorithm of Balas [1965], was applied to a small example.

Based on the tour scheduling approach of Bailey [1985], Ozkarahan and Bailey [1988] presented a goal programming model for nurse scheduling with a fixed staff size. Demands were given per time period, and goals included meeting the workforce requirements as well as matching the derived shift numbers to tours. A sequential solution approach was used, determining daily schedules after day-off scheduling. Tests were carried out on real-world data, and significant savings were reported with regard to the manual planning method used before.

Weil et al. [1995] developed a constraint programming (CP) model for nurse scheduling, using a simple search strategy on a small real-world example. In a strongly constrained setting including preassigned shifts, Cheng et al. [1997] presented another CP method for nurse scheduling. Redundant modelling was used, coupling two alternative constraint programming models by so-called channelling constraints. The technique was successively applied to real-world data from a Hong Kong hospital. Meisels et al. [1997] described the development of a nurse scheduling system involving a combined CP/knowledge-based approach. Qualification and shift compatibility constraints were incorporated as well as nurse preferences.

The setting of Berrada et al. [1996] considers constraints on the number of consecutive and weekly working days as well as restrictions on the placement of days off and preferences. The model was solved by integer programming and tabu search. Contrary to Berrada et al. [1996], Dowsland [1998] regarded demands for non-overlapping shifts as hard constraints. An involved two-phase tabu search algorithm was proposed, using strategic oscillation and chain neighbourhoods. Tabu search was reported to be more robust than simulated annealing or random descent.

Jaumard et al. [1998] described a general framework for the solution of nurse scheduling problems by branch-and-price. Their setting is closer to general workforce scheduling as demand figures are given by so-called demand periods and shifts are allowed to overlap. Preferences and skills are considered as well as understaffing. Additionally, the model includes matching conditions for contiguity between scheduling periods. The authors used a column generation formulation, representing working hours, weekends off, consecutive days on, holidays and shift type constraints as constrained resources in a shortest path subproblem. Preliminary tests on data from a Montréal hospital indicated rather high running times.

Another approach for tour scheduling in a nursery application was described by Mason and Smith [1998]. Similarly to Jaumard et al. [1998], demand figures are given per segment in which no shifts start and end, and column generation was used in an integer programming formulation. The model includes constraints on the number of days on and off as well as preferences with regard to the choice of shifts, shift transitions and general workstretch features. A column generator generates minimum-cost workstretches and combines them into roster lines, using a network flow formulation and dynamic programming. Furthermore, it was shown how working hour constraints can be respected.

Schaerf and Meisels [1999] used generalised local search to find an assignment of nurses to shifts and functions such that the least number of constraints is violated. Their model includes nurse availabilities as well as qualifications. Carter and Lapierre [2001] summarised scheduling techniques for emergency room physicians in hospitals around Montréal. A multitude of relevant restrictions was described. It was shown how schedules used in two hospitals could be slightly improved by manual changes and simple tabu search techniques.

Most recently, Bellanti et al. [2004] described a monthly nurse scheduling problem in an Italian hospital. The model includes preferences as well as holidays. Starting with the assignment of holidays, requested days off and night shifts, a greedy algorithm builds an initial solution. Solution improvement by tabu search and iterated local search yielded solutions which were superior to manually created schedules, but none of the two improvement methods turned out to strictly outperform the other.

2.7. Cyclic Roster Approaches

A limited number of publications deal with cyclic rosters on which workers rotate in a cyclic fashion, cf. Section 1.2.4. The main advantage of cyclic rosters is their fairness because all employees work on the same pattern [Bennett and Potts, 1968]. However, rotating schedules entail an equal coverage for every week for which they are unrolled, meaning that it is not possible to adapt to changing labour demands [Baker, 1976] [Ernst et al., 2004].

Some publications which refer to cyclic rosters have already been cited before, namely the combinatorial approaches for day-off scheduling by Lowerre [1977], Emmons [1985], Burns and Koop [1987] and Koop [1988] as well as the shift assignment problems treated in van den Berg and Panton [1994], Lau [1994], Lau [1996a] and Lau [1996b]. Although cyclic rostering is a special case of tour scheduling, it is interesting to note that no publication has treated cyclic rostering in an integrated way, determining daily shifts and shift/day-off schedules simultaneously.

Bennett and Potts [1968] devised a two-step approach for cyclic scheduling of bus drivers and operators, sequentially assigning days off and shifts of given types. Special goals for the distribution of days off were partly solved to optimality by combinatorial analysis.

In an application to police and fire departments, Laporte et al. [1980] developed an integer programming approach for generating cyclic rosters. In their model, shift changes between three non-overlapping shifts are disallowed within workstretches. Integer programming was used for the selection of building blocks among the set of all feasible workstretches, and solutions containing disconnected subcycles were cut off by additional constraints. An alternative integer programming approach for cyclic nurse scheduling with only one shift type was proposed by Rosenbloom and Goertzen [1987] who generated valid weekly sequences of days off and shifts in a first step and defined a matrix of week transitions respecting all constraints. An integer program was then used to choose among the valid transitions.

Balakrishnan and Wong [1990] presented a cyclic rostering problem with a given workforce size and demand figures per shift type. Shift changes within workstretches were disallowed, and the first workstretch was assumed to start on a Monday in order to break cyclicity. Incorporating coverage constraints in a Lagrangian relaxation approach, the subproblem was formulated and solved as a network flow model. The possible duality gap was closed by the solution of a K-shortest path problem.

Panton [1991] divided a cyclic roster of fixed size into *modules* of several weeks. The day-off scheduling and shift assignment problems were then solved within each module. In their heuristic solution approach, weekends off are assigned first, and labour requirements are evenly divided between the modules. Days off are scheduled by complete enumeration, respecting matching conditions between weeks. Finally, shifts are assigned using the out-of-kilter algorithm for network flows. The approach was evaluated on scheduling problems for Casino security officers.

Several publications describe the development of a scheduling system named ROMAN, developed at the Singapore Information Technology Institute. The system incorporates algorithms for cyclic and non-cyclic named rostering. Chew [1991] gave a description the underlying cyclic roster algorithm in an application to airport baggage services. Due to special rules, the model only allows for a small number of feasible workstretches. Chew [1991] formulated an integer program on the workstretch level, but outlined only a manual solution procedure, assigning shifts one by one with non-increasing start times.

System overviews and applications to the health care and transportation sector were reported in Khoong and Lau [1992] and Khoong et al. [1994]. Khoong and Lau [1992] also described a sequential approach for cyclic roster generation, starting with the determination of required shift numbers and the necessary workforce size. Subsequently, days off and shifts were assigned, using different heuristic methods, branch-and-bound search and allowing for user interaction on different levels. Khoong et al. [1994] additionally mentioned a module for individual rostering which e.g. incorporates employee preferences. The system allows for the definition of work rules in a specification language and provides means for what-if analyses of work policies, employment levels, demand levels and deployment profiles. Khoong et al. [1994] shortly surveyed solutions methods, including heuristics, local search, branch-and-bound, graph algorithms and manual interaction on different subproblems.

Mason et al. [1998] described the development of another rostering system for the Customs authority at Auckland airport. Their methods starts by determining labour requirements per time period, using heuristics and iterated simulation of passenger arrivals. A shift scheduling formulation is then used to cover demands by full-time and part-time staff. Finally, full-time shifts are positioned in a cyclic roster by a method described by Panton [1991].

Building upon the work of Balakrishnan and Wong [1990], Millar and Kiragu [1998] presented a network flow model for cyclic and non-cyclic anonymous nurse rostering. As only two non-overlapping shift types are used, all feasible workstretches can be enumerated and represented as nodes in a flow formulation. Incorporating shift coverages as well as constraints on weekends off and weekly working hours, the model was solved by a commercial integer programming package.

One of the best recent references for cyclic rostering is given by Mason [1999]. The author reviewed IP models for the generation of cyclic rosters for given shift numbers. He proposed to represent workstretches by columns, taking different shift types and shift changes within workstretches into account. In order to avoid disconnected solutions, a novel three-way branch was presented which seems to be more powerful than the cutting scheme of Laporte et al. [1980].

Taking a half-automatic heuristic approach, Muslija et al. [2000] solved a cyclic rostering problem for given workforce sizes and shift requirements. After determining lengths of work blocks, blocks of shifts and days off are distributed subject to weekend-off constraints. Finally, valid sequences of shifts are generated and assigned to the work blocks. The algorithm is part of a commercial software package.

Felici and Gentile [2004] described an integer programming algorithm for cyclic rostering at Alitalia, using a formulation which initially exhibits strong symmetry. A number of valid inequalities was derived in order to tighten the LP bound, making use of special problem structures with regard to weekends off and days on and off. Specialised branching strategies aim at breaking symmetry. Real-world test problems were used for the experimental evaluation.

2.8. Break Placement Flexibility and Implicit Modelling

Special attention has been given to the representation of relief and lunch breaks in shift and tour scheduling. Aykin [1996] notes that in shifts of at least nine hours, employees usually receive one lunch break of 30 or 60 minutes and two relief breaks of 15 minutes each. Other authors report on settings in which only one lunch breaks is required, e.g. Brusco and Johns [1996]. Furthermore, some organisations use *split shifts* which Thompson [1992] defines as shifts with a meal break of at least 1.5 hours, see also Segal [1972], Holloran and Byrn [1986] and Jacobs and Bechtold [1993].

Clearly, workers are not available for covering workloads during their break. Breaks are usually restricted to take place within certain time intervals which can e.g. be governed by canteen opening hours [Bechtold and Jacobs, 1990]. The placement of breaks then provides flexibility which can be exploited in scheduling [Dantzig, 1954] [Segal, 1972]. The consideration of flexible breaks in shift and tour scheduling based on the above set covering model has attracted considerable research.

A straightforward approach for the incorporation of breaks consists in replicating shift variables for each valid break placement [Dantzig, 1954]. The columns in the set covering formulation (2.1)-(2.3) then only equal 1 if none of the breaks is taken at the given moment. Considering more than one break implies creating one shift variable for each combination of break start times. If breaks can only be taken within rather limited intervals, this is a feasible approach, see e.g. Bennett and Potts [1968], Thompson [1990] and Brusco [1998]. With larger degrees of freedom, the explicit incorporation of breaks is generally judged intractable [Bechtold, 1988].

Keith [1979] and Schindler and Semmel [1993] propose to consider break placement only in a second step. Other authors completely ignore breaks (e.g. Morris and Showalter [1983], Bailey and Field [1985] and Brusco and Jacobs [2001]) or use only fixed breaks (e.g. Brusco and Jacobs [1993b], Brusco and Johns [1995]). However, Showalter and Mabert [1988] and Jacobs and Bechtold [1993] have shown that break placement flexibility plays an important role in the creation of efficient shift plans and rosters. Clearly, flexible breaks can be more easily incorporated in heuristic solution approaches, see e.g. Henderson and Berry [1976].

For shift scheduling scenarios with three breaks, Panton and Ryan [1999] and Mehrotra et al. [2000] proposed to use column generation to enumerate break combinations and showed that this approach is competitive with other models. However, it does not exploit size advantages related to so-called *implicit formulations*.

Gaballa and Pearce [1979] were the first to use sets of *break variables*. A distinct break variable is attributed to each shift and break start time. Break variables reduce workforce availabilities in the time periods covered by the breaks. By one constraint per shift, the sum of breaks is restricted to be equal to the numbers of shifts.

Bechtold and Jacobs [1990] improved this basic idea by sharing break variables between different shifts. A set of so-called forward and backward constraints and one equality constraint ensures the availability of sufficient breaks for all shifts. In a post-processing step, breaks are matched to the shifts. The equivalence of this model to a formulation incorporating breaks explicitly in the shift variables was proved by Bechtold and Jacobs [1996]. In comparison to Gaballa and Pearce [1979], the approach of Bechtold and Jacobs [1990] reduces the number of constraints considerably. However, it only applies to less-than-24h operations and makes rather restrictive assumptions: shifts contain only one break, all break durations are equal, and break time windows do not exhibit so-called *extraordinary overlap*.

These restrictions do not apply to the approach of Aykin [1996] who generalised the Gaballa/Pearce model by using one set of break variables for each of three breaks in a shift. In Aykin [1998], this model was applied to cyclic shift scheduling problems. More recently, Aykin [2000] showed that even if his formulation requires more break variables, the number of nonzeros in the constraint matrix is less than in the extended formulation of Bechtold and Jacobs [1990]. On a set of artificial shift scheduling problems, optimal solutions were shown to be obtained in lower runtimes. However, Aykin did not make use of a constraint substitution idea given in Bechtold and Jacobs [1990] which reduces the number of nonzero elements in the constraint matrix.

Topaloglu and Ozkarahan [1998] compared the break scheduling approaches of Bechtold and Jacobs [1990] and Aykin [1996] on a tour scheduling formulation based on Bailey [1985], generalising the Bechtold/Jacobs model to several breaks per shift. A comparative evaluation showed that on many tour scheduling settings, the Bechtold/Jacobs approach entails lower computation times.

Other researchers have also adopted the Bechtold/Jacobs approach for representing flexible breaks, including Jarrah et al. [1994], Thompson [1995] and Thompson [1996a]. Brusco and Jacobs [2000] showed how to overcome the restriction to discontinuous operations by the introduction of so-called wrap-around break variables. Rekik et al. [2003] gave a new interpretation of the forward/backward constraints of Bechtold and Jacobs [1990] by Benders' reformulation and elimination of redundant constraints. Furthermore, they showed the equivalence of the models of Bechtold and Jacobs [1990] and Aykin [1996]. Rekik et al. [2003] remarked that the Bechtold/Jacobs formulation bears size advantages especially if shifts have many break start times in common.

Implicit modelling has been used in other shift and tour scheduling applications as well. The basic idea of implicit models consists in using several categories of variables which are coupled by constraints, avoiding redundancy when shifts or tours share common properties. Coupling constraints encode feasibility conditions on the assignment of different classes of objects, leaving the generation of actual solutions for a postprocessing step, see e.g. Bailey [1985], Bechtold and Jacobs [1990] and Thompson [1995].

Moondra [1976] proposed an implicit model for representing start time flexibility in shift scheduling. Thompson [1995] combined this idea with the implicit break model of Bechtold and Jacobs [1990]. Doubly implicit shift scheduling was also used by Thompson [1996a] in order to evaluate the effects of different numbers of action times which consist of shift and break start and end times. Implicit shift models are restricted to settings with large scheduling flexibility and assume shift costs to be proportional to the number of covered time periods [Mehrotra et al., 2000]. Thompson [1995] showed that the implicit shift model exhibits substantial size advantages if a shift scheduling setting conforms with these assumptions.

Bailey [1985] presented a model for implicit handling of start time flexibility (*float*) in tour scheduling. He showed that if different shift types are allowed in a weekly tour, feasible day-on/off patterns can be represented by a distinct set of variables which is coupled to variables determining the shifts per day. If these assumptions are too general, it may still be sufficient to restrict weekly shifts to an interval of admissible start times. An implicit model for overlapping *start-time bands* was described by Jacobs and Brusco [1996]. Restricting each tour to a set of start times, the authors coupled the tour variables to shift variables of the respective start-time band. In Brusco and Jacobs [2000], this formulation was integrated with the implicit break placement formulation of Bechtold and Jacobs [1990]. Rekik et al. [2003] proposed a doubly implicit formulation for start-time bands and break placement flexibility. They showed how to overcome a limitation of Brusco and Jacobs [2000], allowing for overlapping start-time bands between the days by the introduction of wrap-around shift variables.

Thompson [1990] developed an implicit model for the handling of limited employee availabilities in named shift scheduling. So-called *regions* represent sets of shifts (and equivalently intervals of time) in which more than one employee is available. Employees are assigned to regions, and the set of region variables is coupled to the shift variables.

Thompson [1992] conducted a study on using service personnel for blocks of controllable work (e.g. maintenance or cleaning tasks) when no uncontrollable work (due to customer arrivals) has to be carried out. In their approach, a set of variables represents start times for blocks of controllable work. Two integer programming formulations were compared which are very similar to the break models of Gaballa and Pearce [1979] and Bechtold and Jacobs [1990]. By a computational study on artificial test problems, the second formulation was shown to be superior in terms of model sizes and solution times.

Çezik et al. [2001] coupled seven daily shift scheduling formulations into a discontinuous tour scheduling formulation, restricting shifts on consecutive days by forward and backward constraint systems as in Bechtold and Jacobs [1990]. In another interesting paper, Çezik and Günlük [2002] gave a general interpretation of implicit models by projecting out variables of a bipartite flow formulation. They showed how different assumptions lead to different representational complexities. Their analysis was illustrated with the implicit break model of Bechtold and Jacobs [1990] and the tour scheduling formulation of Çezik et al. [2001].

2.9. Working Subset Methods

Besides implicit modelling, scientists have proposed to use *working subsets* of the shift types to reduce problem sizes. Some of these propositions have already been mentioned throughout the preceding exposition. Several authors have treated the minimisation of different shift types as implicit or explicit objectives in shift or tour scheduling models, see Holloran and Byrn [1986], Bechtold and Brusco [1994a], Brusco et al. [1995] and Musliu et al. [2004]. Mabert and Watts [1982] mentioned that the use of working subsets not only reduces computational complexity, but is also convenient for the staff. Brusco and Jacobs [2001] pointed out that restrictions on shift starting times avoid administrative burden and ease the organisation of briefing sessions for employees. Thompson [1996a] evaluated the impact of action times (i.e. shift and break start and end times) on service organisations. In the following, we shortly summarise publications which explicitly tackle working subset problems.

Henderson and Berry [1976] replicated shift variables in the classical set covering formulation for each possible placement of three breaks. From the resulting shift realisations, they chose a working subset by selecting a first random shift and repeatedly adding shifts of highest differences with regard to the number of covered periods. They showed that for low working subset sizes, this method is superior to random selection of shifts. Furthermore, Henderson and Berry [1976] stated that only 40 to 50 shifts with fixed breaks are generally sufficient to find efficient solutions.

In a tour scheduling application for part-time staff in a bank, Mabert and Watts [1982] proposed a heuristic reduction method for the number of shifts. A priori, shifts of 4, 5 and 6 hours were admitted with respective numbers of 5, 4 and 3 shifts per tour. Using relative frequencies of shifts in the solution of a weekly shift scheduling formulation, Mabert and Watts [1982] selected working subsets of shift types and workdays by biased sampling.

Easton and Rossin [1991] used column generation to implicitly represent tours with a high degree of scheduling flexibility. In their contribution, only tours with nonzero values in the LP relaxation are used for a tour scheduling algorithm. The authors showed that this method is superior to the approaches of Henderson and Berry [1976] and Mabert and Watts [1982] as well as LP-based rounding techniques.

A comparative evaluation of different working subset methods was presented by Bechtold and Brusco [1994b], solving discontinuous tour scheduling problems by integer programming. They distinguished between structural methods using only information on the shifts (like in Henderson and Berry [1976]) and demand-based procedures (as in Mabert and Watts [1982]). In addition to the aforementioned techniques, the authors proposed two further structural methods and one demand-based method. It was shown that best results for different working subset sizes from 10 to 50 shifts are obtained by either the new structural or the new demand-based method. Furthermore, it was demonstrated that global optimal solutions can usually be obtained with working set sizes of 40 or 50 shifts. Bechtold and Brusco [1994b] mentioned that structural and demand-based working subset methods could be combined with the “refinement procedure” of Easton and Rossin [1991].

The United Airlines scheduling system described by Brusco et al. [1995] incorporates another method for shift start time selection which Brusco and Jacobs [1998b] presented in more detail. The method is based on the superposition of daily demands, resulting in a shift scheduling formulation. This problem is solved by a heuristic column generation algorithm, using dynamic sets of admissible start times, different shift selection rules and random elements. The subsequent simulated annealing algorithm for tour scheduling only employs shifts used in the shift scheduling solution. Brusco and Jacobs [1998b] showed that this method yields results which exceed the LP relaxation only slightly.

Based on the algorithm of Brusco [1998], Brusco and Jacobs [2001] evaluated the impact of the numbers and choices of shift starting times in continuous tour scheduling. Restricting all shifts in a tour to start at the same time, they extended the basic IP formulation by starting time constraints. By an experimental evaluation with different work policies on demand profiles from service industries, they found out that efficient schedules can often be found with as little as three to five starting times. However, it was also stated that the number of starting-time subsets which are capable of providing optimal workforce sizes is often small, and poor selections can lead to substantial penalties. Furthermore, a case study for a call centre was described, using heuristic starting time selection procedures in a spreadsheet application.

2.10. Complexity Results

A number of authors have shown that classes of problems arising in workforce scheduling are $\mathcal{NP}$-hard. Bartholdi III [1981] proved that cyclic staffing with intermittently available resources is $\mathcal{NP}$-hard. This problem e.g. covers shift scheduling problems with breaks and cyclic employee availability as well as cyclic tour scheduling problems. However, Bartholdi III [1981] also showed that cyclic staffing problems are solvable in polynomial time if the columns in a cyclic set covering formulation contain contiguous blocks of ones. Vohra [1988] gave a polynomial-time algorithm for problems with equal demands. The combinatorial approaches described in Section 2.3 represent further classes of polynomial subproblems.

Van den Berg and Panton [1994] analysed the *continuous shift assignment problem* (CoSA) which consists in attributing given shifts to gaps in a cyclic roster such that consecutive days are attributed equal shift types. By a reduction from the PARTITION problem, they showed that this problem is $\mathcal{NP}$-hard. Closely related is the *changing shift assignment problem* (CSAP) of Lau [1994], prescribing valid pairs of shift types on consecutive days. CSAP was shown to be $\mathcal{NP}$-hard by Lau [1994] and Lau [1996b] even under restrictive assumptions, using a reduction from 3SAT. Both publications provide a further proof for the $\mathcal{NP}$-hardness of cyclic rostering.

Kortsarz and Slany [2001] studied the relation of the *minimum shift design problem* considered in Musliu et al. [2004] to the minimum edge-cost flow problem. The problem involves the choice of shift types and the determination of the number of employees assigned to each shift. Kortsarz and Slany [2001] showed that the minimum shift design problem is $\mathcal{NP}$-hard. Furthermore, they stated that there is a constant $c < 1$ such that approximating the shift design problem within $c \ln n$ is $\mathcal{NP}$-hard.

2.11. Relationship to Crew Rostering

Related to general workforce scheduling is the rostering of airline crews, i.e. pilots, flight engineers and attendants. Much attention has been given to crew rostering since cabin personnel is usually the second most important cost factor for airlines (after fuel costs) [Anbil et al., 1992]. Overviews on airline crew scheduling and rostering can be found in Bodin et al. [1983], Teodorović [1989] and, more recently, in Kohl and Karisch [2004]. For annotations on parallels between general workforce and crew scheduling, see also Dowling et al. [1997], Rekik et al. [2003] and Ernst et al. [2004].

Crew rostering problems are usually solved in two stages [Ryan, 1992]. In a first step, the *flight legs* (single nonstop flights, also *segments*) are grouped into so-called *pairings* (equivalently, *rotations*). A pairing is a sequence of flight legs which can be carried out by one crew, starting and finishing at a crew base [Anbil et al., 1992]. While pairings for short-haul flights usually span over one or two days, long-haul flights often entail pairings of three or four days. Union and legal regulations usually impose minimum rest times between working days as well as working hour constraints [Vance et al., 1997]. *Crew pairing optimisation* is usually formulated as a set partitioning problem[1] and solved by integer programming, see e.g. Anbil et al. [1992], Hoffman and Padberg [1993] and Vance et al. [1997].

In the subsequent *crew rostering* phase, given pairings are covered by rosters [Ernst et al., 2004]. In contrast to crew pairing, crew rostering is usually carried out on a named basis, i.e. employees for the single roster lines are known. Solution approaches usually take preassigned activities like ground duties, trainings and simulator sessions into account, see e.g. Gamache and Soumis [1998] and Day and Ryan [1997]. Additionally, staff qualifications may be explicitly considered [Gamache et al., 1999]. The alternative *bidline* approach which is most frequent in North America assumes that rosters are anonymous [Kohl and Karisch, 2004]. Each crew member then chooses a roster line in order of seniority. A third possibility is *preferential bidding*, meaning that the anonymous roster generation method takes preference scores into account, usually weighted by seniority. Crew rostering problems often decompose by crew base and aircraft type since e.g. pilots are usually only qualified for one aircraft type [Gamache et al., 1998]. The scheduling horizon is usually four weeks or a month [Ryan, 1992] [Gamache et al., 1999].

As crew pairing, crew rostering is usually formulated as a generalised set partitioning model (e.g. Ryan [1992], Gamache et al. [1999]):

$$\min \sum_{j\in 1}^{m} \sum_{r\in R_j} c_r X_r$$

subject to

$$\sum_{j=1}^{m} \sum_{r\in R_j} a_{pr} X_r = b_p \qquad \forall p \in \{1,\dots,n\} \tag{2.4}$$

$$\sum_{r\in R_j} X_r = 1 \qquad \forall j \in \{1,\dots,m\} \tag{2.5}$$

$$X_r \in \{0,1\}\ \forall\, r \in \bigcup_{j=1}^{m} R_j \tag{2.6}$$

where n is the number of pairings and m the number of employees. R_j denotes is an index set for feasible roster lines for employee j. The binary decision variables X_r choose among the

[1]Set partitiong problems can be stated as $\sum_{j\in J} c_j X_j$ subject to $\sum_{j\in J} a_{ij} X_j = 1\, \forall i \in I$, $X_j \in \{0,1\}\, \forall j \in J$ with $c_j \in \mathbb{R}_+$ and $a_{ij} \in \{0,1\}$.

roster lines such that each employee is assigned exactly one roster line (constraints (2.5)) and each pairing is assigned the required number b_p of crew members (*crew complement*, constraints (2.4)). The enumeration of feasible roster lines is often treated as a separate problem [Christou et al., 1999] [Kohl and Karisch, 2004]. The predominant solution method for the above model is integer programming, often in conjunction with column generation, see e.g. Ryan [1992], Day and Ryan [1997], Gamache et al. [1998] and Gamache et al. [1999]. Fahle et al. [2002] and Sellmann et al. [2002] have proposed solution methods combining integer programming with constraint programming.

More recently, crew rostering has been applied to railway applications, see e.g. Caprara et al. [1997], Caprara et al. [1998] and Ernst et al. [2000]. The basic setting is closely related to aircrew rostering. In a first step, *round trips* (pairings) are generated for different crew bases which are rostered in a second step. However, the scheduling horizon is usually one week, rosters are anonymous and can be cyclic or non-cyclic [Ernst et al., 2000]. In the cyclic roster applications described by Caprara et al. [1997] and Caprara et al. [1998], one crew starts on each day of the roster instead of attributing one worker or crew to each roster week (as in the cyclic rosters of Section 2.7).

At first sight, there are considerable similarities between crew rostering and general workforce scheduling. Both settings impose similar constraints like minimum rest times, restrictions on consecutive days on and off as well as working hour constraints. While workforce scheduling problems are often formulated as set covering programs, set partitioning models and algorithms are the predominant basis for crew rostering approaches. In both settings, column generation approaches have received considerable attention in recent years.

However, there are some important differences. While in general workforce scheduling, employees may only be scheduled within given limits of shift lengths and start times, crew duties result from the given pairings. Breaks are not explicitly considered, but result from rest times (*sits*) between subsequent trips. Crew scheduling furthermore incorporates linking constraints which guarantee geographic contiguity [Dowling et al., 1997]. Finally, crew scheduling generally refers to named scheduling, and rosters are non-cyclic with exceptions in railcrew rostering.

A further application of anonymous rostering is the scheduling of *bus drivers* which is also referred to as *run-cutting* in North America. A large number of contributions was presented on international workshops on computer-aided scheduling of public transport [Wren, 1981] [Rousseau, 1985] [Daduna and Wren, 1988] [Desrochers and Rousseau, 1992] [Daduna et al., 1995] [Wilson, 1999]. An overview on bus driver scheduling can e.g. be found in Wren and Rousseau [1995].

A bus schedule defines so-called *vehicle blocks*, i.e. daily tours of a bus, starting and ending at a depot. Furthermore, it defines *relief points* which correspond to stopovers away from the depot at which drivers may change. Trips between such relief points build basic *tasks* to be covered by the driver schedule. The problem is either stated as a set partitioning model (e.g. Falkner and Ryan [1992] or as a set covering relaxation with one coverage constraints for each task, see e.g. Smith and Wren [1988] or Desrochers and Soumis [1989]. Overcoverage of tasks is treated in a subsequent step or accepted as so-called *deadheading* with drivers as passengers. Falkner and Ryan [1992] have proposed a model for simultaneous scheduling of vehicles and drivers. As for aircrew scheduling (see e.g. Kohl and Karisch [2004]), much progress in bus driver scheduling results from the development of commercial systems like HASTUS and CREW-OPT in North America [Blais and Rousseau, 1988] [Desrochers and Soumis, 1989], IMPACS in the United Kingdom [Smith and Wren, 1988] and HOT in Germany [Daduna and Mojsilovic, 1988].

As in airline and railway crew rostering, bus driver scheduling is driven by the tasks of the bus schedule, and there is no notion of shift types like in general workforce scheduling [Wren and Rousseau, 1995]. Meal breaks are usually taken into account [Smith and Wren, 1988] [Desrochers and Soumis, 1989]. Furthermore, models sometimes include shift overtime and split shifts, consisting of two spells separated by a break of several hours [Wren and Rousseau, 1995].

3. Local Search in Constraint Programming

All intelligent thoughts have already been thought;
what is necessary is only to try to think them again.
— Johann Wolfgang von Goethe

In the following, we give a short introduction to *constraint programming* which has become a popular solution method for discrete optimisation problems. We will start by describing fundamental concepts. While constraint programming usually uses branch-and-bound search, we will review work on constraint programming in a local search context. Furthermore, we will describe different methods for representing vehicle routing problems in a constraint programming framework. In Chapters 4 and 5, these techniques will be applied to two problems arising in airport workforce planning.

3.1. Constraint Programming

Constraint programming (CP) is a paradigm which aims at solving combinatorial optimisation problems [Baptiste et al., 2001]. Like other solution methods, CP comprises two components: a *search strategy* and a *search space reduction strategy*. Constraint programming uses a technique called constraint propagation for search space reduction while branch-and-bound is the predominant solution technique. Constraint propagation is based on the formulation of combinatorial search problems as constraint satisfaction problems (CSP). The image interpretation settings of Huffman [1971], Clowes [1971] and Waltz [1975] were the first publications on constraint satisfaction problems. For surveys on CSP solving, see Meseguer [1989] and Kumar [1992], for an in-depth treatment of constraint processing e.g. Tsang [1993] and Dechter [2003].

Formally, a CSP can be described by a triple $P = (V, \mathit{DOM}, \mathit{CONS})$:

- $V = \{x_1, \ldots, x_n\}$ is a finite set of *variables*;
- $\mathit{DOM} = \{D(x) \,|\, x \in V\}$ is a set of *domains* with $D(x)$ containing all values that can be assigned to x;
- $\mathit{CONS} = \{c_1, \ldots, c_m\}$ is a set of *constraints*.

Domains are usually finite. A constraint $c \in \mathit{CONS}$ is a function $c : D(x_{i_1}) \times \ldots \times D(x_{i_k}) \rightarrow \{\text{true}, \text{false}\}$ on a base set $V' = \{x_{i_1}, \ldots, x_{i_k}\}$. $|V'|$ is called the *arity* of the constraint. For a set $V = \{x_1, \ldots, x_n\}$, an *assignment* is an element of $D(x_1) \times \ldots \times D(x_n)$. If $c \in \mathit{CONS}$ is a constraint with base set $V' = \{x_{i_1}, \ldots, x_{i_n}\}$, an assignment $a \in D(x_{i_1}) \times \ldots \times D(x_{i_n})$ *satisfies* c iff $c(a_{i_1}, \ldots, a_{i_n}) = \text{true}$. *Solving* a CSP $P = (V, \mathit{DOM}, \mathit{CONS})$ is equivalent to finding an assignment a which satisfies all constraints $c \in \mathit{CONS}$.

Constraint satisfaction problems are often represented as (hyper-)graphs with the node set V, node labels from *DOM* and a (hyper-)edge on the base set V' of each constraint $c \in \mathit{CONS}$. If all constraints in *CONS* are binary ($|V'| = 2$), the constraint graph will be an ordinary undirected graph.

While for constraint satisfaction problems, we seek a *feasible* solution, the task associated with a *constraint optimisation problem* (COP) consists in finding an *optimal* feasible solution with regard to an objective function z. z is a function $z : D(x_1) \times \ldots \times D(x_n) \rightarrow \mathbb{R}$, attributing a value $z(a)$ to each assignment. Without loss of generality, it can be assumed that z is to be minimised. A COP instance P can thus be described by a quadruple $P = (V, \mathit{DOM}, \mathit{CONS}, z)$. General CSP and COP solving is $\mathcal{NP}$-hard [Dechter, 2003].

Clearly, finite domain CSPs or COPs can be solved by full enumeration, generating all possible assignments for the variables. However, this will only be a practical approach for very small examples. Constraint propagation provides a more evolved search space reduction technique. The basic idea of constraint propagation is to eliminate inconsistent variable assignments by repeated application of implicit constraints, the so-called *consistency tests*. For the application of consistency tests, we maintain variable-specific subsets $\delta(x_i) \subseteq D_(x_i)$ $(x_i \in V)$ or higher-order working sets $\delta(x_{i_1}, \ldots, x_{i_l}) \subseteq D(x_{i_1}) \times \ldots \times D(x_{i_l})$. A consistency test has a condition part A and an instruction part B: Each time A is satisfied, execute B which possibly restricts the working subset of combined domains.

Different levels of consistency can be achieved by consistency tests. Montanari [1974] introduced the basic notions of node, arc, and path consistency, relating to problems with only binary constraints. Imagine that domains $\delta(x)$ are maintained for each single variable $x \in V$. *Node consistency* is simply achieved if the values in each variable's domain satisfy the constraints on that variable. A constraint network is *arc consistent* if for every value in $\delta(x_i)$, there is a value in the domain $\delta(x_j)$ of each variable $x_j \neq x_i$ such that the combined assignment obeys all constraints. Finally, a model is *path consistent* if for each pair (x_i, x_j) of variables, each consistent assignment from $\delta(x_i) \times \delta(x_j)$ and each third variable x_k, there is a value in the domain $\delta(x_k)$ such that the combined assignment satisfies all binary constraints on the variables.

If we include higher-order (non-binary) constraints, the notion of *generalised arc consistency* can be used to describe the state of a constraint model. A variable x_i is then called generalised arc-consistent relative to a constraint $c \in \mathit{CONS}$ involving x_i iff for any value from $\delta(x_i)$, there is an assignment from $\delta(x_j)$ for each variable $x_j \neq x_i$ involved in c such that c is fulfilled. The constraint model is said to be arc-consistent if each of its variables is arc-consistent with regard to all constraints [Dechter, 2003].

More general is the notion of *k-consistency* [Freuder, 1978]. We say that a partial assignment $(a_{i_1}, \ldots, a_{i_k})$ is *k-feasible* if it satisfies all constraints which at most contain these variables. Algorithms for obtaining k-consistency maintain sets of combined assignments for variable subsets of order $k-1$. k-consistency is then achieved if for any $(k-1)$-feasible assignment from $\delta(x_{i_1}, \ldots, x_{i_{k-1}}) \subseteq D(x_{i_1}) \times \ldots \times D(x_{i_{k-1}})$ and for any further variable x_{i_k}, there exists an assignment for x_k taken from a set $\delta(x_{i_k}) \subseteq D(x_{i_k})$ such that the combined assignment is k-feasible.

However, managing $(k-1)$-dimensional working subsets is only practical for small values for k. Therefore, the concept of *domain consistency* has been introduced which only necessitates one-dimensional subsets $\delta(x_i) \subseteq D(x_i)$ for each variable $x_i \in V$ [Dorndorf et al., 2000]. A problem is *k-d-consistent* if for all variable sets $V' := \{x_{i_1}, \ldots, x_{i_{k-1}}\}$ of order $k-1$ and every variable x_{i_k}, every instantiation $a_{i_k} \in \delta(x_{i_k})$ of x_{i_k} can be extended to a feasible assignment, i.e. there is a k-feasible assignment $(a_{i_1}, \ldots, a_{i_k})$ with $a_{i_j} \in \delta(x_{i_j})$. Note that k-d-consistency corresponds to generalised arc consistency if the maximum arity over the constraints is less or equal to k.

If domains are maintained by intervals $\delta(x_i) = [l_i, r_i] := [l_i, l_i{+}1, \ldots, r_i]$ (given $D(x_i) \subseteq \mathbb{N}_0$), we speak of *bound consistency*. The definition of bound consistency is analogous to domain consistency: The working subsets $\delta(x_i)$ are *k-b-consistent* iff for all variable subsets $V' := \{x_{i_1}, \ldots, x_{i_{k-1}}\}$ of order $k-1$, for every k'th variable x_{i_k} and each $a_{i_k} \in [l_{i_k}, r_{i_k}]$, there are values $a_{i_1} \in \delta(x_{i_1}), \ldots, a_{i_{k-1}} \in \delta(x_{i_{k-1}})$ such that $(a_{i_1}, \ldots, a_{i_k})$ is k-feasible. In some contexts, domains can be naturally defined by intervals, meaning that domain and bound consistency coincide.

The repeated application of consistency tests can be interpreted as a fixed point iteration. Uniqueness of the fixed point can be proved if consistency tests are monotonous, see e.g. Dorndorf et al. [2000].

Constraint programming systems usually implement some combination of domain and bound consistency and normally only guarantee k-consistency for $k = 2$ (arc consistency). A series of algorithms for achieving arc consistency (AC-3, AC-5, etc.) have been proposed, see Dechter [2003] for an overview. Algorithm 1 shows the basic scheme of the AC-3 algorithm. A queue Q is used to maintain pairs of variables which must be checked for consistency. The REVISE procedure (Algorithm 2) is repeatedly called to delete inconsistent values. If a domain is changed, all constraints involving the respective variable are re-evaluated.

Algorithm 1 AC-3$(V, DOM, CONS)$

```
for all pairs $(x_i, x_j) \in V \times V$ do
    $Q \leftarrow Q \cup \{(x_i, x_j), (x_j, x_i)\}$
end for
while $Q \neq \emptyset$ do
    select and delete $(x_i, x_j)$ from $Q$
    if REVISE$(x_i, x_j) =$ true then
        $Q \leftarrow Q \cup \{(x_k, x_i) \mid k \neq i, k \neq j\}$
    end if
end while
```

More evolved algorithms maintain values supporting the possible assignments to a variable. Constraint programming systems usually use some variant of the AC-5 algorithm [Hentenryck et al., 1992]. Consistency tests are usually only triggered on specific events of the involved domains: *domain events* (any change to the domain), *range events* (change of the minimum or maximum value) and *bound events* (a single value remains in the domain). A consistency test for enforcing difference between two variables e.g. only triggers on bound events because as long as there is more than one value in a variable's domain, nothing can be concluded with regard to the other domain.

Algorithm 2 REVISE(x_i, x_j)

```
deleted $\leftarrow$ false
for all $a_i \in \delta_i$ do
    for all constraints $c_{x_i x_j}$ on $(x_i, x_j)$ do
        if there is no $a_j \in \delta_j$ such that $c_{x_i x_j}(a_i, a_j) =$ true then
            delete $a_i$ from $\delta_i$
            deleted $\leftarrow$ true
        end if
    end for
end for
return deleted
```

The concepts of constraint satisfaction and propagation are often part of special programming languages ("constraint logic programming") like PROLOG, CHIP and ECLiPSe. Additionally, programming libraries and modelling environments in C++ and Java have become available (e.g. Ilog Solver or Coalog Solver).

Clearly, there is a tradeoff between achieving higher consistency and the complexity of consistency tests. Higher consistency results in less search effort, but may be costly. On the other hand, if consistency is low, branch-and-bound search will often run into dead-ends of inconsistent variable assignments. In practice, consistency tests are usually designed to have low polynomial time complexity.

Constraint programming systems often provide *global constraints*, i.e. high-level abstractions for modelling complex combinatorial problems in a natural way. Global constraints often make use of specialised algorithms like flow or matching algorithms. The most prominent example is the *alldifferent* constraint which ensures that all variables in a given set are assigned pairwise different values [Régin, 1994].

3.2. Large Neighbourhood Search

Constraint programming is a very successful solution technique for complex discrete optimisation problems. It allows for an easy incorporation of a multitude of side constraints. In realistic problems, sometimes not all constraints are known in advance, and further restrictions may come up with the refinement of optimisation models. Traditional algorithmic paradigms are often limited in this case because the incorporation of additional constraints would entail a complete redesign. In CP-based algorithms, additional constraints usually pose no problems and may even allow for a more efficient solution. Constraint models are often used in powerful branch-and-bound algorithms for the exact solution of problems [Tsang, 1993].

However, exact solution approaches are generally very costly, and consequently, construction heuristics are most frequently used to tackle large-scale optimisation problems. These can be combined with a solution improvement phase. The improvement phase often involves local exchanges like the classical 2-opt, 3-opt or Or-opt exchanges [Lin, 1965] [Or, 1976] which are powerful for simple travelling salesman and vehicle routing settings. Generalisations to more complex problems, e.g. involving time windows, are sometimes possible, cf. e.g. Potvin and Rousseau [1995]. Generally, the classical operators fail on more constrained problems because local exchanges do not always find improving solutions which obey all constraints.

The idea of large neighbourhood search (LNS) is the repeated relaxation of some of the decisions made in the construction phase and a subsequent reoptimisation, using CP-based branch-and-bound. Each single relaxation and reoptimisation can be seen as a local step in a possibly very large neighbourhood. In comparison to classical local improvement operators, the number of relaxed decisions will generally be much larger. Additionally, reoptimisation is done with the full freedom of tree-based search and not according to a fixed scheme. The underlying constraint model helps in maintaining feasibility, and the advantages of CP-based algorithms mentioned above remain valid. In strongly constrained problems, a search in large neighbourhoods will be the only possibility to find improving steps at all.

Constraint-based local improvement was first introduced for the solution of job-shop scheduling problems. Applegate and Cook [1991] presented a shuffle technique which was inspired by the shifting bottleneck procedure in Adams et al. [1988]. Another implementation of CP-based reoptimisation for job-shop scheduling can be found in Caseau and Laburthe [1995].

More specifically to a travelling salesman/vehicle routing context with time windows, Pesant and Gendreau showed how to reinterpret classical local search operators in a CP framework. They devised a special neighbourhood CP model for the local move which is coupled by interface constraints to the master model [Pesant and Gendreau, 1996] [Pesant and Gendreau, 1999]. The approach is applied to a direction-preserving 3-opt neighbourhood for the TSPTW (and in Pesant and Gendreau [1999] to a physician scheduling problem). While the approach remains restricted to classical operators, the authors make out a trend towards the exploration of larger neighbourhoods.

In an application to vehicle routing, Shaw [1998] developed an involved scheme for constraint-based reoptimisation and was the first to use the notion of large neighbourhood search. In each local step, decisions for a set of interdependent visits are relaxed, using a measure of relatedness. Random elements are used to diversify the search. Instead of using full branch-and-bound, limited discrepancy search (LDS) is used in order to heuristically limit the size of the search tree [Harvey and Ginsberg, 1995]. The approach is evaluated on a number of standardised VRP and VRPTW problem instances including Solomon's VRPTW instances [Solomon, 1987]. The initial solution is built by using one vehicle per visit. The approach is shown to be highly competitive and often superior to leading solution techniques, yielding the best-known results for some of the test problems.

Kilby et al. [2000] have conducted a detailed study of the impact of the number of constraints on the quality of different construction and improvement methods. For the solution construction, the traditional savings heuristic of Clarke and Wright [1964] and two constraint-based insertion heuristics are used. For solution improvement, LNS was compared to traditional operators from the VRP literature. These methods were applied to vehicle routing problems with and without time windows and with additional precedence and same-tour constraints. For solution construction, the constraint-based methods are shown to be superior to the savings heuristic. While on simple VRP problems, the traditional exchange operators are almost as good as LNS for solution improvement, large neighbourhood search is shown to be clearly superior on strongly constrained test problems. In [Kilby et al., 1998], implementation details for the two aforementioned publications were given.

Caseau and Laburthe embedded local search in a construction method for VRP and VRPTW problems, minimising the number of tours and travel distances [Caseau and Laburthe, 1999]. Construction and improvement are based on constraint programming in order to address complex constraints including capacity restrictions, but the local steps are classical 2-opt and 3-opt exchanges. For the minimisation of the number of vehicles in the VRPTW, an additional relax-and-reoptimise algorithm is devised which uses limited discrepancy search. The method is shown to be highly competitive, often outperforming the tabu search reference of Rochat and Taillard [1995] when the travelling distance is minimised.

The following two chapters will describe applications of large neighbourhood search. Chapter 4 will present an algorithm for levelling workloads arising in workforce planning on airports, using a non-linear objective function. In Chapter 5, a complex shift planning problem based on routing and scheduling problems is tackled by CP-based local search.

3.3. Constraint-Based Solution Methods for Vehicle Routing

Before entering into these algorithms, we will shortly review different constraint-based solution methods for vehicle routing problems with time windows (VRPTWs). In vehicle routing, we are given a set I of customers which must be served by a set K of vehicles. The visit at customer $i \in I$ must be carried out within a time interval $[a_i, b_i]$. Early arrival is allowed, but entails waiting until the earliest service time a_i. Between two customers i and j, we have a travel time of $d_{i,j}$. We will assume that travel times obey the triangle inequality, $d_{i_1,i_3} \leq d_{i_1,i_2} + d_{i_2,i_3}\ \forall i_1, i_2, i_3 \in I$. We will not consider capacity restrictions for the moment being. In vehicle routing, different goals can be pursued, e.g. the minimisation of the number of vehicle routes or the minimisation of travel distances [Toth and Vigo, 2001a].

Constraint-based methods for time-constrained vehicle routing problems can follow different approaches. Each formulation entails different sets of constraint variables, consistency tests and search algorithms. Depending on characteristics of the problem at hand (e.g. the magnitude of travel times, the widths of time windows, further constraints), one or the other method will be appropriate. We will shortly review different techniques used in the literature.

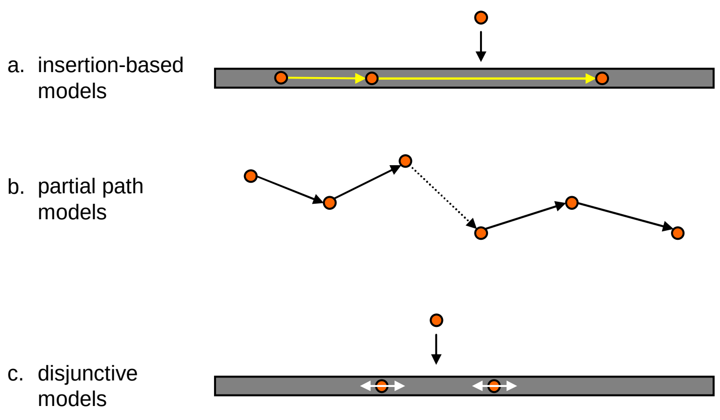


Figure 3.1.: Basic ideas of constraint-based methods for vehicle routing.

3.3.1. Insertion-Based Methods

Probably the most intuitive approach for vehicle routing is given by *insertion-based methods* (see Fig. 3.1a, nodes in orange colour represent visits, grey bars are vehicle routes). The basic idea is to insert visits one by one into an emerging set of vehicle routes. Customers are therefore explicitly assigned to vehicles, and partial solutions consists of incomplete vehicle routes along with unassigned visits. Clearly, insertion-based models are not restricted to being solved by constraint programming. In fact, insertion-based methods have mostly been implemented by ad-hoc data structures and procedures. An example which has gained much attention is the insertion heuristic of Solomon [1987] who uses a mix of travel time and temporal criteria to select customers for insertion. Based on Solomon's heuristic, Potvin and Rousseau [1993] propose a parallel route building algorithm.

We will show how an insertion-based approach can be followed in the constraint programming paradigm, using a model which is similar to the one described by Kilby et al. [2000]. We introduce a temporal domain $\delta_i^t = [\alpha_i, \beta_i]$ denoting the interval of valid service times for customer i[1]. δ_i^t is naturally initialised to the original time window for customer i ($[\alpha_i, \beta_i] := [a_i, b_i]$). $\delta_i^{veh} \subseteq K$ will denote the set of vehicles by which i can be covered. $\pi_i \in I$ and $\sigma_i \in I$ represent the customers which precede and succeed customer i in a given (partial) set of tours. As suggested by the notation, π_i and σ_i are no domain variables in the proper sense, but change their values upon insertion of additional customers.

Furthermore, we let $\delta_i^{ip} \subseteq I$ denote the set of valid insertion positions of i. Without loss of generality, we will assume that δ_i^{ip} represents valid *successor* visits, i.e. $j \in \delta_i^{ip}$ if i can be inserted before j. Note that in contrast to the model of Kilby et al. [2000], we assume that δ_i^{ip} not only contains inserted visits, but in general also comprises still uninserted visits which may finally become the successor of i. If i is inserted, δ_i^{ip} thus contains σ_i as well as unassigned customers which can be inserted between i and σ_i.

If $i \in I$ is a customer which has already been inserted into a route, the following rules describe valid updates for the temporal domains of i and its surrounding visits:

$$\alpha_i + d_{i,\sigma_i} > \alpha_{\sigma_i} \Longrightarrow \alpha_{\sigma_i} := \alpha_i + d_{i,\sigma_i} \tag{3.1}$$
$$\beta_i - d_{\pi_i,i} < \beta_{\pi_i} \Longrightarrow \beta_{\pi_i} := \beta_i - d_{\pi_i,i} \tag{3.2}$$

While (3.1) describes the forward propagation of i's earliest start time to its successor, (3.2) is the backward propagation of latest start times. We will thus create on consistency tests for

[1]As indicated by the notation, we will treat domains as units in their own right without making reference to the original variables (like x_i in the above presentation).

each customer i, performing the updates according to rules (3.1) and (3.2) and triggering upon the range events of δ_i^t. The update of the earliest start time of σ_i will then trigger the respective test for σ_i, possibly updating the earliest start time of σ_{σ_i} and so on. Similarly, updates of latest start times are propagated by a linear backward pass over the route. This propagation scheme is known as *push-forward/push-backward propagation* [Solomon, 1987] [Caseau and Laburthe, 1999].

After propagating all temporal constraints, the domains $\delta_i^t = [\alpha_i, \beta_i]$ are consistent for all i, and each customer can realise any service time within its temporal domain. A visit i can therefore be inserted between two scheduled visits j_1 and j_2 ($j_1 := \pi_{j_2}$ if

$$\alpha_{j_1} + d_{j_1,i} \leq \beta_i \wedge \max(\alpha_{j_1} + d_{j_1,i}, \alpha_i) + d_{i,j_2} \leq \beta_{j_2} \tag{3.3}$$

This observation can be used to update the set δ_i^{ip} of valid insertion positions of i, i.e. we remove j_2 from δ_i^{ip} if (3.3) is not fulfilled.

Clearly, the basic decisions of a search algorithm correspond to inserting visits into the routes. Note that inserting i into a route k amounts to setting δ_i^{veh} to $\{k\}$ and specifying the predecessor and successor visits $\pi_i := j_1$ and $\sigma_i := j_2$, respectively. In the model of Kilby et al. [2000], insertion decisions are only taken extrinsically, i.e. insertions are not triggered by constraint propagation. However, if the insertion of one visit leaves only one possible insertion position for another visit, this could be exploited implicitly. We therefore introduce boolean domains δ_i^{ins} for each $i \in I$. δ_i^{ins} is initially set to $\{\text{true}, \text{false}\}$ and reduced to $\{\text{true}\}$ when i is inserted.

We can now conceive a consistency test which triggers as soon as $|\{j \in \delta_i^{ip} \mid \delta_j^{ins} = \{\text{true}\}| = 1$, i.e. there is only one remaining insertion position for i. We then insert i in the corresponding route and insertion position. Additionally, we revise the insertion positions domain of i:

$$\delta_i^{ip} := \{j \in \delta_i^{ip} \mid \delta_j^{ins} = \{\text{true}, \text{false}\} \vee j = \sigma_i\}$$

Note that insertion also activates temporal propagation for the inserted visit (rules (3.1) and (3.2)).

In insertion methods, the sequence of trips is explicitly resolved. Furthermore, visits are assigned to dedicated vehicles. If vehicles cannot be distinguished, this introduces symmetry because assigning a tour to any of the vehicles results in isomorphic solutions. However, different vehicle characteristics or multiple depots can be easily accounted for. Campbell and Savelsbergh [2004] have recently noted that complicating side constraints like shift time limits, variable delivery volumes and multiple routes per vehicle can be easily incorporated in insertion-based methods.

3.3.2. Partial Path Methods

As insertion-based formulations, *partial path methods* are based on a previous/next representation for sequences of visits. However, visits are not explicitly assigned to routes. Basic decisions correspond to fixing the predecessors or successors, linking chains of trips (see Fig. 3.1b).

To illustrate a constraint logic for partial path methods, we let $\delta_i^{succ} \subseteq I$ and $\delta_i^{pred} \subseteq I$ denote the valid successors and predecessors of visit i, respectively. As above, $\delta_i^t = [\alpha_i, \beta_i]$ denotes the interval of valid start times for i. Note that we do not use any vehicle domains.

We will give examples of typical consistency tests in partial path methods. Temporal propagation can be carried out by using the following rules:

$$\min_{j \in \delta_i^{pred}} (\alpha_j + d_{j,i}) > \alpha_i \Longrightarrow \alpha_i := \min_{j \in \delta_i^{pred}} (\alpha_j + d_{j,i}) \tag{3.4}$$

$$\max_{j \in \delta_j^{succ}} (\beta_j - d_{i,j}) < \beta_i \Longrightarrow \beta_i := \max_{j \in \delta_j^{succ}} (\beta_j - d_{i,j}) \tag{3.5}$$

In contrast to the update rules (3.1) and (3.2) in insertion-based methods, (3.4) and (3.5) require minimisation or maximisation operations over all valid predecessors or successors. While this potentially yields a higher degree of consistency, it entails $\mathcal{O}(|I|)$ runtime for each execution. Clearly, analogous update rules could also be realised in insertion-based methods if predecessor and successor domains (like the above δ_i^{ip}) were used. However, this would sacrifice the runtime advantage of push-forward/push-backward propagation.

Using temporal information, we can adapt the successor domain δ_i^{succ} for each i as follows:

$$\alpha_i + d_{i,j} > \beta_j \Longrightarrow j \notin \delta_i^{succ}$$

In a final solution, each visit can only appear in one successor domain. We therefore impose an *alldifferent* constraints on the successor domain δ_i^{succ} [Focacci et al., 2002]. Predecessors and successors are then synchronised by the following rules:

$$i \notin \delta_j^{pred} \Longleftrightarrow j \notin \delta_i^{succ}$$

Note that these updates are not as strong as the above insertion checks (3.3) for insertion-based methods.

To the knowledge of the author, partial path methods have only been applied to travelling salesman problems with and without time windows (TSP/TSPTW), see e.g. Caseau and Laburthe [1997], Pesant et al. [1998] and Focacci et al. [2002]. In this class of problems, one can make use of the fact that we build only a single path. Additional propagational effect can then be gained by maintaining sets of visits which *must* precede or succeed a given customer [Langevin et al., 1993] [Pesant et al., 1998].

In travelling salesman and vehicle routing settings, we face the problem of subtours, i.e. closed circuits returning to a given customer or city [Dantzig et al., 1954] [Lawler et al., 1985]. Caseau and Laburthe [1997] and Pesant et al. [1998] independently proposed a *nocycle* constraint which keeps track of the beginning β_i and ending visit ϵ_i of the path in which each $i \in I$ is involved. With this information, we can impose

$$\delta_i^{succ} = \{j\} \Longrightarrow \beta_i \notin \delta_{\epsilon_j}$$

in order to avoid the end of the path to become connected to its beginning. Clearly, the problem of disconnected solutions is only important if temporal restrictions are loose. If time windows are tight, the temporal orientation of visits will make subtours unlikely, and the additional cost of *nocycle* may not pay off.

Extending partial path methods to vehicle routing settings is possible, but requires additional effort. Imposing restrictions on the vehicle level is more difficult since paths are initially built without any reference to vehicles. Basic decisions in search algorithms amount to fixing predecessor and successor variables. While insertion-based methods allow for inserting customers between two scheduled visits, predecessor and successor decisions are rather myopic [Caseau and Laburthe, 1997]. If we aim at minimising travel times, powerful lower bounds can be used, e.g. by assignment and minimum spanning tree relaxations [Pesant et al., 1998] [Focacci et al., 2002] or the Lagrange method [Caseau and Laburthe, 1997].

3.3.3. Disjunctive Methods

While the above methods have been developed in the field of vehicle routing and travelling salesman problems, *disjunctive scheduling* has its roots in research on job-shop scheduling [Roy and Sussman, 1964]. If travel times are interpreted as sequence-dependent setup times (see e.g. Allahverdi et al. [1999]), disjunctive scheduling also applies to routing problems [Beck et al., 2002].

Disjunctive constraints prevent the simultaneous or overlapping processing of activities (visits) without specifying their exact order. The basic idea corresponds to enforcing two customers i, j

which are assigned to the same vehicle to be placed either "i before j" or "j before i" (Fig. 3.1c). Using exemplary consistency tests, we shortly illustrate how this idea can be represented in constraint models. For an introduction to disjunctive scheduling, see e.g. Baptiste et al. [2001].

We only need two domains $\delta_i^{veh} \subseteq K$ and $\delta_i^t = [\alpha_i, \beta_i]$ for each visit $i \in I$. If for two visits i, j, we have $b_i + d_{i,j} \leq a_i$ ($b_j + d_{j,i} \leq a_j$), i can only be placed before (after) i. Otherwise, there is a disjunctive constraint between i and j which becomes operational by an appropriate consistency test. This test becomes active as soon as i and j are assigned to the same vehicle ($\delta_i^{veh} = \{k\} = \delta_j^{veh}$):

$$\alpha_i + d_{i,j} > \beta_j \Longrightarrow \alpha_i := \max(\alpha_i, \alpha_j + d_{j,i}) \quad (3.6)$$
$$\alpha_j + d_{j,i} > \beta_i \Longrightarrow \alpha_j := \max(\alpha_j, \alpha_i + d_{i,j}) \quad (3.7)$$

Rule (3.6) reflects the fact that if i cannot be placed before j, it must take place after j. The time window of i is then reduced accordingly. Update (3.7) applies to the symmetric situation, placing i before j. We therefore choose among the two different orders of i and j if one of the rules applies [Carlier and Pinson, 1989].

If we know that i and j cannot be placed on behind the other, we can enforce that i and j are assigned to different vehicles. If we assume that e.g. i is bound to vehicle k ($\delta_i^{veh} = \{k\}$), we can apply the following update:

$$\alpha_i + d_{i,j} > \beta_j \wedge \alpha_j + d_{j,i} > \beta_i \wedge \delta_i^{veh} = \{k\} \Longrightarrow \delta_j^{veh} := \delta_j^{veh} \setminus \{k\} \quad (3.8)$$

It should be mentioned that the above rules do not always ensure sufficient consistency for assignment feasibility. Imagine three customers i_1, i_2 and i_3 with initial start time windows $[a_{i_j}, b_{i_j}] = [0,1]\ \forall j \in \{1,2,3\}$ and travel times of 1 between all visits ($d_{i_{j_1}, i_{j_2}} = 1\ \forall j_1 \neq j_2$). If i_1, i_2 and i_3 are assigned to the same vehicle, the reader can easily verify that none of the rules (3.6), (3.7) or (3.8) has any effect. However, it is clear that the trips cannot be carried out by the same vehicle. A search algorithm must therefore take explicit disjunction decisions for customers whose order has not been implicitly defined by propagation.

The above basic scheme can be refined by generalising disjunctions to sets of activities [Caseau and Laburthe, 1994] [Caseau and Laburthe, 1995]. A review of consistency tests and degrees of consistency in disjunctive scheduling can be found in Dorndorf et al. [2000].

While insertion-based formulations assign customers to vehicles and fix their sequence at the same time, disjunctive methods take these decisions separately. Like in insertion methods, it is easy to incorporate constraints on the vehicle level.

Disjunctive scheduling has yielded quite impressive results on shop and machine scheduling problems, see e.g. Carlier and Pinson [1989] and Baptiste et al. [1995]. Beck et al. [2002] have recently described how vehicle routing and open shop scheduling problems can be mutually reformulated. Using constraint programming libraries for job-shop scheduling (using disjunctive methods) and routing problems (using previous/next representations) on original and reformulated problems, the authors argue that each technology performs best on the class of problems for which it has originally been developed.

For the following two chapters, we have therefore opted for insertion-based methods. As we will see, these techniques are able to incorporate multitudes of relevant constraints in airport demand and shift planning.

4. Workload Levelling in a Vehicle Routing Environment

What is chiefly needed is skill
rather than machinery.
— Wilbur Wright

Using the techniques described in the preceding chapter, we now tackle the *workload levelling problem* which was shortly described in Section 1.2.2. Its goal is the smoothing of a demand curve which is the superposition of tasks planned in tours. While workload levelling is based on the vehicle routing problem with time windows (VRPTW), the objective function aims at avoiding sharp demand peaks which are difficult to cover by staff. We will show that the problem is $\mathcal{NP}$-hard and present an algorithm for solution improvement by repeated relaxation and CP-based reoptimisation.

4.1. Introduction

Demand planning is a first and important step in the management and scheduling of ground handling workforce and equipment. Planners frequently try to get an overview of workforce demands by visualising the temporal superposition of all work tasks. The visualisation of a histogram of workloads in their temporal evolution ("demand curve") is helpful for

- an analysis of the magnitude of workload at different times of the day and week;
- an analysis of peak times with high workforce demand and traffic congestion;
- the determination of the temporal extent of workload within the day and the times in which nothing has to be done (e.g. in the night);
- an estimation of the necessary workforce size or equipment dimension;
- an assessment of the ability of different shift types to cover peak times (times of highest workload) efficiently.

Demand planning can also be seen as a preparatory step for shift planning. On the one hand, an analysis of demands can give an idea of how to cover the bottom-line demand by full time workers and peak demands by additional part-time staff. On the other hand, shift planning and rostering models usually aim at covering workloads given in discrete intervals. Furthermore, covering demand-based workloads can give a good approximation to task-level shift planning if the characteristics of the handling tasks are sufficiently homogeneous (see Section 1.2.3).

The workload is initially given as a set of tasks which is the result of matching task generation rules (engagement standards) to flight events. Each task is defined by a length, a location and an interval of admissible start times. Between the tasks, the equipment or worker possibly has to move from one location to another. We will always assume that the trip between tasks is performed directly before starting the latter tasks. Time windows are hard constraints: A worker

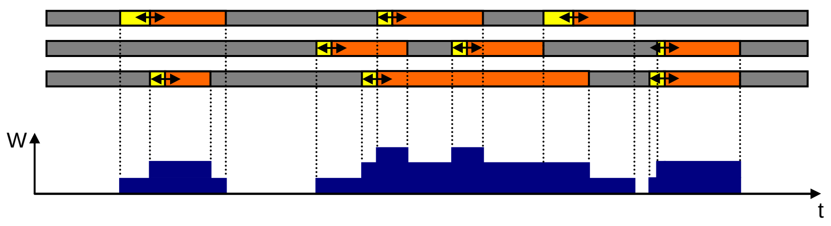


Figure 4.1.: Basic setting of workload levelling: (crew) tasks in tours, demand curve.

arriving too early must wait until the task's earliest start time, and a task must not start after its latest start time.

When planning mobile equipment like baggage vehicles, push-back tractors or catering trucks, we have a typical vehicle routing setting: Vehicles start a central depot, perform a set of tasks within the bounds of their time windows and return to the depot. We will not impose any capacity constraints: While some of the vehicles (like push-back tractors) are not used to carry goods, others (like baggage lorries) are configured ad-hoc to the necessary size for the given baggage. Since equipment is constantly available, a global tour model seems appropriate, i.e. the extent of tours equals the given planning time horizon which is typically one week (see the upper part of Fig. 4.1).

This global tour model is also a good approximation for planning workforce demands. Like mobile equipment, workers must carry out tasks, driving from one location to the other. Especially on the apron, travel times can be considerable and should not be neglected in the estimation of workloads. The work tasks for staff will also be planned in global tours, and we will account for work and travel times in the demand curve. Clearly, this only gives an approximation of actual workloads because workers are not constantly available. The global tour model will generally underestimate travel times because real workers start and end their shifts at prescribed times.

In contrast to equipment tasks, workforce tasks may be grouped into blocks of crew tasks which which will be carried out by teams of workers. Such crew tasks must be performed as much in parallel as possible. Working in teams is common e.g. in the cleaning of aircraft cabins.

The basic setting for demand planning is thus the vehicle routing problem with time windows (VRPTW) with additional crew constraints. Especially when planning equipment, we will be interested in a minimisation of the number of tours, giving a minimum number of devices which must be acquired. This is the objective of an algorithm which has already been used in the planning system before, see Kwasniok [1994]. It solves the routing problem as an assignment problem by a network flow algorithm for fixed tasks [Desrosiers et al., 1993] and by the classical Solomon insertion heuristic for movable tasks [Solomon, 1987]. If a scenario only contains fixed tasks, we cannot do much more except for minimising travel times.

In workforce planning, the tour model is mainly used to get an approximation of travel times, and we will be more interested in the temporal evolution of workforce demands. Demand peaks should be avoided by performing movable tasks at times of lower workload. While the minimisation of the number of tours tends to minimise the overall demand peak, it is not sufficient for avoiding local peaks. As a consequence, it does not make use of the freedom of fixing tasks at different times of their time windows. Generally speaking, we aim at a demand curve which is as smooth as possible in order to minimise the necessary workforce. This will avoid workforce activation problems related to sudden increases or decreases of workloads. The associated problem will be called the *workload levelling problem* (WLP), see also Fig. 4.1.

When levelling workloads, we will generally not be willing to increase the number of tours. The smoothing procedure will therefore be based on the result of tour minimisation. We can then try to fix the tasks to start times which give a levelled demand. The result of this initial phase can be improved by considering reattributions of tasks to different tours, giving them higher temporal degrees of freedom. Local search is a good means for this task. While well-known local exchange operators like 2-opt and 3-opt have proven to helpful in basic travelling salesman and vehicle routing problems, their usefulness is limited for the time window case [Caseau and Laburthe, 1999]. For more complex scenarios like the workload levelling problem (e.g. requiring crew task parallelism), the exploration of larger neighbourhoods is more appropriate. In order to enlarge the degrees of freedom in local search, *large neighbourhood search* (LNS) has proved to be a successful optimisation technique, see Section 3.2.

Large neighbourhood search combines advantages of local search methods with those of constraint programming. While good solutions can already be obtained in little time, LNS yields high-quality solutions if more runtime can be invested. Furthermore, constraint-based algorithms generally do not require a fundamental redesign if further constraints (e.g. precedence or different-tour constraints) come into play which is often the case in real-world optimisation problems.

The procedure not only needs to be robust with regard to future constraints. Planning scenarios on airports effectively exhibit quite different characteristics. As an example, the number of tasks will range from an order of 10^3 to 10^5, tasks can be movable only over some minutes or several hours, all tasks or none of the tasks can be grouped in crews of different sizes, and travel times can be absent or reach durations of up to one hour. The goal of this work is thus the development of an algorithm which is robust in terms of solution quality on a wide range of test cases.

The chapter is structured as follows: we first show that the workload levelling problem bears strong similarities to the resource levelling problem in project scheduling. Afterwards, a mathematical definition of workload levelling will be given. Paragraph 4.4 shows that the workload levelling problem is $\mathcal{NP}$-hard, justifying the search for appropriate heuristic algorithms. We present a constraint model for workload levelling which is the basis of a branch-and-bound algorithm, making use of powerful lower bounds. The overall local search scheme will be described in Section 4.8, and Section 4.9 shows how the results of the preceding tour minimisation phase can be preprocessed for the local improvement phase. Experimental results on realistic airport planning scenarios are given in Section 4.10. The chapter concludes by giving a summary and some future research directions.

4.2. Vehicle Routing and Resource Levelling

A task which is similar to workload levelling is investigated unter the term *resource levelling* in project scheduling. Resource levelling does not refer to a unique objective function; according to Brucker et al. [1999], a project scheduling problem is of a resource levelling type if “the objective function to be minimised represents some measure of the variation of resource utilisation”[1]. This comprises the following objectives:

1. Reduction of fluctuations in the pattern of resource usage [Easa, 1989];
2. Minimisation of the deviation from a target resource level [Brucker et al., 1999];
3. Minimisation of the excession of a target resource level [Brucker et al., 1999];
4. Making the resource utilisation approach a rectangular shape [Harris, 1990];
5. Gradual buildup of resource requirements towards a single peak and subsequent decline [Ahuja, 1976].

[1]In contrast, the minimisation of the number tours in vehicle routing can be compared to the resource investment problem, see Brucker et al. [1999].

Objective (1) is usually grasped by minimising the sum of the squares of resource requirements [Zimmermann and Engelhardt, 1998], (2) can be tackled by the sum of the squares of the absolute differences between requirements in consecutive periods. The sum of the positive differences will tend towards objective (3) [Brucker et al., 1999]. For objective (4), the sum of the absolute values of the differences can be taken [Neumann and Zimmermann, 1999], objective (5) is achieved by minimising the sum of the squares of resource changes [Ahuja, 1976].

All of these objective functions are *non-regular* because their value may decrease with higher makespans. Therefore, the minimum makespan is calculated in a first step and fixed in resource levelling. Likewise, we will base workload levelling on an existing algorithm for tour minimisation and impose an upper bound on the number of tours used in workload levelling.

Basic considerations of resource levelling were presented by Ahuja [1976]. The basic problem was described with the objective to attain a parabolic resource profile which gradually builds up and declines after a peak. A variance measure of the difference in resource usage from one period to the other is minimised by shifting non-critical tasks within their time windows. A brute-force approach for very small examples as well as a simple greedy heuristic are presented, fixing one activity after the other.

Easa [1989] devised an exact integer programming approach for minimising the absolute deviations from a target resource level. The decision variables are the shifts of the non-critical tasks within their time windows. The algorithm seems appropriate only for very small examples. No experimental results are presented.

The heuristic algorithm of Harris [1990] consists of sequentially fixing all tasks with the objective of making the histogram of a single resource approach a rectangle. Base intervals in which movable activities must take place are scheduled first to have a better guidance for the algorithm.

Seibert and Evans [1991] conduct a comprehensive study of important considerations in project scheduling with regard to the temporal evolution of resource demands, using early-start/late-start resource curves. The authors stress the importance of resource levelling especially in planning staff to avoid short-term hiring and firing with negative cost effects. A project scheduling software package is used for generating a levelled resource usage curve; its quality is measured by the sum of the squares of residuals to an initially assumed resource profile.

Bandelloni et al. [1994] describe a dynamic programming algorithm for resource levelling, minimising the squared deviations of requirements from their mean. The method starts by building a graph expressing interdependencies between noncritical activities due to succession relationships or objective function interactions. The components of this graph are successively split up by fixing start times of activities with few interactions.

Resource levelling with minimum and maximum time lags and with or without resource limits (restricted resource levelling) is treated by Zimmermann and Engelhardt [1998]. The study refers to different objective functions like maximum resource usage, sum-of-squares, deviation from a target level and absolute differences in requirements of consecutive time periods. The authors develop powerful lower bounds for the case of unrestricted resources and devise a branch-and-bound algorithm. The search tree is heuristically cut by filtered beam search. Test cases of up to 200 activities and five different resources are solved.

Neumann and Zimmermann [1999] show that resource levelling with minimum and maximum time lags is $\mathcal{NP}$-hard on the set of objective functions already used in Zimmermann and Engelhardt [1998]. Heuristic algorithms are presented to solve the restricted and unrestricted case by fixing activity start times one by one. A clever idea allows to reduce the complexity from pseudo-polynomiality in the time window widths to a polynomial runtime in the number of activities. For the restricted case, already fixed activities may have to be revised. Experimental results show that activities should be fixed in decreasing sizes of their time windows and on the basis of resource demands including base time intervals of near-critical activities.

More specific to airport staff planning, Nobert and Roy [1998] integrate the levelling of aircargo workloads in a shift scheduling model. Workloads are given by kilograms of freight which can be processed with interruptions. After solving the LP relaxation, resulting demands are rounded up. The resulting standard shift scheduling problem is then solved by integer programming.

To the knowledge of the author, no articles have been published on workload levelling in a vehicle routing scenario. One possible reason for this is that vehicle routing and project scheduling were investigated separately for a long time. Proposing an algorithm for the workload levelling problem will be a step in combining ideas from project scheduling and vehicle routing.

Still, the workload levelling problem differs from resource levelling in some aspects. Resources are normally levelled over the whole span of a project with a day as the smallest time unit [Ahuja, 1976]. If resources refer to skilled workers, resource levelling is used because staff needs training for a job, and operations will not be efficient if personnel is frequently employed or set free [Seibert and Evans, 1991]. In contrast, workload levelling will be used on a more tactical or operational level. It deals with workload fluctuations over a day or a week and will usually be performed on a minute discretisation.

For workload levelling, minimising the sum of the squares of workloads seems appropriate. With this objective, potentially overlapping tasks will be fixed in a way to avoid local demand peaks (within the limits of their time windows and tours). The global demand maximum may also be decreased because the objective of the algorithm mentioned in Section 4.1 is tour minimisation, and it is heuristic in nature. Since we will consider travel times in the demand curve, the workload levelling algorithm will also tend to avoid unnecessary travel times. Because the main source of improvement lies in the fixation of task start times, workload levelling will only be applied to scenarios containing movable tasks. Typical airport applications comprise cabin cleaning or operations departments.

While transferring ideas from resource levelling (like time window propagation and search strategies) can be fruitful, none of the approaches in the literature appears to be sufficient for workload levelling. Many approaches in the literature are restricted to very small test cases. More important is the difference in problem complexity: in project scheduling, temporal relations between activities are fixed, i.e. there is a fixed graph of potential temporal influences between activities. This is not the case in workload levelling since temporal dependencies are a consequence of the attribution of tasks to positions in tours.

We therefore have two degrees of freedom: The attribution of tasks to tours and the temporal fixing of tasks within their tours. These decisions are interdependent since a task's time window is restricted by its surrounding tasks. Experimental results will indicate that reattributing tasks to different tours has a strong impact on solution quality.

4.3. Mathematical Model

We now give a mathematical description of the workload levelling problem. Let I be the set of tasks and R the tours as given by the result of the preliminary tour minimisation algorithm. T is the (discretised) time horizon, covering all tasks. Each tour will start at an artificial origin task and end at a destination task; I^o and I^d will denote the sets of all origin and destination tasks, respectively. Some or all tasks may be grouped into blocks $C \subseteq I$ of *crew tasks*; the set of all crews will be denoted by $\mathcal{C}$[2].

We introduce the following *parameters*:

[2]Note that the uppercase letter C suggests that crews are sets of tasks, see also Appendix A.

$[a_i, b_i] \subset T$	start time window of task $i \in I \cup I^o \cup I^d$
l_i	length of task $i \in I \cup I^o \cup I^d$ (0 for delimiter tasks)
$d_{i,j}$	travel time between tasks $i \in I \cup I^o, j \in I \cup I^d$

Origin tasks $i \in I^o$ will be fixed to the scenario start ($a_i = b_i = \min T$), destination delimiters $i \in I^d$ to the scenario end ($a_i = b_i = \max T$). Note that by fixing the depot travel times $d_{i,j}$ for $i \in I^o$ or $j \in I^d$, we only allow for single-depot models.

The following *variables* are used:

R_i	tour of task $i \in I \cup I^o \cup I^d$
$succ_i$	successor of $i \in I \cup I^o$
$pred_i$	predecessor of $i \in I \cup I^d$
T_i	start time of $i \in I \cup I^o \cup I^d$

Tours of delimiter tasks $i \in I^o \cup I^d$ are fixed in advance since each tour has unique origin and destination tasks. Note that the variable information is somewhat redundant since if all predecessors are fixed, we also know the successors and tours. *Schedules* Σ are given by the fixation of predecessor, successor, tour and start time for each work task i, i.e. $\Sigma = \{(R_i, (pred_i, succ_i), T_i) \mid i \in I\}$.

For a given schedule Σ, the number of tasks (workforce demand) whose travel times or work durations take place at $t \in T$ is

$$W(t) := |\{i \in I \mid T_i - d_{pred_i,i} \leq t < T_i + l_i\}|,$$

see also Fig. 4.1. We now define the workload levelling problem as

$$\min z = \sum_{t \in T} W(t)^2 \tag{4.1}$$

subject to

Tours

$$R_i \in R \;\; \forall i \in I \cup I^o \cup I^d \tag{4.2}$$

Time windows

$$T_i \in [a_i, b_i] \;\; \forall i \in I \cup I^o \cup I^d \tag{4.3}$$

Tour temporal relation

$$T_i + l_i + d_{i,succ_i} \leq T_{succ_i} \;\; \forall i \in I \cup I^o \tag{4.4}$$

Predecessor-successor consistency

$$pred_i = j \Leftrightarrow succ_j = i \;\; \forall i \in I \cup I^d, \forall j \in I \cup I^o \tag{4.5}$$

Predecessor-tour consistency

$$\exists\, j : pred_i = j \wedge R_i = r \Rightarrow R_j = r \;\; \forall i \in I \cup I^d \tag{4.6}$$

Tour difference

$$R_i \neq R_j \;\; \forall i, j \in C, i \neq j, C \in \mathcal{C} \tag{4.7}$$

Crew temporal relation

$$\begin{array}{r} T_j \geq T_i + (a_j - a_i) \\ T_j + l_j \leq T_i + l_i + (a_j - a_i) \end{array} \;\; \forall C \in \mathcal{C}, \forall i, j \in C : i \neq j, l_i \geq l_j \tag{4.8}$$

Constraint (4.2) restricts tasks to use the given tours. Start times are restricted in terms of time windows and successor/predecessor relations by (4.3) and (4.4). Equations (4.5) and (4.6)

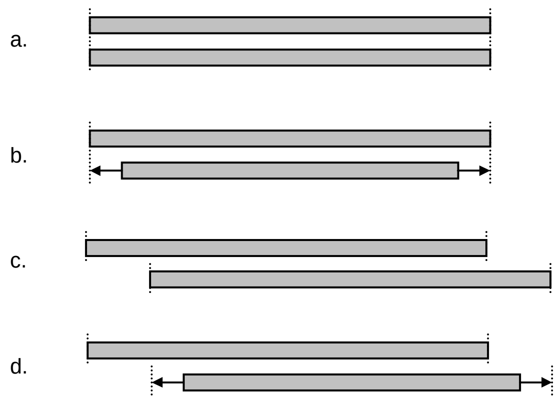


Figure 4.2.: Temporal relation of crew tasks.

ensure consistency between predecessors and successors as well as between predecessors and tours. Equation (4.7) constrains crew tasks to be placed in different tours. The parallelism of crew tasks is expressed by (4.8).

Let us consider the effect of constraint (4.8) on the temporal relation of two tasks $i, j \in C$ of the same crew $C \in \mathcal{C}$ (Fig. 4.2). Let first i, j have equal lengths $l_i = l_j$ and same earliest start times $a_i = a_j$. Then (4.8) constrains i, j to be placed in parallel (see Fig. 4.2a). Let now the duration of the first task be greater ($l_i > l_j$) but the start times still be equal. Then (4.8) says that j should start after i but should end before j, i.e. j is to be placed under i (Fig. 4.2b).

We now consider the case of identical durations but different start times. W.l.o.g. let $a_j > a_i$. Then j must take place exactly $a_j - a_i$ time units later than i (Fig. 4.2c), i.e. $a_j - a_i$ is an offset between i and j. If finally $l_i > l_j$ and $a_j > a_i$ (Fig. 4.2d), then j must be placed within a time window identical to the length of i, but with an offset of $a_j - a_i$ with regard to i's time window.

Constraints (4.8) thus ensure that crew tasks with $a_i = a_j$ take place as much in parallel as possible. If $a_j - a_i \neq 0$, the difference $a_j - a_i$ can be interpreted as an offset for the relative position of the time windows. A typical application is a supervisor arriving somewhat earlier at a working location to prepare jobs for a group of e.g. cabin cleaners. The offset in the original earliest start times a_i, a_j then always entails an equal offset in the planned start times T_i, T_j.

For the realisation of constraints (4.8), crew tasks will be sorted by decreasing durations. Nevertheless, (4.8) still describes a number of constraints which is quadratic in the number of crew tasks. Let $i, j, k \in C$ be pairwise different tasks of $C \in \mathcal{C}$ with $l_i \geq l_j \geq l_k$. Then constraints (4.8) for the task pairs (i, j) and (j, k) read as

$$\begin{gathered} T_j \geq T_i + (a_j - a_i) \\ T_j + l_j \leq T_i + l_i + (a_j - a_i) \end{gathered}$$

and

$$\begin{gathered} T_k \geq T_j + (a_k - a_j) \\ T_k + l_k \leq T_j + l_j + (a_k - a_j) \end{gathered}$$

It follows

$$\begin{gathered} T_k \geq T_j + (a_k - a_j) \geq T_i + (a_j - a_i) + (a_k - a_j) = T_i + (a_k - a_i) \\ T_k + l_k \leq T_j + l_j + (a_k - a_j) \leq T_i + l_i + (a_j - a_i) + (a_k - a_j) = T_i + l_i + (a_k - a_i) \end{gathered}$$

which are exactly the constraints (4.8) for the task pair (i, k). As a consequence, the crew temporal relation is transitive, and it suffices to define constraints for adjacent tasks in an ordering by decreasing lengths, i.e. we need $2(|C| - 1)$ constraints for a crew $C \in \mathcal{C}$.

4.4. Computational Complexity

General resource levelling is known to be $\mathcal{NP}$-hard [Neumann and Zimmermann, 1999]. Similarly, the $\mathcal{NP}$-hardness of workload levelling can be shown.

Theorem 1. *The workload levelling problem (WLP) is $\mathcal{NP}$-hard.*

Proof. The decision problem corresponding to WLP is: Is there a schedule Σ with objective function value $z(\Sigma) \leq B$? To show that WLP is $\mathcal{NP}$-hard, we will use a polynomial transformation from the PARTITION problem which is $\mathcal{NP}$-hard [Garey and Johnson, 1979]. The PARTITION problem is: Given a finite set E and a size $s(e) \in \mathbb{N}_0$ for each $e \in E$, is there a subset $E' \subseteq E$ such that

$$\sum_{e \in E'} s(e) = \sum_{e \in E \setminus E'} s(e)$$

For the WLP, the time horizon is chosen as $T = [0, 2[$, all travel times $d_{i,j}$ are 0. For each given e with size $s(e) \in \mathbb{N}_0$, we build a crew of $s(e)$ tasks all of which have a start time window $[0, 1]$ and a length of 1. Note that this set of crew tasks has an offset of 0 and thus has to be placed in parallel. Clearly, for each e with $s(e) = 1$, we construct a single non-crew task. It is evident that this transformation is polynomial.

We now set $B := 2\left(\frac{|E|}{2}\right)^2$. For a given partition of E into E' and $E \setminus E'$, we can build a solution to WLP by fixing all tasks corresponding to elements in E' to start time 0 and tasks of $E \setminus E'$ to 1. This gives a schedule Σ for WLP with objective function value $z(\Sigma) = B$.

Conversely, if $z(\Sigma) \leq B$ for the given problem with $h := \sum_{e \in E} s(e)$ tasks, this (optimal) solution must be given by $\frac{h}{2}$ tasks being fixed to start time 0 and $\frac{h}{2}$ tasks fixed to 1, respectively. Let E' be the tasks corresponding to the crews (or single tasks) fixed to 0 (note that a block of crew tasks starts at either 0 or 1). Now E' and $E \setminus E'$ is a partition of E with $\sum_{e \in E'} s(e) = \frac{h}{2} = \sum_{e \in E \setminus E'} s(e)$. □

Because WLP is $\mathcal{NP}$-hard, we cannot expect that a polynomial time algorithm exists which solves the problem exactly. Empirical experience shows that typical test cases are very demanding as well, justifying the search for powerful heuristics.

4.5. Constraint Model

We will now devise a constraint model for the workload levelling problem. It will be based on an insertion logic, inserting tasks sequentially into an emerging set of shifts (see Section 3.3). Travel times can be asymmetric (e.g. due to one-way connections on the apron), but are assumed to obey the triangle inequality. The following domains will be used, given with the corresponding decision variable in the mathematical model:

Name	mathematical variable	CP domain
start time	T_i	$\delta_i^{start} = [\alpha_i, \beta_i]$
inserted variable	—	δ_i^{ins}
current predecessor	$pred_i$	π_i
current successor	$succ_i$	σ_i
predecessor variable	$pred_i$	δ_i^{pred}
insertion positions	$succ_i$	δ_i^{ip}
tour variable	R_i	δ_i^{tour}

The domain δ_i^{start} gives the start time window of task i. It is initialised to $\delta_i^{start} = [\alpha_i, \beta_i] := [a_i, b_i]$ and will be adapted if time windows are restricted by inserted tasks.

The binary variable domain δ_i^{ins} will be $\{\text{false}, \text{true}\}$ for relaxed tasks and will become $\{\text{true}\}$ upon insertion. An inserted task has current predecessor and successor tasks π_i and σ_i in its tour. As described in Section 3.3.1, π_i and σ_i are no constraint variables in the proper sense. Instead of shrinking monotonically, these variables give the (intermediary) predecessors and successors in the partial schedule, containing single values which change with new insertions. For relaxed tasks, π_i and σ_i are set to *NIL*.

In contrast, δ_i^{pred} gives the potential predecessor tasks: for inserted tasks, this includes the actual predecessor in the tour as well as unscheduled tasks which may be placed before the task. For tasks which are not inserted, the domain gives the tasks which may become predecessors of i in a tour. Similar to the predecessor variable, δ_i^{ip} gives the potential successors for an uninserted task i. While δ_i^{pred} converges to the predecessor in the final schedule, δ_i^{ip} becomes empty when i is inserted. Insertion positions are thus only temporarily used for tasks which are not yet inserted.

The reason for this asymmetry is propagation efficiency. The evaluation of insertion positions which are feasible with regard to the time windows is most efficiently done from the unscheduled task's perspective. As soon as a task is inserted, these insertion checks will be disabled by setting the insertion positions δ_i^{ip} to $\emptyset$. Valid insertion positions are primarily stored as successors because a set of successors can efficiently be synchronised with the predecessor domain of another task (cf. the insertion positions test below). Contrasting to the insertion positions variable, we need full predecessor information in δ_i^{pred} which always includes the actual predecessor. This information will be used for the scheduling of potential travel times (see Section 4.7).

Finally, δ_i^{tour} gives the potential tours to which i may be attributed. For scheduled tasks, this domain will contain a single value equal to the task's actual tour.

In the following, we will describe the consistency tests. It will become clear that the insertion-based model is particularly well-suited for local reoptimisation which can e.g. be exploited for an efficient evaluation of insertion positions. For each local step, the insertion and start time decisions for a set I^{rel} of work tasks will be released, and for each $i \in I^{rel}$ the possible insertion positions will be evaluated. Let

$$\begin{aligned}\Phi(i) := \{(j_1, j_2) \,|\, j_1, j_2 \in I \setminus I^{rel},\, &\alpha_{j_1} + l_{j_1} + d_{j_1,i} \le b_i \,\wedge \\ &\max(\alpha_{j_1} + l_{j_1} + d_{j_1,i}, a_i) + l_i + d_{i,j_2} \le \beta_{j_2}\}\end{aligned}$$

be the set of pairs of tasks between which i can be inserted and $\Phi := \bigcup_{i \in I^{rel}} \Phi(i)$ the set of all insertion positions. We will express the time complexity of consistency tests in terms of the cardinalities $|I^{rel}|$ and $|\Phi|$. Note that because a task can always be inserted at its former position, we have $|I^{rel}| \le |\Phi|$.

Tour temporal relation For each work task $i \in I$, we have the following update rules:

$$\begin{aligned}\alpha_{\pi_i} + l_{\pi_i} + d_{\pi_i,i} > \alpha_i &\Longrightarrow \alpha_i := \alpha_{\pi_i} + l_{\pi_i} + d_{\pi_i,i} \\ \beta_i - d_{\pi_i,i} - l_{\pi_i} < \beta_{\pi_i} &\Longrightarrow \beta_{\pi_i} := \beta_i - d_{\pi_i,i} - l_{\pi_i}\end{aligned}$$

The consistency test will be executed upon each change of δ_i^{start}, π_i or σ_i. Its time complexity is in $\mathcal{O}(1)$ for each task i.

Crew temporal relation Let $C = (i_1, \dots, i_n) \in \mathcal{C}$ be a crew such that $l_{i_j} \ge l_{i_{j+1}}$ for all j.

The start times of adjacent crew tasks i_j, i_{j+1} are updated by the following rules:

$$\begin{aligned}
\alpha_{i_j} + (a_{i_{j+1}} - a_{i_j}) > \alpha_{i_{j+1}} &\Longrightarrow \\
&\alpha_{i_{j+1}} := \alpha_{i_j} + (a_{i_{j+1}} - a_{i_j}) \\
\alpha_{i_{j+1}} - (l_{i_j} - l_{i_{j+1}}) - (a_{i_{j+1}} - a_{i_j}) > \alpha_{i_j} \quad &\Longrightarrow \\
&\alpha_{i_j} := \alpha_{i_{j+1}} - (a_{i_{j+1}} - a_{i_j}) + (l_{i_{j+1}} - l_{i_j}) \\
\beta_{i_j} + (l_{i_j} - l_{i_{j+1}}) + (a_{i_{j+1}} - a_{i_j}) < \beta_{i_{j+1}} &\Longrightarrow \\
&\beta_{i_{j+1}} := \beta_{i_j} + (l_{i_j} - l_{i_{j+1}}) + (a_{i_{j+1}} - a_{i_j}) \\
\beta_{i_{j+1}} - (a_{i_{j+1}} - a_{i_j}) < \beta_{i_j} \quad &\Longrightarrow \\
&\beta_{i_j} := \beta_{i_{j+1}} - (a_{i_{j+1}} - a_{i_j})
\end{aligned}$$

This consistency test is executed upon the range events of $\delta_{i_j}^{start}$ and $\delta_{i_{j+1}}^{start}$. As for the tour temporal relations, its time complexity is constant for each pair (i_j, i_{j+1}).

Insertion positions We evaluate insertion positions for all released tasks $i \in I^{rel}$. The potential successor set δ_i^{ip} is divided into a set of unscheduled tasks $\Gamma(i)$ and a set of inserted tasks, defining pairs $\Xi(i)$ of tasks between which i may be inserted:

$$\begin{aligned}
\Gamma(i) &:= \{j_2 \,|\, j_2 \in \delta_i^{ip}, \delta_{j_2}^{ins} = \{\text{false}, \text{true}\}\} \\
\Xi(i) &:= \{(j_1, j_2) \,|\, j_2 \in \delta_i^{ip}, \delta_{j_2}^{ins} = \{\text{true}\}, j_1 = \pi_{j_2}\}
\end{aligned}$$

Note that the predecessors j_1 in $\Xi(i)$ are the inserted tasks of δ_i^{pred}. The sets $\Gamma^{del}(i)$ and $\Xi^{del}(i)$ give the uninserted tasks which cannot be successors of i and the pairs of inserted tasks between which i cannot be inserted due to temporal restrictions:

$$\begin{aligned}
\Gamma^{del}(i) &:= \{j_2 \in \Gamma(i) \,|\, \alpha_i + l_i + d_{i,j_2} > \beta_{j_2}\} \\
\Xi^{del}(i) &:= \{(j_1, j_2) \in \Xi(i) \,|\, \alpha_{j_1} + l_{j_1} + d_{j_1,i} > \beta_i, \\
&\qquad \max(\alpha_{j_1} + l_{j_1} + l_{j_1 i}, \alpha_i) + l_i + d_{i,j_2} > \beta_{j_2}\}
\end{aligned}$$

The potential successors and predecessors are updated as follows:

$$\begin{aligned}
\delta_i^{ip} &:= \delta_i^{ip} \setminus (\Gamma^{del}(i) \cup \{j_2 \,|\, (j_1, j_2) \in \Xi^{del}(i)\}) \\
\delta_i^{pred} &:= \delta_i^{pred} \setminus \{j_1 \,|\, (j_1, j_2) \in \Xi^{del}(i)\}
\end{aligned}$$

This check must be performed upon changes of the start time windows δ_j^{start} of potential predecessors and successors, upon their insertions (δ_j^{ins} domain events) and upon changes of i's start time window δ_i^{start}. The cost of the test is in $\mathcal{O}(|\Phi|)$.

Predecessor update Predecessor information is adjusted upon updates of insertion position domains. All tasks which are not inserted and do not allow for i as successor can be removed from δ_i^{pred}:

$$\delta_i^{pred} := \delta_i^{pred} \setminus \{i' \in \delta_i^{pred} \,|\, \delta_{i'}^{ins} = \{\text{false}, \text{true}\}, i \notin \delta_{i'}^{ip}\}$$

If i is inserted, we will additionally remove all scheduled predecessors except for its actual predecessor π_i:

$$\delta_i^{ins} = \{\text{true}\} \Longrightarrow \delta_i^{pred} := \delta_i^{pred} \setminus \{i' \in \delta_i^{pred} \,|\, \delta_{i'}^{ins} = \{\text{true}\}, \pi_i \neq i'\}$$

Note that this consistency test is unidirectional and does not remove potential successor tasks from an insertion positions domain. One predecessor update constraint is used for each task in $I^{rel} \cup \bigcup_{i \in I^{rel}} \{j_2 \,|\, (j_1, j_2) \in \Phi(i)\}$. The test will be triggered for each change of $\delta_{j_1}^{ip}$ for j_1 such that $(j_1, j_2) \in \Phi(i)$ as well as upon insertion of i (δ_i^{ins} domain event). The complexity of the test is in $\mathcal{O}(|\Phi|)$ if containment in δ_i^{ip} can be checked in constant time (e.g. by a bit vector representation).

Different tours Each crew task must be inserted into a different tour. An *alldifferent*$(\{\delta_i^{tour} \mid i \in C\})$ global constraint is used for every crew $C \in \mathcal{C}$. The *alldifferent* constraint with incremental updates triggers on all domain events and has a runtime in $\mathcal{O}(|C|^2|R|^2)$ for a complete branch of the search tree [Régin, 1994] [Régin, 2000].

Tour-task consistency While insertion positions and predecessors are restricted by the insertion positions constraint, tour domains may be constrained by the different tours test. This information must be made consistent. If a task is inserted, nothing has to be done. If $\delta_i^{ins} = \{\text{false}, \text{true}\}$, we perform the following updates:

$$\begin{aligned}
\delta_i^{ip} &:= \delta_i^{ip} \setminus \{j \in \delta_i^{ip} \mid \delta_j^{tour} = \{r\}, r \notin \delta_i^{tour}\} \\
\delta_i^{pred} &:= \delta_i^{pred} \setminus \{j \in \delta_i^{pred} \mid \delta_j^{tour} = \{r\}, r \notin \delta_i^{tour}\} \\
\delta_i^{tour} &:= \{r \in \delta_i^{tour} \mid \exists j \in \delta_i^{ip} : \delta_j^{ins} = \{\text{true}\}, \delta_j^{tour} = \{r\}\}
\end{aligned}$$

The test could be made somewhat stronger by deleting insertion positions j such that $\delta_j^{tour} \cap \delta_i^{tour} = \emptyset$, but for efficiency reasons, the rule propagates for $|\delta_j^{tour}| = 1$. It is therefore only triggered on the bound events of the tour variables of insertion positions (predecessors and successors) and on domain events of δ_i^{tour} and δ_i^{ip}. Because for uninserted tasks, δ_i^{ip} and δ_i^{pred} are always conform in giving the same insertion positions, triggering on δ_i^{pred} would be redundant.

Note that the rule for updating δ_i^{tour} only considers inserted tasks j. This is correct because if there is no inserted task j in tour r before which i can be inserted, i can never be attributed to r. For the same reason, there is no meaning in triggering the constraint upon insertions of potential predecessors and successors of i.

The consistency test adapts the predecessor information in δ_i^{pred}, but because the predecessor update rule described above is unidirectional, this information will never become propagated to other tasks' successor domains δ_j^{ip}. What may seem a "consistency gap" will be little harmful in practice because insertion positions are only evaluated for uninserted tasks j. Because the tour variable δ_j^{tour} of uninserted tasks rarely becomes bound, we will not expect that hardly ever such a task j has to be deleted from δ_i^{pred}.

Again assuming set containment checks to run in constant time, the predecessor and successor update run in $\mathcal{O}(|\Phi|)$ time each. Because the number of possible tours for i cannot be greater than its insertion positions, the tour domain update has the same complexity, yielding an overall runtime in $\mathcal{O}(|\Phi|)$.

Task insertion When the set $\{j \in \delta_i^{ip} \mid \delta_j^{ins} = \{\text{true}\}\}$ of insertion positions for i reduces to a single task j_2, i can be inserted between $j_1 := \pi_{j_2}$ and j_2, entailing the following updates:

$$\begin{aligned}
\delta_i^{ip} &:= \emptyset \\
\delta_i^{tour} &:= \{r\} \\
\delta_i^{ins} &:= \{\text{true}\} \\
\pi_i &:= j_1 \quad \sigma_i := j_2 \\
\sigma_{j_1} &:= i \quad \pi_{j_2} := i
\end{aligned}$$

The constraint only triggers on domain events of δ_i^{ip}. Additional triggering upon insertions of any of the $j \in \delta_i^{ip}$ would be redundant because an increase in the number of inserted successors can never result in an insertion of i. The main purpose of the task insertion constraint is to trigger the temporal propagation for i. Its cost is in $\mathcal{O}(|\Phi|)$.

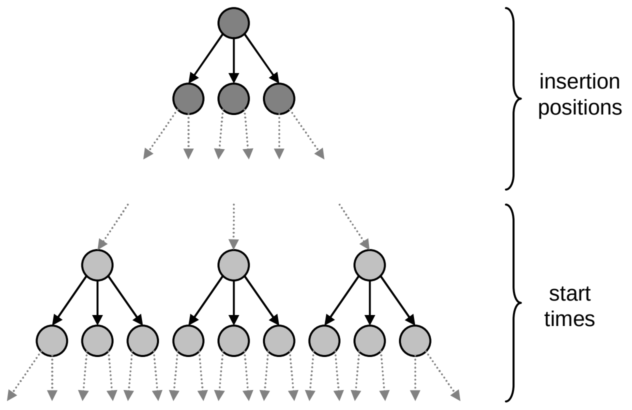


Figure 4.3.: Structure of the search tree.

Only the insertion positions, predecessor update and tour-task consistency tests trigger upon domain changes of tasks which depend on the set of relaxed tasks or on the insertion positions. We will therefore dynamically add these constraints for each local step, registering the tests only on the respective domains of insertion positions (taken from $\Phi(i)$). This avoids redundant executions of consistency tests as far as possible. Note that even if the task insertion test is only necessary for released tasks, we can make it a static part of the constraint model because it will effectively only be triggered for released tasks.

The constraint model presented above exhibits some symmetry which can deteriorate the efficiency of branch-and-bound algorithms. One symmetry is with regard to the tours: The chains of tasks in a solution can be assigned to any of the $|R|$ tours, yielding equivalent solutions (see also Section 3.3.1). This symmetry is not harmful in local search because we will never release a total tour, and the task context in a tour will always break the symmetry.

Another potential symmetry relates to the tasks. Especially crew tasks often have identical time windows, lengths and take place at the same location. Every single task can be attributed to any permutation of the tours, yielding isomorphic solutions. To avoid this symmetry, tasks of identical time windows, lengths, locations and crews (or non-crew tasks) are grouped. A partial ordering is defined on the tasks and tours such that the tasks of each such group are assigned in non-decreasing order to the tours. This assignment will be ensured by a special consistency test, preventing the search algorithm from repeatedly evaluating isomorphic situations.

4.6. Branch-and-Bound

As described in Section 3.2, large neighbourhood search (LNS) consists of the repeated relaxation of decisions and reoptimisation. While the relaxation amounts to resetting tour attributions of tasks and their time windows, reoptimisation will consist of a restricted branch-and-bound search. Before it is shown how the search tree can be heuristically restricted to promising parts of the solution space, the basic branch-and-bound scheme will be described.

For each of the relaxed tasks, two types of decisions must be taken: Tasks have to be inserted into the plan, and they must be fixed in time. Because a task's time windows can be restricted by insertion, we will first insert tasks and then fix start times. As a consequence, the search tree will be structured into two levels: an upper level where insertion decisions are taken and a lower level on which tasks are fixed to start times (cf. Fig. 4.3).

The basic branch-and-bound search is outlined by Algorithm 3. The input is a partial schedule $\hat{\Sigma}$ (the result of relaxing task decisions) and an incumbent solution Σ^* of value z^* which is identical to the schedule before relaxation. The algorithm uses a stack $\mathcal{N}$ of tree nodes which still have to be processed.

Algorithm 3 WLP branch-and-bound.

```
1: $\mathcal{N} \leftarrow (\hat{\Sigma})$
2: **while** there are nodes on $\mathcal{N}$ **do**
3:     pop a partial schedule $\Sigma'$ from $\mathcal{N}$
4:     **if** there are uninserted tasks in $\Sigma'$ **then**
5:         choose task $i$ with $\delta_i^{ins} = \{\text{false}, \text{true}\}$
6:         **for all** insertion positions $j \in \delta_i^{ip}$ **do**
7:             perform trial insertion of $i$ before $j$ ($\delta_i^{ip} := \{j\}$)
8:             calculate lower bound $LB(\Sigma', \delta_i^{ip} := \{j\})$
9:         **end for**
10:        **for all** insertion positions $j \in \delta_i^{ip}$ such that $LB(\Sigma', \delta_i^{ip} := \{j\}) < z^*$ **do**
11:            create a child of $\Sigma'$, assigning $\delta_i^{ip} := \{j\}$
12:            push child on node stack $\mathcal{N}$
13:        **end for**
14:    **else if** there are unfixed tasks **then**
15:        choose a task $i$ with $|\delta_i^{start}| > 1$
16:        **for all** start times $\tau \in \delta_i^{start}$ **do**
17:            perform trial fixation of $\delta_i^{start}$ to $t$
18:            calculate lower bound $LB(\Sigma', \delta_i^{start} := \{t\})$
19:        **end for**
20:        **for all** start times $t \in \delta_i^{start}$ such that $LB(\Sigma', \delta_i^{start} := \{t\}) < z^*$ **do**
21:            create a child of $\Sigma'$, assigning $\delta_i^{start} := \{t\}$
22:            push child on node stack $\mathcal{N}$
23:        **end for**
24:    **else if** $z(\Sigma') < z^*$ **then**
25:        $\Sigma^* \leftarrow \Sigma'$
26:        $z^* \leftarrow z(\Sigma')$
27:    **end if**
28: **end while**
```

Constraint propagation is performed throughout the search tree, i.e. after each insertion or fixation of start times, the consequences on other variables are propagated. This may fix decisions for further tasks. As an example, when a crew task is fixed in time, all of its adjacent crew tasks will become fixed if all tasks have equal lengths.

Most of the consistency tests described above turned out to have little immediate cost consequences. For the trial insertions and fixations (lines 7 and 15 of the algorithm), propagation was therefore restricted to the temporal consistency tests.

Some design decisions were left open so far. In lines 5 and 13, decisions for tasks i which are to be inserted or fixed are taken. The order of task insertions and fixations has a considerable impact on the algorithm's performance. *Dynamic variable ordering* will be used, i.e. these decision will depend on the state of the constraint model [Dechter, 2003]. In both cases, variants of the fail-first principle will be used, meaning that we should first take tasks whose insertion positions or start times are most restricted [Haralick and Elliott, 1980].

For task insertions, we will first insert tasks belonging to larger crews because larger crew blocks will be more restricting. Within the tasks of same crew size, tasks with few insertion positions are preferred, i.e. tasks are sorted by increasing sizes of δ_i^{ip}. If these are still equal, tasks with smaller time windows δ_i^{start} are preferred. Remaining ties are broken arbitrarily. When fixing start times, tasks with smaller time windows δ_i^{start} are taken first. In case of ties, tasks of larger crews are preferred. Remaining ambiguities are resolved arbitrarily (e.g. by task order).

Furthermore, we must specify in which order the children of a search node are explored, i.e. in which order they are written on the node stack (lines 10 through 13 and 20 through 23 of Algorithm 3). Because the lower bound which will be described in Section 4.7 gives good hints of promising branches, this lower bound is always used as a first criterion. For tie breaking, the minimum objective function resulting from an insertion or start time fixation has shown to provide good guidance.

Even for modestly sized sets of relaxed tasks, a complete branch-and-bound search would be too costly in terms of computation time. Additionally, each tree search only represents one local exchange while considerable improvements on large-scale scenarios can only be achieved by many local steps. The single steps should then not necessitate too much time. Among the different schemes to prune the search space heuristically, *limited discrepancy search* (LDS) proposed by Harvey and Ginsberg [1995] is one of the most successful techniques in recent years.

The basic observation which has led to the development of LDS is that pure construction heuristics, taking decisions one by one, often lead to good solutions. On some instances, optima may be reached, while on others, only few decisions of the heuristic are wrong. Interpreted in a tree search context, a construction heuristic corresponds to a descent on the best branches of the tree if the children at each node are ordered by the heuristic criterion. If now the heuristic fails to find a better solution, the intuition says that we should follow its decisions at all but one decision point. This discrepancy from the best path can be taken at any node. If there is no improvement, we should try two discrepancies and so on. If the best decisions correspond to left branches, we thus allow a limited number of deviations to the right. Note that this first corresponds to a depth-first tree traversal on the leftmost branch while the breadth of search is iteratively increased.

For a tree search with discrepancy D, we visit leaves in which a maximum of D discrepancies from the best decisions is summed up over all of the decision points. Limited discrepancy search now increases the number D of allowed discrepancies up to a given limit L_D, starting with $D = 0$ which corresponds to following the best valuation at all decision points. This of course means that in a run for a given D, the parts of the search trees for $D' = 0, \dots, D - 1$ will be revisited. But because we may have found good solutions in preceding runs, the gain in upper bound quality usually outweighs the extra effort. It can be easily verified that despite of repetitions, complexity of LDS is in $\mathcal{O}(n^{L_D})$ if n is the number of decision points.

From a local search viewpoint, LDS allows the exploration of a neighbourhood in a time which is polynomial in the number of relaxed tasks if the discrepancy limit L_D is fixed. Clearly, the runtime is still exponential in L_D.

First experiments have shown that when low discrepancy limits were used, the algorithm often failed to find improving solutions. On the other hand, solution times grew rapidly with higher limits. A variation of the basic LDS schema was therefore used: Instead of consuming discrepancies for all deviations from the best path, discrepancy is only accounted if the first decision on a branch results in a feasible improving solution. This means that each time an improving solution is found, this fact is backpropagated to all decision points which have led to the solution. When the next alternative at a decision point is evaluated, discrepancy is only consumed if the preceding branch was marked as successful.

As a consequence, an improving solution can always be found with a discrepancy limit of $L_D := 0$. The limit L_D thus has a strong impact on the number of improving solutions which are explored. For an improvement method of the workload levelling problem, this approach seems very appropriate because the starting solution was created without considering levelling criteria. As a consequence, improving solutions can be found very quickly in the first steps while later on, improvements can only be found by investing more search effort.

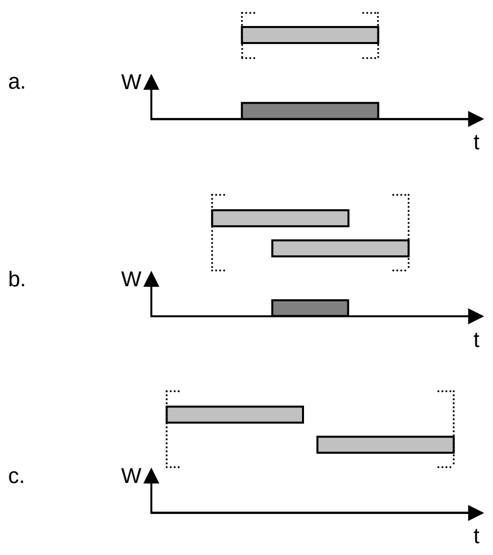


Figure 4.4.: Base histogram: critical, near-critical and non-critical tasks.

4.7. Lower Bounding

The use of good lower bounds is crucial for the performance of branch-and-bound type algorithms. If the lower bound of a partial solution Σ' is already greater or equal to the value of the incumbent solution, the current branch can be abandoned. In this way, large portions of the search tree can be pruned if lower bounds are tight. Additionally, lower bounds can be used to guide the tree search as described above.

A basic lower bound could be calculated by superposing the demands of all tasks which are inserted and whose start times are fixed. We can do better by a simple observation. Fig. 4.4 distinguishes three basic situations for a task's time window (given by dotted brackets). Tasks which are fixed in time (Fig. 4.4a) can be totally accounted in the demand curve (sketched below the task). If a task's start time is not yet fixed, we may still have the situation of Fig. 4.4b: If the latest start time of a task is less than its earliest completion time, we already know that it must take place within a certain *base interval* [Harris, 1990] [Neumann and Zimmermann, 1999]. Only if the time window is very large (case 4.4c), nothing can be done.

The resulting demand profile will be called *base histogram*. In analogy to project scheduling terminology, tasks with fixed time windows will be called critical and unfixed tasks with a non-empty base interval near-critical[3].

In the context of workload levelling, we must include travel times which are placed directly before the start times of the tasks. In a partial schedule Σ' as given by the constraint model in a search node, the final travel times are not known for all tasks because some of the tasks may not yet be inserted. For lower bound calculation, we can nevertheless include a minimum travel time d_i^{min} which must be performed before a task i:

$$d_i^{min} := \min_{j \in \delta_i^{pred}} d_{j,i}$$

For a current start time window $\delta_i^{start} = [\alpha_i, \beta_i]$ and the minimum travel time d_i^{min}, the base time interval of task i will be $[\beta_i - d_i^{min}, \alpha_i + l_i[$. The number $\hat{W}(t)$ of tasks which must be

[3]In project scheduling, activities are critical if there is no slack, cf. e.g. Zimmermann [2005].

performed at a time $t \in T$ can thus be defined as

$$\hat{W}(t) := |\{i \in I \,|\, \beta_i - d_i^{min} \leq t < \alpha_i + l_i\}|$$

Using this demand, a first lower bound on the final objective function value of a partial schedule Σ' is given by

$$\hat{z}(\Sigma') := \sum_{t \in T} \hat{W}(t)^2$$

By the insertion of tasks and start time decisions, the time windows will be narrowed in the course of solution construction, increasing the sizes of the base intervals. $\hat{z}(t)$ will finally converge toward the final objective value $z(t)$ for a full schedule.

For a partial schedule Σ', let $I' \subseteq I$ be the set of tasks which are not yet fixed in time, i.e. $I' := \{i \in I \,|\, \alpha_i < \beta_i\}$. Similarly to Zimmermann and Engelhardt [1998], these tasks can be used to strengthen the lower bound. Note that we have already scheduled a workload of $\max(\alpha_i + l_i - (\beta_i - d_i^{min}), 0)$ for all tasks $i \in I'$. If we do not want to tackle interdependencies of start time decisions for tasks $i \in I'$ explicitly, we can regard the remaining task minutes

$$w_i := l_i + d_i^{min} - \max(\alpha_i + l_i - (\beta_i - d_i^{min}), 0)$$

as a uniform workload which will be attributed unitwise to times of lowest demands. It is clear that the total workload $W' := \sum_{i \in I'} w_i$ must be carried out within the time interval

$$T' = [A, B[= \left[\min_{i \in I'}(\alpha_i - d_i^{min}), \max_{i \in I'}(\beta_i + l_i)\right[$$

The workload will be distributed in steps of increasing intervals with the left boundary A being fixed. The tasks $i \in I'$ are sorted by latest end times $\beta_i + l_i$. Let i_1 be the task with lowest latest end time. The workload w_{i_1} is now distributed unitwise over the interval $T_1 := [A, \beta_{i_1} + l_{i_1}]$, each time increasing $\hat{W}_t$ at the time t of lowest workload. The workload w_{i_2} of the next task in order is then distributed over the interval $T_2 := [A, \beta_{i_2} + l_{i_2}]$ and so on. In this way, the total workload is uniformly scheduled in increasing intervals. By construction, it is clear that the value $\tilde{z}(\Sigma') := \sum_{t \in T} \tilde{W}_t$ obtained from the final histogram $\tilde{W}_t$ is still a lower bound.

If periods are stored by their demand $\hat{W}_t$, the procedure can be implemented in $\mathcal{O}(|T'| + |W'|)$ time (and $\mathcal{O}(|T'|)$ space) complexity.

The quality of the lower bound depends on the time period T' covered by the tasks in I'. The quality of the bound will normally be better if a small number of tasks is unscheduled which is true for large neighbourhood search. Even if it is helpful in guiding the search tree traversal, the lower bound $\tilde{z}(\Sigma')$ calculated before any tour attributions and temporal fixations will generally not give a good indication of the optimal objective function value.

4.8. Large Neighbourhood Search

The local search algorithm consists of repeated steps of decision relaxation and reoptimisation. In Section 4.6, the reinsertion and fixing of tasks by branch-and-bound was described. We will now present the choice of tasks whose insertion and start time decisions are relaxed.

Tasks should be chosen as to allow for an objective function improvement when being reoptimised. It will thus not make sense to relax tasks which are far away from each other because insertion and fixation decisions for such tasks will not be interdependent. We should prefer tasks which are somehow related. As in Shaw [1998], a relatedness measure will be used for the choice of task sets. Note that the choice of task sets for relaxation is related to clustering methods [Taillard, 1993].

Crew tasks generally have a strong interaction: If e.g. crew tasks have same lengths, the total crew block will be fixed when the start time of one of the tasks is fixed. We will therefore always relax whole blocks of crew tasks. Furthermore, the symmetry constraints described in Section 4.5 impose considerable restrictions for the tour attribution. Non-crew tasks will therefore only be relaxed along with their symmetry group.

The procedure will consist in choosing a first task and then collecting a set of related tasks. Because most improvements will stem from start time changes (and not from changes in travel time), only movable tasks are considered in the first step. Different cost-based criteria like regret-based selection were tried for the selection of a first task. Astonishingly, none of these criteria could clearly outperform a random task selection. Selection criteria cannot generally grasp the improvement potential of a task's surroundings and will therefore be very local in nature. Since the LNS algorithm quickly finds local improvements, such criteria will only bias the search when tasks have been moved to their locally optimal solution. The first task is therefore chosen with a uniform distribution over the set of movable tasks. Along with the first task, its crew or symmetry group tasks are chosen.

Because only tasks which are near in time interact with the first task, the further choice is restricted to tasks which can overlap with a time window $[\alpha_i - TD, \alpha_i + l_i + TD]$ around the first task i where TD is a parameter. Note that all time windows are fixed when selecting tasks for relaxation, i.e. $\alpha_i = \beta_i \, \forall i \in I$. A relatedness measure $REL(i,j)$ between two tasks i, j is used which is a weighted sum of four terms:

$$REL(i,j) := \lambda_{TW} \cdot TW(i,j) + \lambda_{OL} \cdot OL(i,j) + \lambda_{DIST} \cdot DIST(i,j) + \lambda_{TT} \cdot TT(i,j)$$

A higher relatedness means that these tasks are assumed to have a better chance of improving the solution when being released. All terms are designed to be commutative in their arguments:

- $TW(i,j)$ is the size of the (original) time windows: $TW(i,j) := (b_i - a_i) + (b_j - a_j)$
- $OL(i,j)$ gives the current overlap between i and j: $OL(i,j) := \max(\min(\alpha_i + l_i, \alpha_j + l_j) - \max(\alpha_i, \alpha_j), 0)$
- $DIST(i,j)$ is a measure of the distance between the tasks ($DIST^{fix}$ is a parameter of the algorithm):

$$DIST(i,j) := \begin{cases} \frac{1}{|\alpha_j - \alpha_i| + |(\alpha_j + l_j) - (\alpha_i + l_i)|} & \text{if } |\alpha_j - \alpha_i| + |(\alpha_j + l_j) - (\alpha_i + l_i)| > 0 \\ DIST^{fix} & \text{else} \end{cases}$$

- $TT(i,j)$ gives a (symmetric) distance between i and j (with parameter TT^{fix}):

$$TT(i,j) := \begin{cases} \frac{1}{t_{ij} + t_{ji}} & \text{if } t_{ij} + t_{ji} > 0 \\ TT^{fix} & \text{else} \end{cases}$$

If I^{rel} are the tasks chosen so far, the valuations $\sum_{i \in I^{rel}} REL(i,j)$ are calculated for all tasks $j \in I \setminus I^{rel}$ which are near to the first tasks. The tasks are then sorted by decreasing valuations. This means that tasks which are near in time or space to the tasks chosen so far, which overlap with them or have large time windows will come first. Let RS be the number of elements in this sorting.

If we relied too much on the relatedness sorting, we would possibly miss sets of tasks which can improve the solution. As proposed in Shaw [1998], some random influence is introduced in order to diversify the search. A random number $\mu \in [0, 1[$ is chosen, and with a choice randomness parameter $CR \in \mathbb{N}$, an index

$$\kappa := \lfloor RS \cdot \mu^{CR} \rfloor$$

is calculated. We now choose the κ'th element in the relatedness sorting (along with its crew or symmetry group tasks) to be relaxed. If CR is high, μ^{CR} will be close to 0, and we will choose one of the first elements in the relatedness sorting. If CR is rather low, the influence of randomness will be greater.

This task selection procedure I^{rel} is repeated until no more tasks can be chosen without exceeding a given limit on the number of tasks. The tasks in I^{rel} are then relaxed, i.e. their start time, tour and scheduled domains are reset; the current predecessor/successor values are set to *NIL*. We then evaluate possible insertion positions $\Phi(i)$ in the tours. These insertion positions are recorded in the insertion positions and predecessor domains of the relaxed tasks as well as in the predecessor domains of the insertion positions. As described above, the insertion positions, predecessor update and tour-task consistency tests are added dynamically to the formulation, and tasks are reinserted and fixed by the LDS algorithm described above.

Finally, we must specify how the neighbourhood size, i.e. the number of relaxed tasks, is controlled. Clearly, the most costly step in large neighbourhood search is the reinsertion procedure whose complexity scales with the number of relaxed tasks. For efficiency reasons, we should try to gain improvements by the least number of tasks. The procedure therefore starts with only one relaxed task at a time. After a certain number of steps, no further improvements will be possible. To avoid stalling, the step size is increased by one after a number *SWI* of steps which do not yield any improvement. Clearly, there is a tradeoff: Instead of accepting many non-improving steps at a low stepwidth, it may pay off to force higher numbers of tasks to be relaxed in order to gain substantial improvements.

A general advantage of large neighbourhood search is that the solution quality smoothly scales with the invested runtime: While with little runtime, reasonable results are obtained, the results gradually get better if more time is available. If arbitrarily large neighbourhood sizes and discrepancies can be used, LNS amounts to a complete search for the given optimisation problem.

4.9. Preprocessing

As described before, the levelling procedure is preceded by a tour minimisation algorithm. The improvement phase will build upon the task attribution as given by the preceding tour minimisation phase. We still have some degrees of freedom for fixing tasks within their remaining time windows. Before the implementation of the levelling procedure, tasks were fixed to their earliest possible start times, neglecting the possibility of shifting tasks within their tours in order to level demands. This earliest start solution is a possible starting point for the levelling algorithm. However, a better starting solution can be generated by letting a preprocessing phase decide at which times the tasks are scheduled.

Assume first that the scenario does not contain crew tasks. Consider a tour whose tasks are not yet fixed in time, containing a set of tasks I_r in fixed order. If the tour is traversed timewise, the possible start times of a task i_2 only depend on the fixation of the preceding task i_1. This means that a dynamic programming (DP) algorithm with state space $I_r \times T$ can be applied. The cheapest objective function value for each state (i_2, t_2) is calculated by minimisation over the start times t_1 of the predecessor $i_1 = \pi_{i_2}$ such that $t_1 + l_{i_1} + d_{i_1,i_2} \leq t_2$. At each node, the increase in objective function by fixing i_2 to t_2 is added.

This dynamic programming algorithm can make use of the constraint model, see also Focacci and Milano [2001]. When a DP state (i_1, t_1) is expanded to possible start times for successor task i_2, this can be done by setting $\delta_{i_1}^{start} := \{t_1\}$ and propagating. The resulting objective function is recorded, and each of the start times in $\delta_{i_2}^{start}$ is tried and propagated. If the best value at state (i_1, t_1) plus the objective function increase is less than the current value at the state (i_2, t_2), its best

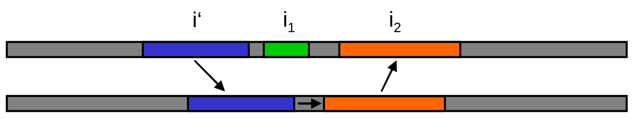


Figure 4.5.: Temporal influence over crew tasks.

value is updated. This procedure can be optimised for start times t_1 of i_1 with $t_1+l_{i_1}+d_{i_1,i_2} < a_{i_2}$, i.e. when the fixation of i_1 to t_1 does not have any influence on possible start times for i_2. We can then first minimise over the states (i_1, t_1) with the aforementioned property and then expand to all start times $t_2 \in [a_{i_2}, b_{i_2}]$ of i_2.

This procedure provides local optima for a single tour in a specified context as given by the tasks in other tours. The context should be as expressive as possible. Imagine that no other tasks are recorded in the demand curve. Then the tourwise dynamic programming as described above will yield identical costs for every possible fixation. On the other hand, if the tasks of all other tours are fixed, the optimisation may have a very “local character”, meaning that we cannot expect much improvement.

Two dynamic programming schemes were evaluated:

- We use the given task attribution, but let the time windows open. Base time intervals of all tasks are recorded as described in Section 4.7, i.e. we start from a base workload histogram. The tours are then fixed one by one by the above dynamic programming procedure. This successive fixing procedure will be referred to as SFDP.

- We take the earliest start time fixation. For each tour in turn, the start times for all tasks in the tour are relaxed and reoptimised by dynamic programming. This is repeatedly done on all tours until no more improvement is possible, i.e. we have reached a local optimum. This is the case if no improvement was possible for $|R|$ tours. We will refer to this procedure as repeated local dynamic programming (RLDP).

Note that RLDP will traverse more tours than SFDP, but the extra runtime may pay off because the fixation of a tour is evaluated in different contexts. For SFDP, the context in which a tour is optimised will be less expressive, especially for the first tours which are fixed.

If the scenario contains crew tasks, some extra effort must be spent. In the RLDP scheme, start time decisions should be relaxed for complete crews because otherwise, a crew task would have little or no degrees of freedom. Potential problems for dynamic programming arise from the more complex temporal relations between the tasks: While within the tours, possible start times for a task i_2 only depend on its predecessor i_1, valid start times for i_2 may additionally be influenced by tasks before i_1 via crew task relations. An example can be found in Fig. 4.5 where tasks of the same colours represent blocks of crew tasks and temporal influences are sketched by arrows. In such cases, the optimal start times determined in dynamic programming may not be feasible. In practical scenarios, this phenomenon will be rare, and we will simply discard the DP results in such cases.

For comparison with the dynamic programming approaches, a third greedy fixing (GF) method is evaluated. All time windows are left open, and fixing is done on the basis of the base histogram. All tasks are traversed in order of increasing time windows, i.e. decisions for tasks with most degrees of freedom are left until the end. Each task in turn is fixed to its (locally) optimal start time. This approach was successfully applied to resource levelling [Harris, 1990] and corresponds to the fail-first principle [Haralick and Elliott, 1980].

The best solution of these preprocessing algorithms will give a hint of which objective function values can be attained within the given attribution of tasks to tours. The degree by which the local

No.	days	tasks	movable tasks	average time window	task minutes	crews	crew tasks	range of travel times	tours	groups
A01	8	229	199	217.4	14274	0	0	[0,7]	4	229
A02	8	1819	1073	168.2	80985	637	1351	[0,7]	20	1105
A03	7	3420	3197	71.1	46080	0	0	[0,0]	57	2720
B01	7	410	99	96.8	43255	33	99	[0,11]	34	82
B02	7	748	138	32.9	10673	19	38	[0,8]	11	692
B03	7	929	26	35.1	61870	0	0	[0,10]	20	929
B04	7	1615	628	255.7	34025	134	1615	[0,10]	42	134
B05	7	1121	338	269.3	27795	0	0	[0,24]	11	777
B06	8	627	609	109.2	18450	94	188	[0,18]	7	533
B07	8	734	696	104.4	22555	145	290	[0,29]	7	581
B08	8	4252	4241	116.6	45547	0	0	[0,54]	57	4203
B09	7	11325	8961	56	321464	0	0	[0,12]	789	7423
B10	7	4867	1032	77.5	75313	1572	4867	[0,0]	43	1572
B11	7	623	422	190.3	59629	72	623	[0,0]	28	72
B12	7	3234	3172	87.8	50758	580	3234	[0,1]	21	580
B13	7	1836	1588	52.8	65452	739	1836	[0,1]	22	739
B14	7	3731	742	9.5	20025	0	0	[0,3]	12	3583

Table 4.1.: Scenario data.

search algorithm is able to improve the solution will furthermore give an indication of how much the quality of tour attributions contributes to overall solution quality.

4.10. Experimental Results

All of the aforementioned algorithms were implemented in Visual C++ 7.1. The toolkit *ctk* was used for the implementation of constraint propagation and the branch-and-bound algorithm. Tests were carried out on a personal computer with AMD Athlon 2000+ processor (1.67 MHz), 512 MB main memory and operating system Windows XP SP1.

Since workload levelling is a novel optimisation problem which is especially relevant to airport operations, realistic airport planning scenarios were chosen for the evaluation. Scenarios cover seven or eight days with a discretisation of one minute and often have a large scale as can be seen from Table 4.1. Test cases of class A represent pure equipment planning scenarios while class B scenarios apply to staff planning.

The number of tasks vary between 229 (scenario A01) and 11325 (scenario B09) of which an average of 59.9% are movable (column *tasks* includes movable tasks in Table 4.1). While in some scenarios, nearly all tasks are movable, only 2.8% of the tasks are movable in scenario B03, meaning that the potential for levelling will be rather low. The average time windows as given in Table 4.1 refer to movable tasks only. As can be seen from the problem data, the tasks are partly highly movable with a time window of up to 269 minutes. Tasks have average lengths of between 5.4 and 105.5 minutes. 10 out of 17 scenarios contain crew tasks with average crew sizes between 2 and 12. Some of the scenarios have vanishing travel times while others use travel durations of up to 54 minutes. It can be seen that the test cases cover a wide range of different characteristics with regard to the number of tasks, their movabilities, durations and the use of crews.

The figure *tours* gives the result of the initial tour minimisation algorithm. Note that while scenario A01 comprises only 4 tours, 789 tours are needed to cover the tasks of scenario B09. The column *groups* refers to the symmetry breaking described in Section 4.5. It can be seen that especially the test cases involving crew tasks exhibit strong symmetry. In other scenarios, there is no or nearly no advantage in using symmetry breaking constraints. As an example, each task

No.	EST	SFDP			RLDP			GF		
	obj. fct.	obj. fct.	improvement	runtime (sec.)	obj. fct.	improvement	runtime	obj. fct.	improvement	runtime (sec.)
A01	34808	30004	13.80%	1801.62	31950	8.21%	2271.68	29940	13.99%	1.14
A02	866586	810704	6.45%	215.78	866534	0.01%	0.91	812956	6.19%	1.14
A03	309680	306010	1.19%	9.81	308410	0.41%	8.10	307030	0.86%	0.31
B01	564373	549397	2.65%	42.63	549397	2.65%	131.28	552037	2.19%	0.17
B02	34946	34412	1.53%	0.16	34642	0.87%	0.29	34414	1.52%	0.03
B03	631072	630766	0.05%	0.10	630916	0.02%	3.54	630766	0.05%	0.02
B04	552403	530297	4.00%	29.61	543331	1.64%	80.23	528493	4.33%	0.55
B05	124539	120645	3.13%	2.90	121803	2.20%	15.10	120911	2.91%	0.18
B06	57671	49871	13.52%	96.82	54239	5.95%	56.60	49663	13.89%	0.48
B07	86393	76359	11.61%	63.26	82531	4.47%	16.62	76497	11.45%	0.59
B08	318996	315790	1.01%	70.37	318684	0.10%	1316.61	316734	0.71%	5.18
B09	28045088	27901034	0.51%	345.07	28045178	0.00%	264.81	27956676	0.32%	28.56
B10	1158001	1114627	3.75%	0.98	1152755	0.45%	1.32	1115295	3.69%	0.44
B11	882205	787145	10.78%	9.99	796789	9.68%	5.41	787913	10.69%	1.26
B12	505034	467734	7.39%	137.71	505034	0.00%	1.00	467566	7.42%	1.99
B13	707076	659146	6.78%	143.68	707076	0.00%	0.74	659506	6.73%	1.49
B14	93818	91326	2.66%	0.30	92160	1.77%	1.37	91396	2.58%	0.26

Table 4.2.: Preprocessing runtimes and results.

builds its own group in scenario A01.

Table 4.2 shows results and runtimes (in seconds) of the three preprocessing algorithms SFDP, RLDP and GF in comparison to earliest start fixing (EST). It can be seen that SFDP dominates RLDP in solution quality. The disappointing results of RLDP can be explained by its local character since it reoptimises only one tour at a time with all other tours being fixed. This prevents RLDP from overcoming local optima. In contrast to expectations, SFDP is not faster than RLDP on all scenarios. Nevertheless, SFDP only consumes more time on scenarios on which results are clearly superior to RLDP, i.e. RLDP terminates early because no further improvements can be found.

Astonishingly, the results of GF preprocessing are competitive with the SFDP results. While on most scenarios, the quality of GF results is slightly below the SFDP outcome, it is even better on 4 of the 17 scenarios. Even if the fixing idea of GF is simple, the ordering of tasks by increasing time windows seems to pay off, confirming the experience of Harris [1990] in the resource levelling context. GF needs only little computation time while SFDP consumes up to more than 30 minutes for preprocessing. The high runtimes of SFDP are due to the testwise fixations and propagations of start time decisions in dynamic programming. On average, SFDP consumes more than 67 times the runtime of GF. As expected, the improvements by preprocessing are generally higher on scenarios involving many tasks with wide time windows.

The results of GF preprocessing were taken as a starting point for large neighbourhood search. In preliminary experiments, the parameters for the choice of release tasks and for reinsertion were tuned. The maximum distance TD for tasks which can be relaxed around the first chosen task was set to 180 minutes. The relatedness weight parameters λ_{TW}, λ_{OL}, λ_{DIST} and λ_{TT} were all set to 1, i.e. the criteria are equally weighted. The values $DIST^{fix}$ and TT^{fix} were fixed to 10. A value of 50 was used for the choice randomness parameter CR, meaning that the influence of randomness is rather low.

As described in Section 4.6, the search depth can be controlled by the discrepancy limit L_D. Astonishingly, a limit of $L_D = 0$ provided best results on average. As a consequence, only one improving solution is explored in every iteration. Clearly, it becomes more and more difficult to find improving solutions when the quality of the incumbent solution gradually increases.

A planner may sometimes not be able to invest too much runtime. To show how the solution quality scales with the invested runtime, Table 4.3 gives results for runs with a limit of 10, 20 and 60 minutes runtime. The runtimes include the GF preprocessing, improvement figures are given relative to the GF results. Different values were used for the parameter *SWI*, denoting the number

No.	GF result	runtime 10 min.			runtime 20 min.			runtime 60 min.		
	obj. fct.	avg. obj. fct.	improvement	rel. $\tilde{\sigma}$	avg. obj. fct.	improvement	rel. $\tilde{\sigma}$	avg. obj. fct.	improvement	rel. $\tilde{\sigma}$
A01	29940	27643.6	7.67%	0.14%	27523.0	8.07%	0.41%	27410.0	8.45%	0.18%
A02	812956	774513.8	4.73%	0.57%	767191.6	5.63%	0.17%	764722.8	5.93%	0.13%
A03	307030	299750.4	2.37%	0.30%	296525.6	3.42%	0.19%	295039.2	3.91%	0.11%
B01	552037	541517.8	1.91%	0.12%	541518.2	1.91%	0.12%	541521.0	1.90%	0.12%
B02	34414	33002.6	4.10%	0.22%	32714.4	4.94%	0.30%	32432.0	5.76%	0.62%
B03	630766	626745.6	0.64%	0.04%	626381.4	0.70%	0.08%	624930.8	0.93%	0.08%
B04	528493	527568.0	0.18%	0.12%	527339.8	0.22%	0.09%	526241.8	0.43%	0.15%
B05	120911	114923.0	4.95%	0.35%	114242.0	5.52%	0.21%	113896.0	5.80%	0.14%
B06	49663	45308.6	8.77%	0.98%	44656.4	10.08%	0.48%	44620.4	10.15%	0.33%
B07	76497	67294.6	12.03%	0.36%	66403.8	13.19%	0.27%	65793.0	13.99%	0.22%
B08	316734	267878.6	15.42%	2.09%	248517.6	21.54%	1.67%	242099.0	23.56%	0.55%
B09	27956676	27927436.0	0.10%	0.02%	27897576.4	0.21%	0.02%	27790665.2	0.59%	0.03%
B10	1115295	1075460.6	3.57%	0.22%	1072787.0	3.81%	0.05%	1071994.6	3.88%	0.04%
B11	787913	770067.8	2.26%	0.34%	768371.0	2.48%	0.09%	766377.0	2.73%	0.28%
B12	467566	443968.8	5.05%	0.33%	438915.8	6.13%	0.15%	436433.8	6.66%	0.36%
B13	659506	637238.6	3.38%	0.10%	633412.8	3.96%	0.08%	632818.0	4.05%	0.09%
B14	91396	87865.8	3.86%	0.14%	87111.2	4.69%	0.16%	86672.6	5.17%	0.26%

Table 4.3.: Results of large neighbourhood search based on GF results.

of steps without improvement after which the number of relaxed tasks is increased: For the 10, 20, and 60 minute runs, *SWI* was set to 20, 50, and 100, respectively. This means that for the longer runtimes, smaller neighbourhoods are explored more thoroughly. Each result is an average over 5 runs with different random seeds. To show the closeness of results with different initialisations of the random number generator, an estimation

$$\tilde{\sigma} = \sqrt{\frac{\sum_{i=1}^{n}(z_i - \bar{z})^2}{n-1}}$$

for the standard deviation was calculated with z_i being the results of the different runs, $\bar{z}$ the average results as given in Table 4.3 and $n := 5$ the number of runs. The relative standard deviation in Table 4.3 is equal to $\frac{\tilde{\sigma}}{\bar{z}}$. Additionally to the LNS results in Table 4.3, Table 4.4 shows how many steps are performed on average and how many tasks are relaxed in the final steps.

From Table 4.3, it can be seen that the LNS procedure improves the preprocessing results considerably. If we assume that the freedom of fixing tasks within their tours is well exploited by preprocessing, this means that the reattribution of tasks to different tours is a considerable source of improvement. This however varies between the scenarios. As an example, scenario B04 is improved by 4.33% in preprocessing, but LNS only finds an additional 0.43% amelioration. In contrast, LNS improves scenario B08 by 23.56% while the improvement of preprocessing can nearly be neglected. Other scenarios (like scenario B07) are substantially levelled by preprocessing *and* LNS. The results of LNS were again subjected to the tour-wise dynamic programming reoptimisation used in RLDP preprocessing. On none of the scenarios further improvement could be realised, demonstrating the quality of start time decisions in LNS.

From the results for different runtimes, it can be seen that much of the improvement can generally be achieved in relatively low runtimes. After a certain number of steps, large neighbourhood search hardly yields any improvement. Fig. 4.6 shows the typical evolution of solution quality on a LNS run on scenario B08. The time after which saturation occurs differs from scenario to scenario. As an example, the best solution for scenario B01 comprising little movable tasks is already found after 10 minutes runtime.

Relative standard deviations for the different runs are generally quite low. As expected, deviations diminish with higher runtimes: While the 10 minutes results have an average deviation of 0.38%, the 20 minutes and 60 minutes results exhibit relative standard deviations of 0.27% and 0.22%, respectively.

No.	runtime 10 min.		runtime 20 min.		runtime 60 min.	
	avg. steps	avg. tasks	avg. steps	avg. tasks	avg. steps	avg. tasks
A01	562.0	5.2	1100.2	5.0	2109.0	5.0
A02	1081.8	4.6	3010.6	5.0	5356.6	5.2
A03	1184.4	4.6	3566.6	4.6	7285.0	5.0
B01	320.4	12.0	651.8	12.4	1157.0	12.0
B02	460.6	6.8	1112.2	6.0	2446.8	6.0
B03	761.4	12.6	1764.8	10.8	4411.8	9.8
B04	420.0	26.0	970.0	25.2	2112.2	26.4
B05	725.4	4.4	1681.2	4.4	3376.4	5.4
B06	852.2	5.4	2315.0	5.2	4214.6	5.4
B07	1166.0	5.4	2950.4	5.6	5497.4	5.8
B08	5269.6	7.0	12106.6	8.4	18744.4	7.6
B09	121.0	1.0	242.6	1.0	716.6	1.0
B10	1123.2	11.2	2404.0	10.0	5041.8	13.4
B11	406.8	19.4	973.6	19.2	2304.0	23.4
B12	1375.6	15.4	3644.6	13.4	7308.0	15.2
B13	1156.4	7.8	3217.0	7.0	5254.0	8.0
B14	1445.0	9.8	3100.0	10.2	5870.6	10.4

Table 4.4.: Final numbers of steps and tasks in large neighbourhood search.

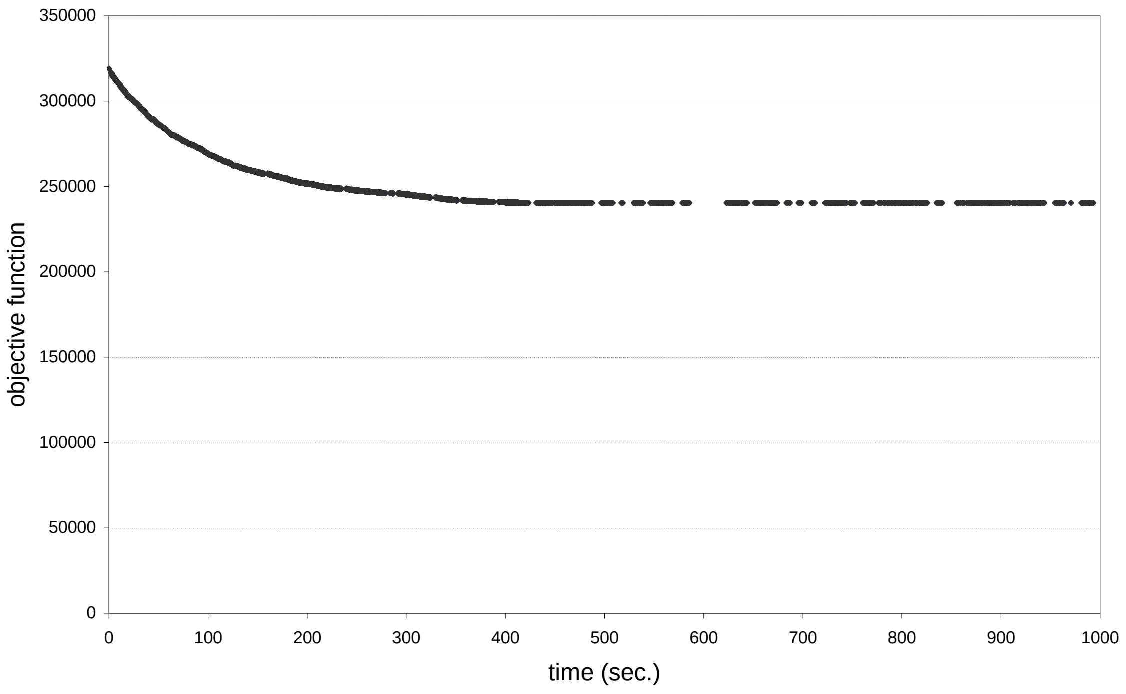


Figure 4.6.: Objective function development on scenario B08.

	10 min.		20 min.		60 min.	
	GF	EST	GF	EST	GF	EST
A01	27643.6	28086.2	27523.0	27832.6	27410.0	27667.8
A02	774513.8	773665.0	767191.6	766787.2	764722.8	765033.4
A03	299750.4	300008.0	296525.6	296498.0	295039.2	294751.2
B01	541517.8	541829.4	541518.2	541578.2	541521.0	541775.0
B02	33002.6	33016.0	32714.4	32763.6	32432.0	32227.8
B03	626745.6	626989.8	626381.4	626329.8	624930.8	625145.8
B04	527568.0	530519.0	527339.8	528272.0	526241.8	525609.0
B05	114923.0	114742.6	114242.0	114085.4	113896.0	113887.8
B06	45308.6	45242.6	44656.4	44612.0	44620.4	44593.4
B07	67294.6	67778.8	66403.8	66770.0	65793.0	66332.0
B08	267878.6	269470.8	248517.6	248398.2	242099.0	240663.0
B09	27927436.0	28007548.8	27897576.4	27976051.2	27790665.2	27857366.8
B10	1075460.6	1075764.6	1072787.0	1072961.0	1071994.6	1072239.8
B11	770067.8	767937.0	768371.0	768273.0	766377.0	768273.0
B12	443968.8	447729.2	438915.8	442574.2	436433.8	439751.2
B13	637238.6	639134.4	633412.8	635564.6	632818.0	634708.2
B14	87865.8	87770.6	87111.2	87136.4	86672.6	86685.2

Table 4.5.: Overall effect of GF preprocessing.

The final numbers of steps and tasks as given in Table 4.4 reflect the different characteristics of the test cases. As an example, large neighbourhood search on the large-scale scenario B09 consists of repeated relaxations and reoptimisations of only one task. It can be clearly expected that the results on this scenario will improve if further runtime is invested. On the other end of the scale, up to 20367 steps are performed on scenario B08, and up to 28 tasks are released in a run on scenario B04.

To measure the overall effect of GF preprocessing, large neighbourhood search was run an additional five times on the initial solution in which all tasks are fixed to their earliest possible start times. Table 4.5 contrasts the average 10, 20, and 60 minutes results with earliest start time fixing (EST) to the GF/LNS results.

It can be seen that the GF improvements are quickly reconstructed by pure large neighbourhood search. After only 10 minutes, the GF/LNS results are better on only 12 of the 17 scenarios while on average, EST/LNS results are still 0.26% above the GF/LNS outcomes. After 20 and 60 minutes, the average gap is 0.19% and 0.12%, respectively, and preprocessing does not pay off any more.

Additionally to the objective function summing up quadratic workloads, it is interesting to observe the effects of levelling on the overall workload maximum and travel times even if these are no explicit optimisation criteria. A summary of demand peaks and travel times before (with EST fixing) and after optimisation (including GF preprocessing) is given in Table 4.6. As before, optimisation results are averaged over five runs with 60 minutes of runtime.

Obviously, workload levelling is also beneficial for global demand maxima. On 8 out of 17 scenarios, the demand maximum can be lowered by the levelling procedure. It is interesting to note that the large-scale scenario B09 figures among these 8 scenarios; in one run, the global maximum is even reduced from 129 to 125. A visual inspection of this scenario shows that the levelling results are better than the maximum 0.95% objective function improvement (over EST fixing) might suggest.

No.	demand maximum		travel times	
	before optimisation	after optimisation	before optimisation	after optimisation
A01	4	3.0	530	377.2
A02	20	16.6	2247	2322.2
A03	12	10.0	0	0.0
B01	31	28.0	374	374.0
B02	7	6.0	1705	1419.4
B03	20	20.0	3094	2691.2
B04	42	42.0	1708	1735.6
B05	11	11.0	2380	2014.6
B06	7	6.8	861	819.6
B07	7	7.0	1534	1387.2
B08	10	8.8	8905	2842.4
B09	129	126.6	135900	135900.0
B10	43	43.0	0	0.0
B11	28	28.0	0	0.0
B12	18	18.0	1884	1821.8
B13	22	22.0	876	911.6
B14	11	11.0	3255	2793.2

Table 4.6.: Effects on global demand maxima and travel times.

The algorithm also tends to reduce travel times. A reduction can be observed on 9 of the 14 scenarios involving travel times while on three scenarios, travel times slightly increase. Travel time reductions raise up to a 69.4% improvement on a scenario B08 run. Clearly, it must be noted that travel times are not an optimisation criterion of the initial tour building algorithm.

4.11. Conclusions and Future Research

We have presented a novel workload levelling problem in a vehicle routing environment. Its objective is the smoothing of workloads arising as work tasks for mobile staff and equipment on airports. Superpositioning work tasks and travel times as a demand curve, workload levelling exploits the freedom of positioning movable tasks at times of low workloads. Unnecessary demand peaks are therefore avoided. The resulting demand curve provides a suitable basis for realistic estimations of workforce demands and for the planning of staff and equipment.

We have shown that workforce levelling is closely related to resource levelling in project scheduling. Like resource levelling builds upon the result of a makespan optimisation algorithm, workload levelling starts from the outcome of a tour minimisation procedure. Consequently, workload levelling has been conceived as a solution improvement algorithm. Furthermore, we have proved that workload levelling is $\mathcal{NP}$-hard, justifying the search for heuristic algorithms.

We have shown that constraint propagation is a good choice for capturing the temporal interdependencies between the tasks as well as the non-linear objective function. Additionally, CP allows for an easy incorporation of constraints which may come up in the future. We have presented a large neighbourhood search algorithm which repeatedly relaxes and reoptimises insertion and start time decisions. A new variant of limited discrepancy search has been used for reoptimisation, exploiting a powerful lower bound. The solution scheme gradually explores larger neighbourhoods and provides easy means of weighing runtime against solution quality.

As an interface between tour minimisation and local search, three different greedy and local dynamic programming procedures for the initial fixing of start times have been presented.

The algorithms has been tested on a wide range of real-world scenarios of different sizes and characteristics. Surprisingly, the greedy preprocessing algorithm has shown to be the best basis for solution improvement, providing competitive results within very low runtimes. Large neighbourhood search has been able to improve considerably upon the preprocessing results, showing that a considerable potential for improvement lies in the reattribution of tasks. While with short runtimes, preprocessing still pays off, comparable results can be achieved by pure large neighbourhood search if more runtime is invested. The overall algorithm has been shown to be robust in providing level demand curves on quite different test cases. Furthermore, overall demand maxima and travel times have been reduced on many planning scenarios.

The workload levelling procedure is part of a commercial software application and has proven to be a very valuable tool for demand planning and for the preparation of workloads for staff planning.

In general, one can expect that levelled workloads provide a better basis for demand-level planning of shift duties. If, however, shift planning is the only focus of workload levelling, the model may be refined. As an example, the length of tours could be restricted such that tours fit into shifts [Ernst et al., 2004]. A disadvantage of the separation of workload levelling and (demand-based) shift planning is that levelling does not always schedule tasks at positions which are most appropriate with regard to shift planning. A theoretical deficiency is that the demand-level shift planning approach does not provide a lower bound for task-level shift planning.

Further experiments with large neighbourhood search could e.g. evaluate the efficiency of alternative heuristic pruning techniques. Different alternatives to limited discrepancy search have been proposed in recent years, including improved limited discrepancy search [Korf, 1996], depth-bounded discrepancy search [Walsh, 1997] and interleaved depth-first search [Meseguer, 1997].

5. An Improvement Algorithm for Complex Shift Planning

The art of flying
is to throw yourself at the ground
and miss.
— Douglas Adams

Shift planning at airports can be carried out in different ways. Models in the literature regularly build upon workforce demands given in discrete time periods, see Chapter 2. If information on flights and passenger/load figures is little detailed, this is also an appropriate approach for ground staff planning. If, however, we are interested in an operative planning which is carried out shortly before operations, shift planning should aim at covering the individual work tasks (see also Section 1.2.3). Task-level shift planning is a novel optimisation problem which combines aspects of vehicle routing and shift scheduling. Additionally, our scenario requires the handling of preemptive tasks and crew constraints. In the past, a heuristic construction method has been used to tackle the large scale and considerable complexity of airport shift planning problems. While this method generally provides good solutions, it sometimes lacks robustness and exhibits deficiencies on individual planning instances. In the following, we will develop a mathematical description of task-level shift planning and describe the design and implementation of a constraint-based method for local reoptimisation of shift plans.

5.1. Introduction

Ground handling companies and airline ground handling departments face the problem of covering large sets of work tasks with appropriate staff. Handling tasks include baggage loading and unloading, aircraft cleaning, fuelling as well as check-in and boarding services. In the preceding chapter on workload levelling, it was shown how tasks can be prepared for a graphical demand analysis as well as demand-based scheduling. We will now deal with covering work tasks by appropriate shift duties which are assigned to workers in a downstream step.

For a given department, we will assume a set of tasks to be given, spanning over a planning horizon which will frequently be one week. Each task is basically characterised by a window of possible start times, a length and a location at which it takes place. The goal of shift planning is to cover these tasks by shifts of any of a set of given types. Shift types define start and end times within the day and give information on the duration and placement of one or several breaks (e.g. relief or lunch breaks). Tasks must be placed one after the other such that time window and break placement restrictions are met and sufficient time for travel between locations is available.

This basic scenario is often enriched by further constraints. Workers like aircraft cleaning staff are often planned in crews or teams, imposing constraints on the parallelism of tasks and shifts. Some of the tasks can be preemptive or splittable, meaning that the task is handed over to another person at some time. Some tasks may have to be covered by staff with special skills like language skills (for check-in and boarding services) or being trained for a special aircraft type (e.g. for baggage loading/unloading). Further restrictions affect the mix of shift types in the final plan, reflecting the numbers of workers with different contracts and worktime agreements.

Task-level shift planning consists in building tours of tasks, obeying time window constraints. This can be interpreted as a special vehicle routing problem with time windows (VRPTW) with the shift types imposing restrictions on the tours. Crews and splittable (preemptive) tasks introduce novel modelling aspects which are particularly challenging for the design of optimisation algorithms. Task-level shift planning is thus an integration of classical shift scheduling with vehicle routing and further complicating constraints.

Only a limited number of publications is loosely related to this problem. Thompson [1992] describes the scheduling of blocks of controllable work which can be interpreted as splittable tasks. Bailey et al. [1995] describe a shift scheduling setting in which workloads are given as activities in a project scheduling environment while in Nobert and Roy [1998], workforce demands relate to the weight of aircargo. Campbell and Savelsbergh [2004] limit the extent of tours by a given length without explicitly taking shift types or breaks into account. Crew and driver scheduling relates to task-level scheduling in covering trips by shifts duties (see Section 2.11). However, driver and crew duties are driven by the placement and lengths of trips, and there is no notion of shift types. In the following, we will show how all aspects of complex task-level shift planning can be represented in an optimisation model and how efficient algorithms can be designed, yielding robust solutions on a large variety of staff scheduling scenarios.

Realistic airport shift planning is often carried out on a very large scale. Frequently several thousand tasks with partly wide time windows must be covered, and several hundred shift types may be used. Furthermore, several dozens of qualifications and related restrictions are used. Due to 24×7 operations, it may not be possible to decompose larger scheduling horizons without sacrificing optimisation quality. In Chapter 7, it will be shown that even a very restricted version of the shift planning problem is $\mathcal{NP}$-hard. While scenarios using little constraints can still be solved to optimality (see Chapter 7), we cannot expect to obtain optimal solutions on larger and more complex scenarios. Nevertheless, the use of advanced optimisation algorithms pays off because with the large scale of airport operations, small improvements translate to large savings.

Heuristic methods allow for the construction of solutions of good quality in low runtimes. In the past, a construction method has successfully been used for several years. The algorithm makes use of the approximation of workloads by a workload histogram and initially solves a set covering formulation of the demand-level shift scheduling problem by linear programming and heuristic rounding. The levelled workload histogram of Chapter 4 provides a starting point for this initial phase which considers only a small subset of the constraints. In a second phase, the tasks are inserted into the shifts of the first phase one by one. Because the workload histogram only provides an approximation to actual workloads, further shifts may have to be created in a final phase to cover all tasks.

Each task is handled only once in the course of solution construction, entailing low runtimes even for large-scale scenarios. Since the algorithm was used and tuned for several years, the solutions generally exhibit a fairly good quality and were generally well-accepted by customers. Nevertheless, the algorithm shares typical traits with construction heuristics: While on average, good solutions are provided in little runtime, solutions can have considerable deficiencies on selected scenarios. The more obvious such flaws are, the less planners will accept the algorithm. Tuning of parameters sometimes helps, but changing parameters generally improves results on some scenarios while quality decreases on others.

The goal of this work was thus the conception and implementation of an algorithm which improves the initial solution by local reoptimisation, compensating for the flaws of the construction phase. By designing an improvement procedure, we can benefit from the generally good quality of the construction algorithm and from other advantageous features of initial solutions like the grouping of tasks of similar qualifications. The algorithm should be generic and robust on a wide range of real-world planning scenarios.

As described in Chapter 3, solution techniques based on constraint programming are particularly appropriate for problems comprising many constraints. We will therefore develop a constraint model representing the multitude of constraints which make up the shift planning problem. On this basis, a relax-and-optimise approach similar to the solution approach of the workload levelling problem will be devised. Constraint programming has the additional advantage of an easy incorporation of additional constraints which are likely to come up in the future, e.g. by model refinements or customer-specific requirements. The algorithm will allow for the determination of good solutions in reasonable runtimes. With further invested runtime, solution quality increases further, resulting in shift plans that are partly considerably better than the starting solutions.

The chapter is organised as follows: In the subsequent section, the shift planning problem is described formally, providing a basis for the mathematical model in Section 5.3. Sections 5.4 and 5.5 present basic considerations for the design of a CP-based optimisation algorithm. The constraint model is described in Section 5.6. The calculation of lower bounds presented in Section 5.7 provides a basis for the restricted branch-and-bound algorithm of Section 5.8 while Section 5.9 deals with design decisions for the implementation of large neighbourhood search. The minimisation of task overlaps which are partly allowed in the optimisation model is the subject of Section 5.10. Section 5.11 presents experimental results. A summary concludes the chapter.

5.2. Problem Description

In the following, we will characterise basic entities in task-level shift planning and introduce some notation. We will furthermore describe how task splitting, overlapping and crews are handled. An overview of mathematical symbols can be found in Appendix A.

5.2.1. Tasks and Shifts

Workloads are given as a set I of *(work) tasks*. A task $i \in I$ is characterised by an interval $[a_i, b_i]$ of possible *start times* (start time window) and a length l_i. All times are assumed to be given as integer numbers on an appropriate discretisation level (typically minutes).

The output of the shift planning process is a set S of shifts. Different constraints apply on the shift level. *Shift types* K specify when shifts can start and end. A shift type $k \in K$ can be realised on days $N_k \subseteq N$ where N is the set of all days of the planning horizon (often one week). The realisation of shift type $k \in K$ on a day $n \in N_k$ starts at a time st_{kn} and ends at et_{kn}. Note that a shift is always attributed to the day on which it starts. It will be assumed that shift types cover at most 24 hours. For a shift of type $k \in K$, costs of c_k are incurred. Shift costs will often be proportional to their length. Additionally, night shifts may entail higher costs.

Furthermore, a shift type defines up to three *break rules* for the lunch or relief breaks a shift must contain (see Fig. 5.1). In practice, shifts will often contain a lunch break of 30 or 60 minutes length and up to two relief breaks of 15 minute length [Schindler and Semmel, 1993] [Aykin, 2000]. The (possibly empty) early, main and late break rules of a shift type k are denoted by br_k^{eb}, br_k^{mb} and br_k^{lb}, respectively. If a shift type prescribes the use of less breaks, it will be assumed that the corresponding break rules have the value *NIL* (e.g. $br_k^{eb} = \mathit{NIL}$). The set of all break rules will be denoted by BR. A break rule $br \in BR$ is defined by a start time window $[a_{br}, b_{br}]$ relative to the shift start and a length l_{br}.

A *shift* $s \in S$ can be regarded an instance of a shift type $k \in K$ on a day $n \in N_k$, see e.g. Koop [1988]. The shift start and end times of a shift s are then given by st_{kn} and et_{kn}. An origin task i_s^o and a destination task i_s^d will delimit the start and end of a shift s. The start time windows of these tasks are naturally defined as $[a_{i_s^o}, b_{i_s^o}] := [st_{kn}, st_{kn}]$ and $[a_{i_s^d}, b_{i_s^d}] := [et_{kn}, et_{kn}]$. The sets of all origin and destination tasks will be denoted by I^o and I^d, respectively.

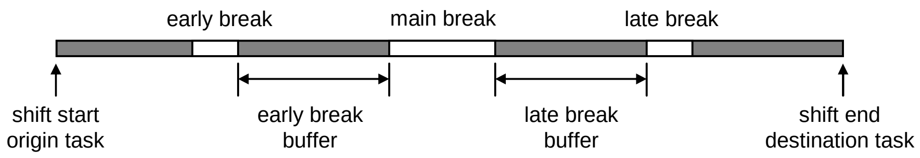


Figure 5.1.: Structure of a shift with origin, destination and break tasks.

A shift can have up to three breaks as defined by the break rules of its shift type. Shifts of type k will contain one *break task* for each break rule of k. The break rules of shift type k thus induce up to three break tasks $I_s^{break} = \{i_s^{eb}, i_s^{mb}, i_s^{lb}\}$ in a shift s. Start time windows of break tasks are defined by the break rules relative to the start time st_{kn} of the shift type realisation (k, n):

$$\begin{aligned}
[a_{i_s^{eb}}, b_{i_s^{eb}}] &:= [st_{kn} + a_{br_k^{eb}}, st_{kn} + b_{br_k^{eb}}] \\
[a_{i_s^{mb}}, b_{i_s^{mb}}] &:= [st_{kn} + a_{br_k^{mb}}, st_{kn} + b_{br_k^{mb}}] \\
[a_{i_s^{lb}}, b_{i_s^{lb}}] &:= [st_{kn} + a_{br_k^{lb}}, st_{kn} + b_{br_k^{lb}}]
\end{aligned}$$

Analogously, break task lengths are given by the break rules:

$$\begin{aligned}
l_{i_s^{eb}} &:= l_{br_k^{eb}} \\
l_{i_s^{mb}} &:= l_{br_k^{mb}} \\
l_{i_s^{lb}} &:= l_{br_k^{lb}}
\end{aligned}$$

While the shift attributions of work tasks $i \in I$ will be decision variables of the optimisation process, the shift assignment of break tasks is fixed. The set of all break tasks will be denoted by $I^b := \bigcup_{s \in S} I_s^b$.

A main break rule $br := br_k^{mb}$ can furthermore define *break buffer* times bb_{br}^{eb} and bb_{br}^{lb}. While bb_{br}^{eb} defines the minimum time between the end of the early break and the start of the main break, bb_{br}^{lb} is a minimum buffer time between main and late break.

Tasks can take place at different locations, entailing *travel times* between the tasks. If a task $i \in I \cup I^d$ is placed in a shift with a predecessor work task $j \in I$, a travel time $d_{j,i}$ must be performed directly before i's start. While in vehicle routing settings, the minimisation of travel distances is often an explicit objective (see Section 1.4), they only impose feasibility conditions in our setting. Clearly, time windows must be respected, i.e. an arrival at i before its earliest start time a_i entails waiting time. Shifts start and end at a central depot. Depot travel times are given by $d_{i_s^o,i}$ and d_{i,i_s^d} for $i \in I$.

Breaks will take place at the location of their predecessor work tasks, i.e. there are no travel times before breaks. We will therefore define $d_{i,j} = 0$ if $j \in I^b$. Because the predecessor of a break can be another break, the determination of travel times of tasks after breaks may require a recourse over several break tasks. If $pred_i$ denotes the predecessor of a task i in a shift, we define the first regular predecessor $rpred(i) \in I \cup I^o$ of a task $i \in I \cup I^b \cup I^d$ as

$$rpred(i) := \begin{cases} pred_i & \text{if } pred_i \in I \cup I^o \\ rpred(pred_i) & \text{else} \end{cases}$$

With this definition, the travel time which must be accomplished before a task i (possibly a break task) is $d_{rpred(i),i}$. As in Chapter 4, travel times are assumed to obey the triangle inequality.

The goal of shift planning consists in covering all work tasks by shifts of appropriate shift types. The set S of shifts along with the shift type K_s and day N_s for each shift $s \in S$ are thus decision variables of the optimisation process[1]. Furthermore, S_i will denote the shift attribution of task

[1]As in Chapter 4, we will represent decision variables by uppercase letters with indices for the entities to which they refer (e.g. K_s, N_s, S_i), see also Appendix A.

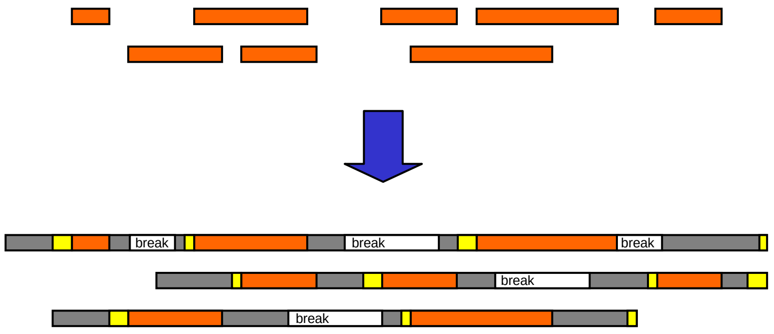


Figure 5.2.: Task-level shift planning.

$i \in I \cup I^b \cup I^o \cup I^d$. Note that for break and delimiter task, this shift attribution is fixed. As in Chapter 4, we will use an insertion-based model and therefore maintain the predecessor $pred_i$ and successor task $succ_i$ for each $i \in I \cup I^b \cup I^o \cup I^d$. The start times T_i for each $i \in I$ will also be regarded as decision variables even if they do not have any impact on the objective function.

The general setting of shift planning is sketched in Fig. 5.2. In the upper part, the (possibly movable) work tasks are displayed which are the input of the optimisation process. The output is a set of shifts (shown in the lower part), including travel times (yellow) between the tasks. Note that there are no travel times before breaks.

5.2.2. Qualifications

Each work task can have qualification requirements, corresponding to basic skills which a worker must meet. Each task $i \in I$ thus requires a set $Q_i \subseteq Q$ of qualifications where Q is the set of all elementary qualifications.

A shift derives its qualification requirements from its tasks. Clearly, if a task requires the worker to have a certain skill, a person working a shift must cover all qualification requirements of tasks in the shift. The qualification profile Q_s of a shift s is thus given by

$$Q_s = \bigcup_{\substack{i \in I \\ S_i = s}} Q_i$$

Since skilled workers are generally scarce, the combination of qualification requirements in shifts must generally be restricted. First of all, this will be done by restricting the number of qualifications per shift by a constant $q^{max} \in \mathbb{N}$ ($|Q_s| \leq q^{max}$). Note that this qualification restriction is closely related to capacity constraints in vehicle routing (see e.g. Toth and Vigo [2001a]). This implies that all tasks must obey this limit ($|Q_i| \leq q^{max}$) since otherwise, a task could never be assigned to a shift.

Furthermore, we introduce an objective function term for penalising the use of scarce qualification combinations. This term should be regarded an approximation for obtaining solutions reflecting the skills of the workforce at hand. More exact approaches limiting shifts along workforce restrictions may be subject of future research. Qualification penalties will be a subordinate objective, i.e. we will not accept better qualification combinations if this entails additional shift costs.

We will assume that *qualification preferences* $qp(Q') \in [0, 1]$ can be given for each set $Q' \subseteq Q$ with $qp(\emptyset) = 1$ and $qp(Q_1) \geq qp(Q_2)$ if $Q_1 \subseteq Q_2$. Qualification preferences should reflect the relative frequencies of skill combinations among the staff, i.e. scarce qualification combinations should have low preference values. Note that while an analysis of ways to obtain appropriate and consistent values is beyond the scope of this work, the determination of preferences must be regarded a difficult problem in itself.

Qualification combinations Q' with $qp(Q') = 0$ will be interpreted as *incompatible qualifications*, i.e. such qualifications must not be combined in shifts. A typical example for incompatible qualifications are water and faeces services on airports which are not combined for hygienic reasons.

Some tasks in a scenario may require qualifications which can only be covered by few employees. A wide-spread use of such special qualifications may prevent a shift plan from being covered by appropriate staff. Knowing the number of staff with different skills, planners can often specify maximum limits on the numbers of shifts containing a specific qualification. This will be represented by *qualification restrictions* QR. A qualification restriction $qr \in QR$ is defined by a qualification $q_{qr} \in Q$ and a limit $m_{qr} \in \mathbb{N}$ on the number of shifts requiring q_{qr}. Imposing strict maximum limits may prevent an algorithm from finding feasible solutions. We will therefore use a large penalty M^{QR} if a qualification limit is exceeded.

5.2.3. Crews

The planning practice on airports sometimes requires workers to be planned in *crews* (teams). This comprises the task parallelism constraints which have already been described in Section 4.3 for the workload levelling problem. A shift planning scenario may thus define a number of *task crews* $\mathcal{C}$ (corresponding to the crews of Section 4.3) with $C \subseteq I \, \forall C \in \mathcal{C}$.

Sometimes it may already be sufficient to impose temporal restrictions on crew tasks. In other cases, we may not only want the tasks to be performed in parallel, but due to legislative or union regulations, the staff may work in fixed groups on parallel shifts, performing groups of parallel tasks during their shifts. For a given shift planning scenario, the size $cs \in \mathbb{N}$ of each crew of workers will be fixed. If thus a scenario with $cs > 1$ is given, the set of shifts S will consist of groups of cs shifts each.

A typical example is the planning of cabin cleaning personnel [Stern and Hersh, 1980]. The staff in cleaning departments often consists of crews of size $cs = 4$. We thus have to create blocks of four shifts each which we will call *shift crews*. Each shift crew $H \in \mathcal{H}$, $H \subseteq S$ is thus a group of cs shifts.

The crew size cs generally corresponds to the size of typical task groups. In the above cabin cleaning example, we would thus expect that tasks are usually blocked as groups of four tasks because the cleaning of a typical aircraft may require this number of workers. To avoid splitting up staff crews, each such task group should always be assigned in total to a shift crew.

Clearly, crews should not only work on shifts of parallel types, but their blocks of tasks should also be performed as much in parallel as possible. However, the task parallelism constraints of task crews are often imposed on larger blocks of tasks. As an example, a wide-body aircraft may require two crews of four people to work in parallel in order to accomplish the cleaning work within the groundtime. Task parallelism would then refer to a group of eight tasks while blocks of four tasks are assigned to parallel shifts.

A task crew $C \in \mathcal{C}$ may thus be split up into one or several *subcrews* each of which has a maximum size of cs. If all tasks in a task crew have identical time windows and lengths, task crews may be split arbitrarily. However, this is not the case in general. If $cs > 1$, we will therefore assume that a set $\mathcal{B}$ of subcrews is given such that each subcrew $B \in \mathcal{B}$ is a subset of the tasks of a task crew. Note that if $cs = 1$, a scenario can contain task crews without any subcrews, i.e. only task parallelism constraints are imposed.

We will restrict each subcrew $B \in \mathcal{B}$ to be assigned to a block $H \in \mathcal{H}$ of shift crews. Note that subcrews can have sizes less than cs, and crew planning scenarios may even comprise single non-crew tasks. We will allow such ungrouped tasks to be covered either by crew shifts or by

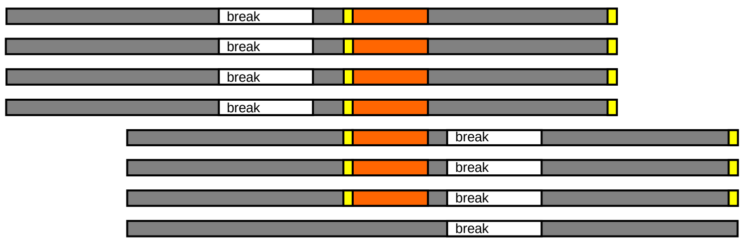


Figure 5.3.: Handling of task crews and subcrews.

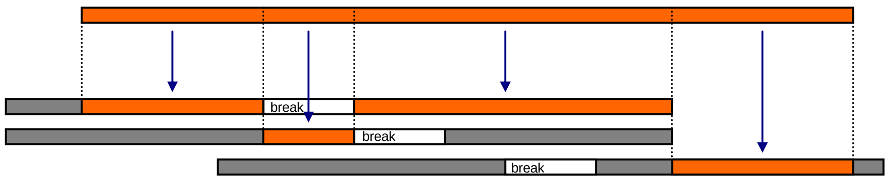


Figure 5.4.: Task splitting.

singleton shifts which are not grouped in crews. Crew planning scenarios with given crew size cs will therefore generally involve blocks of crew shifts of order cs and single shifts. Even if each crew member can cover different sets of tasks, we will restrict the breaks of shift crews to take place in parallel.

Figure 5.3 illustrates the handling of crew tasks for a department with $cs = 4$. In the example, seven tasks build a task crew whose tasks are placed in parallel. The task crew consists of two subcrews each of which is placed in a block of crew shifts of size 4. Note that not only the crew tasks are placed in parallel (assuming that the time windows are not fixed a priori), but also the break tasks.

5.2.4. Task Splitting

Due to their lengths, some tasks may not fit into any shift type. As an example, occupations for ticketing or check-in counters or help desks frequently span over the whole day, but shift types typically have maximum lengths of 8 to 12 hours. Nevertheless, it may not be obligatory to attribute such tasks to a single person. Thus it is possible to split it at some point in time and assign the resulting task parts to different shifts, introducing the concept of *preemptive tasks*. Figure 5.4 shows the basic idea: A long task is interrupted at the start time of the break of the first shift. While the third part of the task is again attributed to the same shift, the break time interval and the final part are attributed to separate shifts.

Planners on airports usually know which types of tasks must be split in order to ensure that a feasible shift attribution can be found. Preemptive tasks should generally not be split into arbitrary many parts since a frequent switching between different workers disrupts operations. We therefore assume that for each task i, a *minimum split length* msl_i is given, denoting the minimum length of a task part after splitting. If $msl_i = \infty$, we will assume that i cannot be split. Crew tasks will always be non-preemptive.

We could try to determine appropriate *split parts* (parts which result from splitting) before shift planning, i.e. the split parts would already be input of the algorithm. This would clearly fix the number of split parts, their lengths and split points. These figures should however depend on the shift types and the context given by surrounding tasks. Task splitting is therefore integrated into shift planning, i.e. the times at which tasks are split will be decision variables of the optimisation process.

For each task i with $msl_i < \infty$, we will determine the number of split parts beforehand, replacing i by its split parts in the input task set I. The number of generated split parts will be taken to be an upper bound on the number of effectively used split parts in a schedule.

The obvious upper bound $\lfloor l_i/msl_i \rfloor$ on the number of split parts is generally too weak. This number can be reduced if we take some considerations about meaningful split points with regard to the shift types into account. In practice, planners often try to cover a long task principally by one shift, only taking out the time span of a break (for an example, cf. to the first shift in Fig. 5.4). We will determine the shift type that covers the longest part of the task without taking care of breaks, but respecting displacements of the task within its start time interval. For each break, we assume that one additional split part has to be created. If i leaves a part which does not fit into any shift type, the procedure is repeated for the remaining task duration.

If a and b are earliest and latest start times and l is a length (initially, a, b and l will be equal to a_i, b_i and l_i), we can calculate the part of a task which is covered by shift type k on day n:

$$cov(i,k,n,a,b,l) := \begin{cases} \min(et_{kn} - d_{i,i_k^d}, \max(st_{kn} + d_{i_k^o,i}, a) + l) & \text{if } st_{kn} \leq b \\ -\max(st_{kn} + d_{i_k^d,i}, a) & \\ 0 & \text{else} \end{cases}$$

If $nb(t_1,t_2)$ is the number of breaks in the time interval $[t_1,t_2]$[2], the number of split parts for task i with earliest start time $a_i =: a$, latest start time $b_i =: b$ and length $l_i =: l$ is iteratively calculated as

$$sparts(a,b,l) := \begin{cases} 0 & \text{if } l = 0 \\ 1 + nb(st, st + tcov) + sparts(st + cv, st + cv, l - cv) & \\ \text{with } cv := \max_{k',n'} cov(k',n',a,b,l) & \\ \qquad (k,n) := \text{argmax}_{k',n'} cov(k',n',a,b,l) & \text{else} \\ \qquad st := \max(a, st_{kn}) & \end{cases}$$

Knowing the number $m := sparts(a_i,b_i,l_i)$ of parts to be created from an original task i, we create n split parts $i_1,\ldots,i_m$. Each of these parts will have the same start time interval $[a_{i_j},b_{i_j}] := [a_i,b_i]$ as the original task i. Additionally, we define an ordered sequence of the split parts $P := (i_1,\ldots,i_m)$ denoted as *split task* of total length $l_P := l_i$. Split parts will inherit the location of the original task, i.e. travel times from and to split parts are defined as for i. Split task and split parts replace the original task i as part of the input data.

The set of all split tasks will be denoted by $\mathcal{P}$. The lengths of the single split parts will be variable, and there is no meaning in attributing lengths l_{i_j} to split parts i_j. We therefore set $l_{i_j} = 0 \, \forall i_j \in P, P \in \mathcal{P}$. The split task P inherits the minimum split part length from the original task: $msl_P := msl_i$. For a split part i_j of P, we will write $i_j \in P$, and the split task $P_{i_j} \in P \cup \{\mathit{NIL}\}$ of a $i \in I$ will be P if $i \in P$ for a $P \in \mathcal{P}$ and *NIL* otherwise.

Split points will be represented by the start times T_{i_j} of the single split parts. Their end times are equal to the start time of the next split part, i.e. $T_{i_{j+1}}$ also represents the end time of i_j. Note that while the start times of interior split parts are generally not fixed, the start of the first split part and the end of the last split part are movable if and only if the original task is movable. If these outer times are movable, they are still constrained to obey a distance which is equal to the length of the original task.

In order to define an end time for the last split part i_m, a delimiting *pseudo split part* i_P^{pseudo} is defined for each split task $P \in \mathcal{P}$. The start time $T_{i_P^{pseudo}}$ of this task represents the end

[2]We take a pessimistic approach and define a break task i_s^b to be in time interval $[st,et]$ iff $\max(st, a_{i_s^b}) < \min(et, b_{i_s^b} + l_{i_s^b})$.

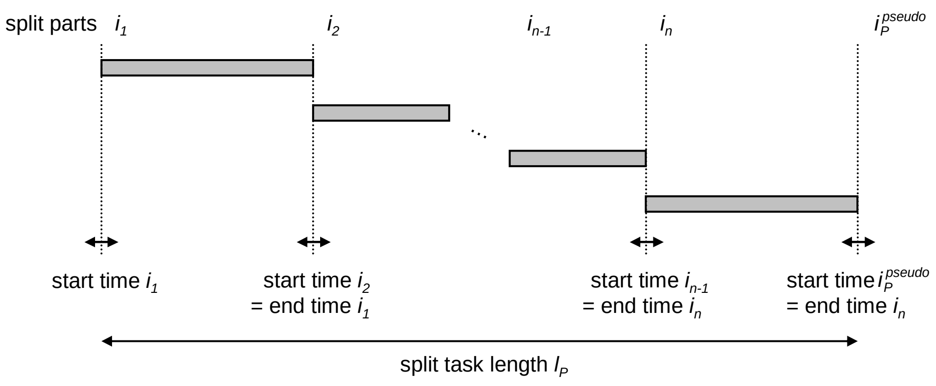

split parts i_1
i_2
i_{n-1}
i_n
i_P^{pseudo}
start time i_1
start time i_2 = end time i_1
start time i_{n-1} = end time i_n
start time i_P^{pseudo} = end time i_n
split task length l_P

Figure 5.5.: Structure of a split task.

time of split part i_m. Note that the pseudo split part is only introduced for convenience reasons since $T_{i_P^{pseudo}} = T_{i_1} + l_P$. The start time window of the pseudo task is $[a_{i_P^{pseudo}}, b_{i_P^{pseudo}}] = [a_{i_1} + l_P, b_{i_1} + l_P]$. Since i_P^{pseudo} is not a "real" task, we define $S_{i_P^{pseudo}} := \mathit{NIL}$. The structure of a split task is illustrated in Fig. 5.5.

We define the split part successor $ssucc_i$ of a split part i as

$$ssucc_i := \begin{cases} i_{j+1} \text{ if } P_i = (i_1, \ldots, i_m) \text{ and } i = i_j, j < n \\ pt_P \text{ if } P_i = (i_1, \ldots, i_m) \text{ and } i = i_m \\ \mathit{NIL} \text{ if } i = i_P^{pseudo} \text{ for a } P \in \mathcal{P} \end{cases}$$

and the split part predecessor $spred_i$ as

$$spred_i := \begin{cases} \mathit{NIL} \text{ if } P_i = (i_1, \ldots, i_m) \text{ and } i = i_1 \\ i_{j-1} \text{ if } P_i = (i_1, \ldots, i_m) \text{ and } i = i_j, j > 1 \\ i_m \quad \text{if } i = i_P^{pseudo} \text{ for a } P \in \mathcal{P} \end{cases}$$

The length of a split part i can be calculated from its start time T_i and the start time T_{ssucc_i} of its split part successor. The actual length $al(i)$ of a task $i \in I$ is therefore defined as

$$al(i) := \begin{cases} l_i & \text{if } P_i = \mathit{NIL} \\ T_{ssucc_i} - T_i & \text{else} \end{cases}$$

5.2.5. Task Overlapping

The lengths of work tasks on airports are hardly deterministic. Especially in peak times, employees are often encouraged to work harder, perform several tasks in parallel or in shorter times. In a deterministic model, this practice cannot be represented in full detail. To defuse some of the strict character of deterministic models, a task $i \in I$ will be allowed to overlap up to a maximum task-dependent duration of tol_i^{max} minutes. This maximum overlap can depend on task attributes like the staff group by which it will be performed.

For easier handling, we will generalise tol_i^{max} to a maximum overlap ol_i^{max} for general tasks $i \in I \cup I^b \cup I^o \cup I^d$. Because an overlap with origin, destination or break tasks would mean performing parts of a task within a break or outside a shift, we set $ol_i^{max} = 0$ for all $i \in I^b \cup I^o \cup I^d$. Furthermore, no more than two tasks should overlap, meaning that the overlap ol_i^{max} of work tasks $i \in I$ should be limited by half the length of the task.

For split parts, the task length is a consequence of the start times of adjacent split parts. Since allowed overlaps will normally be substantially shorter than minimum split part durations, we will relate the overlap limit to the minimum split part length. This leads to the following definition:

$$ol_i^{max} := \begin{cases} 0 & \text{if } i \in I^b \cup I^o \cup I^d \\ \min\left(tol_i^{max}, \left\lfloor \frac{l_i}{2} \right\rfloor\right) & \text{if } i \in I \text{ and } P_i = \mathit{NIL} \\ \min\left(tol_i^{max}, \left\lfloor \frac{msl_{P_i}}{2} \right\rfloor\right) & \text{if } i \in I \text{ and } P_i \neq \mathit{NIL} \end{cases}$$

If possible, overlap should be avoided. Because we will generally accept overlap if shift costs can be lowered, we will not try to handle overlap minimisation as part of the optimisation model. Instead, overlap will be minimised as a part of a post-processing step, cf. Section 5.10.

5.2.6. Shift Number Restrictions

Different employment categories among the workforce often impose considerable restrictions, and solutions will fall short of planners' requirements if these are not represented. As an example, it may be advantageous to create many short shifts to cover the demand in peak times, i.e. periods of high workload. In general, short shifts can only be covered by part time staff, and part time workers are only available to a certain extent due to union and legal regulations, see e.g. Bennett and Potts [1968], Schindler and Semmel [1993] and Brusco et al. [1995].

We therefore introduce *shift number restrictions*, limiting the numbers of shifts of certain types, including *minimum* and *maximum* as well as *relative* and *absolute* restrictions. Each absolute restriction $r \in R_{abs}^{min}$ ($r \in R_{abs}^{max}$) imposes a minimum (maximum) limit of $m_r \in \mathbb{N}$ shifts on shift type realisations from $K_r \times N_r$ where K_r and N_r are sets of reference shift types and days. In contrast, a relative shift number restriction $r \in R_{rel}^{min}$ ($r \in R_{rel}^{max}$) defines a minimum (maximum) proportion of $p_r \in \mathbb{R}_+$ on the number of shifts from $K_r \times N_r$ with regard to all shifts on the reference days N_r (i.e. shifts from $K \times N_r$).

While minimum restrictions can always be obeyed, it may not be possible to obey maximum limits, and we will impose a large penalty M^{SNR} if a maximum shift number restriction (absolute or relative) cannot be obeyed.

5.3. Mathematical Model

We are now ready to define the shift planning problem by summarising its parameters and decision variables and defining constraints and optimisation criteria.

The parameters (input) of the algorithm are given by:

I	set of work tasks
$[a_i, b_i] \subset \mathbb{N}$	start time window of task $i \in I$
$l_i \in \mathbb{N}$	length of task $i \in I$
$Q_i \subseteq Q$	qualifications of task $i \in I$
$ol_i^{max} \in \mathbb{N}$	maximum overlap for $i \in I \cup I^b \cup I^o \cup I^d$
$\mathcal{P}$	set of split tasks
$\mathcal{C}, \mathcal{B}$	sets of task crews, subcrews
$d_{i_1,i_2} \in \mathbb{N}$	travel time between $i_1 \in I \cup I^o$ and $i_2 \in I \cup I^d$
K	set of shift types
$c_k \in \mathbb{R}_+$	cost of shift type $k \in K$
$N_k \subseteq N$	days of validity of shift type $k \in K$
$st_{kn}, et_{kn} \in \mathbb{N}$	start and end time of shift type realisation $k \in K$ on day $n \in N_k$

$br_k^{eb}, br_k^{mb}, br_k^{lb} \in BR$	early, main and late break rule for shift type $k \in K$
$[a_{br}, b_{br}] \subset \mathbb{N}$	break start time relative to shift start given by break rule br
$l_{br} \in \mathbb{N}$	break length given by break rule br
$q^{max} \in \mathbb{N}$	maximum number of qualifications in shifts
$qp(Q') \in \mathbb{R}_+$	qualification penalties for qualification set Q'
QR	qualification restrictions
$m_{qr} \in \mathbb{N}$	maximum limit given by qualification restriction $qr \in QR$
$R_{abs}^{min}, R_{abs}^{max}$	absolute shift number restrictions
$R_{rel}^{min}, R_{rel}^{max}$	relative shift number restrictions
$m_r \in \mathbb{N}$	minimum/maximum limit for absolute shift restrictions $r \in R_{abs}^{min} \cup R_{abs}^{max}$
$p_r \in \mathbb{R}_+$	minimum/maximum proportions for relative shift restrictions $r \in R_{rel}^{min} \cup R_{rel}^{max}$

The following are the decision variables:

S	set of shifts
$K_s \in K$	shift type for each shift $s \in S$
$N_s \in N$	day for each shift $s \in S$
$\mathcal{H}$	shift crews
$S_i \in S$	shift for each task $i \in I$
$pred_i \in I \cup I^o \cup I^b$	predecessor of task $i \in I \cup I^d \cup I^b$
$succ_i \in I \cup I^d \cup I^b$	successor of task $i \in I \cup I^o \cup I^b$
$T_i \in T$	start time of task $i \in I \cup I^b$

Note that while the objective function will not depend on the start times, the start time variables T_i are used for validity checks and are useful as part of the output. Equally, an output of realised travel times and qualification profiles Q_s of the shifts can be interesting.

As described above, shift costs are the main optimisation criterion. The planner may furthermore want to optimise the qualification mix in the shifts. As described in Section 5.2.2, qualification penalties are regarded a subordinate objective, i.e. objectives are lexicographically ordered. The qualification penalty term will therefore be normalised by means of the minimum positive qualification preference

$$qp_{min} := \min_{\substack{Q' \subseteq Q \\ qp(Q')>0}} qp(Q')$$

By multiplication with a weight w^{QP}, we keep a means of adjusting the relative significance of qualification preferences[3].

As described above, exceeding the upper limits of qualification or maximum shift number restrictions is penalised by large values M^{QR} and M^{SNR}, respectively.

Sometimes it may not be possible to place all tasks in shifts, e.g. if there are no shift types covering the total duration of a task. Splitting may be a remedy, but there could be times of the day which are not covered by any shift types at all, or splitting may not be desired. If task splitting is allowed, we want to assign as many task minutes as possible. We therefore impose a penalty M^{UT} for each task minute which is not assigned to a shift. M^{UT} will be chosen such that covering a task pays off even if qualification or shift restrictions are exceeded.

The shift planning problem is defined by

[3]The equivalent weights technique for lexicographically ordered objectives was originally proposed by Sherali [1982], see also Berrada et al. [1996] for a workforce scheduling application.

$$\min \Bigg\{ \sum_{s\in S} c_{K_s} \tag{5.1}$$

$$+ w^{QP} \frac{qp_{min}}{|S|} \sum_{s\in S} qp(Q_s)^{-1} \tag{5.2}$$

$$+ M^{QR} \sum_{qr\in QR} \max\left(|\{s\in S | q_{qr}\in Q_s\}| - m_{qr}, 0\right) \tag{5.3}$$

$$+ M^{SNR} \sum_{r\in R^{max}_{abs}} \max\left(\sum_{k\in K_r}\sum_{n\in N_r} |\{s\in S|K_s = k, N_s = n\}| - m_r, 0\right) \tag{5.4}$$

$$+ M^{SNR} \sum_{r\in R^{max}_{rel}} \max\Bigg(\sum_{k\in K_r}\sum_{n\in N_r} |\{s\in S|K_s = k, N_s = n\}| - \tag{5.5}$$

$$p_r \cdot \sum_{k\in K}\sum_{n\in N_r} |\{s\in S|K_s = k, N_s = n\}|, 0\Bigg)$$

$$+ M^{UT} \cdot \sum_{\substack{i\in I\\ S_i = NIL}} al(i) \Bigg\} \tag{5.6}$$

subject to
Time windows

$$T_i \in [a_i, b_i] \;\; \forall i \in I \cup I^b \cup I^o \cup I^d \tag{5.7}$$

Shift temporal relation

$$\begin{aligned} T_{ssucc_i} + d_{rpred(succ_i),succ_i} - \min(ol_i^{max}, ol_{succ_i}^{max}) &\le T_{succ_i} \text{ if } P_i \neq NIL\\ T_i + l_i + d_{rpred(succ_i),succ_i} - \min(ol_i^{max}, ol_{succ_i}^{max}) &\le T_{succ_i} \text{ else}\\ &\forall i \in I \cup I^b \cup I^o \end{aligned} \tag{5.8}$$

Crew temporal relation

$$\left.\begin{aligned} T_{i_{j+1}} &\ge T_{i_j} + (a_{i_{j+1}} - a_{i_j})\\ T_{i_{j+1}} &\le T_{i_j} + l_{i_j} + (a_{i_{j+1}} - a_{i_j}) \end{aligned}\right\} \forall C = (i_1,\ldots,i_m) \in \mathcal{C}, \forall 1 \le j < n \tag{5.9}$$

Shift type days

$$N_s \in N_{K_s} \;\; \forall s \in S \tag{5.10}$$

Shift start and end times

$$\left.\begin{aligned} T_{i_s^o} &= st_{K_sN_s}\\ T_{i_s^d} &= et_{K_sN_s} \end{aligned}\right\} \forall s \in S \tag{5.11}$$

Break buffers

$$\begin{aligned} T_{i_s^{eb}} + l_{i_s^{eb}} + bb^{eb}_{br} &\le T_{i_s^{mb}} \;\; \forall s \in S : br^{eb}_{K_s} \neq NIL \neq br^{mb}_{K_s} =: br\\ T_{i_s^{mb}} + l_{i_s^{mb}} + bb^{eb}_{br} &\le T_{i_s^{lb}} \;\; \forall s \in S : br := br^{mb}_{K_s} \neq NIL \neq br^{lb}_{K_s} \end{aligned} \tag{5.12}$$

Predecessor-successor consistency

$$pred_i = j \Leftrightarrow succ_j = i \;\; \forall i \in I \cup I^b \cup I^o \tag{5.13}$$

Shift-predecessor consistency

$$pred_i = j \wedge S_i = s \Rightarrow S_j = s \;\; \forall i \in I \cup I^b \cup I^o \tag{5.14}$$

Shift difference

$$S_{i_1} \neq S_{i_2} \;\; \forall C \in \mathcal{C}, \forall i_1, i_2 \in C, i_1 \neq i_2 \tag{5.15}$$

Subcrew attribution

$$\exists H \in \mathcal{H} : S_{i_1}, S_{i_2} \in H \;\; \forall B \in \mathcal{B}, i_1, i_2 \in B, i_1 \neq i_2 \tag{5.16}$$

Crew shift types

$$\left.\begin{array}{l} K_{s_1} = K_{s_2} \\ N_{s_1} = N_{s_2} \end{array}\right\} \forall s_1, s_2 \in S, \exists H \in \mathcal{H} : s_1, s_2 \in H \tag{5.17}$$

Crew break parallelism

$$\left.\begin{array}{l} T_{i^{eb}_{s_1}} = T_{i^{eb}_{s_2}} \\ T_{i^{mb}_{s_1}} = T_{i^{mb}_{s_2}} \\ T_{i^{lb}_{s_1}} = T_{i^{lb}_{s_2}} \end{array}\right\} \forall s_1, s_2 \in S, \exists\; H \in \mathcal{H} : s_1, s_2 \in H \tag{5.18}$$

Split task length

$$T_{i_1} + l_P = T_{i_P^{pseudo}} \;\; \forall P = (i_1, \ldots, i_m) \in \mathcal{P} \tag{5.19}$$

Minimum split part lengths

$$\left.\begin{array}{r} T_i + msl_{P_i} \leq T_{ssucc_i} \text{ if } ssucc_i \neq succ_i \\ T_i = T_{ssucc_i} \text{ else} \end{array}\right\} \forall i \in I : P_i \neq NIL \tag{5.20}$$

Shift qualification profiles

$$Q_s = \bigcup_{\substack{i \in I \\ S_i = s}} Q_i \tag{5.21}$$

Maximum qualification requirements

$$|Q_s| <= q^{max} \;\; \forall s \in S \tag{5.22}$$

Incompatible qualifications

$$Q_s \nsubseteq Q' \;\; \forall Q' \subseteq Q : qp(Q') = 0, \forall s \in S \tag{5.23}$$

Absolute minimum shift number restrictions

$$\sum_{k \in K_r} \sum_{n \in N_r} |\{s \in S | K_s = k, N_s = n\}| \geq m_r \;\; \forall r \in R^{min}_{abs} \tag{5.24}$$

Relative minimum shift number restrictions

$$\left.\begin{array}{l} \displaystyle\sum_{k \in K_r} \sum_{n \in N_r} |\{s \in S | K_s = k, N_s = n\}| \geq \\ \qquad\qquad \displaystyle p_r \cdot \sum_{k \in K} \sum_{n \in N_r} |\{s \in S | K_s = k, N_s = n\}| \end{array}\right\} \forall r \in R^{min}_{rel} \tag{5.25}$$

The objective function term (5.1) describes the shift costs as induced by the shift types while (5.2) imposes qualification penalties. Since the number of used shifts should not an impact, the sum of the inverted qualification preferences is divided by the number $|S|$ of shifts. Note that all qualification preferences are strictly positive due to constraint (5.23). For normalisation, the qualification penalty term is multiplied by qp_{min}. Consequently, the qualification term has values in $[0, 1]$, rendering tuning of qualification influences via w^{QP} easier.

Terms (5.3), (5.4) and (5.5) impose large penalties if qualification or maximum shift number restrictions cannot be obeyed. Finally, (5.6) accounts for task minutes which cannot be assigned to shifts, making use of the actual length $al(i)$ defined in Section 5.2.4.

Equation (5.7) restricts all tasks (including origin, destination and break tasks) to start within their start time window. Constraint (5.8) enforces tasks to be placed one behind the other in shifts. While for non-preemptive tasks, the end time is given by the start time plus its length, the end of split parts is given by the start time T_{ssucc_i} of the split part successor. Tasks are allowed to overlap by at most $\min(ol_{i_1}^{max}, ol_{i_2}^{max})$ time units.

Crew tasks should be placed as much in parallel as possible as described by equations (5.9). Again, it suffices to use a linear number of constraints for adjacent pairs (i_j, i_{j+1}) of crew tasks, cf. Section 4.3. The difference $a_{i_{j+1}} - a_{i_j}$ between the original start times of adjacent crew tasks is interpreted as an offset for the realised start times.

Equation (5.10) restrains a shift to be on a day which is valid for its shift type K_s, and (5.11) synchronises the shift type realisation with the shift start and end times.

Constraints (5.12) impose minimum buffer times between early/main and main/late break while equations (5.13) and (5.13) guarantee consistency between predecessor, successor and shift variables.

To ensure a fair assignment of the work tasks to the members of a team, no more than one task of a subcrew is assigned to the same worker as described by (5.15) (note that this is often implicit in (5.9). While subcrews are part of the input, the shifts are grouped into shift crews, and subcrews are always assigned to such blocks of shifts (equation (5.16)). All shifts of a shift crew ask for the same shift type realisation (constraints (5.17)), and their breaks take place in parallel (constraints (5.18)).

Split parts must cover the total length of their split task which is ensured by constraints (5.19). Each split part has a minimum length as given by the value msl_{P_i} (constraints (5.20)). Note that the minimum distance between start times is only imposed if two split parts are not successors in a shift. Split parts which are placed as successors in a shift are interpreted as a single effective part. A solution to the shift planning problem can thus use less effective parts than the logical parts which were originally generated. The start times of split parts which are placed one behind the other are assumed to be equal.

Equation (5.21) relates shift qualification profiles to the qualification requirements of the tasks, and (5.22) restricts qualification requirements to a maximum size q^{max}. Constraint (5.23) forbids the use of incompatible qualifications. Absolute and relative minimum shift number restrictions are represented by (5.24) and (5.25).

While in the bulk of the literature on shift planning, the workload is represented by a histogram of workforce requirements per time interval, model (5.1)-(5.25) covers a set of work tasks. The shifts contain tours of tasks. Problem (5.1)-(5.25) can thus be interpreted as a specialised vehicle routing problem with time windows (VRPTW). Instead of using an intermediate histogram abstraction, the tasks are directly attributed to a set of appropriate shifts in an integrated approach. To the knowledge of the author, this is the first time that task-level shift planning is addressed.

The basic VRPTW problem is not only enriched by restrictions referring to the shifts, but also by complicating task-level constraints, including qualifications, task splitting, overlapping and crews. The crew temporal constraints (5.9) and break parallelism constraints (5.18) refer to tasks in different shifts. Therefore, they introduce vertical relations between the shifts while the classical shift temporal relations (5.8) have a horizontal character. Task splitting (preemptive tasks) add further vertical constraints to the problem since start time decisions for split parts generally involve two shifts.

In Chapter 7, it will be shown that even a simple subproblem of the shift planning problem is $\mathcal{NP}$-hard in the strong sense. While we will see that simple classes of shift planning problems can still be solved to optimality, this cannot be expected for general large-scale scenarios involving many constraints. Before this work, a construction heuristic for the shift planning problem has been used with success. While this algorithm generally produces good results in very low runtimes, solution quality sometimes suffers from evident flaws which has adverse effects on customer satisfaction.

It was therefore searched for a procedure which is able to provide a remedy for the deficiency of the initial algorithm. This was supposed to be done without sacrificing beneficial properties of the construction algorithm like its performance and solution properties like the grouping of tasks of different qualifications. It was therefore decided to implement a solution improvement algorithm, using the result of the initial algorithm as a starting point. By repeated local steps, obvious flaws of the solution were supposed to be overcome and the general solution quality enhanced. Because local steps only change small parts of the solutions, many properties of the initial solution can be conserved.

Traditional local search operators could not be expected to provide reasonable means for the improvement of shift plans. The good experience with the levelling algorithm of Chapter 4 was the reason for opting for a solution by a CP-based relax-and-optimise approach. Clearly, constraint programming is the method of choice for representing the complex constraints of the shift planning problem (5.1)-(5.25). Large neighbourhood search gives a framework for the relaxation and reoptimisation of parts of a given solution, using a restricted branch-and-bound scheme.

In the following, we will first develop a constraint programming model for the shift planning problem described above. This model is not restricted to the local search context, but some design decisions make it particularly appropriate for the reoptimisation setting. As in Chapter 4, we use an insertion-based model which allows for an efficient propagation of start times [Caseau and Laburthe, 1999] and is flexible in incorporating further constraints which may come up in the future [Campbell and Savelsbergh, 2004]. For performance reasons, we will accept a somewhat lower degree of consistency with regard to the determination of insertion positions. Effectively, a full evaluation would mean that some of the consistency tests would have to trigger on many events. Before explaining the domains and consistency tests, we will describe some general ideas with regard to the creation of shifts and the handling of unassigned tasks.

5.4. Shift Creation

Our goal is to find a set S of shifts covering the tasks I at minimum costs. In order to represent the complex constraints of task-level shift planning, we will use a constraint programming model. As in Chapter 4, different domains and consistency tests will pertain to the basic entities. As an example, we will use a start time domain for each task. Similarly, we need domains and consistency tests for the shifts. As an example, we will represent valid start times of break tasks as well as qulaification profiles of shifts by appropriate domain variables.

Such domains and consistency tests cannot be generated dynamically, but must be created beforehand. For the work tasks, this does not impose any problems since the tasks are part of the input data. In contrast, the set of shifts is part of the decision variables of the problem, and the composition of a “good” set of shifts is not obvious. We therefore have to create domains and consistency tests for a set of shifts which is not known in advance.

As a remedy, we will create so-called *shift templates*. These templates should be chosen such that any possibly optimal solution can be represented by the related domains and consistency tests. We therefore need upper bound approximations on the number of required shifts of different types. For the calculated shift numbers, we then domains and consistency tests. Clearly, we can leave shift templates empty, meaning that such shifts are not part of the final solution.

The starting solution for the improvement procedure clearly gives an indication for creating a suitable set of shift templates, but we will generally require additional shifts of other types. We will thus only make use of the set of tasks for determining an upper bound on the number of required shifts. Since there is no obvious way to determine which combinations of tasks will be covered by the same shifts, we will take the worst-cast approximation of covering each task by a separate shift.

In a first approach, we could create one set of templates for each shift type realisation. Many constraints effectively relate to the shift type realisation. As an example, this would fix the shift start and end times and define initial start time domains for the break tasks. Nevertheless, the approach has some disadvantages. First, the upper bound approximation for the number of required shift templates will be poor: For each task i, we would have to create a shift template for each shift type/day combination covering i, resulting in a huge number of shift templates. Second, the consequences for the logic of a solution algorithm are unfavourable. In practice, shifts can often be shortened or prolonged within the set of given shift types without affecting key figures like the number of breaks. If now the shift type is fixed for each template, the use of longer or shorter shift types requires a reattribution of all tasks to another shift template.

Instead of fixing shift types, we first only settle the day $N_s = n_s$ on which a shift template s starts and leave its actual shift type open. This means that the start time domains of origin, destination and break tasks are initialised to rather general values. As an example, the start time domain of an origin task incorporates all start times of the valid shift types for the template. Additionally, the initial break start time domains will contain all possible start times for any of the possible shift types. This means that we have to adapt the set of possible break times as soon as the shift type is known. Clearly, this can be carried out by a consistency test. Similarly, we will need consistency tests to ensure possible break buffer times.

Since the consistency tests for a shift template cannot change dynamically, we must choose the set of possible shift types such that the basic information on breaks (break durations, buffer times) is fixed for a template. As can be easily observed, all of this information relates to the set of break rules of the shift type. Consequently, we will fix the set of break rules for each shift template, i.e. a given shift template can be fixed to any shift type k on day n_s with the same set of break rules.

To formalise this idea, we introduce the notion of *break rule days* BRD. A break rule day $brd \in BRD$ is defined as a pair $brd := (n_{brd}, BR_{brd})$ of a reference day n_{brd} and a possibly empty set BR_{brd} of break rules such that each break rule type (early, main, late break) occurs at most once in the set. We furthermore define a set K_{brd} of all shift types which are covered by a break rule day brd:

$$K_{brd} := \{k \in K \,|\, n_{brd} \in N_k \wedge \{br_k^{eb}, br_k^{mb}, br_k^{lb}\} = BR_{brd}\}$$

We define the earliest start time st_{brd} for a break rule day brd as the minimum start time over its shift type realisations ($st_{brd} := \min_{k \in K_{brd}} st_{kn_{brd}}$) and the latest end time et_{brd} as the maximum over the end times ($et_{brd} := \max_{k \in K_{brd}} et_{kn_{brd}}$). Each shift template s will have a fixed break rule day brd_s (and therefore also a fixed day $N_s = n_{brd}$) while its shift type K_s will be determined in the course of the algorithm.

With these considerations and definitions, an easy temporal checking of tasks can be performed: A task i may be covered by a shift s of break rule day $brd := brd_s$ if

$$st_{brd} \leq b_i \wedge \min(st_{brd} + d_{i^o,i}, a_i) + l_i + d_{i,i^d} \leq et_{brd}$$

where $d_{i^o,i}$ and d_{i,i^d} are travel times from and to the depot. Note that with this definition, we neglect that there may be breaks preventing the task from being covered. To avoid checking all shift types of a break rule day, we will neglect the effect of breaks.

Two cases have to be distinguished. If the crew size cs is 1, the scenario will not comprise subcrews, and no shift crews must be created. We can therefore determine the number cnt_{brd} of shifts for each break rule day brd as

$$cnt_{brd} := |\{i \in I \mid st_{brd} + d_{i^o,i} \leq b_i \wedge \max(st_{brd}, a_i) + l_i + d_{i,i^d} \leq et_{brd}\}|$$

If there are split tasks in the input data ($\mathcal{P} \neq \emptyset$), split parts – which have vanishing lengths l_i – will also be counted in this term. While cnt_{brd} remains a valid upper bound on the shifts to be created, we may seek for more exact approximations for split tasks, e.g. taking minimum split part lengths into account. But since minimum lengths are only obeyed if split parts are placed in different shifts, we will content ourselves with the above definition.

If the crew size cs is greater than 1, we must create blocks of cs shifts. While most of the tasks will generally be grouped in subcrews in a such scenario, there may be singleton tasks which are not attributed to subcrews. Clearly, we do not need a whole shift crew to cover singleton tasks. We therefore count the number of shifts which are needed for non-crew tasks separately:

$$\begin{aligned} cnt_{brd}^{single} := |\{i \in I \mid i \notin \bigcup_{B \in \mathcal{B}} B \wedge \\ st_{brd} + d_{i^o,i} \leq b_i \wedge \\ \max(st_{brd}, a_i) + l_i + d_{i,i^d} \leq et_{brd}\}| \end{aligned}$$

When checking if a subcrew might fit into a break rule day, we must check if all tasks of the subcrew can be placed in the break rule day, i.e. we use the minimum over all latest start times and the maximum over all earliest end times for temporal checks with the break rule day. Thus, the number cnt_{brd}^{crew} of shift crews to be created for subcrew tasks is

$$\begin{aligned} cnt_{brd}^{crew} := |\{B \in \mathcal{B} \mid st_{brd} \leq \min_{i \in B}(b_i - d_{i^o,i}) \wedge \\ \max_{i \in B}(\max(st_{brd}, a_i) + l_i + d_{i,i^d}) \leq et_{brd}\}| \end{aligned}$$

In total, we will create

$$\left\lceil \frac{cnt_{brd}^{single}}{cs} \right\rceil + cnt_{brd}^{crew}$$

shift crews for a break rule day brd, each consisting of cs shift templates.

Because each of the shifts can finally be fixed to different shift types, the break rule day approach creates considerably less shifts than fixed shift type method. The more shift types require the same sets of break, the clearer the savings will be. We will never create more templates than in the fixed shift type approach. If many break rule combinations are used in the shift types, memory requirements may still be high. If this is the case, we can heuristically introduce upper bounds on the number of created templates per day or per break rule day.

Note that even if shift templates are used, only those shifts which finally comprise work tasks will be part of the output of the algorithm. Nevertheless, we will let S denote the set of all shift templates.

5.5. Unassigned Task Handling

As described in Section 5.3, it may not be possible to assign all tasks to shifts. If a task remains unassigned ($S_i = NIL$), a penalty of $M^{UT} \cdot al(i)$ proportional to its actual length is imposed. Allowing tasks not to be assigned to any shift can be cumbersome in algorithmic handling. We will therefore create a dummy shift $s = s_i^{dummy}$ for each task i which can be left unassigned.

If the task cannot be covered by a regular shift, this will be represented by the assignment to its dummy shift: $S_i := s_i^{dummy}$.

Note that if a task which can be covered is left unassigned in the initial solution, this must be regarded a flaw the construction algorithm. Furthermore, if splitting is used, it may be possible to assign larger portions of a split task than in the starting solution. In general, we can assume that only a small amount of the tasks is unassigned in the initial solution.

We first treat the case of non-split tasks i. For each such task i, a dummy shift s_i^{dummy} is created. Furthermore, we create a special dummy shift type k_l^{dummy} for each unassigned task length l and assign dummy shifts to the corresponding shift type: $K_{s_i^{dummy}} = k_{l_i}^{dummy}$. The cost of a dummy shift type k_l^{dummy} is given by $c_{k_l^{dummy}} = M^{UT} \cdot l$, reflecting the unassigned task penalties. The start and end times of dummy shift types are set to limits which guarantee that every task can be placed into a shift of the shift type, i.e. they only have one realisation and will generally be longer than 24 hours.

For split parts i, we cannot account unassigned task penalties via dummy shift types since lengths are not fixed. We will define one dummy shift per split task, meaning that logically unassigned split parts of the same split task P are placed in the same dummy shift s_P^{dummy}. To avoid penalties on the shift level, these dummy shifts have a fixed shift type k_0^{dummy} of costs $c_{k_0^{dummy}} = 0$. Penalties for unassigned split parts will thus be accounted separately.

A special break rule day $brd^{dummy} \in BRD$ is defined to cover all dummy shift types. All dummy shifts are assigned to this break rule day: $brd_{s_i^{dummy}} := brd^{dummy}$. Dummy shifts will not be regarded parts of the shift set S because they require a dedicated handling.

5.6. Constraint Model

We now come to the description of details of the constraint model. After showing which domain variables are used, we will describe the consistency tests which are used to propagate information on tasks, shifts, qualifications and so on. The main novelty of the model lies in the temporal propagation. In comparison to the model of Chapter 4, we not only have to incorporate shift types and breaks, but we must also devise rules which cover regular work tasks as well as split parts.

5.6.1. Domains

We start by describing the domains used in the CP model. The possible start times of each task i are maintained in a domain δ_i^{start} which is represented by a range $\delta_i^{start} = [\alpha_i, \beta_i]$. For work tasks $i \in I$, the start time domain is naturally initialised to $[a_i, b_i]$. The initial time windows of origin, destination and break tasks are initialised to the extent of the break rule day of the shift:

$$\delta_{i_s^o}^{start} = \delta_{i_s^d}^{start} = \delta_{i_s^{eb}}^{start} = \delta_{i_s^{mb}}^{start} = \delta_{i_s^{lb}}^{start} := [st_{brd_s}, et_{brd_s}]$$

The length of a split part i is the consequence of the values of two adjacent temporal variables δ_i^{start} and $\delta_{ssucc_i}^{start}$ of i and its split task successor $ssucc_i$, respectively. To provide control over the length of a split part, we introduce a split part length domain δ_i^{sl}. Again, possible lengths can be represented as a range which is initialised to

$$\delta_{i_j}^{start} := [0, l_P]$$

for each part i_j of a split task $P \in \mathcal{P}$. Note that several parts of a split task can be superposed, representing only one effective split part. Only the last part of such a task chain will then have a

Name	Mathematical variable	CP domain
start time	T_i	δ_i^{start}
split task part length	—	δ_i^{sl}
running predecessor	$pred_i$	π_i
running successor	$succ_i$	σ_i
shift assignment	S_i	δ_i^{shift}
predecessor task	$pred_i$	δ_i^{pred}
inserted flag	—	δ_i^{ins}

Table 5.2.: Overview of task-related variables in the CP model.

positive length. The last split part i_m of a split task P will therefore always have a positive length, and its length variable is initialised to

$$\delta_{i_m}^{start} := [msl_P, l_P]$$

As an insertion-based model is used, we will keep track of current predecessors and successors in a variable π_i for each task $i \in I \cup I^b \cup I^d$ and the current successor σ_i for each task $i \in I \cup I^b \cup I^o$. π_i and σ_i are not constraint variables in the proper sense as they contain single values which are adapted with insertions (see also Section 4.5). With a given initial shift plan, the initialisation of predecessor and successor variables is straightforward. For empty shifts, origin, break and destination task build a chain of predecessor and successor tasks.

For each task $i \in I \cup I^b \cup I^o \cup I^d$, the set of shifts to which it may be assigned is maintained in a domain δ_i^{shift}. While for the starting solution the shift assignment is given, we will show in Section 5.9.3 how the set of shifts is reinitialised within local search. Clearly, the shift domain is fixed for origin, destination and break tasks.

Additionally to the shift assignment, we keep track of potential predecessor tasks in a domain δ_i^{pred}. While the current predecessor task π_i denotes the unique current predecessor after all insertions made so far, the predecessor variable δ_i^{pred} specifies all tasks after which i may be inserted in a shift (if i is not yet inserted) or all tasks which may still be inserted in front of i (if i was already inserted). The predecessor domains will be reinitialised for each local step.

The boolean domain δ_i^{ins} will indicate if i is inserted into a shift. Origin, destination and break tasks are always inserted ($\delta_i^{ins} := \{\text{true}\}$) while domains of work tasks are initialised to $\delta_i^{ins} := \{\text{false}, \text{true}\}$ when their assignment is relaxed in a local step.

All task-related domains with their equivalents in the mathematical model are given in Table 5.2.

Turning to shift-related domains, δ_s^K will specify the shift types which may be attributed to a shift s; note that the day is fixed for each shift template. Initially, all potential shift types of brd_s are part of the shift type domain:

$$\delta_s^K := K_{brd_s}$$

The shift type of dummy shifts s_i^{dummy} is always fixed to the dummy shift type which corresponds to i's length:

$$\delta_{s_i^{dummy}}^K := \{k_{l_i}^{dummy}\}$$

Since shift costs only arise for shifts containing work tasks $i \in I$, we keep track of used shift templates s via boolean domains δ_s^{used}. As shift templates are initially empty, we initialise $\delta_s^{used} := \{\text{false}, \text{true}\}$. As soon as a work task $i \in I$ is inserted into the shift, the value false will be removed from the domain.

Name	Mathematical variable	CP domain
shift type	K_s	δ_s^K
shift used flag	—	δ_s^{used}
qualifications	Q_s	δ_s^Q
qualification penalty	$qp(Q_s)^{-1}$	δ_s^{qp}

Table 5.3.: Overview of shift-related variables in the CP model.

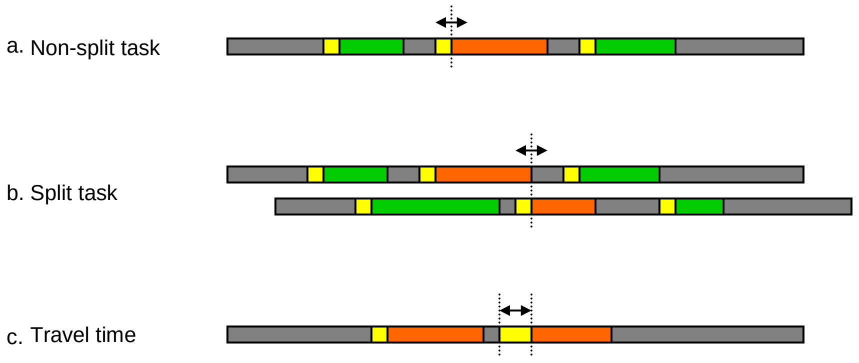


Figure 5.6.: Temporal propagation for split and non-split tasks and for travel times.

The qualification domain δ_s^Q describes the set of qualification requirements for each shift $s \in S$. Since CP domains shrink monotonically and qualification requirements augment with each task insertion, δ_s^Q will represent the inverse of the qualification requirements, i.e. $Q \setminus Q_s$. Since empty shifts do not have qualification requirements, δ_s^Q is initialised to the set Q of all qualifications.

A further domain δ_s^{qp} is introduced for the calculation of qualification penalties according to objective function term (5.2). The domain is represented as a range between the actual penalty $(qp(Q_s)^{-1})$ and the maximum value qp_{min}^{-1} with qp_{min} defined as in Section 5.3. Because $qp(\emptyset) = 1$, we initialise δ_s^{qp} to

$$\delta_s^{qp} = [1, qp_{min}^{-1}]$$

for empty shifts. Additionally to calculating actual qualification penalties, δ_s^{qp} will be used to cause a search failure when a task insertion results incompatible qualifications in a shift. The decision is then automatically revised (cf. Section 5.6.3).

All shift-related domains are summarised in Table 5.3.

5.6.2. Shift Temporal Constraints

Predecessor and successor relationships define temporal relations between subsequent tasks in shifts. We will now define rules for updating start time windows upon changes of the temporal information of other tasks. We will use push-forward/push-backward propagation rules which allow for the update of time windows by a linear forward pass for earliest start times and a linear backward pass for latest start times. Checks for possible insertion positions can then be done in constant time, see also Section 3.3.1.

Due to split parts, the temporal propagation will be more complicated than in the CP model for the workload levelling problem. According to Fig. 5.6, the following cases must be distinguished:

- The time window of a non-split task i is updated; such an update is typically due to a task insertion (orange task in Fig. 5.6a). We must propagate this information forward to the successor task σ_i and backward to the predecessor task π_i in the shift.

- The time window of a split part i is restricted (Fig. 5.6b). Note that the time window δ_i^{start} of a split part denotes the start time window of one task (orange task in lower shift) as well as the end time of its split part predecessor $spred_i$ (orange task in upper shift). Consequently, we must propagate the temporal update backward to the predecessor π_i of task i and forward to the successor σ_{spred_i} of its split part predecessor $spred_i$. A further constraint between the start and end time windows of a split part will guarantee that minimum split part lengths are observed (see below).

- Upon each travel time change, this means that the minimum distance between two subsequent task may have increased (Fig. 5.6c). This must be propagated forward from the first to the second task and backward from the second to the first. Since breaks take place at the location of their predecessor work task, we must pay special attention to tasks after breaks whose travel times can change when a task is inserted before the break.

We will define building blocks for start time propagation in these different cases. The forward propagation rule *FWDPROP*(i_1, i_2) describes the earliest start time propagation from a task i_1 (non-split task or split part) to the earliest start time of i_2. Symmetrically, *BWDPROP*(i_1, i_2) will propagate changes of the latest start time of task i_2 to the latest end time of task i_1 (again, i_1 may be split part or not).

As mentioned before, we must be careful with the interaction of breaks and travel times. If the direct predecessor of a break task is another break task, the determination of travel times can mean a recourse to a work task over several breaks. When a task is inserted into a shift, not only its own travel time is changed, but also the travel time of the first successor task which is not a break. For travel time propagation, we define a first work task predecessor $wpred(i)$ and a first work task successor $wsucc(i)$ as

$$wpred(i) := \begin{cases} \pi_i & \text{if } \pi_i \in I \cup I^o \\ wpred(\pi_i) & \text{if } \pi_i \in I^b \end{cases}$$

$$wsucc(i) := \begin{cases} \sigma_i & \text{if } \succ_i \in I \cup I^d \\ wsucc(\sigma_i) & \text{if } \succ_i \in I^b \end{cases}$$

Furthermore, we have to take into account that the minimum distance between subsequent tasks i_1 and i_2 can be relaxed by an overlap which is limited by $\min(ol_{i_1}^{max}, ol_{i_2}^{max})$. The forward propagation *FWDPROP*(i_1, i_2) for tasks $i_1 \in I \cup I^b \cup I^o$ and $i_2 \in I \cup I^b \cup I^d$ is thus given by

$$\begin{aligned}&\textit{FWDPROP}(i_1, i_2) : P_{i_1} = \textit{NIL} \vee P_{i_2} = \textit{NIL} \vee P_{i_1} \neq P_{i_2} \Longrightarrow \\ &\begin{bmatrix} t := \alpha_{i_1} + l_{i_1} + d_{wpred(i_2),i_2} - \min(ol_{i_1}^{max}, ol_{i_2}^{max}) > \alpha_{i_2} \Longrightarrow \alpha_{i_2} := t \text{ if } P_{i_1} = \textit{NIL} \\ t := \alpha_{ssucc_{i_1}} + d_{wpred(i_2),i_2} - \min(ol_{i_1}^{max}, ol_{i_2}^{max}) > \alpha_{i_2} \Longrightarrow \alpha_{i_2} := t \text{ if } P_{i_1} \neq \textit{NIL} \end{bmatrix}\end{aligned}$$

The backward propagation *BWDPROP*(i_1, i_2) of latest end times is defined symmetrically:

$$\begin{aligned}&\textit{BWDPROP}(i_1, i_2) : P_{i_1} = \textit{NIL} \vee P_{i_2} = \textit{NIL} \vee P_{i_1} \neq P_{i_2} \Longrightarrow \\ &\begin{bmatrix} t := \beta_{i_2} - d_{wpred(i_2),i_2} - l_{i_1} + \min(ol_{i_1}^{max}, ol_{i_2}^{max}) < \beta_{i_1} \Longrightarrow \beta_{i_1} := t \qquad \text{if } P_{i_1} = \textit{NIL} \\ t := \beta_{i_2} - d_{wpred(i_2),i_2} + \min(ol_{i_1}^{max}, ol_{i_2}^{max}) < \beta_{ssucc_{i_1}} \Longrightarrow \beta_{ssucc_{i_1}} := t \text{ if } P_{i_1} \neq \textit{NIL} \end{bmatrix}\end{aligned}$$

Some comments are in place to understand these building blocks. As a first observation, both *FWDPROP*(i_1, i_2) and *BWDPROP*(i_1, i_2) will entail updates only if not both i_1 and i_2 are split parts of the same split task $P_{i_1} = P_{i_2}$. Otherwise, i_1 and i_2 are subsequent split parts of the same split task, and other consistency tests will apply.

The propagation rules describe how temporal information is transferred between the end of i_1 and the start of i_2. While the time window $\delta_{i_2}^{start}$ always describes the domain of start times for

i_2, we have to make a distinction for the determination of i_1's end time. If i_1 is a non-split task ($P_i = \mathit{NIL}$), the range of end times can be calculated by adding the length l_{i_1} to the start time window $\delta^{start}_{i_1} = [\alpha_{i_1}, \beta_{i_1}]$. If however i_1 is a split part ($P_{i_1} \neq \mathit{NIL}$), the range of end times is retrieved from $\delta^{start}_{ssucc_{i_1}}$.

Note also that a travel time $d_{wpred(i_2),i_2}$ is used in the propagation rules. Effectively, i_1 may be a break task, meaning that it inherits the location of the first work task predecessor. If i_2 is a break task, $d_{wpred(i_2),i_2}$ is 0 by definition.

We can now come to the description of consistency tests for the different settings of Fig. 5.6.

Non-split task temporal relation For each non-split task $i \in I \cup I^b \cup I^o \cup I^d$ ($P_i = \mathit{NIL}$), start times are updated as follows:

$$\begin{aligned} i \notin I^o, \pi_i \neq \mathit{NIL} &\Longrightarrow \mathit{BWDPROP}(\pi_i, i) \\ i \notin I^d, \sigma_i \neq \mathit{NIL} &\Longrightarrow \mathit{FWDPROP}(i, \sigma_i) \end{aligned}$$

This consistency test will be performed each time the start time domain δ^{start}_i, the predecessor π_i or the successor σ_i change. For each i, the time complexity is in $\mathcal{O}(1)$.

Split part temporal relation For each split part $i \in I$ ($P_i \neq \mathit{NIL}$), start times are adapted by

$$\begin{aligned} \pi_i \neq \mathit{NIL} &\Longrightarrow \mathit{BWDPROP}(\pi_i, i) \\ spred_i \neq \mathit{NIL}, \sigma_{spred_i} \neq \mathit{NIL} &\Longrightarrow \mathit{FWDPROP}(spred_i, \sigma_{spred_i}) \end{aligned}$$

Latest start time are propagated backward as for non-split tasks. Forward propagation however affects the split part predecessor and its shift successor if i is not the first part of its split task. The rule triggers upon reductions of δ^{start}_i as well as on changes of i's predecessor π_i and $spred_i$'s successor σ_{spred_i}, executing in $\mathcal{O}(1)$ runtime for each i.

When a task i is inserted, the temporal information between i and its predecessor and successor is already completely propagated by the consistency tests described before. If breaks are involved, we additionally need the following update rule.

Travel time propagation For each break task $i^b \in I^b$, surrounding start times are adapted by

$$\pi_{i^b} \neq \mathit{NIL} \Longrightarrow \begin{cases} \mathit{FWDPROP}(wpred(i^b), wsucc(i^b)) \\ \mathit{BWDPROP}(wpred(i^b), wsucc(i^b)) \end{cases}$$

This test clearly triggers on changes of π_{i^b}. As for the tests above, the time complexity is in $\mathcal{O}(1)$.

The temporal propagation rules described so far reflect the shift temporal constraints (5.8).

5.6.3. Shift Constraints

In the course of reoptimisation, task insertions will entail restrictions of the temporal domains $\delta^{start}_{i^o_s}$ and $\delta^{start}_{i^d_s}$ of origin and destination tasks, respectively. Clearly, this also restricts the set of valid shift types which is maintained in domains δ^K_s. Vice versa, earliest and latest start times of origin and destination tasks can be updated using the set of valid shift types.

Shift types For a shift $s \in S$, the set δ_s^K of valid shift types and the shift boundaries given by $\delta_{i_s^o}^{start}$ and $\delta_{i_s^d}^{start}$ will be updated as follows:

$$K_s^{valid} := \{k \in \delta_s^K \mid \alpha_{i_s^o} \leq st_{kn_s} \leq \beta_{i_s^o} \wedge \alpha_{i_s^d} \leq et_{kn_s} \leq \beta_{i_s^d}\} \Longrightarrow \\ \begin{bmatrix} \delta_s^K := K_s^{valid} \\ \min\limits_{k \in K_s^{valid}} st_{kn_s} > \alpha_{i_s^o} \Longrightarrow \alpha_{i_s^o} := \min\limits_{k \in K_s^{valid}} st_{kn_s} \\ \max\limits_{k \in K_s^{valid}} st_{kn_s} < \beta_{i_s^o} \Longrightarrow \beta_{i_s^o} := \max\limits_{k \in K_s^{valid}} st_{kn_s} \\ \min\limits_{k \in K_s^{valid}} et_{kn_s} > \alpha_{i_s^d} \Longrightarrow \alpha_{i_s^d} := \min\limits_{k \in K_s^{valid}} et_{kn_s} \\ \max\limits_{k \in K_s^{valid}} et_{kn_s} < \beta_{i_s^d} \Longrightarrow \beta_{i_s^d} := \max\limits_{k \in K_s^{valid}} et_{kn_s} \end{bmatrix}$$

The test is triggered upon changes of δ_s^K, $\delta_{i_s^o}^{start}$, $\delta_{i_s^d}^{start}$. Its complexity is in $\mathcal{O}(|K|)$.

Because shift templates do not have fixed shift types, the start time domains $\delta_{i^b}^{start}$ of break tasks $i^b \in I^b$ were initialised to rather unspecific values (cf. Section 5.6.1). In the course of the algorithm, break start time windows will be adapted with further information on valid shift types becoming available. In the opposite direction, task insertions around breaks generally cause changes to their start time domains which can also affect valid shift types. As break rules define break windows with reference to the shift start, the following break window constraints involve the start time variables of origin and break tasks.

Break windows Let $s \in S$ be a shift of break rule day brd_s. For each break rule $br \in BR_{brd_s}$, the following updates are applied to the start time windows of the origin task i_s^o and the break task i^b corresponding to br, using the break time window $[a_{br}, b_{br}]$ given by br relative to the shift start:

$$\begin{aligned} \alpha_{i_s^o} + a_{br} > \alpha_{i^b} &\Longrightarrow \alpha_{i^b} := \alpha_{i_s^o} + a_{br} \\ \beta_{i_s^o} + b_{br} < \beta_{i^b} &\Longrightarrow \beta_{i^b} := \beta_{i_s^o} + b_{br} \\ \alpha_{i^b} - b_{br} > \alpha_{i_s^o} &\Longrightarrow \alpha_{i_s^o} := \alpha_{i^b} - b_{br} \\ \beta_{i^b} - a_{br} < \beta_{i_s^o} &\Longrightarrow \beta_{i_s^o} := \beta_{i^b} - a_{br} \end{aligned}$$

The update triggers on changes of $\delta_{i_s^o}^{start}$ and $\delta_{i^b}^{start}$, usually caused by some task insertion. The test runs in constant time for each break.

Furthermore, minimum buffer times may be defined between early and main or main and late break. These can be formulated as consistency tests between the start time domains of the involved break tasks. The main difference between break window and break buffer constraints is that the former limits minimum and maximum distances while the latter only defines minimum buffer times.

Break buffers Let $s \in S$ be a shift with early, main and late break, i.e.

$$BR_{brd_s} = \{br_s^{eb}, br_s^{mb}, br_s^{lb}\}$$

and let bb_{br}^{eb} and bb_{br}^{lb} be the early and late break buffer times defined by the main break rule $br := br_s^{mb}$. Then the following reduction rules apply to the start time domains $\delta_{i_s^{eb}}^{start} = [\alpha_{i_s^{eb}}, \beta_{i_s^{eb}}]$, $\delta_{i_s^{mb}}^{start} = [\alpha_{i_s^{mb}}, \beta_{i_s^{mb}}]$ and $\delta_{i_s^{lb}}^{start} = [\alpha_{i_s^{lb}}, \beta_{i_s^{lb}}]$ of the break tasks:

$$\begin{aligned} \alpha_{i_s^{eb}} + l_{i_s^{eb}} + bb_{br}^{eb} > \alpha_{i_s^{mb}} &\Longrightarrow \alpha_{i_s^{mb}} := \alpha_{i_s^{eb}} + l_{i_s^{eb}} + bb_{br}^{eb} \\ \beta_{i_s^{mb}} - bb_{br}^{eb} - l_{i_s^{eb}} < \beta_{i_s^{eb}} &\Longrightarrow \beta_{i_s^{eb}} := \beta_{i_s^{mb}} - bb_{br}^{eb} - l_{i_s^{eb}} \\ \alpha_{i_s^{mb}} + l_{i_s^{mb}} + bb_{br}^{lb} > \alpha_{i_s^{lb}} &\Longrightarrow \alpha_{i_s^{lb}} := \alpha_{i_s^{mb}} + l_{i_s^{mb}} + bb_{br}^{lb} \\ \beta_{i_s^{lb}} - bb_{br}^{lb} - l_{i_s^{mb}} < \beta_{i_s^{mb}} &\Longrightarrow \beta_{i_s^{mb}} := \beta_{i_s^{lb}} - bb_{br}^{lb} - l_{i_s^{mb}} \end{aligned}$$

If we only have an early *or* a late break, only one pair of reduction rules is used. The test is triggered upon changes of the involved start time domains and entails an $\mathcal{O}(1)$ runtime for each shift.

This consistency test implements the break buffer constraints (5.12) of the mathematical model. As described above, the qualification penalties are calculated via domains δ_s^{qp}. This variable will be updated upon each change of the inverse qualification profile δ_s^{qp} for each shift s.

Qualification combinations Let $s \in S$ be a shift and qp_{min} as defined in Section 5.3. The qualification penalty domain δ_s^{qp} is recalculated from δ_s^Q as follows:

$$\delta_s^{qp} := \begin{cases} \delta_s^{qp} \backslash]-\infty, qp(Q \setminus \delta_s^Q)^{-1}[& \text{if } qp(Q \setminus \delta_s^Q) > 0 \text{ and } |Q \setminus \delta_s^Q| \leq q^{max} \\ \delta_s^{qp} \backslash]-\infty, qp_{min}^{-1} + 1[& \text{else} \end{cases}$$

With an appropriate representation of qualification sets, the test executes in constant runtime.

The rule basically calculates qualification penalties as defined by the objective function term (5.2). If the shift's qualification profile $Q \setminus \delta_s^Q$ contains incompatible qualifications (i.e. $qp(Q \setminus \delta_s^Q) = 0$) or if the size of the qualification profile exceeds the limit q^{max}, the new minimum of δ_s^{qp} is set to $qp_{min}^{-1} + 1$. Because the maximum of the qualification penalty domain δ_s^{qp} was initialised to qp_{min}^{-1} (cf. Section 5.6.1), δ_s^{qp} will then become empty. Consequently, if a task insertion entails combining incompatible qualifications or exceeding the limit for the qualification profile size, the state of the constraint model becomes invalid. In constraint programming, invalid assignments are discarded and entail backtracking, i.e. the insertion decision causing failure will be revised. The consistency test thus additionally assures that constraints (5.22) and (5.23) are observed.

Note that the propagation of incompatible qualifications and maximal qualification profile sizes could be stronger by adapting shift domains of uninserted tasks. However, this would mean costly checks of potential qualification profiles. Furthermore, consistency tests triggering on many events would have to be used. Because in practice, incompatible qualifications and qualification profile sizes are not very limiting, it was decided to use less costly a-posteriori checks as described above.

5.6.4. Crew Constraints

Additionally to the horizontal relations between subsequent tasks in shifts, crews define vertical dependencies between tasks in different shifts. These can be efficiently represented if crew tasks are sorted by length, cf. Section 4.3. Additionally to crew tasks, the breaks of a shift crew must be placed in parallel. In fact, temporal restrictions on crew tasks and breaks are equivalent if breaks have equal lengths and earliest start times, cf. to constraints (5.9) and (5.18) in the mathematical model. But since the shifts of a shift crew have equal shift types, their break rules will be identical. This means that their lengths are in fact equal. Furthermore, since break start times were initialised using break rule days (cf. Section 5.6.1), their initial start times also correspond. We can thus use the same consistency test for the implementation of constraints (5.9) and (5.18).

Crew temporal relation Let

$$C = (i_1, \dots, i_m) \in \mathcal{C} \cup \bigcup_{\{s_1,\dots,s_n\} \in \mathcal{H}} \left\{ (i_{s_1}^{eb}, \dots, i_{s_n}^{eb}), (i_{s_1}^{mb}, \dots, i_{s_n}^{mb}), (i_{s_1}^{lb}, \dots, i_{s_n}^{lb}) \right\}$$

be a tuple of crew tasks or break tasks of crew shift, and let $i_j, i_{j+1} \in C$ be adjacent tasks in the sorting of C by decreasing lengths ($l_{i_j} \geq l_{i_{j+1}}$). Then the following domain reduction rules apply for $1 \leq j < n$:

$$\begin{aligned}
\alpha_{i_j} + (a_{i_{j+1}} - a_{i_j}) > \alpha_{i_{j+1}} &\Longrightarrow \\
&\alpha_{i_{j+1}} := \alpha_{i_j} + (a_{i_{j+1}} - a_{i_j}) \\
\beta_{i_{j+1}} - (a_{i_{j+1}} - a_{i_j}) < \beta_{i_j} &\Longrightarrow \\
&\beta_{i_j} := \beta_{i_{j+1}} - (a_{i_{j+1}} - a_{i_j}) \\
\alpha_{i_j} - (l_{i_{j+1}} - l_{i_j}) + (a_{i_{j+1}} - a_{i_j}) > \alpha_{i_{j+1}} &\Longrightarrow \\
&\alpha_{i_{j+1}} := \alpha_{i_j} - (l_{i_{j+1}} - l_{i_j}) + (a_{i_{j+1}} - a_{i_j}) \\
\beta_{i_{j+1}} + (l_{i_{j+1}} - l_{i_j}) - (a_{i_{j+1}} - a_{i_j}) < \beta_{i_j} &\Longrightarrow \\
&\beta_{i_j} := \beta_{i_{j+1}} + (l_{i_{j+1}} - l_{i_j}) - (a_{i_{j+1}} - a_{i_j})
\end{aligned}$$

If a shift crew does not contain a full set of early, main and late breaks, the update is only performed for the breaks which are defined by the break rules. The test is performed upon each change of the involved variables $\delta^{start}_{i_j}$ and $\delta^{start}_{i_{j+1}}$ and has $\mathcal{O}(1)$ time complexity for each pair $(j, j+1)$.

Additionally to the temporal constraints, there are further restrictions on crew tasks. The following test implements the above constraint (5.15).

Different shifts The tasks of each crew $C \in \mathcal{C}$ must be placed in different shifts. This is enforced by an *alldifferent*$(\{\delta^{shift}_i \,|\, i \in C\})$ global constraint of complexity $\mathcal{O}(|C|^2|S|^2)$ for a whole branch of the search [Régin, 1994] [Régin, 2000].

Furthermore, all tasks of a subcrew $B \in \mathcal{B}$ must be assigned to the same shift crew $H \in \mathcal{H}$ (constraint 5.16). This is assured by the following consistency test:

Subcrew attribution Let $B \in \mathcal{B}$ be a subcrew and

$$S(B) := \{s \in S \,|\, \exists i \in B : \delta^{shift}_i = \{s\}\}$$

the set of regular shifts to which tasks of B are already assigned. If $s' \in S(B) \neq \emptyset$, we can enforce the assignment of all tasks $i \in B$ to the same shift crew by

$$|\delta^{shift}_i| > 1 \Longrightarrow \delta^{shift}_i := \delta^{shift}_i \cap H_{s'}$$

It should be noted that the set $S(B)$ will not include dummy shifts s^{dummy}_i. Because the consistency test is performed each time the shift domain of a subcrew task is bound to a value, the set $S(B)$ will only contain shifts of the same shift crew. We can therefore use an arbitrary shift s' to determine this shift crew $H_{s'}$ and prevent all other tasks of the subcrew to be assigned to shifts which are not part of $H_{s'}$.

With an appropriate implementation of set operations, the test consumes $(|B|)$ runtime for each subcrew $B \in \mathcal{B}$.

By definition, all shifts of a shift crew have identical break rule days. But we must also ensure that all crew shifts finally have same shift types (constraint (5.17)). We may update all shift type domains each time one of them changes, but this would amount to costly set operations. Instead, we note that a shift type $k \in K$ can be uniquely determined from given start and end times $st = st_{kn}$ and $et = et_{kn}$ for a fixed day n and its set of break rules. Since the set of break rules of a shift template s follows from its break rule day brd_s, we can achieve full consistency of shift types by synchronisation of the start time domains $\delta^{start}_{i^o_s}$ and $\delta^{start}_{i^d_s}$.

Shift start times For each shift crew $H \in \mathcal{H}$, the start time domain $\delta_{i_s^o}^{start}$ for each $s \in H$ are updated as follows:

$$\begin{aligned}\max_{s' \in H} \alpha_{i_{s'}^o} > \alpha_{i_s^o} &\Longrightarrow \alpha_{i_s^o} := \max_{s' \in H} \alpha_{i_{s'}^o} \\ \min_{s' \in H} \beta_{i_{s'}^o} < \beta_{i_s^o} &\Longrightarrow \beta_{i_s^o} := \min_{s' \in H} \beta_{i_{s'}^o}\end{aligned}$$

The test triggers on changes of $\delta_{i_s^o}^{start}$.

Shift end times For a shift crew $H \in \mathcal{H}$ and each shift $s \in H$, we update the temporal domains of the destination tasks i_s^d as follows:

$$\begin{aligned}\max_{s' \in H} \alpha_{i_{s'}^d} > \alpha_{i_s^d} &\Longrightarrow \alpha_{i_s^d} := \max_{s' \in H} \alpha_{i_{s'}^d} \\ \min_{s' \in H} \beta_{i_{s'}^d} < \beta_{i_s^d} &\Longrightarrow \beta_{i_s^d} := \min_{s' \in H} \beta_{i_{s'}^d}\end{aligned}$$

The rule triggers on changes of the $\delta_{i_s^d}^{start}$. Both tests have $\mathcal{O}(|H|)$ runtime complexity.

In conjunction with the shift type consistency test described above, these tests ensure that the domains δ_s^K of all shifts in H correspond.

For shift cost accounting, shift templates are marked as used via a domain δ_s^{used} as soon as a task is inserted. In crew handling, we must consider that a shift crew can only be used as a unit. We will therefore mark all crew shifts as used as soon as a task is inserted into one of them.

Shift crew accounting Let $H \in \mathcal{H}$ be a shift crew. The following update rule is used to account for full shift crews:

$$\exists s \in H : \delta_s^{used} = \{\text{true}\} \Longrightarrow delta_{s'}^{used} := \{\text{true}\} \; \forall s' \in H$$

This update is executed each time one of the flag variables δ_s^{used} is set to $\{\text{true}\}$, consuming $\mathcal{O}(|H|)$ runtime.

5.6.5. Split Task Constraints

Split parts require special attention since their lengths are variable. Several split parts can be placed one behind the other in a shift, representing only one effective split part. If we do not impose further restrictions on the successive placement, several states of the model will represent the same logical situation since redundant split parts can be piled up in different ways. Such symmetries should be avoided since they inflate the search space.

We will therefore restrict the succession of split parts to the end of a split task $P = (i_1, \ldots, i_m)$, i.e. if two parts i_j and i_{j+1} are direct successors in a shift and no further task will be placed between them, we enforce that all tasks $i_{j'}$ with $j' > i$ succeed. Vice versa, if two parts cannot be placed one behind the other, none of the tasks $i_{j'}$ with $j' < i$ will be placed as successors in shifts.

Shift succession of split parts interacts with their lengths: If we know that two parts are placed one behind the other, the first part is restricted to a length of 0. If the tasks cannot succeed, the minimum split length msl_P has to be obeyed. These two aspects are represented by the following consistency test, triggering on changes of the predecessor domain δ_i^{pred}:

Split part succession Let $P = \{i_1, \ldots, i_m\}$ a split task and $i \in P$, $i \neq i_1$, one of its split parts. The following update rules apply:

$$\begin{array}{l}
\delta_i^{pred} = \{spred_i\} \Longrightarrow \\
\begin{bmatrix} ssucc_i \neq i_P^{pseudo} \Rightarrow \delta_{ssucc_i}^{pred} := \{i\} \\ \delta_{spred_i}^{sl} := \delta_{spred_i}^{sl} \backslash]0, \infty[\end{bmatrix} \\
spred_i \notin \delta_i^{pred} \Longrightarrow \\
\begin{bmatrix} spred_{spred_i} \neq \mathit{NIL} \Rightarrow \delta_{spred_i}^{pred} := \delta_{spred_i}^{pred} \setminus \{spred_{spred_i}\} \\ \delta_{spred_i}^{sl} := \delta_{spred_i}^{sl} \backslash] - \infty, msl_P[\end{bmatrix}
\end{array}$$

The test is executed upon changes of δ_i^{pred} and has constant runtime.

If the predecessor domains $\delta_{spred_i}^{pred}$ or $\delta_{ssucc_i}^{pred}$ are updated by the split part succession rule, this will cause the execution of the same consistency test for $spred_i$ or $ssucc_i$, respectively. Therefore, decisions for shift successions are propagated backward while decisions for different shifts will be propagated toward the end of a split task.

Symmetrically to the split part succession rule, a further test is devised which triggers on changes of the length domain δ_i^{sl}: If the minimum in this domain is greater than 0, we know that we cannot superpose i with its split successor $ssucc_i$ in the same shift. Vice versa, if the maximum length falls below msl_{P_i}, $ssucc_i$ must be placed behind i.

Split part length Let $P = \{i_1, \ldots, i_m\} \in \mathcal{P}$ be a split task and $i \in P$, $i \neq i_m$, one of its split parts. Using the values $\delta_i^{sl} = [sl_i^{min}, sl_i^{max}\}$ of i's length domain, the following updates are performed:

$$\begin{aligned}
sl_i^{min} > 0 &\Longrightarrow \delta_{ssucc_i}^{pred} := \delta_{ssucc_i}^{pred} \setminus \{i\} \\
sl_i^{max} < msl_P &\Longrightarrow \delta_{ssucc_i}^{pred} := \{i\}
\end{aligned}$$

The test is triggered upon changes of δ_i^{sl} and has $\mathcal{O}(1)$ runtime.

The split part length is synchronised with the temporal domains:

Split part length synchronisation For a split part $i \in I$, the following reduction rules synchronise the split part length $\delta_i^{sl} = [sl_i^{min}, sl_i^{max}]$ with the temporal domains δ_i^{start} and $\delta_{ssucc_i}^{start}$ of i and its split part successor $ssucc_i$:

$$\begin{aligned}
\alpha_i + sl_i^{min} > \alpha_{ssucc_i} &\Longrightarrow \alpha_{ssucc_i} := \alpha_i + sl_i^{min} \\
\beta_i + sl_i^{max} < \beta_{ssucc_i} &\Longrightarrow \beta_{ssucc_i} := \beta_i + sl_i^{max} \\
\alpha_{ssucc_i} - sl_i^{max} > \alpha_i &\Longrightarrow \alpha_i := \alpha_{ssucc_i} - sl_i^{max} \\
\beta_{ssucc_i} - sl_i^{min} < \beta_i &\Longrightarrow \beta_i := \beta_{ssucc_i} - sl_i^{min} \\
\alpha_{ssucc_i} - \beta_i > sl_i^{min} &\Longrightarrow sl_i^{min} := \alpha_{ssucc_i} - \beta_i \\
\beta_{ssucc_i} - \alpha_i < sl_i^{max} &\Longrightarrow sl_i^{max} := \beta_{ssucc_i} - \alpha_i
\end{aligned}$$

The consistency test triggers changes of δ_i^{start}, $\delta_{ssucc_i}^{start}$ and δ_i^{sl} and has $\mathcal{O}(1)$ complexity.

The combination of the split part succession test and the length synchronisation rule guarantees that minimum lengths according to constraint (5.20) are observed.

Additionally to synchronising split part lengths, we must guarantee that the lengths of all split parts sum up to the total length l_P of the split task $P = (i_1, \ldots, i_m)$. This is realised by a temporal constraint between the start time domains of the first and the pseudo split part:

Split task length Let $P = (i_1, \dots, i_m) \in \mathcal{P}$ be a split task, i_1 its first split part and i_P^{pseudo} the delimiting pseudo split part. The following test expresses that P must observe a total length of l_P:

$$\begin{aligned}
\alpha_{i_1} + l_P > \alpha_{i_P^{pseudo}} &\Longrightarrow \alpha_{i_P^{pseudo}} := \alpha_{i_1} + l_P \\
\beta_{i_1} + l_P < \beta_{i_P^{pseudo}} &\Longrightarrow \beta_{i_P^{pseudo}} := \beta_{i_1} + l_P \\
\alpha_{i_P^{pseudo}} - l_P > \alpha_{i_1} &\Longrightarrow \alpha_{i_1} := \alpha_{i_P^{pseudo}} - l_P \\
\beta_{i_P^{pseudo}} - l_P < \beta_{i_1} &\Longrightarrow \beta_{i_1} := \beta_{i_P^{pseudo}} - l_P
\end{aligned}$$

The test triggers on changes of $\delta_{i_1}^{start}$ and $\delta_{i_P^{pseudo}}^{start}$ and has $\mathcal{O}(1)$ runtime complexity.

5.6.6. Insertion Position Constraints

Within large neighbourhood search, we will repeatedly relax the insertion decisions for a set $I^{rel} \subseteq I$ of tasks. For each $i \in I^{rel}$, the possible insertion positions $\Phi(i)$ will be determined. Let $\Phi := \bigcup_{i \in I^{rel}} \Phi(i)$ be the set of all insertion positions for the relaxed tasks. The complexity of the following consistency tests will be given with reference to the figures $|I^{rel}|$ and $|\Phi|$. Section 5.9.3 will show how the sets $\Phi(i)$ are determined.

In the constraint model, insertion positions are maintained on the task level (δ_i^{pred}) and as shifts (δ_i^{shift}). Different constraints apply to the two levels both of which are coordinated by the following rule.

Shift-predecessor consistency For each task $i \in I$, the predecessor and shift domains δ_i^{pred} and δ_i^{shift} are synchronised as follows:

$$\begin{aligned}
\delta_i^{pred} &:= \delta_i^{pred} \setminus \{j \in \delta_i^{pred} \mid \delta_j^{shift} = \{s\} \wedge s \notin \delta_i^{shift}\} \\
\delta_i^{shift} &:= \{s \in \delta_i^{shift} \mid \exists j \in \delta_i^{pred} : \delta_j^{ins} = \{\text{true}\} \wedge \delta_j^{shift} = \{s\}\}
\end{aligned}$$

From δ_i^{pred}, we remove all inserted tasks which are assigned to shifts which are not valid for i. Furthermore, a shift in δ_i^{shift} cannot be used to cover i if there is none of its tasks is part of δ_i^{pred}. Note that the rule could be made somewhat stronger by deleting inserted predecessors j with $\delta_i^{shift} \cap \delta_j^{shift} = \emptyset$, but this would involve more costly set operations.

With careful implementation, the test has $\mathcal{O}(|\Phi|)$ time complexity.

A central part of the propagation logic is the decision to insert a task into a shift. The shift attribution δ_i^{shift} of a task i does not uniquely determine its position in the shift. We must therefore resolve the succession relationship via the predecessor domain δ_i^{pred}. The set of predecessors contains tasks which are already inserted as well as uninserted tasks. The exact insertion position of i is uniquely determined when δ_i^{pred} contains only one inserted task.

Task insertion Let $i \in I$ be a work task and

$$\Psi^{ins}(i) := \{i' \in \delta_i^{pred} \mid \delta_{i'}^{ins} = \{\text{true}\}\}$$

the set of inserted possible predecessors of i. For the insertion of i into a shift, we perform the following update:

$$\delta_i^{ins} \neq \{\text{true}\} \wedge \Psi^{ins}(i) = \{i^{pred}\} \wedge i^{succ} := \sigma_{i^{pred}} \wedge \delta_{i^{pred}}^{shift} = \{s\} \Longrightarrow$$
$$\left[\begin{array}{l} \delta_{i^{succ}}^{pred} := \delta_{i^{succ}}^{pred} \setminus \{i' \in \delta_{i^{succ}}^{pred} \,|\, i' \neq i \wedge \delta_{i'}^{ins} = \{\text{true}\}\} \\ \delta_i^{pred} := \delta_i^{pred} \setminus \{i' \in \delta_i^{pred} \,|\, i' \neq i^{pred} \wedge \delta_{i'}^{ins} = \{\text{true}\}\} \\ \delta_{i'}^{pred} := \delta_{i'}^{pred} \setminus \{i\} \; \forall i' \in I \cup I^b \cup I^o \cup I^d, i' \neq i^{succ}, \delta_{i'}^{ins} = \{\text{true}\}, |\delta_{i'}^{pred}| > 1 \\ \pi_i := i^{pred}, \sigma_i := i^{succ}, \pi_{i^{succ}} := i, \sigma_{i^{pred}} := i \\ \delta_i^{shift} := \{s\} \\ \delta_i^{ins} := \delta_i^{ins} \setminus \{\text{false}\} \\ \delta_s^{used} := \delta_s^{used} \setminus \{\text{false}\} \\ \delta_s^Q := \delta_s^Q \setminus Q_i \end{array}\right]$$

The test is triggers upon each a change of δ_i^{pred}. The cost of the update is in $\mathcal{O}(|I^{rel}| + |\Phi| + |Q|)$.

The set $\Psi^{ins}(i)$ of inserted predecessors of i will generally contain at least one task because otherwise, δ_i^{pred} would only contain tasks which are not inserted. But we cannot expect that i can be inserted after the insertion of one of its predecessors, say i', since then, δ_i^{pred} would have to contain at least one of the inserted predecessors of i'. Consequently $\Psi^{ins}(i) = \emptyset$ means that we cannot insert i at all.

The insertion rule performs a number of updates. It sets predecessor domains, adapts running predecessors and successors, sets the shift and inserted domains of the task as well as the shift's "used" domain and updates the qualification domain. Note that δ_s^Q contains the inverse of a shift's actual qualification profile, i.e. all qualifications which are *not* required by the shift.

In order to exclude all other insertion positions, the test traverses all inserted tasks with unbound predecessor variables which are not equal to the actual successor i^{succ}. To make this update more efficient, the set $\{i' \in I \cup I^b \cup I^o \cup I^d \,|\, |\delta_{i'}^{pred}| > 1\}$ can be maintained in a global constraint variable. Each time the predecessor domain $\delta_{i'}^{pred}$ of a task i' becomes bound to a value, the global domain will be updated by a simple consistency test.

Some comments on the constraint model are in place. First of all, the model uses an insertion-based logic, i.e. tasks are inserted one by one into an emerging shift plan. The current state of the insertions is maintained in the current predecessor and successor variables π_i and σ_i. The insertion test described before is responsible for setting these variables, triggering the temporal propagation within shifts. The efficient propagation of time windows by forward/backward passes is a main advantage of the model. Note that while the predecessor/successor variables are not constraint variables in the proper sense, they preserve monotonicity of temporal propagation. Because the triangle inequality is obeyed, the length and travel time of a newly inserted task sums up to a duration which is at least as long as the temporal distances accounted before insertion.

Possible insertion positions for tasks are maintained in predecessor and shift domains. In contrast to the model for the workload levelling problem, there are no intermediate checks for temporal validity of insertion positions. Instead, the search strategy will evaluate which of the insertion positions are valid with regard to temporal and other constraints (like compatibility of qualifications). This lower degree of consistency was accepted because refined models require more costly consistency tests triggering on many events. Preliminary tests showed that this does not generally pay off for the shift planning problem.

An explanation can be found by an analysis of possible effects of predecessor domain reductions. The best we can expect from a deletion of insertion positions is the insertion of further tasks. But this will only happen if a single inserted task remains in a predecessor domain. However, a task can generally be inserted in many places, including at least one empty shift template. As a consequence, we cannot expect substantial effects from additional consistency tests, and runtime seems to be better invested in the additional cost of a systematic search space exploration.

5.6.7. Avoiding Symmetry

Especially when planning scenarios comprise crews, there are often several tasks of identical characteristics. Every permutation of these tasks then results in an isomorphic solution. Such symmetries should be avoided to prevent the search from evaluating equivalent solutions (see also Section 4.5). These symmetries do not affect split parts which are variable in length. We therefore build groups of non-split tasks having identical

- time windows,
- lengths,
- task crews and subcrews (or are non-crew tasks),
- locations (i.e. travel times to surrounding tasks),
- maximum overlaps and
- qualification profiles.

It should be clear that all tasks of each such group are isomorphic. Partial orders are then imposed on the tasks of each group and all shifts. By a special consistency test, the tasks of a group are restricted to be assigned to shifts in increasing order, resolving the aforementioned symmetry.

5.6.8. Objective Function Calculation

We can now specify how the objective function value can be calculated from the constraint model. A given state (schedule) Σ of the model will represent a complete solution if

- All tasks are assigned to shifts (possibly dummy shifts), i.e. the domains δ_i^{shift} are bound to single values for all $i \in I$.
- The shift types of all used shifts are fixed, i.e.

  $$\delta_s^{used} = \{\text{true}\} \Rightarrow |\delta_s^K| = 1$$

  While in simple cases, we can always fix the shift type domains to the cheapest valid shift type, shift number constraints entail interdependencies, requiring an adequate search procedure for the determination of an overall best solution.
- The lengths of split parts which are assigned to their dummy shift are fixed, i.e. their length domain δ_i^{sl} is bound. Again, there are interdependencies between decisions for different tasks, and an easy fixing to the shortest possible lengths (avoiding penalties for unassigned task minutes) is not possible.

Note that the specification of a full solution does not require the fixing of start time variables δ_i^{start}, meaning that a search algorithm will only have to fix the above domains.

The objective value z of a complete schedule Σ will be the sum of five components:

$$z(\Sigma) := z^{SC}(\Sigma) + z^{UT}(\Sigma) + z^{QP}(\Sigma) + z^{QR}(\Sigma) + z^{SNR}(\Sigma)$$

z_{SC} is the shift cost component. Since in the CP model, we use possibly empty shift templates, only used shifts are considered, i.e. shifts for which $\delta_s^{used} = \{\text{true}\}$. In partial solutions, we can determine minimum shift costs using information on valid shift types:

$$z^{SC}(\Sigma) := \sum_{\substack{s \in S \\ \delta_s^{used} = \{\text{true}\}}} \min_{k \in \delta_s^K} c_k$$

Via costs of dummy shift types, $z^{SC}(\Sigma)$ already contains unassigned task penalties for non-split tasks (cf. Section 5.5). Meanwhile, unassigned split parts must be penalised separately. Their lengths are retrieved from the domains δ_i^{sl} which is bound in a complete solution. If sl_i^{min} is the minimum length (complete solution: effective length) of a split part i (minimum value of δ_i^{sl}), the unassigned split part penalty $z^{UT}(\Sigma)$ of a schedule Σ is

$$z^{UT}(\Sigma) := M^{UT} \cdot \sum_{\substack{i \in I : P_i \neq NIL \\ \delta_i^{shift} = \{s_{P_i}^{dummy}\}}} sl_i^{min}$$

Penalties for qualification combinations are summed up by $z^{QR}(\Sigma)$. As described above, the individual penalties for the shifts are maintained in domains δ_s^{qp}. Since empty shift templates account for penalties of 1, a correction term is used. We furthermore normalise qualification penalties by the sum of used shifts and the minimum qualification preference qp_{min}, i.e.

$$z^{QP}(\Sigma) := w^{QP} \frac{qp_{min}}{|\{s \in S \mid \delta_s^{used} = \{\text{true}\}\}|} \left(\sum_{s \in S} qp_s^{min} - |\{s \in S | \delta_s^{used} = \{\text{true}\}\}| \right)$$

where qp_s^{min} is the minimum value in the domain δ_s^{qp}.

The calculation of penalties for exceeding qualification restrictions is straightforward:

$$z^{QR}(\Sigma) := M^{QR} \cdot \sum_{qr \in QR} \max(|\{s \in S \mid q_{qr} \notin \delta_s^Q\}| - m_{qr}, 0)$$

In this term, only used shifts are considered because only then $Q \setminus \delta_s^Q$ possibly contains a qualification q_{qr}.

Shift number restrictions impose constraints on the composition of a shift plan. Minimum restrictions can always be observed by adding empty shifts to a solution. In the model, we can simply add the cost

$$c_{min}^K(r) := \min_{k \in K_r} c_k$$

of the cheapest shift type in the reference set K_r for each shortage. Meanwhile, maximum restrictions impose large penalties M^{SNR} for each extra shift. All shift number restrictions can therefore be represented as objective function components.

For penalty calculation, the actual number $ACT(r)$ of shifts for a shift restriction r is defined as:

$$ACT(r) := |\{s \in S \mid \delta_s^{used} = \{\text{true}\}, n_s \in N_r, \exists k \in K_r : \delta_s^K = \{k\}\}|$$

Relative shift number restrictions $r \in R_{rel}^{min} \cup R_{rel}^{max}$ define minimum and maximum proportions of shifts of types in K_r relative to all shifts on the respective days. To calculate penalties for shortage (minimum restrictions) and surplus shifts (maximum restrictions), we additionally need the number of all reference shifts $REF(r)$ on the affected days:

$$REF(r) := |\{s \in S \mid \delta_s^{used} = \{\text{true}\}, n_s \in N_r\}|$$

When a shift plan is filled up in order to observe a relative minimum restriction $r \in R^{min}_{rel}$, we must consider that each additional shift also changes $REF(r)$. The number ρ of additional shifts to be created should be chosen to obey

$$\begin{aligned} & ACT(r) + \rho \geq p_r(REF(r) + \rho) \\ \Longrightarrow \quad & \rho \geq \tfrac{p_r \cdot REF(r) - ACT(r)}{1 - p_r} \end{aligned} \tag{5.26}$$

The minimum number ρ of additional shifts is thus

$$\rho := \left\lceil \frac{p_r \cdot REF(r) - ACT(r)}{1 - p_r} \right\rceil$$

With these considerations, the shift restriction penalty term $z^{SNR}(\Sigma)$ can be calculated as

$$z^{SNR}(\Sigma) := M^{SNR} \sum_{r \in R^{max}_{abs}} \max(ACT(r) - m_r, 0) + \tag{5.27}$$

$$c^K_{min}(r) \sum_{r \in R^{min}_{abs}} \max(m_r - ACT(r), 0) + \tag{5.28}$$

$$M^{SNR} \sum_{r \in R^{max}_{rel}} \max(ACT(r) - (p_r \cdot REF(r)), 0) + \tag{5.29}$$

$$c^{min}_k(r) \sum_{r \in R^{min}_{rel}} \max\left(\left\lceil \frac{p_r \cdot REF(r) - ACT(r)}{1 - p_r} \right\rceil, 0 \right) \tag{5.30}$$

A drawback of this term is that it can only be calculated for complete solutions. Only then all shift types are fixed and numbers of actual and reference shifts for relative restrictions are known. Clearly, we should also have guidance for the search when a solution is not complete. The following section describes how this can be achieved.

5.7. Lower Bounding

The existence of good lower bounds, i.e. lower estimations on the final objective function value of a partial solution, is crucial for the performance of branch-and-bound algorithms. In addition to pruning large parts of the search tree, lower bounds can be used to guide the search, indicating which branches are promising and should be followed.

Some of the above terms for the evaluation of the objective function value already try to anticipate costs. The shift cost term z^{SC} already considers cheapest valid shift types, and the unassigned task penalty z^{UT} takes minimum task lengths into account. On the other hand, the term z^{SNR} for shift number restrictions is rather weak because it can only be evaluated for a complete solution. However, it will be essential to take promising decisions in early stages of the search if it is meant to find a shift mix observing all restrictions. This requires good measures for penalties arising in complete solutions. Additionally, it will be shown that information on uninserted tasks can be used to anticipate further shift costs.

5.7.1. Task Look-Ahead

Imagine a task i which is not yet inserted but can be attributed to shifts which are not used so far, i.e. $\delta^{used}_s = \{\text{false}, \text{true}\} \; \forall s \in \delta^{shift}_i$. Then i will cause additional shift costs upon insertion. In the local steps of large neighbourhood search, tasks will be forbidden to be inserted into dummy shifts in order to reduce the number of unassigned tasks. We can therefore infer additional costs for uninserted tasks by the set of regular shift types. Clearly, it must be considered that two such tasks may be covered by the same shift. The resulting interdependencies will be implicitly represented by the following approach.

Imagine that the cost of a shift type is distributed over its length, resulting in minutewise costs for the shift type. Different shift types can cause different costs for covering the same minute. We can define minimum minutewise costs $mc(t)$ for a time period t as

$$mc(t) := \min_{\substack{k \in K, n \in N_k \\ st_{kn} \leq t < et_{kn}}} \frac{c_k}{et_{kn} - st_{kn}}$$

If i is a non-split task, its insertion will entail costs of at least

$$\min_{t \in \delta_i^{start}} \sum_{t'=t}^{t' < t + l_i} mc(t')$$

Covering a split part i by an empty shift results in costs of at least

$$\min_{t \in \delta_i^{start}} \sum_{t'=t}^{t' < t + msl_{P_i}} mc(t')$$

These costs can be added to the shift costs for each i fulfilling the above conditions, resulting in a stronger lower bound.

5.7.2. Maximum Restrictions

For absolute maximum restrictions, the calculation of lower bounds is easy. $ACT^{lb}(r) := ACT(r)$ is effectively a lower bound on the number of actual shifts in a complete solution. The term

$$M^{SNR} \sum_{r \in R_{abs}^{max}} \max(ACT^{lb}(r) - m_r, 0) \tag{5.31}$$

is thus a lower bound on the penalties for absolute maximum restrictions. Note that the bound could be refined by using an actual number $ACT'(r) := |\{s \in S \mid \delta_s^{used} = \{\text{true}\} \wedge \delta_s^K \subseteq K_r\}|$. $ACT'(r)$ is however more costly to evaluate, and we will content ourselves with the above expression (5.31).

For relative maximum restrictions, we additionally need an upper bound $REF^{ub}(r)$ on the number of reference shifts. We will first include all used shifts on days N_r. In a partial solution, some tasks may not yet be assigned to shifts and will be covered by still empty shift templates. We will assume that each of these tasks will be covered by one shift. For a given restriction $r \in R_{rel}^{max}$, it suffices to consider tasks which can be covered by shifts starting on one of the days N_r. Because a shift cannot cover more than 24 hours, we define the set $I^{SNR}(r)$ of relevant tasks for r as

$$\begin{aligned} I(r) := \{i \in I \mid\ & (P_i = \mathit{NIL} \wedge t_1 + d_{i^o,i} \leq b_i \wedge a_i + l_i + d_{i,i^d} \leq t_2) \vee \\ & (P_i \neq \mathit{NIL} \wedge t_1 + d_{i^o,i} \leq b_i \wedge a_{ssucc_i} + d_{i,i^d} \leq t_2)\} \end{aligned}$$

where $[t_1, t_2]$ is the time range of the set $\bigcup_{n \in N_r} \{n, n+1\}$ of days and $d_{i^o,i}$ and d_{i,i^d} are depot travel times. The number $NT(r)$ of not inserted tasks on the days affected by r is then

$$NT(r) := \left|\left\{i \in I(r) \mid \delta_i^{ins} \neq \{\text{true}\}\right\}\right| \tag{5.32}$$

and

$$REF^{ub}(r) := |\{s \in S \mid \delta_s^{used} = \{\text{true}\}, n_s \in N_r\}| + NT(r)$$

gives an upper bound on the number of reference shifts. $REF^{ub}(r)$ decreases monotonically because each time one of the tasks is inserted, one task less is accounted, but there is at most one additional used shift on the affected days. Because all tasks are assigned in a complete solution, $REF^{ub}(r)$ converges to $REF(r)$. A lower bound on the penalties for relative maximum restrictions is thus given by

$$M^{SNR} \sum_{r \in R_{rel}^{max}} \max(ACT^{lb}(r) - (p_r \cdot REF^{ub}(r)), 0) \tag{5.33}$$

By additional constraint variables, $REF^{ub}(r)$ can be efficiently calculated as a part of the CP model. We therefore introduce global domains containing the unused shift templates for each day. Because the number of shift templates for each day is fixed, the first part of the formula for $REF^{ub}(r)$ can be calculated from these domains. The second part can be retrieved from variables storing all unassigned tasks for each day on which these can take place.

5.7.3. Minimum Restrictions

We now come to the handling of absolute minimum restrictions. From (5.28), we may think of calculating an upper bound on $ACT(r)$ to obtain a lower bound on the penalties for absolute minimum restrictions. But because penalties for minimum restrictions are related to shift type costs ($c_{min}^K(r) := \min_{k \in K_r} c_k$), stronger bounds can be obtained.

The basic idea is that costs for violated minimum restrictions are imposed anyway: If a minimum restrictions r falls short of its limit and a yet uninserted task is covered by a shift in K_r, this will entail costs of at least $\min_{k \in K_r} c_k$ in the shift cost term. If another shift type is used, the same costs will arise for the violated restriction r in the penalty term (5.28).

We can thus use the original number $ACT(r)$ of used shifts in $K_r \times N_r$ and basically impose costs of

$$c_{min}^K(r) \sum_{r \in R_{abs}^{min}} \max(m_r - ACT(r), 0)$$

(cf. (5.28)) for absolute minimum restrictions.

However, we must take care about the interaction of shift costs and restriction penalties. In the shift cost term z^{SC}, we have already accounted minimum costs for used shifts. As long as not all of the shift types are not fixed, we do not know if a shift will finally realise a type K_r. We have to avoid accounting such shifts twice, i.e. in the shift cost and restriction terms. The following shifts $DA(r)$ are concerned:

$$DA(r) := \{s \in S \mid \delta_s^K \cap K_r \neq \emptyset, n_s \in N_r, |\delta_s^K| > 1, \delta_s^{used} = \{\text{true}\}\}$$

Because this problem only exists when using minimum shift number restrictions, we will not try to refine the shift cost term, but realise the remedy as a part of restriction penalties.

Imagine a shift s_1 in $DA(r)$ with $c_{min}^{shift}(s_1) := \min_{k \in \delta_{s_1}^K} c_k \geq \min_{k \in K_r} c_k =: c_{min}^{snr}(r)$, i.e. its current shift costs are already higher than the minimum costs for shift types in K_r. Since the costs c_{min}^{shift} are realised anyway, the error can be attributed to the additional penalty of $c_{min}^{snr}(r)$ which should be subtracted from the restriction penalties. For a shift $s_2 \in DA(r)$ with $c_{min}^{shift}(s_2) < c_{min}^{snr}(r)$, we would locate the flaw in the shift costs $c_{min}^{shift}(s_2)$ because if s_2 finally has a shift type in K_r, it will incur costs of at least $c_{min}^{snr}(r)$.

Now imagine that a partial solution contains both s_1 and s_2, and a restriction $r \in R_{abs}^{min}$ asks for a minimum number of one shift of types in K_r. We have to take a pessimistic approach and subtract the highest costs to obtain a valid lower bound on the penalty costs (5.28). If $(s_1, s_2, \ldots, s_n)$ are all of the shifts in $DA(r)$ ordered by decreasing minimum costs $c_{min}^{shift}(s_i)$, we would take the shifts of highest costs and subtract $\min(c^{shift}_{min}(s_1), c^{snr}_{min}(r))$, $\min(c^{shift}_{min}(s_2), c^{snr}_{min}(r))$ etc. from the penalties. The number of shift costs to be deducted is

$$\nu = \min(|DA(r)|, \max(m_r - ACT(r), 0))$$

We could thus use

$$c^{snr}_{min}(r) \cdot \max(m_r - ACT(r), 0) - \sum_{i=1}^{\nu} \min(c^{shift}_{min}(s_i), c^{snr}_{min}(r)) \tag{5.34}$$

as a lower bound approximation on the penalties for r. The drawback of this term is its computational complexity. Such penalty costs would have to be recalculated upon each change of $DA(r)$ due to the bounding of one of the shift types or because shifts are newly used. Furthermore, the term would have to be updated after each change to the shift costs $c^{shift}_{min}(s_i)$ or the number $ACT(r)$ of actual shifts.

Equation (5.34) defines penalties with deductions for doubly accounted shifts. An alternative consists in penalising less shifts, i.e. to define deductions on the number of shifts and not on the cost level. Using $c^{snr}_{min}(r) \geq \min(c^{shift}_{min}(s_i), c^{snr}_{min}(r))$, the above penalty term can be relaxed:

$$\begin{aligned} & c^{snr}_{min}(r) \cdot \max(m_r - ACT(r), 0) - \sum_{i=1}^{\nu} c^{snr}_{min}(r) \\ \geq\ & c^{snr}_{min}(r) \cdot (\max(m_r - ACT(r), 0) - \nu) \\ =\ & c^{snr}_{min}(r) \cdot \max(m_r - ACT(r) - |DA(r)|, 0) \end{aligned}$$

Because the determination of $DA(r)$ involves costly calculations of set intersections, $DA(r)$ is replaced by a simpler set $DA'(r)$ of used and unfixed shifts whose break rule days allow for a fixing to one of the types K_r on days N_r:

$$\begin{aligned} DA'(r) := \{s \in S \mid\ & K_{brd_s} \cap K_r \neq \emptyset, n_s \in N_r, \\ & |\delta^K_s| > 1, \delta^{used}_s = \{\text{true}\}\} \supseteq DA(r) \end{aligned}$$

We must be sure that monotonicity properties still hold. Assume the case that a task is inserted into a shift of one of the break rule days

$$BRD(r) := \{brd \in BRD \mid K_{brd} \cap K_r \neq \emptyset\}$$

If the shift type is not directly bound, the shift will be part of $DA'(r)$, and if r is not yet observed, one penalty less will be accounted. Since the shift is newly used, it causes an increase of at least $\min_{k \in K_{brd_s}} c_k$ in the shift cost term, but with the definition above, the shift restriction penalties would decrease by $c^{snr}_{min}(r)$ at the same time. If the new minimum shift costs $c^{shift}_{min}(s)$ are less than $c^{snr}_{min}(r)$, the sum of shift costs and restriction penalties would decrease which is not desirable.

To gain a better formulation, we will partly account for costs of less than $c^{snr}_{min}(r)$ for shifts falling short of the minimum limit. We therefore define the minimum costs of all shifts which may later be bound to one of the shift types K_r, using the break rule days $BRD(r)$ as defined above:

$$c^{brd}_{min}(r) := \min_{\substack{brd \in BRD(r) \\ k \in K_{brd}}} c_k$$

Clearly, we have $c^{brd}_{min}(r) \leq c^{snr}_{min}(r)$ because the minimum is taken over a superset of shift types. In general, the difference in costs will not be high because shift restrictions frequently apply to classes of shift types which are strongly related (e.g. shifts for full-time staff) and are likely to be grasped by one break rule day.

Since the above monotonicity problems only arise when tasks are inserted into unused shifts, lower costs of $c^{brd}_{min}(r)$ only have to be accounted as long as there are uninserted tasks. We might therefore determine the tasks fitting into shift types in $BRD(r)$. But because the error resulting from a rougher approximation is low, we simply take the number $NT(r)$ of not inserted tasks on the days affected by r as defined in equation (5.32). With the number $NP_{abs}(r) := \max(m_r - ACT(r) - |DA'(r)|, 0)$ of penalties to be incurred by $r \in R^{min}_{abs}$, a lower bound on the penalties for absolute minimum restrictions is given by

$$\sum_{r \in R^{min}_{abs}} c^{snr}_{min}(r) \cdot \min(NP_{abs}(r), NT(r)) + c^{snr}_{min}(r) \cdot \max(NP_{abs}(r) - NT(r), 0) \tag{5.35}$$

It should be noted that (5.35) is not a lower bound on the penalties (5.28) in itself, but has lower bound properties in combination with shift costs. Effectively, (5.35) may decrease while the sum with the shift costs increases monotonically. Because $DA'(r)$ and $NT(r)$ are empty in a complete solution, it is clear that (5.35) converges towards (5.28).

The lower bound approximation for *relative* minimum restrictions is based on the considerations for absolute minimum restrictions. The main difference is the determination of the number $NP_{rel}(r)$ of penalties. It is important to note that each additional shift contributing to $ACT(r)$ also increases the number $REF(r)$ of reference shifts. However, we must keep in mind that the time when we know that a shift is used and counts for the reference shifts differs from the time when it is considered an actual shift of a type in K_r.

For the calculation of additional shifts for relative minimum restrictions, the lower bound approximation

$$REF^{lb}(r) := REF(r) = |\{s \in S \mid \delta^{used}_s = \{\text{true}\}, n_s \in N_r\}|$$

on the number of reference shifts is used. To cover the minimum requirement of $\lceil p_r \cdot REF^{lb}(r) \rceil$ shifts, we will first assume that

$$COV(r) := \max(\min(\lceil p_r \cdot REF(r) \rceil - ACT(r), |DA'(r)|), 0)$$

shifts from $DA'(r)$ will finally contribute to r. While the shifts of $DA'(r)$ are already counted in $REF^{lb}(r)$, we must take into account that each shift beyond $DA'(r)$ also increases the number of reference shifts. Furthermore, we must consider that we have already taken $COV(r)$ shifts from $DA'(r)$, i.e. the number of actual shifts is $ACT(r) + COV(r)$. Therefore, the number ρ of additional shifts obeys

$$\begin{aligned} & ACT(r) + COV(r) + \rho \geq p_r \cdot (REF^{lb}(r) + \rho) \\ \Rightarrow \quad & \rho \geq \tfrac{p_r \cdot REF^{lb}(r) - (ACT(r) + COV(r))}{1 - p_r} \end{aligned}$$

Consequently, the (minimum) number $NP_{rel}(r)$ of penalties is

$$NP_{rel}(r) := \max\left(\left\lceil \frac{p_r \cdot REF^{lb}(r) - (ACT(r) + COV(r))}{1 - p_r} \right\rceil, 0\right)$$

With this figure given, the penalties for relative minimum restrictions can be calculated analogously to absolute minimum restrictions:

$$\sum_{r \in R^{min}_{rel}} c^{brd}_{min}(r) \cdot \min(NP_{rel}(r), NT(r)) + c^{snr}_{min}(r) \cdot \max(NP_{rel}(r) - NT(r), 0) \tag{5.36}$$

The main difference between absolute and relative minimum restrictions is that the minimum limit imposed by relative restrictions increases within the shift plan construction. Since the limit increases monotonically, we are sure not to violate the lower bound property.

The sum of equations (5.31), (5.33), (5.35) and (5.36) represents the shift number restriction penalty which is used as a lower bound to z^{SNR} in the course of the shift plan reoptimisation, converging to z^{SNR} for a complete solution.

5.8. Branch-and-Bound

As described before, we aim at improving an initial solution by a relax-and-optimise approach. Under the term *large neighbourhood search*, Kilby, Prosser and Shaw [1998] and Shaw [1998] have proposed a framework which was already used for the solution of the workload levelling problem in Chapter 4. Before describing how decisions are relaxed, we describe the restricted branch-and-bound scheme of the reoptimisation phase.

According to the definition of a complete solution in Section 5.6.8, the search consists of the following:

1. Insertion of relaxed tasks into shifts;
2. fixing of shifts to shift types;
3. fixing of lengths of split parts which are assigned to their dummy shift.

Suppose that a task i is to be inserted between two tasks i_1 and i_2. In order to trigger this insertion, all inserted tasks except for i_1 are deleted from the predecessor domain δ_i^{pred}:

$$\delta_i^{pred} := \delta_i^{pred} \setminus \{i' \in \delta_i^{pred} \,|\, \delta_{i'}^{ins} = \{\text{true}\}, i' \neq i_1\}$$

entailing the execution the task insertion rule described in Section 5.6.6. The fixing of shift types and split part lengths is simply done by setting shift type and split part length domains δ_s^K and δ_i^{sl} to single values, respectively. Note that split parts in dummy shifts cannot simply be fixed to their shortest possible lengths because there can be interdependencies via temporal relations via other tasks and shifts.

Decisions are taken in the above order, i.e. first all tasks are inserted, then the shift types of used shifts are fixed and finally the lengths of logically unassigned split parts are determined. The search tree thus consists of three layers corresponding to the different decisions, cf. Fig. 5.7.

Other orderings would be possible, but are less promising. As an example, there is no advantage in fixing shift types earlier because we already account for minimum shift costs. Fixing shift types can restrain possible lengths of split parts in dummy shifts, but this interdependency can be estimated not to be too strong. Generally, it should be tried to keep related decisions e.g. for shift types or for split part lengths close to each other, limiting the depth of backtracking when some of the decisions prove to be bad. This is clearly accomplished by the layered approach.

As long as there are tasks which are not inserted ($I^{rem} := \{i \in I \,|\, \delta_i^{ins} \neq \{\text{true}\}\} \neq \emptyset$), we choose the task $i \in I^{rem}$ with the least number of remaining insertion positions, i.e. the task i for which the number $|\{i' \in \delta_i^{pred} \,|\, \delta_{i'}^{ins} = \{\text{true}\}\}|$ of inserted tasks is minimal. This corresponds to the fail-first principle [Haralick and Elliott, 1980]. If for several tasks, the number of insertion positions is equal, we take the task i which is least movable, i.e. for which $|\delta_i^{start}|$ is minimal.

Trial insertions are then performed for each possible insertion position. The resulting objective values are lower bounds on the final objective function value, giving hints on how promising an insertion is. Branches corresponding to low values are thus explored first.

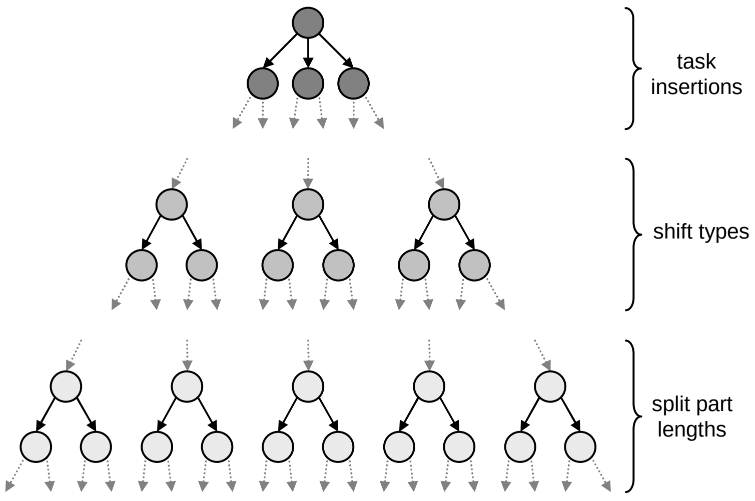


Figure 5.7.: Search tree structure.

After inserting all tasks into shifts, we consecutively choose shifts from

$$S^{rem} := \{s \in S \mid \delta_s^{used} = \{\text{true}\}, |\delta_s^K| > 1\}$$

for shift type fixing. Under the following conditions, the fixing of shift types is easy:

1. The break rule day of the shift is not involved in a shift number restriction, i.e. none of the shift type realisations (k, n_{brd}) for $k \in K_{brd_s}$ is contained in $K_r \times N_r$ for a shift number restriction r.

2. There are no vertical interdependencies with other shifts (which could be affected by shift number restrictions), i.e. the shift does not contain crew tasks or split parts.

If these conditions hold, the shift type decision does not interfere with other shifts, and s can simply be fixed to the cheapest possible shift type. For such shifts, shift type decisions will be taken before all other shifts.

For all other shifts, the number of potential interdependencies is determined, i.e. the number of shift number restrictions and the different split tasks and crews which make shift type fixing difficult. The shifts are then traversed in order of increasing numbers of interdependencies. A trial fixing to each of the shift types in δ_s^K is performed, and the resulting objective function values are used to order subnodes, i.e. cheapest alternatives are chosen first.

Finally, we fix the split parts which are logically unassigned, i.e.

$$SP^{rem} := \{i \in I \mid P_i \neq \mathit{NIL}, \delta_i^{ins} = \{\text{true}\}, \delta_i^{shift} = \{s_i^{dummy}\}, |\delta_i^{sl}| > 1\}$$

Split parts are traversed in decreasing order of the sizes $|\delta_i^{sl}|$ of their length domains. The subnodes are traversed in order of increasing task lengths since shorter durations generally result in lower costs.

Clearly, the lower bound described in Section 5.7 is not only used for the assessment of task insertion and shift type decisions, but also for pruning the search space. As soon as the lower bound exceeds the valuation of a previous best solution, the current state of the model cannot result in an improvement. Consequently, the branch is pruned, and search continues by exploring other assignments.

Since even the search trees for local steps are often too complex to be fully explored, we use limited discrepancy search (LDS) [Harvey and Ginsberg, 1995] in order to heuristically reduce the search space. We thus limit the number of discrepancies with regard to the best search path in the tree by a constant L_D. In contrast to the LDS algorithm for workload levelling (Section 4.6), all branches will be counted for the discrepancies and not only branches resulting in a dead end. Discrepancies are aggregated over all layers, including insertion, shift type and length decisions. For the final levels representing unassigned split part decisions, we will account for more than one discrepancy for each deviation from the best path because length decisions are generally less interdependent.

5.9. Large Neighbourhood Search

In large neighbourhood search, we will repeatedly select sets of tasks for relaxation and reoptimisation by CP-based branch-and-bound described before. Each of the steps can be regarded a local exchange which systematically explores a large neighbourhood. The local steps are performed until termination conditions like maximum runtimes or maximum numbers of steps are met. Initially, we try to achieve improvements in little computation time by reassigning single tasks. As soon as a given number *SWI* of steps is performed without improving the objective function, the number of relaxed tasks is increased by one etc. The steps therefore gradually become more complex and time-consuming, but are able to repair higher-order deficiencies.

In each step, a fixed number nt^{rel} of tasks is released. Clearly, the effectiveness of the algorithm heavily depends on the determination of task sets which promise to yield improvements. Two basic strategies for task selection are conceivable:

- Roll back the insertion decisions for all tasks in a shift such that the shift is no longer used and accounted for. A local step based on this idea aims at removing inefficient shifts from the shift plan, trying to use cheaper shifts to cover the tasks.
- Revise the shift decisions for strongly interdependent tasks in different shifts. An improvement could then stem from a shift being shortened or the qualification mix becoming better.

We will make an alternate use of these two strategies. The first strategy will be called *shift release* and the latter *task release*. Note that both ideas include the rollback of insertion decisions of tasks. For each step, we choose the shift release strategy with probability p. Otherwise, we use the task-oriented strategy (with probability $1 - p$).

The two strategies work as follows: First, one shift/task is selected in an environment which may yield an improvement. Further shifts/tasks are retrieved from a fixed interval around the first shift/task because if there is no temporal dependency between relaxed decisions, there is no advantage in revise the items in common. Clearly, shifts and tasks are only released if the maximum number nt^{rel} of tasks is obeyed.

Local steps can include tasks which were assigned to their dummy shifts. As long as there are tasks in dummy shifts, it will be probable that these will be chosen in a shift release step because this strategy takes shift costs into account. If however a task cannot be assigned at all, e.g. because a time period is not covered by any of the shift types, we should forbid such tasks to be chosen. We therefore devise a preprocessing step determining which tasks cannot be covered. Furthermore, we only allow one logically unassigned task to be included in any local step.

5.9.1. Shift Release

We start with the description of the shift release strategy. For the choice of a first shift, the following valuations are used:

- A reverse utilisation $RUTIL(s)$: The utilisation of a shift is the quotient of the sum of task task lengths (including break tasks) and travel times divided by the shift's duration. The reverse utilisation is then 100% minus the utilisation.

- The marginal shift cost $MCOST(s)$, i.e. the reduction of costs if the shift is not used. This includes shift type costs as well as marginal shift restriction penalties as far as the corresponding shift type is involved in a shift number restriction.

Clearly, a shift with high values for these terms will more probably lead to an improvement. A weighted sum of the two terms is built, and all shifts are sorted by decreasing combined costs. If SN is the number of shifts, $CR \in \mathbb{N}$ is a measure of choice randomness, and μ is a random number in $[0, 1[$, the index κ of the chosen shift in the sorted array is calculated as follows (cf. Kilby et al. [2000]):

$$\kappa := \lfloor SN \cdot \mu^{CR} \rfloor \tag{5.37}$$

If CR is high, μ^{CR} will be close to 0, and we will frequently take the first shift whose valuation is most promising. For smaller values of CR, the influence of randomness will increase, diversifying the search.

After the choice of a first shift s_1, two further valuations are evaluated for all shifts in S:

- The temporal distance to s_1: If a shift s starts before s_1, the temporal distance $DIST(s_1, s)$ is the time between the end of s and the start of s_1. If s_1 is before s, the temporal distance is the difference between the start of s and the end of s_1. If the shifts overlap, we have $SDIST(s_1, s) = 0$.

- The overlap $SOL(s_1, s)$ of the two shifts.

The total valuation $SREL(s)$ for a shift s is then the weighted sum of the above reverse utilisation and marginal shift cost together with the temporal distance and overlap with the first shift. The next shifts are again chosen according to (5.37) with all shifts being sorted according to their combined valuation.

Each time an additional shift s' is chosen, the valuations $DIST(s', s)$ and $OL(s', s)$ are added to the total valuation $SREL(s)$ of all shifts. Consequently, if S^{rel} are the shifts chosen so far, the valuation $SREL(s)$ of a shift $s \in S \setminus S'$ is

$$\begin{aligned} SREL(s) := \lambda_{UTIL} \cdot RUTIL(s) + \lambda_{MCOST} \cdot MCOST(s) + \\ \lambda_{SDIST} \cdot \sum_{s' \in S^{rel}} SDIST(s', s) + \lambda_{SOL} \cdot \sum_{s' \in S^{rel}} SOL(s', s) \end{aligned}$$

Preliminary tests have shown that the temporal distance valuation $DIST(s', s)$ does not sufficiently prevent shifts with high temporal distances from being chosen. Therefore, the added temporal distances

$$\sum_{s' \in S^{rel}} DIST(s', s)$$

of commonly chosen shifts are limited by a constant TD^{shifts}.

The procedure continues as long as there are shifts which can be chosen without exceeding the maximum number nt^{rel} of tasks.

5.9.2. Task Release

The task-oriented release strategy basically works the same. One aspect has special importance in the context of the task release method: If tasks are in subcrews, releasing a single task has only little benefits because the task will have to be placed in the same shift crew as adjacent crew tasks. Therefore, subcrews are always relaxed as a whole. Consequently, we must ensure that when a crew task is chosen, all of its subcrew tasks can also be released within the limit of nt^{rel} tasks.

For the choice of a first task, different strategies were tried, but in the end, choosing a task i_1 from a uniform distribution over all tasks turned out to be the best strategy. Again, releasing tasks which are far away from each other is little promising. Further tasks are therefore limited to overlap with a interval of TD^{tasks} time units around the first task, defined relative to its start and end. If the first task is a crew task, the range is defined from the earliest start and latest end over all subcrew tasks.

All further tasks are chosen as described above: The tasks within the range are sorted according to decreasing valuations expressing a relatedness with the tasks which were relaxed so far. Further tasks are then chosen according to (5.37). If the task is a crew task, the total subcrew is released while non-crew tasks are released along with their group of symmetric tasks (cf. Section 5.6.7). This procedure continues as long as the maximum limit nt^{rel} of relaxed tasks can be obeyed.

The relatedness $TREL$ between two tasks comprises the following terms:

- The overlap $TOL(i_1, i_2)$ which is defined as the minimum overlap resulting from a movement of tasks within in their remaining time windows.
- The temporal distance $TDIST(i_1, i_2)$ summing up the absolute distances between earliest start times and latest end times.
- The affiliation $ST(i_1, i_2)$ to the same split task which is 1 if $P_{i_1} = P_{i_2} \neq \mathit{NIL}$ and 0 otherwise.
- The affiliation $CREW(i_1, i_2)$ to the same task crew which is 1 if there is a $C \in \mathcal{C}$ such that $i_1, i_2 \in C$. Note that a task crew may consist of several subcrews. Then this term will favour subcrews of the same task crew to be chosen.
- The similarity $QUAL(i_1, i_2)$ of qualification profiles which is the cardinality of the intersection of the tasks' qualification requirements: $QUAL(i_1, i_2) := |Q_{i_1} \cap Q_{i_2}|$.

As above, the valuations are aggregated over all tasks which are released so far: If I^{rel} is the set of chosen tasks, $TREL(i)$ is calculated as

$$\begin{aligned} TREL(i) := \lambda_{TOL} \cdot \sum_{i' \in I^{rel}} TOL(i', i) + \lambda_{TDIST} \cdot \sum_{i' \in I^{rel}} TDIST(i', i) + \\ \lambda_{ST} \cdot \sum_{i' \in I^{rel}} ST(i', i) + \lambda_{CREW} \cdot \sum_{i' \in I^{rel}} CREW(i', i) + \\ \lambda_{QUAL} \cdot \sum_{i' \in I^{rel}} QUAL(i', i) \end{aligned}$$

Note that independently from the chosen strategy (task or shift release), we will determine a set I^{rel} of tasks to be relaxed. Nevertheless, we will also revise decisions which do not relate to tasks (like shift types).

5.9.3. Local Step Setup

For setting up the CP model for a local step, it is not sufficient to revise insertion decisions for tasks. If a shift becomes empty, e.g. by revision of insertion decisions in the shift release strategy, we should also reset the "used" as well as the shift type domain, assuring that shift costs are no longer incurred. Even if not the ensemble of all tasks in a shift are relaxed, we will want to have the flexibility of shortening or prolonging the shift by relaxing the shift type domain. We thus have to devise a procedure to determine all items (tasks and shifts) which potentially interfere with the insertion decisions of the tasks to be relaxed.

Starting from the set I^{rel}, we will build sets of dependent tasks I^{dep} and shifts S^{dep}. For all tasks in I^{dep}, we will basically reset the time windows (but not the insertion decisions as for the tasks in $I^{rel} \subseteq I^{dep}$). For the shifts S^{dep}, we will revise all decisions relating to the shift type, i.e. we will enable the shift to be fixed to another shift type in the course of the local step.

If a task $i \in I^{rel}$ is a crew task, we know that i's insertion could have caused temporal restrictions on its adjacent crew tasks. The same is true if i is a split part. The propagation to related tasks of the same task crew or split task may have entailed decisions for shift types, and these decisions should also be rolled back. Clearly, if a shift is part of a shift crew, we should reset the shift types of all crew shifts since otherwise, we would not open any degrees of freedom. The set I^{dep} is therefore recursively defined as the smallest set for which the following holds:

$$\begin{aligned} i \in I^{rel} &\Rightarrow i \in I^{dep} \\ i \in I^{rel} \wedge \exists C \in \mathcal{C} : i \in C &\Rightarrow C \subseteq I^{dep} \\ i \in I^{rel} \wedge P_i \neq \mathit{NIL} &\Rightarrow P_i \subseteq I^{dep} \\ s \in S^{dep} \wedge \delta_i^{shift} = \{s\} &\Rightarrow i \in I^{dep} \end{aligned}$$

Note that this definition makes use of the set S^{dep} (which reversely depends on I^{dep}) since all tasks in dependent shifts are defined to be dependent themselves. Their time windows should be reset because the decision for a specific shift type may have entailed time windows reductions. This is especially true for the origin, destination and break tasks because without resetting these tasks, the shift would not gain any degrees of freedom.

The set S^{dep} of dependent shifts is the smallest set for which the following conditions hold:

$$\begin{aligned} i \in I^{dep} \wedge \delta_i^{shift} = \{s\} &\Rightarrow s \in S^{dep} \\ s \in S^{dep} \wedge \exists H \in \mathcal{H} : s \in H &\Rightarrow H \subseteq S^{dep} \end{aligned}$$

The two sets can be calculated by a fixed point iteration, starting from the set $I^{dep} := I^{rel}$ and successively adding dependent tasks and shifts to I^{dep} and S^{dep}.

We then reset CP domains pertaining to tasks and shifts in I^{dep} and S^{dep}. For each dependent task $i \in I^{dep}$, δ_i^{start} is reset to the original time window $[a_i, b_i]$. For all relaxed tasks $i \in I^{rel}$, we additionally perform the following steps (note that $I^{rel} \subseteq I^{dep}$):

- The successor σ_{π_i} of the predecessor π_i of i is set to its successor σ_i while the predecessor π_{σ_i} of the successor task σ_i is set to π_i; π_i and σ_i are subsequently reset to *NIL*;
- δ_i^{ins} is set to $\{\text{false}, \text{true}\}$;
- δ_i^{pred} and δ_i^{shift} are temporarily reset to empty sets because the determination of insertion positions can only be done in a subsequent step.

For all shifts $s \in S^{dep}$, the following domains are reset:

- δ_s^K is set to all shift types K_{brd_s} allowed by the shift's break rule day brd_s;

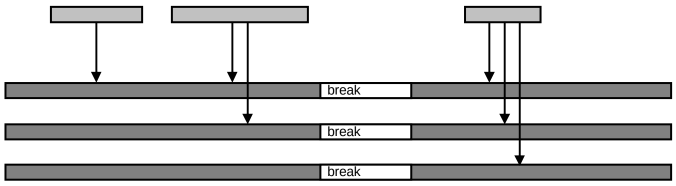


Figure 5.8.: Covering tasks by empty shift templates.

- δ_s^{used} is reset to $\{\text{false}, \text{true}\}$;
- δ_{qual}^s is reinitialised to the set Q of all qualifications;
- δ_s^{qp} is set to $[1, qp_{min}^{-1}]$.

After rolling back the domains to their original states, a local repropagation is performed in order to make the consequences of unrevised insertion decisions visible. Therefore, all consistency tests relating to dependent tasks and shifts are propagated. Afterwards, the time windows reflect the current state of insertions, and the shift type and cost information as well as qualification profiles will be consistent.

Subsequently we can evaluate the positions into which relaxed tasks can be reinserted. All shifts which could cover a task $i \in I^{rel}$ are traversed. If i can be inserted between two tasks i_1 and i_2 (including origin, destination and break tasks), (i_1, i_2) will be part of the set $\Phi(i)$ of insertion positions introduced above. The task i_1 is then added to δ_i^{pred} of potential predecessors of i, i is added to $\delta_{i_2}^{pred}$, and the shift s of i_1 and i_2 to δ_i^{shift}.

Basically the same is done for empty shift templates cover the tasks. Clearly, if a task can be attributed to an empty shift s, it can be covered by any template of the same break rule day. In order to avoid symmetry, we add only one empty shift per break rule day to the shift domain of a task. If a further task can be covered by the same break rule day, we must consider that an empty shift may finally be used to cover both tasks. We therefore allow the task to be assigned to the first empty shift as well as to another empty shift template of the break rule day. For each task fitting into a break rule day, we thus include an additional empty shift, cf. Fig. 5.8. Note that if subcrews are used, the same argument applies to the shift crew level.

The insertion positions are entered into the respective variables δ_i^{pred} and δ_i^{shift}. A repropagation ensures consistency before starting the reoptimisation by the branch-and-bound algorithm described before.

5.10. Overlap Minimisation

As described in Section 5.2.5, two subsequent tasks i_1, i_2 in a shift are allowed to overlap by a maximum amount of $\min(ol_{i_1}^{max}, ol_{i_2}^{max})$, including the travel between i_1 and i_2. The constraint imposing minimum distances between the start times of such tasks i_1 and i_2 was therefore relaxed by this amount. Overlap was introduced to soften the strict deterministic character of the model. It reflects the planning practice to pack tasks somewhat tighter in order to increase the shift utilisation.

Clearly, overlap should be avoided if possible. Overlap minimisation is not explicitly included in the optimisation model for two reasons. On the one hand, it is a subordinate objective, i.e. we will accept more overlap if shift costs can be lowered. On the other hand, the determination of overlaps requires determining task start times. However, this would make the search more costly because we would have to fix all start time domains as a part of the tree search.

When handling overlap minimisation as a secondary objective, we can still exploit the degrees of freedom which are left after local improvement, consisting of the start times of movable tasks and the lengths of split parts. This can be compared to the *finalising* procedures of Campbell and Savelsbergh [2004]. Our goal will be the minimisation of the sum of overlaps on all tasks.

If no crews or split tasks are used, the best solution can easily be found by a dynamic programming pass over the tasks in every shift. As described in Section 5.3, crews and split tasks introduce vertical temporal relations, i.e. their start time decisions affect more than one shift. As a consequence, overlap minimisation is not straightforward anymore. A simple greedy method is therefore used for fixing start times.

The procedure passes once over all shifts and the tasks they contain. For each task i, the start times from δ_i^{start} are tried and the (local) consequences on task overlaps determined. The best start time is then entered into the constraint model and propagated before turning to the next task.

For each non-split task i, we can simply determine the overlaps with the predecessor and successor tasks $pred_i$ and $succ_i$. If i is a split part, we calculate the overlap with the predecessor $pred_i$ and the minimum overlap with its successor $succ_i$, assuming that i will have a minimum length as given by δ_i^{sl}. If i is the first split part of a split task (i.e. $spred_i = NIL$), the end time of the last split part can be determined by adding the total split task length to the start time. We therefore additionally include the overlap of the last split part with its successor and its predecessor, using the task's minimum length. If i is not the first split part, i's start time is also the end time of $spred_i$. We can therefore determine the overlap of $spred_i$ with its successor and its predecessor, using the minimum length from $\delta_{spred_i}^{sl}$.

Often there are ties in choosing the start time causing least overlaps. Non-split tasks are then fixed to the earliest start time while split parts are fixed such that the first split part has the longest possible length. Consequently, only the start time of the first split part is fixed to the earliest value while start times of all other parts are fixed to latest start times. This corresponds to a sequential view of task splitting: the longest possible part of a splittable task is assigned to a shift, and if the task does not fit completely, it is interrupted at the latest possible time.

5.11. Experimental Results

We will evaluate the performance of the large neighbourhood search algorithm on a number of real-world scenarios from the practice of ground handling companies and airlines. Basic information on the test cases is summarised in Tables 5.4 and 5.5. Scenarios belong to one of the three classes B, C and D. Test set B corresponds to the staff planning scenarios used in the workload levelling procedure of Chapter 4. Scenarios of set C use only limited constraints and involve only fixed tasks; these scenarios will also be used for the experiments of Chapter 7. Scenarios of class D partly involve richer sets of constraints while only a limited number or none of the tasks are movable.

Most test cases span over one week or eight days. Some scenarios have a very large scale with up to 11325 tasks (scenario B09) and 2380 shift types (scenario D14). 15 out of 62 test cases allow for splitting of a subset of the tasks with minimum split part lengths between 30 and 300 minutes; *original tasks* in Table 5.4 refers to the number of tasks *before* splitting. Task overlapping up to 2 minutes is only admitted on five scenarios. 15 scenarios comprise task crews of which 8 scenarios involve crew planning with subcrews of sizes 2 or 4. 36 scenarios involve qualifications, but in many cases, qualification constraints are very loose. As an example, scenario B02 allows for four qualifications per shift which exactly equals the number of involved qualifications. Seven test cases comprise qualification restrictions. Absolute shift number restrictions are present in ten scenarios while only two test files include relative restrictions. While the number shift types is partly considerable (scenarios D14, D22), the number of break rule days is mostly moderate with only few exceptions (e.g. B09 and B10).

The algorithm was implemented in Visual C++ 7.1. Tests were carried out an AMD Athlon 2000+ computer (1.67 GHz) with 512 MB RAM and operating system Windows XP SP1. For the implementation of constraint propagation and branch-and-bound, the *ctk* toolkit was used. The improvement phase builds upon solutions of the construction heuristic which was described in Section 5.1. For all tests, we have set $w^{QP} = 5$, $M^{QR} = M^{SNR} = 500$ and $M^{UT} = 10$.

The relaxation strategies were tuned in preliminary tests. With a probability of 70%, the task release strategy is used, but results turned out to be robust over a wide range of different values. The parameters for the task and shift release strategy are given in Table 5.6.

A discrepancy limit of $L_D = 8$ turned out to be a good compromise between search depth and processing times of individual steps. Discrepancy is accounted for over all search tree layers representing task insertion and shift fixation decisions. Shifts which do not involve vertical interdependencies due to split or crew tasks are heuristically fixed to their cheapest available shift types (i.e. a discrepancy of 0 is used). Equally, logically unassigned split parts are always fixed to their shortest possible lengths by accounting for 8 discrepancies for each deviation from the best branch. Additionally to imposing a discrepancy limit, it turned out to be advantageous to limit the number of branches leading to propagation failures to a maximum of 2000. Failures usually arise when the value of a partial solution exceeds the value of the incumbent, but can also result from inconsistent assignments (e.g. due to incompatible qualifications).

The performance of the algorithm was evaluated at runtimes of 5 and 30 minutes. The 5 minute limit is typical of situations in which solutions must be obtained within low response times while a 30 minute run will generally be started for offline execution. For the 5-minute tests, *SWI* (steps without improvement before stepwidth incrementation) was set to 10. For the 30 minute runs, *SWI* was equal to 60, meaning that smaller neighbourhoods are investigated more thoroughly.

Results with regard to objective function improvements are summarised in Table 5.7. Results of the LNS runs represent averages over five runs with different random seeds. As can be seen, local search improves initial shift plans by an average of 3.07% (5 minutes) and 3.88% (30 minutes). Highest improvements are attained on scenarios B13 (47.07%), D09 (19.67%) and D31 (17.87%). On scenarios B13 and D31, these improvements mainly stem from covering still unassigned tasks. While on some scenarios, all or nearly all improvement is accomplished within five minutes (e.g. B02, B13, D01), savings on other scenarios are considerably higher with additional runtime (e.g. B03, C04, D03). Relative standard deviations for the different runs (see Section 4.10) amounted to an average of 0.39% for the 5-minute experiments and 0.46% for the 30-minute tests.

While the objective function comprises several terms, Table 5.8 gives results in terms of shift costs. As can be seen, savings raise up to a maximum of 15.94% on scenario B06. On five scenarios, more than 10% improvement are achieved; on average, savings amount to 2.15% (5 minutes) and 2.74% (30 minutes). Improvements on scenarios involving movable tasks are often higher, especially if crews are involved. On scenario B09, shift costs increase slightly because additional shifts are needed to cover still unassigned tasks.

Furthermore, Table 5.8 indicates shift utilisations which result from dividing total task minutes by total shift durations without breaks [Jacobs and Bechtold, 1993]. Utilisations before local improvement amount to 56.8% with a minimum of 20.9% (B04) and a maximum of 99.7% (D21). While the average utilisation is typical of airport operations (see e.g. Dowling et al. [1997]), the minimum utilisation shows that in scenario B04, shift types do not seem be well-designed for covering the workload. In the 30-minute runs, shift utilisations are increased by up to 9.1% (B06) with an average of 1.4%.

While these savings are already considerable in view of the large scale of airport operations, a main benefit of the local improvement algorithm consists in offering a remedy for deficiencies of the construction algorithm. While scenarios frequently comprise work tasks which cannot be covered by any of the given shift types, the construction phase sometimes leaves tasks unassigned

No.	days	original tasks	splittable tasks	avg. time window width	avg. task length	avg. min. split part length	max. travel time	max. overlap	crew size	task crews	avg. size task crews	subcrews	avg. size subcrews
B01	8	410	0	23.1	105.5	0.0	11	0	1	33	3.0	0	0.0
B02	8	748	0	5.9	14.3	0.0	8	0	1	19	2.0	0	0.0
B03	8	929	0	1.0	66.6	0.0	10	0	1	0	0.0	0	0.0
B04	8	1615	0	99.0	21.1	0.0	10	0	1	134	12.1	0	0.0
B05	7	1121	0	80.9	24.8	0.0	24	0	1	0	0.0	0	0.0
B06	8	627	0	105.1	29.4	0.0	18	0	1	533	1.2	0	0.0
B07	8	734	0	98.1	30.7	0.0	29	0	1	573	1.3	0	0.0
B08	8	4252	0	115.3	10.7	0.0	54	0	1	0	0.0	0	0.0
B09	7	11325	83	43.6	28.4	45.0	12	0	1	0	0.0	0	0.0
B10	7	4867	0	16.2	15.5	0.0	0	0	1	1572	3.1	0	0.0
B11	7	623	0	128.2	95.7	0.0	0	0	1	72	8.7	0	0.0
B12	7	3234	0	85.1	15.7	0.0	1	0	4	580	5.6	968	3.3
B13	7	1836	0	44.8	35.6	0.0	1	0	4	739	2.5	772	2.4
B14	7	3731	0	1.7	5.4	0.0	3	2	1	0	0.0	0	0.0
C01	8	2934	0	0.0	37.1	0.0	0	0	1	0	0.0	0	0.0
C02	8	3337	0	0.0	176.1	0.0	13	0	1	0	0.0	0	0.0
C03	8	2128	0	0.0	27.0	0.0	26	0	1	0	0.0	0	0.0
C04	8	1429	0	0.0	45.5	0.0	26	0	1	0	0.0	0	0.0
C05	8	1027	0	0.0	45.6	0.0	19	0	1	0	0.0	0	0.0
C06	8	1598	0	0.0	47.6	0.0	26	0	1	0	0.0	0	0.0
C07	4	2588	0	0.0	68.2	0.0	0	0	1	0	0.0	0	0.0
C08	4	1816	0	0.0	88.2	0.0	0	0	1	0	0.0	0	0.0
C09	3	1718	0	0.0	71.8	0.0	0	0	1	0	0.0	0	0.0
C10	3	860	0	0.0	112.2	0.0	0	0	1	0	0.0	0	0.0
C11	3	327	0	0.0	66.3	0.0	0	0	1	0	0.0	0	0.0
C12	3	572	0	0.0	58.7	0.0	0	0	1	0	0.0	0	0.0
C13	8	3521	0	0.0	40.1	0.0	6	0	1	0	0.0	0	0.0
C14	8	1297	0	0.0	20.0	0.0	13	0	1	0	0.0	0	0.0
C15	8	149	0	0.0	25.0	0.0	1	0	1	0	0.0	0	0.0
C16	8	255	0	0.0	70.2	0.0	28	0	1	0	0.0	0	0.0
C17	7	2256	0	0.0	38.9	0.0	3	0	1	0	0.0	0	0.0
D01	8	2710	0	0.0	55.7	0.0	5	0	4	609	4.4	706	3.8
D02	8	1764	0	0.0	68.5	0.0	4	0	4	239	7.4	478	3.7
D03	8	292	0	0.0	25.8	0.0	6	0	2	146	2.0	146	2.0
D04	8	476	119	0.0	140.7	45.0	3	0	1	0	0.0	0	0.0
D05	8	348	123	0.0	211.0	45.0	3	0	1	0	0.0	0	0.0
D06	8	7693	0	0.0	57.8	0.0	11	0	4	1497	5.1	1924	4.0
D07	8	2986	0	0.0	28.9	0.0	10	0	2	1493	2.0	1493	2.0
D08	8	842	0	0.0	26.7	0.0	5	0	2	421	2.0	421	2.0
D09	8	441	0	0.0	48.2	0.0	6	0	1	0	0.0	0	0.0
D10	8	1590	0	1.0	53.9	0.0	0	0	1	0	0.0	0	0.0
D11	8	911	0	0.0	58.1	0.0	10	0	1	0	0.0	0	0.0
D12	8	151	0	4.8	432.3	0.0	0	0	1	0	0.0	0	0.0
D13	8	548	19	0.0	165.0	300.0	0	0	1	0	0.0	0	0.0
D14	8	1634	16	0.0	155.2	45.0	60	0	1	0	0.0	0	0.0
D15	8	3011	65	0.0	73.5	30.0	0	2	1	0	0.0	0	0.0
D16	8	1792	0	0.0	89.8	0.0	12	0	1	0	0.0	0	0.0
D17	8	5124	0	0.0	23.8	0.0	12	0	1	0	0.0	0	0.0
D18	7	540	128	0.0	176.8	45.0	0	0	1	0	0.0	0	0.0
D19	8	2075	0	0.0	124.2	0.0	5	0	1	0	0.0	0	0.0
D20	7	187	0	0.0	247.8	0.0	0	0	1	0	0.0	0	0.0
D21	7	126	0	0.0	239.2	0.0	0	0	1	0	0.0	0	0.0
D22	7	3328	83	0.0	83.8	45.0	0	0	1	0	0.0	0	0.0
D23	7	2994	70	0.0	54.0	45.0	0	0	1	0	0.0	0	0.0
D24	7	2164	634	0.0	186.4	45.0	7	0	1	0	0.0	0	0.0
D25	7	2248	68	0.0	43.6	45.0	2	0	1	0	0.0	0	0.0
D26	7	609	7	0.0	29.2	45.0	14	0	1	0	0.0	0	0.0
D27	7	2771	0	0.0	32.5	0.0	0	0	1	0	0.0	0	0.0
D28	7	673	115	0.0	206.5	45.0	1	2	1	0	0.0	0	0.0
D29	7	9759	0	0.0	33.9	0.0	3	2	1	0	0.0	0	0.0
D30	7	3976	332	0.0	75.2	45.0	3	2	1	0	0.0	0	0.0
D31	7	236	97	0.0	251.0	45.0	0	0	1	0	0.0	0	0.0

Table 5.4.: Scenario data.

No.	quali- fications	avg. qual. per task	qual. pref. range	max. qual. per shift	qualification restrictions	shift types	break rule days	range of shift lengths	avg. breaks per shift type	avg. shift type cost	abs. shift restr. (min./max.)	rel. shift restr. (min./max.)
B01	3	1.1	[0,1]	4	7	11	18	[390,510]	2.0	88.2	0/0	0/0
B02	4	1.0	[0.42,0.73]	4	0	9	9	[510,510]	2.0	96.6	0/0	0/0
B03	12	0.4	[0,1]	4	35	33	18	[390,510]	1.0	96.0	0/57	0/0
B04	0	0.0	[1,1]	–	0	41	9	[360,720]	0.0	82.5	0/0	0/0
B05	0	0.0	[1,1]	–	0	17	9	[480,480]	0.0	80.0	0/0	0/0
B06	0	0.0	[1,1]	–	0	94	9	[360,600]	0.0	79.8	0/0	0/0
B07	0	0.0	[1,1]	–	0	94	9	[360,600]	0.0	79.8	0/0	0/0
B08	0	0.0	[1,1]	–	0	3	9	[470,490]	1.0	78.0	0/0	0/0
B09	0	0.0	[1,1]	–	0	68	288	[180,660]	0.7	68.5	0/68	0/0
B10	2	0.6	[0.2,1]	4	0	58	207	[360,540]	0.9	94.0	0/0	0/0
B11	0	0.0	[1,1]	–	0	8	36	[240,570]	0.9	87.8	0/0	0/0
B12	2	1.0	[1,1]	4	0	352	18	[240,600]	0.7	75.2	0/0	0/0
B13	2	1.0	[1,1]	4	0	352	18	[240,600]	0.7	75.2	0/0	0/0
B14	0	0.0	[1,1]	–	0	354	36	[240,600]	0.7	75.3	0/0	0/0
C01	0	0.0	[1,1]	–	0	176	18	[240,480]	0.5	1601.7	0/0	0/0
C02	0	0.0	[1,1]	–	0	270	45	[240,510]	2.4	74.0	0/0	0/0
C03	1	1.0	[1,1]	10	0	270	45	[240,510]	2.4	74.0	0/0	0/0
C04	1	0.5	[1,1]	10	0	270	45	[240,510]	2.4	74.0	0/0	0/0
C05	2	0.6	[1,1]	10	0	270	45	[240,510]	2.4	74.0	0/0	0/0
C06	1	0.6	[1,1]	10	0	270	45	[240,510]	2.4	74.0	0/0	0/0
C07	0	0.0	[1,1]	–	0	12	72	[484,484]	1.0	80.0	8/12	0/0
C08	0	0.0	[1,1]	–	0	12	63	[484,484]	1.0	80.0	8/8	0/0
C09	0	0.0	[1,1]	–	0	17	126	[384,484]	1.0	80.0	12/12	0/0
C10	0	0.0	[1,1]	–	0	11	63	[501,501]	1.0	80.0	0/0	0/0
C11	0	0.0	[1,1]	–	0	17	108	[456,471]	1.0	80.0	0/0	0/0
C12	0	0.0	[1,1]	–	0	14	72	[456,471]	1.0	80.0	0/0	0/0
C13	0	0.0	[1,1]	–	0	9	9	[360,720]	1.0	94.7	14/35	0/0
C14	0	0.0	[1,1]	–	0	41	9	[360,720]	0.0	82.5	0/0	0/0
C15	0	0.0	[1,1]	–	0	94	9	[360,600]	0.0	79.8	0/0	0/0
C16	0	0.0	[1,1]	–	0	94	9	[360,600]	0.0	79.8	0/0	0/0
C17	3	1.1	[1,1]	4	0	5	9	[480,600]	1.0	106.9	0/0	0/0
D01	2	1.0	[1,1]	4	0	384	36	[240,640]	0.8	236.3	0/0	0/0
D02	2	1.0	[1,1]	4	0	384	36	[240,640]	0.8	236.3	0/0	0/0
D03	2	1.0	[1,1]	4	0	384	36	[240,640]	0.8	236.3	0/0	0/0
D04	0	0.0	[1,1]	–	0	270	45	[240,510]	2.4	74.0	0/0	0/0
D05	0	0.0	[1,1]	–	0	270	45	[240,510]	2.4	74.0	0/0	0/0
D06	2	1.0	[1,1]	4	0	174	18	[240,480]	0.6	128.3	0/0	0/0
D07	2	1.0	[1,1]	4	0	174	18	[240,480]	0.6	128.3	0/0	0/0
D08	2	1.0	[1,1]	4	0	174	18	[240,480]	0.6	128.3	0/0	0/0
D09	2	1.3	[0.42,1]	4	7	8	18	[390,510]	2.0	80.5	0/0	0/0
D10	14	1.4	[0,1]	4	7	47	9	[510,510]	1.0	103.2	0/0	0/0
D11	5	0.3	[0,1]	4	21	31	18	[390,510]	1.0	95.7	0/7	0/0
D12	4	0.9	[0,0.63]	4	0	31	18	[390,510]	1.0	95.7	0/7	0/0
D13	5	1.0	[0,1]	4	0	672	9	[240,630]	0.0	93.1	0/0	0/0
D14	11	0.6	[0,1]	4	0	2380	27	[240,645]	0.9	94.4	0/0	0/7
D15	4	0.2	[0,0.55]	2	0	280	45	[360,960]	0.0	98.6	7/0	0/0
D16	0	0.0	[1,1]	–	0	41	9	[360,720]	0.0	82.5	0/0	0/0
D17	0	0.0	[1,1]	–	0	41	9	[360,720]	0.0	82.5	0/0	0/0
D18	0	0.0	[1,1]	–	0	161	27	[195,600]	1.0	76.5	0/0	0/0
D19	8	0.2	[0.04,1]	2	0	9	9	[480,510]	0.0	100.3	0/0	0/0
D20	7	1.0	[0,1]	4	43	324	162	[240,720]	0.5	566.9	0/0	0/0
D21	3	1.0	[0,0.54]	4	0	324	162	[240,720]	0.5	566.9	0/0	0/0
D22	0	0.0	[1,1]	–	0	825	16	[180,480]	0.4	53.7	0/0	0/0
D23	9	1.0	[0,1]	4	0	369	18	[240,480]	0.8	59.2	0/0	0/0
D24	16	1.1	[0,1]	4	0	16	18	[240,480]	0.8	59.2	0/0	0/0
D25	3	0.5	[0.04,1]	4	0	71	135	[240,750]	1.0	94.8	0/0	0/0
D26	1	0.5	[0.03,0.03]	4	0	29	117	[270,570]	1.0	87.0	0/0	0/0
D27	8	1.0	[0,1]	4	0	32	9	[360,540]	0.0	89.2	0/0	0/0
D28	0	0.0	[1,1]	–	0	354	36	[240,600]	0.7	75.3	0/0	0/0
D29	3	0.3	[0.94,1]	4	1	354	36	[240,600]	0.7	75.3	0/9	0/7
D30	7	1.0	[0,1]	4	0	354	36	[240,600]	0.7	75.3	0/0	0/0
D31	1	1.0	[0.91,0.91]	4	0	20	45	[240,570]	0.5	69.3	0/0	0/0

Table 5.5.: Scenario data (continued).

shift release		task release	
CR	10	TD^{tasks}	480
TD^{shifts}	480	λ_{TOL}	100
λ_{UTIL}	20	λ_{TDIST}	10
λ_{MCOST}	10	λ_{ST}	10
λ_{SDIST}	100	λ_{CREW}	10
λ_{SOL}	10	λ_{QUAL}	100

Table 5.6.: Parameters of release strategies.

No.	initial	5 min.		30 min.	
	obj. fct.	obj. fct.	improvement	obj. fct.	improvement
B01	19978.0	19402.0	2.88%	19334.2	3.22%
B02	7182.3	7066.5	1.61%	7066.1	1.62%
B03	37664.3	34035.4	9.63%	32196.1	14.52%
B04	27135.0	25993.0	4.21%	25771.0	5.03%
B05	20715.0	20347.0	1.78%	20187.0	2.55%
B06	6415.0	5485.0	14.50%	5393.0	15.93%
B07	7725.0	6827.0	11.62%	6723.0	12.97%
B08	13797.7	13373.8	3.07%	13217.1	4.21%
B09	730835.5	730501.7	0.05%	730413.0	0.06%
B10	40985.7	40896.0	0.22%	40766.1	0.54%
B11	64392.0	64272.5	0.19%	64263.0	0.20%
B12	20039.0	18241.0	8.97%	17752.6	11.41%
B13	46785.0	25725.5	45.01%	24765.6	47.07%
B14	7789.3	7624.3	2.12%	7597.5	2.46%
C01	258489.5	255088.3	1.32%	254764.6	1.44%
C02	3645650.0	3644874.8	0.02%	3644060.4	0.04%
C03	25513.0	25478.2	0.14%	25478.2	0.14%
C04	24695.0	24639.0	0.23%	24440.0	1.03%
C05	18627.0	18254.6	2.00%	18210.2	2.24%
C06	29782.0	29725.4	0.19%	29629.8	0.51%
C07	56080.0	56080.0	0.00%	56064.0	0.03%
C08	125840.0	125776.0	0.05%	125680.0	0.13%
C09	42400.0	42400.0	0.00%	42400.0	0.00%
C10	26000.0	25968.0	0.12%	25920.0	0.31%
C11	7680.0	7680.0	0.00%	7680.0	0.00%
C12	13520.0	13408.0	0.83%	13328.0	1.42%
C13	102173.0	102173.0	0.00%	102152.0	0.02%
C14	13195.0	12977.1	1.65%	12873.2	2.44%
C15	3265.0	3135.0	3.98%	3111.0	4.72%
C16	8105.0	7173.0	11.50%	7043.0	13.10%
C17	45773.5	45668.8	0.23%	45575.5	0.43%
D01	56858.4	55153.8	3.00%	55141.8	3.02%
D02	57206.3	55672.3	2.68%	55603.5	2.80%
D03	5552.3	5482.7	1.25%	5410.3	2.56%
D04	25707.0	24954.4	2.93%	24605.8	4.28%
D05	35480.0	35480.0	0.00%	35480.0	0.00%
D06	145661.0	144289.8	0.94%	144281.8	0.95%
D07	29321.0	28891.0	1.47%	28699.0	2.12%
D08	12773.0	12374.2	3.12%	12153.0	4.85%
D09	27575.0	23804.0	13.68%	22152.2	19.67%
D10	31091.3	30252.7	2.70%	29852.7	3.98%
D11	21782.2	21049.4	3.36%	20958.2	3.78%
D12	13654.5	13632.0	0.16%	13632.0	0.16%
D13	21090.0	20860.0	1.09%	20711.0	1.80%
D14	66098.3	65549.1	0.83%	65339.6	1.15%
D15	55578.6	55115.9	0.83%	54809.2	1.38%
D16	42335.3	41681.3	1.54%	41367.3	2.29%
D17	37135.0	36839.1	0.80%	36549.2	1.58%
D18	21729.0	21506.3	1.02%	21472.3	1.18%
D19	95574.3	93945.5	1.70%	93470.9	2.20%
D20	60525.7	60522.7	0.00%	60507.2	0.03%
D21	40095.6	38889.6	3.01%	38186.1	4.76%
D22	61060.0	60440.0	1.02%	60141.0	1.51%
D23	39680.5	39572.5	0.27%	39452.5	0.57%
D24	83025.2	82938.2	0.10%	82771.2	0.31%
D25	32720.1	32665.1	0.17%	32662.1	0.18%
D26	14320.9	14121.0	1.40%	14050.9	1.89%
D27	32459.3	32054.2	1.25%	31932.1	1.62%
D28	32928.0	32858.8	0.21%	32743.0	0.56%
D29	133596.6	132964.4	0.47%	132096.8	1.12%
D30	76360.8	76145.8	0.28%	76095.8	0.35%
D31	101460.0	90190.8	11.11%	83327.2	17.87%

Table 5.7.: Average improvement of objective function.

No.	initial solution		5 min.			30 min.		
	shift costs	utilisation	shift cost	improvement	utilisation	shift cost	improvement	utilisation
B01	19973.0	44.0%	19397.0	2.88%	45.1%	19329.2	3.22%	45.2%
B02	7179.0	28.7%	7063.2	1.61%	29.1%	7062.8	1.62%	29.1%
B03	22162.0	53.1%	22233.1	-0.32%	52.9%	21893.8	1.21%	53.7%
B04	27130.0	20.9%	25988.0	4.21%	21.8%	25766.0	5.03%	22.0%
B05	12960.0	34.7%	12592.0	2.84%	35.8%	12432.0	4.07%	36.2%
B06	6410.0	48.0%	5480.0	14.51%	56.1%	5388.0	15.94%	57.1%
B07	7720.0	48.7%	6822.0	11.63%	55.1%	6718.0	12.98%	56.0%
B08	13482.7	58.5%	13058.8	3.14%	60.4%	12902.1	4.31%	61.1%
B09	98900.5	48.1%	98966.7	-0.07%	48.0%	98978.0	-0.08%	48.0%
B10	40981.0	38.1%	40891.3	0.22%	38.2%	40761.4	0.54%	38.3%
B11	21037.0	49.9%	20917.5	0.57%	50.1%	20908.0	0.61%	50.2%
B12	20034.0	50.3%	18236.0	8.97%	55.0%	17747.6	11.41%	56.5%
B13	27260.0	47.1%	25719.6	5.65%	51.8%	24759.6	9.17%	53.7%
B14	7784.3	51.6%	7619.3	2.12%	52.8%	7592.5	2.46%	53.0%
C01	258484.5	54.5%	255083.3	1.32%	54.5%	254759.6	1.44%	54.6%
C02	73125.0	71.2%	72349.8	1.06%	71.2%	71535.4	2.17%	71.3%
C03	25508.0	51.7%	25473.2	0.14%	51.7%	25473.2	0.14%	51.7%
C04	24690.0	58.1%	24634.0	0.23%	58.2%	24435.0	1.03%	58.8%
C05	18622.0	54.4%	18249.6	2.00%	55.2%	18205.2	2.24%	55.3%
C06	29777.0	57.9%	29720.4	0.19%	57.9%	29624.8	0.51%	58.1%
C07	55680.0	59.8%	55680.0	0.00%	59.8%	55664.0	0.03%	59.8%
C08	45440.0	66.5%	45376.0	0.14%	66.6%	45280.0	0.35%	66.8%
C09	42000.0	56.0%	42000.0	0.00%	56.0%	42000.0	0.00%	56.0%
C10	25600.0	68.4%	25568.0	0.13%	68.5%	25520.0	0.31%	68.6%
C11	7280.0	59.0%	7280.0	0.00%	59.0%	7280.0	0.00%	59.0%
C12	13120.0	50.7%	13008.0	0.85%	51.1%	12928.0	1.46%	51.3%
C13	71168.0	34.1%	71168.0	0.00%	34.1%	71147.0	0.03%	34.1%
C14	13190.0	32.8%	12972.1	1.65%	33.3%	12868.2	2.44%	33.6%
C15	3260.0	19.0%	3130.0	3.99%	19.8%	3106.0	4.72%	20.0%
C16	8100.0	36.8%	7168.0	11.51%	41.6%	7038.0	13.11%	42.4%
C17	45768.5	37.6%	45663.8	0.23%	37.7%	45570.5	0.43%	37.8%
D01	56853.4	51.9%	55148.8	3.00%	53.6%	55136.8	3.02%	53.6%
D02	57201.3	54.0%	55667.3	2.68%	54.3%	55598.5	2.80%	54.4%
D03	5547.3	24.6%	5477.7	1.25%	25.0%	5405.3	2.56%	25.3%
D04	25702.0	63.9%	24949.4	2.93%	65.6%	24600.8	4.28%	66.5%
D05	25595.0	74.6%	25595.0	0.00%	74.6%	25595.0	0.00%	74.6%
D06	145656.0	64.9%	144284.8	0.94%	65.0%	144276.8	0.95%	65.0%
D07	29316.0	58.3%	28886.0	1.47%	59.2%	28694.0	2.12%	59.6%
D08	12768.0	34.0%	12369.2	3.12%	35.3%	12148.0	4.86%	36.0%
D09	12770.0	32.3%	11599.0	9.17%	35.5%	11447.2	10.36%	35.9%
D10	31091.0	52.5%	30252.4	2.70%	53.8%	29852.4	3.98%	54.5%
D11	21781.5	46.3%	21048.7	3.36%	47.8%	20957.5	3.78%	48.0%
D12	13652.5	94.4%	13630.0	0.16%	94.4%	13630.0	0.16%	94.4%
D13	21085.0	85.7%	20855.0	1.09%	86.8%	20706.0	1.80%	87.4%
D14	66098.3	78.8%	65549.1	0.83%	79.4%	65339.6	1.15%	79.7%
D15	55578.0	76.0%	55115.3	0.83%	76.7%	54808.6	1.38%	77.2%
D16	42330.3	63.3%	41676.3	1.54%	64.3%	41362.3	2.29%	64.8%
D17	37130.0	54.6%	36834.1	0.80%	55.1%	36544.2	1.58%	55.5%
D18	21724.0	76.5%	21501.3	1.03%	77.3%	21467.3	1.18%	77.4%
D19	72824.0	67.4%	71195.2	2.24%	68.9%	70720.6	2.89%	69.3%
D20	59523.3	98.1%	59520.3	0.01%	98.1%	59504.8	0.03%	98.3%
D21	40092.5	99.7%	38886.5	3.01%	99.8%	38183.0	4.76%	99.8%
D22	61055.0	76.3%	60435.0	1.02%	77.1%	60136.0	1.51%	77.4%
D23	39680.0	71.3%	39572.0	0.27%	71.5%	39452.0	0.57%	71.7%
D24	83025.0	84.8%	82938.0	0.10%	84.9%	82771.0	0.31%	85.0%
D25	32717.5	57.9%	32662.5	0.17%	58.0%	32659.5	0.18%	58.0%
D26	9818.0	35.5%	9618.0	2.04%	36.2%	9637.9	1.83%	36.1%
D27	32458.0	53.6%	32052.9	1.25%	54.3%	31930.7	1.62%	54.6%
D28	32923.0	83.8%	32853.8	0.21%	84.0%	32738.0	0.56%	84.3%
D29	96091.8	68.9%	95859.6	0.24%	69.1%	95492.0	0.62%	69.3%
D30	76357.7	78.5%	76142.7	0.28%	78.7%	76092.7	0.35%	78.8%
D31	13825.0	70.4%	13235.6	4.26%	75.0%	13349.8	3.44%	75.5%

Table 5.8.: Shift costs and utilisations.

No.	5 min.		30 min.	
	steps	tasks	steps	tasks
B01	434.3	5.2	525.4	6.4
B02	462.9	8.4	550.6	10.0
B03	673.6	5.6	910.1	7.8
B04	746.4	4.8	946.7	6.2
B05	646.0	12.4	939.0	18.2
B06	924.6	9.0	1123.3	11.4
B07	997.7	8.6	1222.3	10.4
B08	1111.9	22.6	1444.4	29.2
B09	497.4	2.6	572.4	3.8
B10	373.7	5.2	533.6	7.4
B11	463.4	6.0	613.6	8.4
B12	627.7	7.8	1007.3	12.4
B13	1814.9	8.8	2170.9	12.2
B14	655.0	9.6	880.0	12.8
C01	477.9	5.6	575.1	7.0
C02	737.7	4.0	993.7	5.2
C03	363.0	5.4	407.9	6.6
C04	275.7	4.0	407.4	4.6
C05	453.0	3.8	535.9	4.6
C06	325.9	3.2	484.4	4.0
C07	432.1	6.0	502.0	7.4
C08	411.1	6.2	505.3	7.8
C09	333.0	5.0	397.3	6.2
C10	362.3	6.8	454.3	9.0
C11	294.0	6.8	406.6	9.4
C12	275.7	5.0	378.6	7.2
C13	311.3	3.0	373.9	4.0
C14	510.1	7.2	682.4	9.6
C15	357.4	7.0	473.0	9.2
C16	603.4	6.2	805.9	9.4
C17	460.0	7.8	557.7	9.2
D01	760.7	12.2	771.1	12.4
D02	473.3	6.2	585.0	8.4
D03	421.1	9.6	486.3	10.2
D04	443.4	2.6	610.9	3.2
D05	136.7	2.0	182.0	3.0
D06	962.4	12.0	1050.3	13.8
D07	681.0	12.4	854.0	15.4
D08	637.1	12.0	874.4	16.2
D09	646.7	6.0	792.7	7.6
D10	958.4	7.8	1201.6	10.8
D11	612.1	6.4	782.9	9.2
D12	851.1	19.0	1122.3	25.2
D13	642.1	4.2	937.0	5.8
D14	607.0	2.2	866.6	3.2
D15	735.6	2.2	1134.7	3.2
D16	874.0	6.4	1132.6	8.0
D17	680.7	7.2	1022.4	9.6
D18	508.1	4.2	693.9	6.4
D19	945.0	8.8	1167.9	11.2
D20	162.3	4.0	222.6	4.6
D21	221.0	4.0	315.1	5.8
D22	1123.3	2.4	1419.1	3.8
D23	360.1	1.0	600.6	2.4
D24	465.7	1.0	565.3	1.0
D25	324.1	6.0	407.1	7.2
D26	452.6	9.6	544.1	11.8
D27	1336.7	9.4	1615.4	11.8
D28	149.1	1.0	251.3	1.0
D29	740.9	2.4	1261.9	3.6
D30	317.1	1.0	349.1	1.0
D31	457.7	2.2	872.0	4.6

Table 5.9.: Final steps and stepwidths.

which could be covered. As an example, large neighbourhood search assigns additional 63 tasks in scenario B13 and additional 14 tasks in scenario D31.

Furthermore, the initial algorithm is sometimes not able to avoid maximum restriction violations due to its heuristic nature. Such deficiencies are quickly resolved by local improvement. As an example, LNS avoids up to two qualification restriction violations on scenario D29 and ten violations on scenarios B03 and D09. Furthermore, on scenarios B09 and D29, one and two more maximum shift number restrictions are obeyed after local improvement.

Runtime information is given in Table 5.9. While on some large-scale scenarios, all improvement is attained by releasing one task at a time (scenarios D24 and D30), up to 31 tasks were relaxed in one run on scenario B08. The number of steps over the different 30-minute runs ranged from 124 (one run on D05) to 3079 (one run on B13).

In Table 5.10, we can study the development of task splits on scenarios comprising preemptive tasks. Figures indicate the average number of split parts per split task in the initial solution and after 30 minutes (averages over five runs). It can be seen that even if we do not penalise splitting, the average number of split parts consistently decreases by local improvement.

Table 5.11 compares total task overlaps (in minutes) in the initial solutions and after local

No.	before LNS	after LNS (30 min.)
B09	2.4	2.4
D04	2.9	2.5
D05	5.4	4.1
D13	2.2	2.2
D14	2.0	2.0
D15	4.8	2.5
D20	2.0	1.3
D22	4.0	4.0
D23	5.3	5.3
D24	4.0	4.0
D25	3.5	2.3
D26	7.9	7.9
D28	4.8	3.2
D30	3.4	3.1
D31	3.2	2.6

Table 5.10.: Average numbers of parts per split task.

No.	before LNS	after LNS (30 min.)
B14	2209	2703.0
D15	2990	279.6
D28	27	201.6
D29	64422	3842.0
D30	5399	1479.6

Table 5.11.: Changes in total overlapping.

improvement and application of the algorithm of Section 5.10; only scenarios allowing for overlapping are shown. As can be seen, changes in overlaps are highly problem-dependent. While on scenarios D15, D29 and D30, overlaps decrease by 90.6%, 94.0% and 72.6% on average, scenarios B14 and D28 use 22.4% and 646.7% more overlap after local improvement. On the latter scenarios, overlap is effectively used for a tighter packing of tasks in order to reduce shift costs.

The results of local improvement are generally very satisfactory. The algorithm has been implemented as a part of a commercial staff scheduling system and meets customer demands. Deficiencies of initial heuristic solutions are quickly levelled out, and additional improvement can be obtained by investing more runtime. Results of the combined construction/improvement algorithm are more robust, and the algorithm scales well to large problem instances. However, we will shortly mention some shortcomings of the local improvement scheme.

First, the algorithm has turned out to be sensitive to different kinds of shift models and shift costs. If many shift types are used, the algorithm can exploit more degrees of freedom to shorten or prolong shifts. If shift costs are proportional to shift lengths (which is often the case in practice), the algorithm yields much improvement already in small neighbourhoods. If in contrast only few shift types are present, local search must be applied to much larger neighbourhoods since improving the solution will often necessitate swaps of many tasks. If shift costs are independent from their lengths (e.g. if unit costs are used for all shifts), this means that improvement can

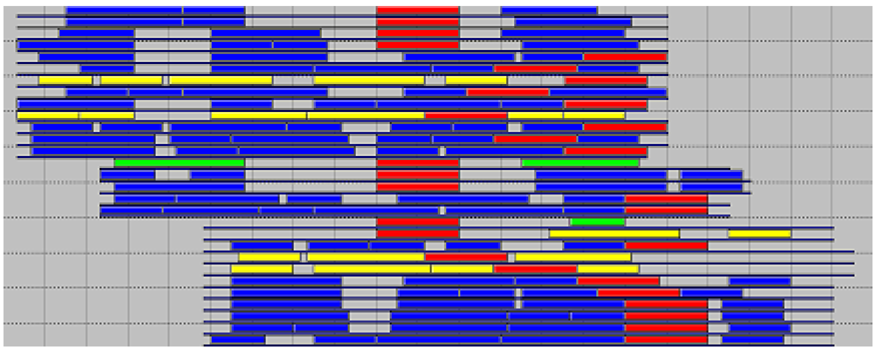


Figure 5.9.: Example from scenario C12: partial solution before improvement.

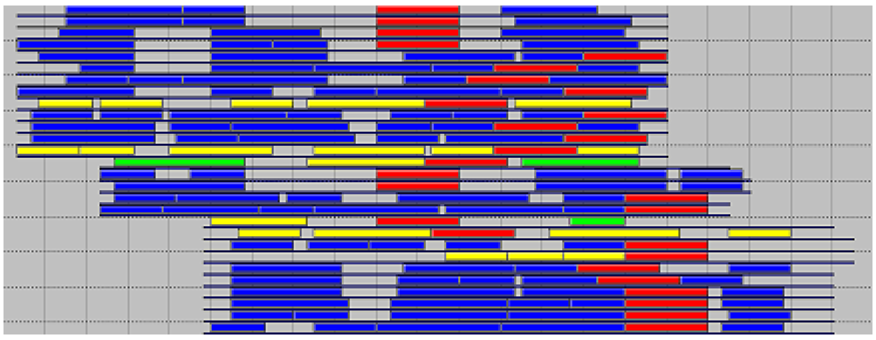


Figure 5.10.: Example from scenario C12: partial solution after improvement.

usually only be achieved by saving shifts.

As an example, scenario D17 uses shift costs which are proportional to shift lengths. In the above experiments, local search achieved average shift cost savings of 1.58% and increased shift utilisation by 0.9%. In an experiment with all shift costs set to 1, shift cost improvements within 30 minutes amounted to an average of 0.61% while utilisation was increased by only 0.2%. Clearly, this can be intentional if the number of shifts is the only optimisation criterion. In fact, the number of shifts was decreased by a maximum of one shift with proportional shift costs while up to four shifts were saved with the unit-cost model.

Another deficiency of large neighbourhood search lies in the asymmetry of task selection and reinsertion procedures: While relaxed tasks are optimised by powerful branch-and-bound search, their choice is based on simple heuristic rules. However, we may necessitate larger and more complex neighbourhoods in order to achieve more global improvement. However, these are generally not easy to find.

Figure 5.9 shows an extract of scenario C12. In order to save one shift in the given solution, all tasks which are marked yellow must be relaxed (tasks in shifts which are also involved in the local step are marked green, breaks have red colours). The result of the local step is given in Fig. 5.10. Note that the yellow tasks represent the minimum task set which is required for this local step. While the branch-and-bound procedure of Section 5.8 immediately finds the improvement, such complex neighbourhoods are not easy to determine. Preliminary experiments with LP-based relaxation strategies turned out to be successful, but required excessive computation times when iteratively applied for finding minimum neighbourhoods. A more efficient solution approach will be presented in the next chapter.

5.12. Conclusions and Future Research

We have tackled a complex task-level shift planning problem on airports. While most literature focuses on covering a demand curve of labour requirements, we have developed a mathematical description for shift planning on the level of single work tasks, integrating shift scheduling with vehicle routing. Task-level shift planning is an appropriate model for detailed operative planning, incorporating a multitude of constraints with regard to shifts and qualifications. Crew restrictions and preemptive tasks represent further outstanding features of the model.

We have shown how task-level shift planning can be incorporated in a constraint programming model. We have demonstrated how classical forward/backward schemes for temporal propagation can be extended to handle crews and preemptive tasks. Shift type flexibility was handled by introducing the notion of break rule days. Furthermore, we have shown how qualification and shift number restrictions can be dealt with.

We have described a constraint-based local improvement algorithm which builds upon solutions of a construction heuristic, resolving deficiencies of initial solutions while preserving advantageous features. Different strategies have been described for the choice of sets of tasks to be reoptimised. Reinsertion is then accomplished by restricted branch-and-bound search, exploiting problem-specific lower bounds. The combined construction-improvement approach offers the possibility of weighing runtime against solution quality. While good solutions even for large-scale scenarios are available within rather low runtimes, better solutions can be obtained if more runtime can be invested.

Experimental results have shown that the improvement algorithm is effective and efficient on a wide range of real-world test cases. Results of the initial algorithm could partly be improved considerably. The algorithm provides a remedy for flaws of the construction phase, including uncovered tasks and violations of shift number and qualification restrictions.

As a part of a commercial staff scheduling package, the algorithm is used by airlines, airports and ground handling companies. Because it provides locally optimal and robust solutions on a very general setting, the method has considerably increased customer satisfaction and improved the quality of shift plans.

Constraint programming has turned out to be a very flexible framework for complex shift planning. The improvement algorithm has already been generalised to different settings, including planning with fixed shifts and partially fixed assignments, optimising shift plans for likely flight delays, cross-utilising shift plans of different departments and checking given shift plans for restriction violations. The constraint model might further be enriched in the future, e.g. by additional qualification constraints, representing the skills of the workforce at hand.

While the algorithm has proven an invaluable tool in practice, a minor deficiency of large neighbourhood search is the asymmetric nature of its relaxation and reoptimisation strategies. Several enhancements are conceivable for the future. On the one hand, it may be promising to combine constraint-based search with tabu search or simulated annealing in order to overcome local optima with moderate neighbourhood sizes. Furthermore, the constraint model described in this chapter could be used for a new construction algorithm. An alternative approach, using integer programming global search, will be described in Chapter 7.

6. Column Generation and Branch-and-Price

The purpose of mathematical programming
is insight,
not numbers.
— Arthur M. Geoffrion

In the preceding chapters, we have used constraint programming to solve the workload levelling problem as well as complex task-level shift planning problems. Constraint programming is a very powerful technique for the handling of multitudes of constraints as well as linear and non-linear objective functions. In general, CP is very strong in representing feasibility aspects of discrete optimisation problems. In contrast, linear programming based methods focus on the optimisation aspect [Grönkvist, 2002]. In the following, we will introduce column generation and branch-and-price techniques which have become popular solution techniques in integer programming. The description will lay the ground for the algorithms of Chapters 7 and 9.

Column generation was originally proposed by Ford and Fulkerson [1958] and Dantzig and Wolfe [1960] and independently by Gilmore and Gomory [1961]. Apart from the original publications, Chvátal [1983], Barnhart et al. [1998] and Desrosiers and Lübbecke [2003] give excellent surveys of Dantzig-Wolfe decomposition, column generation and branch-and-price techniques. A good overview of implementation issues and speedup techniques can be found in Desaulniers et al. [2001]. Lübbecke and Desrosiers [2004] summarise some recent developments in column generation. Another valuable source of information is the PhD thesis of Lübbecke [2001].

The following exposition is mainly based on Chvátal [1983] and Barnhart et al. [1998]. First, the general decomposition idea is presented, leading to a column generation scheme for the implicit handling of large numbers of variables. Dantzig-Wolfe decomposition is shown to be particularly appropriate for block-structured linear programs. Furthermore, we describe how integrality can be achieved by branch-and-price. We shortly sketch the relationship between column generation and Lagrangian relaxation. Finally, some advanced performance issues are dealt with.

6.1. Dantzig-Wolfe Decomposition

Inspired by an idea of Ford and Fulkerson [1958], Dantzig and Wolfe [1960] presented a decomposition method for linear programs. The idea of Dantzig-Wolfe decomposition is to separate the constraint system into a set of complicated coupling constraints and easier-to-handle local constraints. The reformulation leads to a master program containing the complicated constraints while the local constraints are implicit in the newly defined variables. The master program has less constraints, but many more variables than the original compact formulation. The idea of column generation is to deal implicitly with the set of variables (see Ford and Fulkerson [1958]), restricting the master program to a subset of the variables and generating additional variables on demand via dual variable information. Gilmore and Gomory [1961] proposed a similar scheme for the cutting stock problem for which the column generation formulation arises naturally.

In the following, we will assume that we are given a minimisation problem. We therefore start from a linear program

$$\min c^T X \tag{6.1}$$

subject to

$$A^1 X \geq b^1 \tag{6.2}$$
$$A^2 X \geq b^2 \tag{6.3}$$
$$X \geq 0 \tag{6.4}$$

where $A^1 \in \mathbb{R}^{m_1 \times n}$ and $A^2 \in \mathbb{R}^{m_2 \times n}$ are coefficient matrices, $b^1 \in \mathbb{R}^{m_1}$, $b^2 \in \mathbb{R}^{m_2}$ the corresponding ride-hand sides and $c \in \mathbb{R}^n$ a cost vector. Program (6.1)-(6.4) will be called the *compact formulation*.

According to the theorems of Weyl and Minkowski (cf. Nemhauser and Wolsey [1988]), every point of a non-empty polyhedron $\mathcal{X} := \{X \in \mathbb{R}^n_+ \,|\, A^2 X \geq b^2\}$ can be expressed as a convex combination of the extreme points $\{x_q\}_{q \in Q}$ plus a nonnegative linear combination of the extreme rays $\{x_p\}_{p \in P}$ of $\mathcal{X}$:

$$\mathcal{X} = \left\{ \sum_{q \in Q} \lambda_q x_q + \sum_{p \in P} \lambda_p x_p \,\middle|\, \lambda_q \geq 0 \ \forall q \in Q, \lambda_p \geq 0 \ \forall p \in P, \sum_{q \in Q} \lambda_q = 1 \right\}$$

Note that the simplex method systematically enumerates basic feasible solutions (the extreme points of $\mathcal{X}$) or yields a basic feasible direction (an extreme ray of $\mathcal{X}$) of the solution space which is spanned up by its constraint system [Chvátal, 1983].

X thus observes $A^2 X \geq b^2$ if and only if

$$X = \sum_{q \in Q} \lambda_q x_q + \sum_{p \in P} \lambda_p x_p, \sum_{q \in Q} \lambda_q = 1$$

Representing the original variables X by the extreme points and extreme rays of the polyhedron defined by the constraint system $A^2 X \geq b^2$, the linear program (6.1)-(6.4) can be rewritten as

$$\min c^T \left(\sum_{q \in Q} \lambda_q x_q + \sum_{p \in P} \lambda_p x_p \right)$$

subject to

$$A^1 \left(\sum_{q \in Q} \lambda_q x_q + \sum_{p \in P} \lambda_p x_p \right) \geq b^1$$
$$\sum_{q \in Q} \lambda_q = 1$$
$$\lambda_q \geq 0 \quad \forall q \in Q$$
$$\lambda_p \geq 0 \quad \forall p \in P$$

or, rearranging the terms, as

$$\min \sum_{q \in Q} (c^T x_q) \lambda_q + \sum_{p \in P} (c^T x_p) \lambda_p \tag{6.5}$$

subject to

$$\sum_{q \in Q} (A^1 x_q) \lambda_q + \sum_{p \in P} (A^1 x_p) \lambda_p \geq b^1 \tag{6.6}$$

$$\sum_{q \in Q} \lambda_q = 1 \tag{6.7}$$
$$\lambda_q \geq 0 \quad \forall q \in Q \tag{6.8}$$
$$\lambda_p \geq 0 \quad \forall p \in P \tag{6.9}$$

which is called the *master program* associated with (6.1)-(6.4). We will denote its cost vector by $\hat{c}$, the constraint matrix (including the convexity constraint (6.7)) by $\hat{A}$ and the right-hand side by $\hat{b} := [b^1 \; 1]^T$. The decision variables of the master program are the λ_q and λ_p, and the columns of the constraint matrix correspond to extreme points and extreme rays of $\mathcal{X}$.

The master program involves less constraints, but many more variables than the compact formulation. However, the columns of (6.5)-(6.9) need not be fully enumerated when using the revised simplex method [Orchard-Hays, 1968]. Instead, we can use a *restricted master program* (RMP) which contains only a subset of the columns and variables of the master program. Additional columns will be implicitly priced out by a *subproblem* (or *pricing problem*), and columns with negative reduced costs will be added to the restricted master program.

According to the simplex criterion, we must choose an entering column with negative reduced costs in each iteration. In the optimal solution to a given restricted master program, let $\pi \in \mathbb{R}^{m_1}$ denote the dual values associated with constraints (6.6) and μ the dual value of the convexity constraint (6.7). To find an entering column $\hat{a}$ of $\hat{A}$, the following subproblem is solved:

$$\min \, (c - \pi A^1)X \tag{6.10}$$

subject to

$$A^2 X \geq b^2 \tag{6.11}$$
$$X \geq 0 \tag{6.12}$$

If the subproblem has an optimal solution x^* such that $(c - \pi A^1)x^* - \mu < 0$, x^* is a basic feasible solution with negative reduced costs and corresponds to one of the extreme points x_q. The column

$$\hat{a} := \begin{pmatrix} A^1 x^* \\ 1 \end{pmatrix} \tag{6.13}$$

of $\hat{A}$ can thus be added to the RMP with the corresponding cost coefficient in $\hat{c}$ set to cx^*. In the master program, $\hat{a}$ can be used as an entering column.

If the subproblem (6.10)-(6.12) is unbounded, we find a basic feasible direction $\tilde{x}$ such that $(c - \pi A^1)\tilde{x} < 0$. $\tilde{x}$ then corresponds to one of the extreme rays x_p, and the column

$$\hat{a} := \begin{pmatrix} A^1 \tilde{x} \\ 0 \end{pmatrix} \tag{6.14}$$

of $\hat{A}$ can be added to the restricted master program with cost coefficient $c\tilde{x}$. Because its reduced costs are negative, it can be used as an entering column.

If finally the problem (6.10)-(6.12) has an optimal solution x^* of value $(c - \pi A^1)x^* - \mu \geq 0$, we know that each extreme point x_q satisfies $(c - \pi A^1)x_q - \mu \geq 0$, and every extreme ray x_p obeys $(c - \pi A^1)x_q \geq 0$. Since every column $\hat{a}$ of $\hat{A}$ follows either form (6.13) or (6.14), there are no columns with negative reduced costs, and the solution of the restricted master program is optimal.

Section 7.4 will describe a subproblem whose polyhedron is only spanned up by extreme rays, i.e. it does not have non-trivial extreme points. Then upper bounds can be introduced on the variables of the subproblem, cutting the rays. Note that any multiple of an extreme ray represents the same ray.

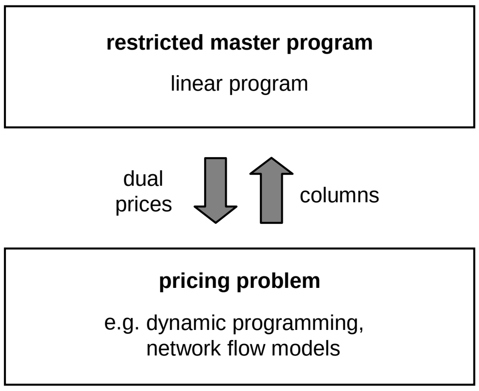

restricted master program
linear program
dual prices
columns
pricing problem
e.g. dynamic programming, network flow models

Figure 6.1.: Column generation process.

When solving model (6.5)-(6.9) by column generation, we start from a restricted master program which guarantees a feasible solution. The two-phase simplex algorithm can be used for this purpose, or a known heuristic solution can be entered as a starting solution [Barnhart et al., 1998] [Desrosiers and Lübbecke, 2003]. Note that the initial dual values which are used for column generation will be "better" when a good starting solution is known. Alternatively, a good dual solution can be used to start the column generation process, see du Merle et al. [1999], Lübbecke and Desrosiers [2004].

The restricted master program is then solved to optimality, and the optimal dual values are transferred to the subproblem. If a column with negative reduced costs can be found, it is added to the RMP. The restricted master program is then reoptimised, resulting in new dual values for the subproblem. The algorithm terminates when no more columns with negative reduced costs are found. Fig. 6.1 shows the iteration between master program and subproblems.

It is noteworthy that it is not necessary to find the column with most negative reduced costs as proposed by the Dantzig rule. Instead, any column with negative reduced costs is a candidate for improving the solution and can thus be added to the master program. Column generation algorithms tend to converge faster if several negative reduced cost columns are added in each iteration [Desrosiers et al., 1984].

6.2. Block-Structured Linear Programs

Clearly, the separation of the constraint matrix into parts A^1 and A^2 should be guided by the problem structure. Dantzig-Wolfe decomposition is often applied to block-angular linear programs in which the constraint matrix exhibits the following structure:

$$\begin{pmatrix} A^{1,1} & A^{1,2} & \cdots & A^{1,K} \\ A^{2,1} & & & \\ & A^{2,2} & & \\ & & \ddots & \\ & & & A^{2,K} \end{pmatrix}$$

$(A^{1,1} \dots A^{1,K})X = b^1$ are then called *coupling* (or *linking*) constraints while the subproblem defined by the block-diagonal matrix A^2 decomposes into separate problems for each $A^{2,k}$. In fact, each of the constraint systems $A^{2,k}X^2 \geq b^{2,k}$ defines a distinct polyhedron on the set X^k of decision variables corresponding to $A^{2,k}$ (and $b^{2,k}$ the corresponding right-hand side). Each such polyhedron can be separately represented by its extreme points $\{x_q^k \,|\, q \in Q^k\}$ and extreme rays $\{x_p^k \,|\, p \in P^k\}$, and the master program will contain one convexity constraint $\sum_{q \in Q^k} \lambda_q^k = 1$ for each subproblem k. If the restrictions on all subsets are equal (i.e. $A^{2,k_1} = A^{2,k_2}, b^{2,k_1} = b^{2,k_2}\, \forall k_1 \neq k_2$), only one subproblem has to be solved, and the convexity constraints generalise to $\sum_{k=1}^{K} \sum_{p \in P^k} \lambda_p^k = K$ [Barnhart et al., 1998].

As an example, the multicommodity flow problem (see e.g. [Ahuja et al., 1993]) gives rise to a block-structured linear program. It was this problem which originally inspired Ford and Fulkerson [1958] to propose dealing implicitly with the columns of a master program. As Desrosiers and Lübbecke [2003] point out, many problems in vehicle routing and crew rostering have multicommodity flow problems as an underlying structure. Other applications give rise to block-structured linear programs as well, see Lübbecke and Desrosiers [2004] for an overview.

In the context of block-structured programs, the decomposition principle has an interesting economic interpretation as decentralised planning with incomplete information at the centre, see Lasdon [1970] and Chvátal [1983].

While Dantzig and Wolfe originally supposed the subproblems to be solved by linear programming, column generation does not prescribe the pricing method. Subproblems can often be naturally formulated as shortest path problems or resource-constrained shortest path problems which are usually solved by dynamic programming or network flow algorithms [Vanderbeck, 2000] [Desaulniers et al., 2001].

Not always is column generation the result of applying Dantzig-Wolfe decomposition. As an example, Gilmore and Gomory [1961] originally proposed their column generation scheme for the cutting stock problem without any reference to a compact formulation. Villeneuve et al. [2003] have recently shown that under relatively mild conditions, a compact formulation exists for every column generation model. However, the compact formulation does not need to be unique, cf. Valério de Carvalho [2002]. Even if a column generation formulation is sometimes more natural, the compact formulation can give additional insight for the integer solution of column generation models as will be seen in the sequel.

Column generation models often avoid problem symmetries which are difficult to circumvent in the compact formulation [Barnhart et al., 1998]. As an example, the straightforward compact formulation for the cutting stock problem which is due to Kantorovich [1960] exhibits strong symmetry which is not present in the column generation formulation [Valério de Carvalho, 2002]. Another example for avoiding symmetries by column generation will be given by the rostering application in Chapter 9.

Another argument for column generation models is the quality of the LP lower bound which is particularly important when solving integer models by branch-and-bound. Since column generation models result from a reformulation, their LP bounds are at least as good as in the compact formulation. For many problems, the column generation bounds are substantially tighter due to the incorporation of integrality constraints in the subproblem. Again, the cutting stock problem provides a good example: While the Kantorovich model provides a poor LP bound (as shown by Valério de Carvalho [2002]), the column generation formulation is known to be very tight, providing an appropriate basis for exact solution approaches by branch-and-bound, see e.g. Vance et al. [1994]. In fact, the subproblems of the cutting stock problem are solved as knapsack problems, meaning that the column generation only admits linear combinations of integer solutions to the subproblems. If however the subproblem is naturally integer, i.e. the LP relaxation only yields integer solutions, the bounds of the compact and column generation formulation coincide [Geoffrion, 1974].

6.3. Branch-and-Price

So far, we have focused our attention on the decomposition of linear programs which often arise as LP relaxations of discrete optimisation problems. When Dantzig-Wolfe decomposition is applied to integer programs which are to be solved by LP-based branch-and-bound, it does not suffice to generate columns at the root node. The initial column set may not provide an optimal integer solution or even an integer-feasible solution. We must therefore generate columns throughout the search tree [Desaulniers et al., 2001]. It is interesting to note that not only is column generation required after branching, but branching may also be required in order to generate all potential variables in the subproblem, see Villeneuve et al. [2003] for an example. The combination of branch-and-bound with column generation is known as *branch-and-price* or *integer column generation* [Barnhart et al., 1998] [Vanderbeck and Wolsey, 1996] [Vanderbeck, 2000].

The first application of branch-and-price for a routing problem was presented by Desrosiers et al. [1984]. Since this first publication, many authors have described successful branch-and-price implementations, see Lübbecke and Desrosiers [2004] for a survey. Application areas include routing and scheduling (see Desrosiers et al. [1984], Desrochers et al. [1990], Dumas et al. [1991], Desrochers et al. [1992] and for an overview Desrosiers et al. [1993]), the aforementioned cutting stock problem (e.g. Vance et al. [1994]), crew pairing and rostering (e.g. Desrochers and Soumis [1989], Ryan [1992], Vance et al. [1997], Gamache et al. [1999]), tour scheduling (e.g. Mason and Smith [1998], Mason and Nielsen [1999], Mason [1999]) and generalised assignment (Savelsbergh [1997]).

A main focus of branch-and-price research is on the design of branching rules. Appelgren [1969] was the first to note that classical variable branching is not compatible with column generation. Assume that in a binary problem (like e.g. crew scheduling), a variable X_j takes on a fractional value $0 < f < 1$ in the relaxed optimal solution. Branching by variable dichotomy would then enforce $X_j = 0$ on one branch and $X_j = 1$ on the other. The 0 branch corresponds to forbidding the variable to be part of the solution. In column generation, however, the subproblem is likely to create this variable again after branching. This clearly has to be prevented. After n branching decisions, n variables must be forbidden if we follow the 0 branches. This is generally not easy to represent in the subproblem structure. If e.g. the subproblem is a shortest path problem, we must forbid whole paths to be generated. Variable branching therefore destroys the subproblem structure and is generally regarded incompatible with column generation [Barnhart et al., 1998].

One of the most frequently applied branching rule in integer column generation is Ryan-Foster constraint branching [Ryan and Foster, 1981] which applies to binary set partitioning problems. Applications of set partitioning models include the aforementioned crew scheduling, cutting stock and vehicle routing problems.

Let A be the binary constraint matrix of the restricted master program of a set partitioning formulation, and suppose that the basic solution λ to $A\lambda = 1, \lambda \geq 0$ is fractional. Then there exist two rows r, s such that

$$0 < \sum_{k:a_{rk}=1,a_{sk}=1} \lambda_k < 1,$$

see Ryan and Foster [1981] for a proof. When we have determined two such rows r, s, we can enforce $\sum_{k:a_{rk}=1,a_{sk}=1} \lambda_k = 0$ on one of the branches (e.g. the left branch) while on the right branch, we set $\sum_{k:a_{rk}=1,a_{sk}=1} \lambda_k = 1$. Consequently, we enforce the two items corresponding to r and s to be covered by the same variable on the right branch while on the left branch, r and s must be covered by different variables. In general, this rule can be easily transferred to the subproblem by an adaptation of the underlying graph structure, see Barnhart et al. [1998].

Applications of this branching rule can e.g. be found in Desrochers and Soumis [1989], Dumas et al. [1991], Ryan [1992], Anbil et al. [1992] and Vance et al. [1994]. The basic idea of Ryan-

Foster branching can also be transferred to set covering problems if only rows are considered for which the covering constraints are fulfilled at equality, see Grönkvist [1998]. Generalisations to non-binary integer programs and general right-hand sides can be found in Vanderbeck and Wolsey [1996] and Vanderbeck [2000]. However, these integer branching rules suppose the subproblems to be solved by mixed integer programming which can be very costly.

More generally, it has been proposed to branch on the variables of the compact formulation, see e.g. Gamache et al. [1999] and Villeneuve et al. [2003]. In fact, the original variables are part of the subproblem, meaning that branching decisions on these variables can be easily transferred to the subproblem without destroying its structure. Ryan-Foster branching is a special case of this general rule when the rows r, s correspond to adjacent nodes in a flow formulation with branching applied to the variable representing the flow between r and s.

6.4. Lower Bounding

Assuming that we are given a minimisation problem, the solution value at a branch-and-price node only represents a lower bound if no more columns with negative reduced costs can be found, i.e. when we have solved the LP relaxation. Lasdon [1970], Farley [1990] and Vanderbeck and Wolsey [1996] describe lower bounds on the final objective function which can be calculated *before* column generation has terminated. As we will see, these lower bounds can be used to terminate column generation even if negative reduced cost columns are still available.

The lower bound of Lasdon [1970] does not make special assumptions on the type and structure of the linear program. However, Lasdon's description only applies to the case of different restrictions on the subsets, i.e. $A^{2,k_1} = A^{2,k_2}, b^{2,k_1} = b^{2,k_2}\, \forall k_1 \neq k_2$ (see Section 6.2). Furthermore, Lasdon assumed that convexity constraints $\sum_{q \in Q^k} \lambda_q^k = 1$ are present in the master program which is not the case if the subproblem polyhedra are only spanned up by extreme rays. In the following, we will show how the Lasdon bound can be generalised to the case of identical restrictions on the subsets when an upper bound on the sum of the variables is known.

We therefore formulate the master program as

$$\min z = \sum_{k \in K} \sum_{p \in P} c_p^k \lambda_p^k \tag{6.15}$$

subject to

$$\sum_{k \in K} \sum_{p \in P^k} \hat{a}_p^k \lambda_p^k = b \tag{6.16}$$

$$\lambda_p^k \geq 0 \tag{6.17}$$

with $\hat{a}_p^k$ denoting the master program column corresponding to variable λ_p^k. Furthermore, we assume that we know an implicit bound L on the sum of the variables:

$$\sum_{k \in K} \sum_{p \in P^k} \lambda_p^k \leq L \tag{6.18}$$

We multiply the constraints (6.16) by their dual values $\pi \in \mathbb{R}^m$ and subtract their sum from (6.15), giving

$$z - \pi b = \sum_{k \in K} \sum_{p \in P^k} \lambda_p^k (c_p^k - \pi b) \tag{6.19}$$

The value $c_p^k - \pi b =: \bar{c}_p^k$ is exactly the reduced cost of variable p in subproblem k. The k'th subproblem in fact calculates $\min_{p \in P^k} \bar{c}_p^k$. Using this minimum in (6.19), we obtain

$$\begin{aligned} z - \pi b &\geq \sum_{k \in K} \sum_{p \in P^k} \lambda_p^k \left(\min_{p \in P^k} \bar{c}_p^k \right) \\ \Longrightarrow z - \pi b &\geq \sum_{k \in K} \left(\min_{p \in P^k} \bar{c}_p^k \right) \sum_{p \in P^k} \lambda_p^k \\ \Longrightarrow z - \pi b &\geq \left(\min_{\substack{k \in K \\ p \in P^k}} \bar{c}_p^k \right) \sum_{k \in K} \sum_{p \in P^k} \lambda_p^k \end{aligned}$$

where $\min_{\substack{k \in K \\ p \in P^k}} \bar{c}_p^k$ is easily calculated by a minimisation over the optimal solutions of the subproblems. Using the upper bound L (equation (6.18)), we can deduce

$$z \geq \left(\min_{\substack{k \in K \\ p \in P^k}} \bar{c}_p^k \right) L + \pi b$$

Because this inequality holds for every value of z which can be obtained from (6.15)-(6.17), it is also valid for the minimum value of z. We therefore have

$$\min z \geq \left(\min_{\substack{k \in K \\ p \in P^k}} \bar{c}_p^k \right) L + \pi b \tag{6.20}$$

From LP duality theory, we know that πb is the objective function value z_B of the current basis. Inequality (6.20) therefore expresses that

$$z_{lb} := z_B + \left(\min_{\substack{k \in K \\ p \in P^k}} \bar{c}_p^k \right) L$$

is a lower bound on the final objective function which can be easily calculated from the data which is available at each column generation iteration.

Assume that the objective function value of an integer solution to (6.15)-(6.17) is always integer. We can then terminate column generation as soon as $\lceil z_{lb} \rceil \geq z_B$ because generating further columns will not improve the lower bound information at the current node. This sometimes help to avoid the so-called *tailing-off effect* which describes the slow convergence of column generation algorithms near the optimum [Lübbecke and Desrosiers, 2004]. If we know an upper bound z_{ub} on the optimal objective function value of (6.15)-(6.17), we can furthermore prune the current branch if $\lceil z_{lb} \rceil \geq z_{ub}$.

Farley [1990] proposes another lower bound for a special class of linear programs with only $\geq$ constraints and non-negative objective function coefficients. The bound of Vanderbeck and Wolsey [1996] is a minor strengthening of Lasdon's bound for identical subproblems when lower and upper bounds on the decision variables are known.

6.5. Lagrangian Relaxation

While column generation is an intrinsically primal method, Lagrangian relaxation can be regarded an equivalent dual method which is often used to solve integer programs [Geoffrion, 1974]

[Lemaréchal, 2001]. Several researchers have compared column generation and Lagrangian relaxation, see e.g. Lübbecke and Desrosiers [2004].

The idea of Lagrangian relaxation applied to integer programs is to relax a set $A^1X \geq b^1$ of constraints which are integrated into the objective function, weighted by a vector $u \geq 0$ of *Lagrange multipliers*. This results in the *Lagrangian subproblem*

$$L(u) := \min_{X \in \mathcal{X}} c^T X - u^T(A^1X - b^1) \tag{6.21}$$

with $\mathcal{X} := \{X \mid A^2X \geq b^2, X \geq 0, X \text{ integer}\}$. Solving (6.21) is algorithmically equivalent to solving the subproblem described above with the dual values replaced by Lagrange multipliers.

$L(u)$ is a lower bound on the optimal value z^* of the above problem since $L(u) \leq \min(c^T X - u^T(A^1X - b^1) \mid A^1X \geq b^1) \leq z^*$. The best lower bound can be found by solving the *Lagrangian dual problem*:

$$\mathcal{L} := \max_{u \geq 0} L(u) \tag{6.22}$$

Imagine that the best set u^* of dual multipliers has been determined by solving (6.21). According to the definition of the Lagrangian subproblem, we have $A^2X \geq b^2$ and by complementary slackness, it can be shown that $u^{*T}(A^1X - b^1) = 0$. We only have to check that $A^1X \geq b^1$. If one of these constraints is violated, primal feasibility must be recovered by some appropriate repair algorithm [Desrosiers and Lübbecke, 2003].

The Lagrangian dual (6.22) is most often solved by subgradient optimisation which is simple and easy to implement, see e.g. Geoffrion [1974]. More recent proposals include bundle methods [Hiriart-Urruty and Lemaréchal, 1993] and the analytic centre cutting plane method [Goffin and Vial, 2002]. Alternatively, imagine that $\mathcal{X}$ is replaced by conv$(\mathcal{X})$ which does not change the optimal solution value. This turns the Lagrangian dual into a linear program. In fact, the Lagrangian dual will yield the same value as the linear program above [Geoffrion, 1974], and the optimal Lagrange multipliers will correspond to the dual variables of the restricted master program [Barnhart et al., 1998].

Lagrangian relaxation and column generation are therefore equivalent, and the choice for one of these methods is mainly due to performance issues. While for a long time, Lagrangian relaxation has found wide-spread use, column generation is nowadays considered superior due to the advent of efficient simplex codes [Barnhart et al., 1998]. However, there are approaches to combine column generation with Lagrangian relaxation [van den Akker et al., 2002], and dual solutions obtained by Lagrangian relaxation can be useful to start column generation [Lübbecke and Desrosiers, 2004].

6.6. Convexification versus Discretisation

In Dantzig-Wolfe decomposition, the solution space of a constraint subset is represented by a convex-linear combination of the extreme points and rays. When solving integer programs, we will usually claim the subproblem solutions (i.e. the extreme points and rays) to be integer. The underlying solution space is then a *convexification* of all integer feasible solutions [Lübbecke and Desrosiers, 2004]. The optimal integer solution can be situated in the interior of the solution space, being a convex-linear combination of the extreme points and rays. We will therefore only require integrality on the original variables while the derived variables λ_q and λ_p can still be fractional.

Imagine the multicommodity flow problem with the subproblem being a shortest path problem (see e.g. Assad [1978]). The elementary paths generated by the subproblem are combined to complex flows in the master program. Integrality is only required on the original flow variables. If we are interested in the individual paths, these integral flows can be recomposed according to the flow decomposition theorem (e.g. Ahuja et al. [1993]). Note that these derived paths need not be identical with the paths which are generated as subproblem solutions.

In other problems however, we may want the new variables λ_q and λ_p to be integer. As pointed out by Vanderbeck [2000], the resulting program is not equivalent to the convex program defined above. As opposed to convexification, the decomposition will be based on a *discretisation* of the solution space. Effectively, the subproblem must be able to generate interior integer points of the solution space. The decomposition is then based on the following theorem (see Nemhauser and Wolsey [1988]):

Theorem 2. *Let $\mathcal{P} := \{X \in \mathbb{R}^n \mid A^2X \geq b^2, X \geq 0\} \neq \emptyset$ and $\mathcal{X} = \mathcal{P} \cap \mathbb{Z}^n$. Then there exists a finite set of integer points $\{x_q\}_{q\in Q}$ and a finite set of integer rays $\{x_p\}_{p\in P}$ of $\mathcal{P}$ such that*

$$\mathcal{X} = \left\{ \sum_{q\in Q} x_q\lambda_q + \sum_{p\in P} x_p\lambda_p \,\middle|\, \sum_{q\in Q} \lambda_q = 1, \lambda \in \mathbb{N}_0^{|Q|+|P|} \right\}$$

This theorem is the integer analogon of the theorems of Weyl and Minkowski, and the decomposition based on discretisation is analogous to Dantzig-Wolfe decomposition. In the special case of an empty set of integer rays, all of the λ_q, $q \in Q$ will be binary, and solving the master program is equivalent to selecting one of the subproblem solutions. Note that as long as the linear programming relaxation is solved, convexification and discretisation coincide. Furthermore, if the original variables are binary, both approaches are equivalent since the solution space does not include interior points [Lübbecke and Desrosiers, 2004].

While the distinction between convexification and discretisation is theoretically important, it is often neglected in practice. Effectively, enforcing integrality on the original variables often entails integrality on the derived variables. The design of branch-and-price algorithms mostly aims at developing branching rules which are empirically sufficient in order to produce integer solutions, see Mason [1999] for an example.

6.7. Advanced Performance Issues

In recent years, several techniques have been proposed to speed up column generation and branch-and-price algorithms, see Desaulniers et al. [2001] for a survey. Some approaches have already been mentioned, e.g. using heuristic primal or dual solutions as a starting point. Furthermore, we have described in Section 6.4 how lower bounds can be used for early termination of column generation.

A further speedup idea consists in fixing the original variables of the compact formulation by using lower and upper bounds. As Desrosiers and Lübbecke [2003] note, dual variable information on the original variables which is not available in the master program can be obtained from the ultimate solution of the pricing problem before reaching the optimum. Fixing original variables corresponds to reduced-cost fixing in linear programming, see Padberg and Rinaldi [1991].

Another approach which has newly attracted attention in general mixed integer programming is the use of cutting planes, see Johnson et al. [2000], Martin [2001] and Marchand et al. [2002]. The combined use of cutting planes and branching is known as *branch-and-cut*, and adding column generation results in *branch-and-price-and-cut* approaches. Cutting planes generally help in finding good lower bounds [Barnhart et al., 1998]. Up to now, only little publications on branch-and-price-and-cut applications have been published, see Nemhauser and Park [1991], Kohl et al. [1999] and Barnhart et al. [2000]. As Irnich [2002] notes, all of these approaches restrict cutting planes to be separated after pricing has converged. Combining pricing and cutting plane separation in a more flexible way (e.g. including lifting of cutting planes [Marchand et al., 2002]) will certainly be subject of future research.

Column generation corresponds to cut generation in the dual program and is thus equivalent to Kelley's cutting plane method for the dual model [Kelley, 1960]. Valério de Carvalho [2002] has proposed to introduce additional dual cuts, i.e. cuts which are valid for the dual model, representing variables of the primal formulation. In an application for the cutting stock problem, he has shown how dual cuts with an intuitive interpretation speed up column generation. While the definition of dual cuts depends on the problem, it may be possible to transfer this idea fruitfully to other applications.

It has been observed that dual values often do not converge smoothly, and oscillations prevent column generation from a fast convergence [Lübbecke and Desrosiers, 2004]. This has led to the proposition of different *stabilisation* techniques. The idea of the BOXSTEP method of Marsten et al. [1975] is to restrict dual values to a box around their current values. If the new duals are situated on the boundary, the box is relocated. Otherwise, the optimum has been reached in the interior of the box. Wentges [1997] and Neame [2000] stabilise the evolution of dual values by using a convex combination of the current dual values and the values of the preceding iteration. Du Merle et al. [1999] propose a combination of primal and dual strategies. As in the BOXSTEP approach, a box is put around the dual values; exceeding the bounds is allowed but penalised. These penalties translate into perturbations for the primal constraints, and violating the constraints is penalised as given by the limits on the dual variables. By dynamic updates, deviations from the original model are driven out of the solution.

As column generation and branch-and-price algorithms are active fields of research, further acceleration ideas will certainly come up in the future.

7. Optimal Shift Planning by Branch-and-Price

That is the trouble with flying:
We always have to return to airports.
Think of how much fun flying would be
if we didn't have to return to airports.
— Henry Minizburg

In Chapter 5, we have introduced complex task-level shift planning which combines elements from shift scheduling and vehicle routing. The problem was complicated by preemptive tasks as well as crew and qualification constraints. In this chapter, a subclass will be tackled, consisting in covering a given set of non-preemptive tasks with fixed start times by appropriate shift duties. Simplified shift planning therefore combines classical shift scheduling with elements from vehicle scheduling and can be interpreted as a specialised multiple depot vehicle scheduling problem (MDVSP). We will show that simplified shift planning is $\mathcal{NP}$-hard in the strong sense which also proves that general task-level shift planning is $\mathcal{NP}$-hard. Furthermore, we will use Dantzig-Wolfe decomposition to derive a branch-and-price algorithm which solves many real-world test cases to proven optimality while consuming moderate computation times.

7.1. Introduction

General task-level shift planning as described in Chapter 5 is a complex optimisation problem. It can involve movable tasks, preemptive tasks, crew constraints as well as qualification and shift-level restrictions. The improvement algorithm of Chapter 5 was designed to tackle all of these constraints, providing a means for generating robust solutions for a multitude of scenarios. While all of these features can be helpful in general airport ground handling, practical shift planning often involves only a small subset of the modelling aspects.

As an example, many planning scenarios comprise only tasks which are fixed in time. Task splitting is often helpful for the planning of passenger services (like check-in, boarding and ticketing) while on the apron, handling tasks usually have short durations, and task interruptions are not desired. Similarly, crew planning is often used for aircraft cleaning personnel while most other airports services can be scheduled without any parallelism constraints. Finally, severe qualification restrictions are usually only imposed in the planning of highly specialised services like operations departments.

This justifies the search for specialised algorithms for a smaller class of shift planning problems, exploiting special structures while still being sufficiently general. In this chapter, we will restrict our attention to shift planning with tasks which are non-preemptive and fixed in time. Furthermore, task overlapping, crew planning, qualifications as well as break buffers and relative shift number restrictions will not be supported.

The resulting shift planning problem thus consists in covering tasks of given start times and lengths by a cost-minimal set of shifts which can be of any of the given types. Note that while work tasks are fixed in time, we can have several movable breaks in each shift. Travel times will be considered, and absolute minimum and maximum restrictions can be imposed on arbitrary sets of shift types.

While the general shift planning of Chapter 5 is based on the vehicle routing problem with time windows (VRPTW), the fixed task case is called the vehicle scheduling problem (VSP) [Desrosiers et al., 1993]. Additionally to the basic VSP setting, shift types restrict the start and end times of tours. Each shift type can be regarded as a depot, imposing temporal restrictions on the departures and arrivals of vehicles. Fixed task shift scheduling can thus be regarded a special case of the multiple depot vehicle scheduling problem (MDVSP).

In Chapter 5, it was shown that some shift planning instances are difficult to tackle by local improvement. This is especially the case if few shift types are used or shift costs are independent from their lengths. Then very large neighbourhoods must be reoptimised in order to achieve improvements, and improving sets of tasks are difficult to find. Our goal will be the development of an algorithm overcoming these deficiencies which are typical of local search approaches.

The local improvement algorithm presented in Chapter 5 did not make assumptions on the structure of the problem instances. In fact, the complexity of local steps depends more on the neighbourhood size than on the scale of the overall scenario. In practice, a considerable number of airports must obey night flying restrictions. Additionally, there may be time periods of low workload within the days. In both cases, realistic scenarios may naturally decompose into separate days or even smaller planning horizons. This can be exploited by decomposition, making the separate problems amenable to an exact solution approach.

The algorithms of Chapters 4 and 5 were based on constraint programming. As described in Section 3.1, CP has its origins in the solution of constraint satisfaction problems. With a focus on feasibility aspects, it is a technique which is appropriate for problems which are strongly constrained or which involve non-linear restrictions or objectives. In contrast, linear programming which is the predominant optimisation method in the Operations Research community is especially powerful in tackling cost aspects while it is limited in the incorporation of complex and non-linear side constraints. For extensive comparisons of strengths and weaknesses of CP and LP approaches, the reader is referred to Heipcke [1999] and Lustig and Puget [2001].

In this chapter, an integer programming approach for the simplified shift planning problem is developed. It will be shown how the structure of a compact network flow formulation of the simplified shift planning problem can be exploited by Dantzig-Wolfe decomposition. The resulting column generation model will be solved by repeated iterations between a master program and subproblems. By branching on the original flow variables, integer solutions can be efficiently obtained in a way which is compatible with the subproblems. In order to generate optimal solutions, column generation is applied throughout the search tree. As will be shown, real-world shift planning problems often decompose into several components which can be optimised separately. Even if simplified shift planning is shown to be $\mathcal{NP}$-hard, the branch-and-price approach is able to provide optimal solutions for many real-world test cases in reasonable times.

Throughout the chapter, basically the same notations as in Chapter 5 will be used. Each work task $i \in I$ will be given along with a start time $a_i = b_i$ and a length l_i. Between two tasks i, j, a travel time of $d_{i,j}$ time units must be performed. Travel times can be asymmetric, but are assumed to obey the triangle inequality. Shifts start and end at a central depot, and breaks take place at the location of their predecessor work task. Day indices n will be omitted from the formulation, assuming that given problems span over only one day. The generalisation to several days is straightforward and will be presented in Section 7.7. A shift type $k \in K$ will thus be characterised by a start time st_k, an end time et_k and costs c_k.

An absolute shift number restriction $r \in R_{abs}^{min}$ ($r \in R_{abs}^{max}$) imposes a minimum (maximum) limit of $m_r \in \mathbb{N}$ shifts on a shift type set K_r. For ease of exposition, we will assume that all shift number restrictions are hard constraints and can be obeyed. Imposing penalties for exceeding maximum limits as in Chapter 5 will be straightforward. We will furthermore suppose that all tasks can be covered by shifts.

The presentation will be structured as follows: Section 7.2 shows that simplified shift planning is $\mathcal{NP}$-hard. Section 7.3 presents a flow formulation for simplified shift planning which is submitted to Dantzig-Wolfe decomposition in Section 7.4. In Section 7.5, it is shown how the reformulated problem can be solved by column generation. Integer branching is the subject of Section 7.6. Section 7.7 shows how realistic shift planning scenarios can often be decomposed into independent optimisation problems. Experimental results are given in Section 7.8, and a summary (Section 7.9) concludes the presentation.

7.2. Computational Complexity

The simplified shift planning problem can be interpreted as a special multiple depot vehicle scheduling problem (MDVSP) [Desrosiers et al., 1993] with the depots corresponding to origin-destination pairs for each shift type. The general MDVSP is known to be $\mathcal{NP}$-hard, see Bertossi et al. [1987] or the alternative proof of Löbel [1997]. Still we cannot conclude that the simplified shift planning problem is $\mathcal{NP}$-hard because it exhibits a special cost and depot structure which may render the problem "easy". We will therefore show the $\mathcal{NP}$-hardness of simplified shift planning in the sequel.

The proof will be based on a reduction from the ONE-IN-THREE 3SAT problem with unnegated literals which can be stated as follows:

> Given numbers $q \geq 3$ and $p \geq 1$, a set $U := \{u_0, \ldots, u_{q-1}\}$ of q boolean variables and a set $\mathcal{C} := \{C_0, \ldots, C_{p-1}\}$ of p clauses over U such that each $C_h \in \mathcal{C}$ has only unnegated variables (literals) and $|C_h| = 3\,\forall h$, is there a truth assignment $tr : U \rightarrow \{\text{true}, \text{false}\}$ such that each clause $C_h \in \mathcal{C}$ has exactly one true variable?

The ONE-IN-THREE 3SAT problem with unnegated literals is known to be $\mathcal{NP}$-hard in the strong sense [Garey and Johnson, 1979].

Theorem 3. *The simplified shift planning problem is $\mathcal{NP}$-hard in the strong sense even if no travel times, shift restrictions or breaks are present.*

Proof. We will first prove that simplified shift planning is $\mathcal{NP}$-hard. We therefore have to show how a shift planning problem can be constructed for a given instance of the ONE-IN-THREE 3SAT problem with unnegated literals. The time axis will be structured into time slices of $2q$ time units for each clause C_h, starting at time 0. An additional period of $2(p-1)q+1$ time units is added after the end of the clause time slices. Furthermore, one time unit is added before start time 0, i.e. the time scale starts at -1. Note that times could be made positive by adding one time unit.

The task set I will be made up of a disjoint union of *literal tasks* I^{lit}, *filling tasks* I^{fill} and *clause tasks* I^{cl}:

- For each variable u_g occurring as a literal in clause C_h, a *literal task* $i \in I^{lit}$ with start time $a_i := 2hq + g$ and length $l_i := q$ is created. The set of literal tasks corresponding to a variable u_g will be denoted by I_g^{lit}.
- If variable u_g occurs as a literal in clause C_{h_1} and its next occurrence is in C_{h_2}, a *filling task* $i \in I^{fill}$ with start time $a_i := (2h_1+1)q+g$ and length $l_i := (2(h_2-h_1)-1)q$ is created, i.e. i is a task which starts at the end of one literal task and ends at the start of the next literal task of the same variable. If u_g occurs in C_{h_1} for the first time, a *filling task* $i \in I^{fill}$ with start time $a_i := -1$ and length $l_i := 2h_1q + g + 1$ is created, i.e. i starts at -1 and ends at the start time of the first literal task of the u_g. If u_g occurs in C_{h_2} for the last time, a *filling*

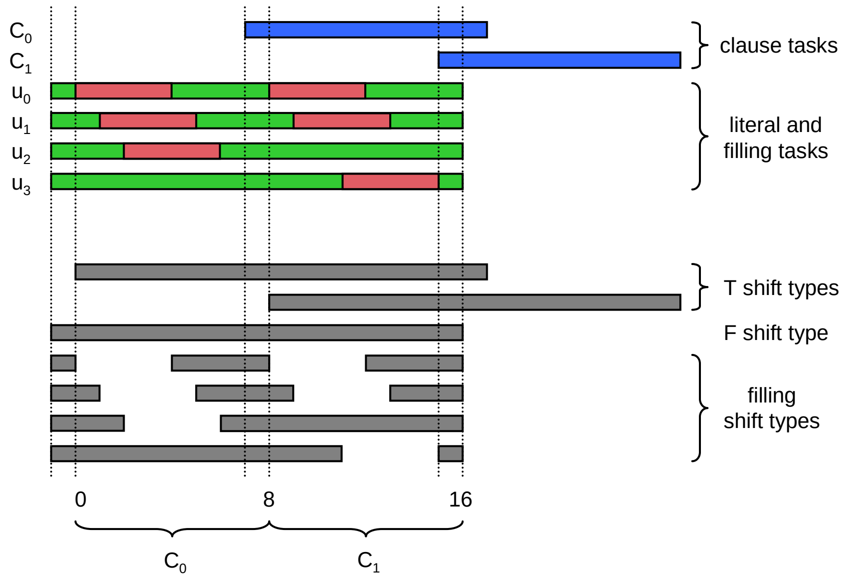


Figure 7.1.: Building blocks of shift planning problem for variable set $U := \{u_0, u_1, u_2, u_3\}$ and clauses $\mathcal{C} := \{\{u_0, u_1, u_2\}, \{u_0, u_1, u_3\}\}$.

task $i \in I^{fill}$ with start time $a_i := (2h_2 + 1)q + g$ and length $l_i := (2(p - h_2) - 1)q - g$ is created, i.e. i starts at the end of the last literal task and ends at $2qp$. The set of all filling tasks for a task u_g will be denoted by I_g^{fill}.

- For each clause C_h, a *clause task* $i \in I^{cl}$ with $a_i := 2(h + 1)q - 1$ and length $l_i := 2q(p - 1) + 2$ is created.

The set K of shift types consists of several *T shift types* K^T, a single *F shift type* k^F and *filling shift types* K^{fill}:

- For each clause C_h, a *T shift type* $k \in K^T$ is created with start time $st_k := 2hq$ and end time $et_k := 2(p + h)q + 1$.
- The *F shift type* k^F has start time $st_{k^F} := -1$ and end time $et_{k^F} := 2pq$.
- For each filling task $i \in I^{fill}$, there is a *filling shift type* $k \in K^{fill}$ which exactly covers the filling task, i.e. it has $st_k := a_i$ and $et_k := a_i + l_i$.

The cost c_k of each shift type $k \in K$ is defined to equal its length $et_k - st_k$. It should be clear that this transformation is polynomial.

Figure 7.1 illustrates the shift planning problem which results from ONE-IN-THREE 3SAT problem with $U := \{u_0, u_1, u_2, u_3\}$ and $\mathcal{C} := \{\{u_0, u_1, u_2\}, \{u_0, u_1, u_3\}\}$ (example taken from Löbel [1997]). The upper part gives the clause tasks (blue) for each $C_h \in \mathcal{C}$ and the blocks $I_g^{lit} \cup I_g^{fill}$ of literal (red) and filling tasks (green) for each $u_g \in U$. The shift types (each of which can be used by several shifts) are given by grey bars in the lower part. On the vertical axis, the time slice of $2q = 8$ time units for each clause C_h are sketched.

The problem is well-defined since all of the tasks can be covered by shifts. Some ideas of the construction shall be given:

- The sets of literal and fixing tasks $I_h^{lit} \cup I_h^{fill}$ for a variable h exactly fit into a shift of the F shift type k^F without any waiting time. Alternatively, filling tasks can be covered by their corresponding filling shift type without any waiting time.

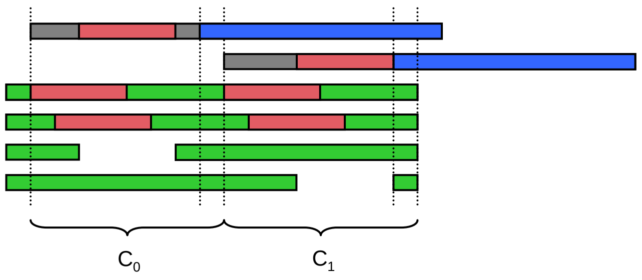


Figure 7.2.: Solution of cost $2(p^2q + pq^2) - pq + p + q = 94$ for the example.

- At a point in time at which a literal task $i \in I_h^{lit}$ ends, no other tasks from $I^{lit} \cup I^{fill}$ end. There is exactly one filling task from I_h^{fill} which starts at this point in time, and no other literal or filling tasks start at this time. Similarly, there is at most one filling task which ends at each point in time. If $i \in I_h^{fill}$ is a filling task which is not a "final" filling task (ending at $2pq$), then there is exactly one task in $I^{lit} \cup I^{fill}$ starting at this time, namely a literal task $i' \in I_h^{lit}$. Two observations follow:

  1. None of the literal tasks $i \in I^{lit}$ can be covered without waiting time by a filling shift type.
  2. If a literal task $i \in I_h^{lit}$ is covered by the F shift type k^F, waiting time can only be avoided if the shift contains all of the literal and filling tasks $I_h^{lit} \cup I_h^{fill}$ created for the variable u_h.

- Each pair $i, j \in I^{lit}$ of literal tasks pertaining to the same clause overlaps by at least one time unit, i.e. literal tasks of a clause cannot be covered by the same shift.

- Each clause task can only be covered by exactly one of the T shift types and none of the other shift types. Covering a clause task by a T shift type leaves a gap which can be used to cover further tasks. The gap is equal to the time slice for the corresponding clause, i.e. it starts at $2kq$ and has a length of $2q - 1$. Only literal tasks fit into this gap – note that filling tasks always cover the time period $2k'q - 1$ for a k'. Since literal tasks of a clause C_k always cover period $(2k + 1)q - 1$, no more than one literal task will fit into the gap. T shift types cannot cover tasks without waiting times.

With these observations, it is easy to see that the total work task minutes amount to $(2pq + 1)q + p(2q(p - 1) + 2) = 2(p^2q + pq^2 - pq + p) + q$.

It is now claimed that there exists a solution of cost $2(p^2q + pq^2) - pq + p + q$ for this shift planning problem if and only if the given instance $(U, \mathcal{C})$ of the ONE-IN-THREE 3SAT problem with unnegated literals has a solution.

"If". Suppose a variable assignment $tr : U \to \{\text{true}, \text{false}\}$ exists such that each clause contains exactly one true variable. We then build one shift for each of the p clause shift types. The shift for clause C_h covers the task which corresponds to the true literal in the respective clause and the only clause task which fits into the shift type. Each filling tasks $i \in I_g^{fill}$ whose variables u_g is assigned the value true is matched to its filling shift type without any waiting time. The blocks $I_g^{lit} \cup I_g^{fill}$ of tasks for all g with $tr(u_g) = $ false are assigned to shifts of the F shift type k^F, again without any gaps. Now all tasks are covered. Figure 7.2 shows the shift plan for the above example with $U := \{u_1, u_2, u_3, u_4\}$, $\mathcal{C} := \{\{u_1, u_2, u_3\}, \{u_1, u_2, u_4\}\}$.

The only waiting time in the plan stems from the clause shift types and amounts to $p(q-1)$. Because the cost of the shift types is equal to their lengths, the cost of the resulting shift plan is the sum of work and waiting time: $(2(p^2q+pq^2-pq+p)+q)+p(q-1) = 2(p^2q+pq^2)-pq+p+q$.

"Only if". Imagine an optimal shift plan of cost $2(p^2q+pq^2)-pq+p+q$ for the above shift planning scenario. We will show that we can deduce a truth assignment for the ONE-IN-THREE 3SAT problem such that each clause has exactly one true literal.

For covering the clause tasks, the shift plan must contain at least one shift of each T shift type, amounting for costs of $p(2pq+1)$. By construction, the clause shifts can additionally cover one literal task at most (and only literal tasks). These tasks will be the literals which are assigned the value true. Note that each T shift type corresponds to a different clause. Even if all of the gaps of the p shifts with T shift types are used for covering literal tasks, the remaining tasks will amount to $(2(p^2q+pq^2-pq+p)+q)-p((2q(p-1)+2)+q) = 2pq^2-pq+q$ task minutes. Because we have already used shifts of cost $p(2pq+1)$, these must be covered by shifts of maximum costs $2(p^2q+pq^2)-pq+p+q-(2p^2q+p) = 2pq^2-pq+q$, i.e. the remaining shift costs equal the remaining task minutes.

On the one hand, this means that we cannot have more than the p shifts of the T shift types because these always entail waiting times. Consequently, exactly one literal in each clause will be assigned the value true. On the other hand, it means that all other literal tasks are assigned to F shifts together with their filling tasks. We will assign the value false to all literals corresponding to these literal tasks. Since we cannot afford waiting times, all tasks in $I_{g_1}^{lit}$ will be assigned to the same shift if u_{g_1} will be assigned the value false. If thus a literal task $i \in I_{g_2}^{lit}$ is not covered by an F shift, none of the tasks in $I_{g_2}^{lit}$ can be covered by an F shift, i.e. we assign consistent values to all variables. Note that the filling tasks $I_{g_2}^{fill}$ for a variable u_{g_2} with $tr(u_{g_2}) =$ true can be covered without waiting times by their filling shift types.

Since the proof does not make use of travel times, shift restrictions or breaks, this shows that even basic shift planning is in fact $\mathcal{NP}$-hard. To show that simplified shift planning is $\mathcal{NP}$-hard in the strong sense, we have to prove that even if the sizes of all entities are polynomially bounded in the length of the input, the problem remains $\mathcal{NP}$-hard, see e.g. Hromkovič [2001, p. 151].

We first note that the ONE-IN-THREE 3SAT problem with unnegated literals is $\mathcal{NP}$-hard in the strong sense since it does not involve any numbers [Garey and Johnson, 1979]. The only numbers which are involved in shift planning without travel times are the lengths of tasks and shifts. But in the above proof, tasks and shifts have maximum durations of $2pq+1$. Lengths are therefore bounded by a polynomial in the input length, showing that simplified shift planning is $\mathcal{NP}$-hard in the strong sense, see also Garey and Johnson [1979, Chapter 4]. $\square$

The proof uses the basic idea of Löbel [1997] of chaining tasks (visits) which are consistently assigned the value false. However, Löbel's proof is based on the network structure of the multi-commodity flow problem associated with a special MDVSP arising in duty scheduling of public transport, i.e. there is no temporal structure (and clearly no shifts or shift types).

7.3. A Flow Model

The simplified shift planning problem will now be formulated as a network flow model. For ease of exposition, it will first be assumed that each shift type $k \in K$ defines exactly one break. Later, generalisations to shift types with several or no breaks are discussed. The formulation will use a distinct network G^k for each shift type k. The construction ensures that every path in the network for shift type k represents a valid shift of type k, covering a (possibly empty) set of the tasks.

For each task $i \in I$, a start time a_i and a length l_i are given. Each shift type $k \in K$ will be delimited by an origin task i_o^k and a destination task i_d^k with fixed time windows $[a_{i_o^k}, b_{i_o^k}] = \{st_k\}$

and $[a_{i_d^k}, b_{i_d^k}] = \{et_k\}$, respectively. The unique break task i_b^k of shift type k has a length of $l_{i_b^k}$ and a start time window $[a_{i_b^k}, b_{i_b^k}]$ such that $a_{i_o^k} \leq a_{i_b^k}$ and $b_{i_b^k} + l_{i_b^k} \leq b_{i_d^k}$. Note that in contrast to Chapter 5, depot and break tasks are defined per shift type (and not per shift).

The network $G^k = (V^k, E^k)$ for shift type $k \in K$ is constructed as follows: The origin and destination tasks of shift type k will be represented by an origin and a destination node v_o^k and v_d^k, respectively. For each task $i \in I$, one node is created for each position in the shift (before or after the break) into which i fits. A task i fits between origin and break task if

$$a_{i_o^k} + d_{i_o^k,i} \leq a_i \wedge a_i + l_i \leq b_{i_b^k}$$

If this condition is true, V_k will thus contain a node for i. A further i node is created if i also fits between break and shift type end, i.e. if

$$a_{i_b^k} + l_{i_b^k} \leq a_i \wedge a_i + l_i + d_{i,i_d^k} \leq b_{i_d^k}$$

The set of nodes for task i and shift type k is denoted by V_i^k and will be empty if i does not fit into k.

For the break i_b^k, one node is created for each possible predecessor task (including the origin task) and each start time which this predecessor task admits. For each possible predecessor i and each start time in $[\max(a_i + l_i, a_{i_b^k}), b_{i_b^k}]$, the break node set V_b^k for shift type k will thus contain one node. The multiplication of break nodes by possible predecessors makes it possible to attribute a break node to the location of its predecessor.

The construction of the edge set E^k is straightforward. If v_i^k and v_j^k are nodes for work tasks i and j ($v_i^k \in V_i^k$, $v_j^k \in V_j^k$), then there is an edge $(v_i^k, v_j^k) \in E^k$ if and only if $a_i + l_i + d_{i,j} \leq a_j$. If v_i^k is a node for a task i fitting into the first part of k (before the break), E^k contains edges (v_o^k, v_i^k) and (v_i^k, v_b^k) for each break node v_b^k which was created for predecessor i. If v_i^k is a node for task i and the position in k after the break, edges (v_i^k, v_d^k) and (v_b^k, v_i^k) are included in E^k if the predecessor task $j \neq i$ and start time t associated with v_b^k are such that $t + l_{i_b^k} + d_{j,i} \leq a_i$. To allow for empty first and second shift parts, E^k contains all arcs (v_o^k, v_b^k) if v_b^k is a break node for predecessor v_o^k and edges (v_b^k, v_d^k) for all break nodes v_b^k.

The result of the construction is a network

$$G^k = \left(\bigcup_{i \in I} V_i^k \cup \{v_o^k, v_d^k\} \cup V_b^k, E^k \right)$$

for each shift type k. In the flow formulation, all edges have a capacity of 1.

An example can be found in Fig. 7.3. In the upper part of the figure, three tasks and a shift type k are sketched. The shift type defines a break which must take place within a time window (including the break's length) as given by the brackets. For compactness of presentation, the time axis is assumed to be discretised such that the break can only realize three different start times. While tasks 1 and 3 only fit into the first and second part of the shift, respectively, task 2 can be placed in either part. Consequently, the network shown in the lower part contains two nodes $2a$ and $2b$. For each admissible start time and predecessor (i_o^k, 1 or $2a$), the graph contains a break node. Note that the network does not contain any node b_1^{2a} because task 2 does not allow for a break placement at its first possible position.

From the construction, it should be clear that every unit flow (or path) from v_o^k to v_d^k in G^k represents a shift of type k, covering some possibly empty set of tasks. In fact, it can be easily verified that each path passes through a break node and travel times can be correctly covered. Furthermore, no path can pass through more than one node of a set V_i^k because tasks are fixed in time. The notions of paths and shifts will thus be used interchangeably in the sequel.

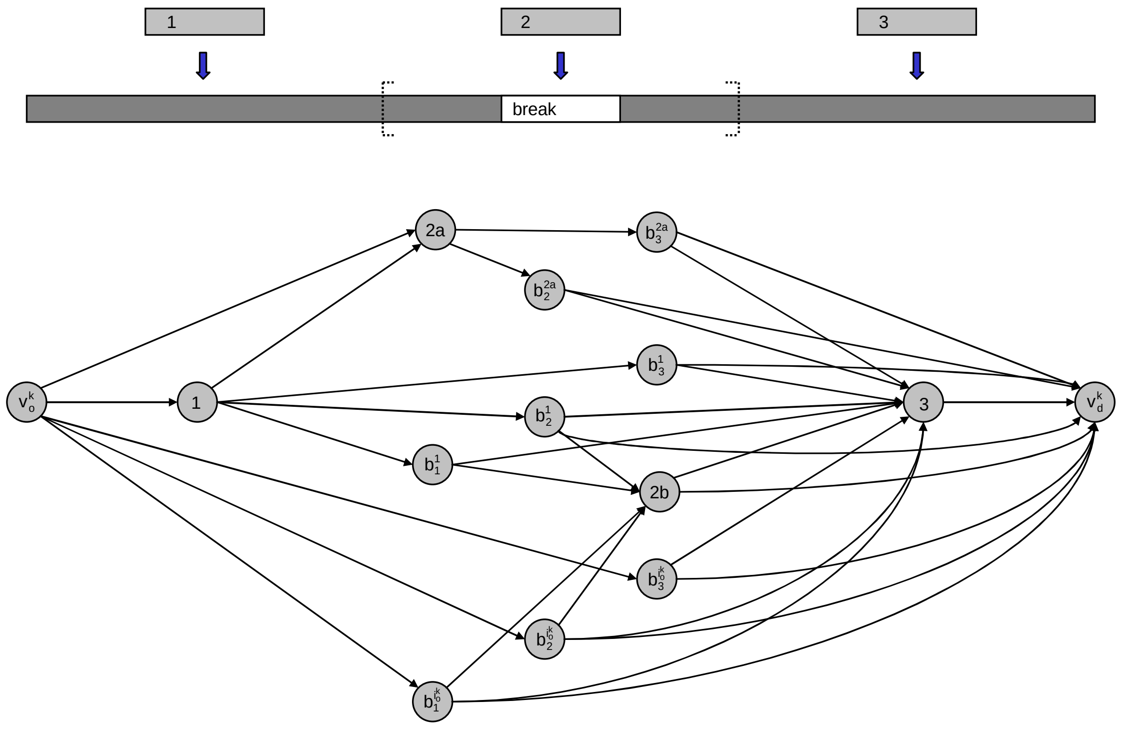


Figure 7.3.: Network construction example.

If TW is the maximum width of the break time windows ($TW := \max_k b_{i_b^k} - a_{i_b^k} + 1$), the total number of nodes is in $\mathcal{O}(|K| \cdot |I| \cdot TW)$, i.e. the construction is pseudo-polynomial in the break window sizes. The number of break nodes can be somewhat reduced if several tasks take place at the same location. Then it suffices to multiply the break nodes by predecessor locations (instead of by the predecessors themselves).

When formulating the shift planning problem as a network flow model, we are interested in the minimisation of shift costs. As a shift corresponds to a path from origin node v_o^k to destination node v_d^k, the costs c_k incurred by shift type k can be attributed to the edges emanating from the origin node. The costs c_{vw}^k of an edge $(v, w) \in E^k$ will thus be defined as

$$c_{vw}^k := \begin{cases} c^k \text{ if } v = v_o^k \\ 0 \quad \text{else} \end{cases}$$

The minimisation of shift costs is a regular objective function since it does not depend on the task start times. For this class of objective functions, the networks above can be simplified by creating break nodes only for the earliest possible start times. In Fig. 7.3, the nodes $b_2^{i_o^k}$, $b_3^{i_o^k}$, b_2^1, b_3^1 and b_3^{2a} thus become redundant.

The network flow model for the shift planning problem combines the flow formulations for the different shift types. The decision variables are the flows X_{vw}^k on the edges $(v, w) \in E^k$. Additionally to flow restrictions, the model will include absolute minimum and maximum shift number restrictions R_{abs}^{min} and R_{abs}^{max}. A minimum (maximum) restriction $r \in R_{abs}^{min} \cup R_{abs}^{max}$ imposes a lower (upper) limit of $m_r \in \mathbb{N}$ on the shifts of types in K_r.

If $\delta^-(v)$ and $\delta^+(v)$ are the in- and outedges of a node v, the flow formulation reads as

$$\min \sum_{k \in K} \sum_{(v,w) \in E^k} c_{vw}^k X_{vw}^k \tag{7.1}$$

subject to

$$\sum_{k \in K} \sum_{v \in V_i^k} \sum_{(v,w) \in \delta^+(v)} X_{vw}^k = 1 \qquad \forall i \in I \tag{7.2}$$

$$\sum_{k \in K_r} \sum_{(v_o^k,w) \in \delta^+(v_o^k)} X_{v_o^k w}^k \geq m_r \qquad \forall r \in R_{abs}^{min} \tag{7.3}$$

$$\sum_{k \in K_r} \sum_{(v_o^k,w) \in \delta^+(v_o^k)} X_{v_o^k w}^k \leq m_r \qquad \forall r \in R_{abs}^{max} \tag{7.4}$$

$$\sum_{(w,v) \in \delta^-(v)} X_{wv}^k - \sum_{(v,w) \in \delta^+(v)} X_{vw}^k = 0 \qquad \forall k \in K, \forall v \in V_b^k \cup \bigcup_{i \in I} V_i^k \tag{7.5}$$

$$X_{vw}^k \geq 0 \text{ and integer } \forall k \in K, \forall (v,w) \in E^k \tag{7.6}$$

Constraints (7.2) ensure that each task is covered exactly once by a shift. (7.3) and (7.4) are the minimum and maximum shift number restrictions, respectively. Equations (7.5) are the flow conservation constraints for inner nodes of the networks. (7.6) imposes nonnegativity and integrality on the flow variables. Note that the flow variables are generally non-binary because in order to obey the minimum restrictions (7.3), we may need several empty shifts of the same type. Model (7.1)-(7.6) is closely related to so-called *three-index formulations* for vehicle routing problems [Toth and Vigo, 2001a].

Generalising the model to shift types with more than one break or no break at all is straightforward. Note that if more than one break is present, additional node instances for the possible positions (between the breaks, at the start or end of the shift) for each task must be created. Because we assume travel times to obey the triangle inequality and the objective does not depend on the gaps between the tasks, it is not necessary to restrict tasks to be covered exactly once. Effectively, if a shift plan covers a task several times, we can simply take out all but one task instance from the plan. The set partitioning constraints (7.2) can therefore be relaxed to set covering constraints ($\geq$ 1) [Toth and Vigo, 2001a] which are numerically more stable [Barnhart et al., 1998] [Desaulniers et al., 2001].

Lübbecke and Desrosiers [2004] additionally note that relaxing set partitioning to set covering constraints halves the dual solution space since dual variables of covering constraints are restricted to be nonnegative. Note that while a solution may cover some of the tasks more than once, an minimum cost solution will never use multiple instances of a whole path.

Equations (7.1) through (7.6) describe an integer multicommodity flow problem with additional quantitative restrictions [Assad, 1978] [Bertossi et al., 1987]. Actually, the shift types can be interpreted as commodities each of which must be routed on a dedicated network. Constraints (7.2) through (7.4) are *coupling constraints* which potentially span over all of the individual flow models (block $[A_1 \cdots A_{|K|}]X \geq b$ in Fig. 7.4[1]). In contrast, the flow constraints 7.5 are local to the flow models. The constraint system thus has a *blockangular* structure (blocks $B_k X = 0$ in Fig. 7.4) with different restrictions on the subsets, cf. Section 6.2.

7.4. Dantzig-Wolfe Decomposition

As mentioned before, the flow model of the shift planning problem gives rise to the block-structured linear program (7.1)-(7.6). We will exploit this special structure by applying Dantzig-Wolfe decomposition to the compact formulation. For related work on the multiple-depot vehicle routing and scheduling problems, the reader is referred to Desrosiers et al. [1993], Ribeiro and Soumis [1994] and Desaulniers et al. [1998]. For the time being, we will relax the integrality conditions of (7.6).

[1]Note that also the maximum shift number restrictions can be formulated as $\geq$ constraints by multiplying with -1.

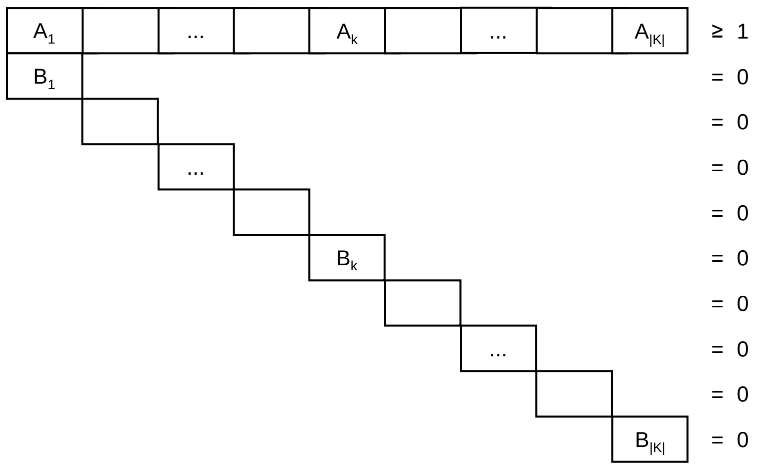


Figure 7.4.: Blockangular structure of the flow model.

As described in Section 6.1, the Dantzig-Wolfe decomposition is based on the representation of the convex polyhedra given by the blocks of local constraints by their extreme points and rays. Applied to problem (7.1)-(7.6), (7.2) through (7.4) are the coupling constraints and thus remain in the master program. The flow conservation constraints (7.5) along with the nonnegativity constraints (7.6) build the subsystems. Since constraints (7.5) define a homogeneous polyhedron, the polyhedra of these subsystems do not have extreme points and can be represented by a linear combination of the extreme rays.

Let $\{(x^k_{vwp} \,|\, (v,w) \in E^k) \,|\, p \in \Omega^k\}$ be the set of extreme rays for shift type k. Each extreme ray $(x^k_{vwp} \,|\, (v,w) \in E^k)$ represents a single path (shift) because it obeys the flow conservation constraints. Note that if $(x^k_{vwp} \,|\, (v,w) \in E^k)$ was a combination of several shifts, it could not be an extreme ray. Without loss of generality, we can assume that $x^k_{vwp} \in \{0,1\}$.

By the theorems of Weyl and Minkowski, the original variables X^k_{vw} can be represented by a linear combination of the extreme rays: $X^k_{vw} = \sum\limits_{p\in\Omega^k} x^k_{vwp}\lambda^k_p$ ($\lambda^k_p \geq 0$) [Nemhauser and Wolsey, 1988]. The following formulation results from replacing all of the original variables in formulation (7.1)-(7.6):

$$\min \sum_{k\in K}\sum_{p\in\Omega^k}\left(\sum_{(v,w)\in E^k} c^k_{vw}x^k_{vwp}\right)\lambda^k_p \tag{7.7}$$

subject to

$$\sum_{k\in K}\sum_{p\in\Omega^k}\left(\sum_{v\in V^k_i}\sum_{(v,w)\in\delta^+(v)} x^k_{vwp}\right)\lambda^k_p \geq 1 \quad \forall i \in I \tag{7.8}$$

$$\sum_{k\in K_r}\sum_{p\in\Omega^k}\left(\sum_{(v^k_o,w)\in\delta^+(v^k_o)} x_{v^k_o wp}\right)\lambda^k_p \geq m_r \ \forall r \in R^{min}_{abs} \tag{7.9}$$

$$\sum_{k\in K_r}\sum_{p\in\Omega^k}\left(\sum_{(v^k_o,w)\in\delta^+(v^k_o)} x_{v^k_o wp}\right)\lambda^k_p \leq m_r \ \forall r \in R^{max}_{abs} \tag{7.10}$$

$$\lambda^k_p \geq 0 \quad \forall k \in K, \forall p \in \Omega^k \tag{7.11}$$

The decision variables of this problem are the coefficients λ_p^k for the extreme rays of the polyhedra given by the flow conservation constraints. We will now introduce new identifiers for the objective function and constraint coefficients which were already grouped by brackets. Let

$$c_p^k := \sum_{(v,w)\in E^k} c_{vw}^k x_{vwp}^k$$

be the cost coefficient for extreme ray $p \in \Omega^k$. As mentioned above, an extreme ray is a single path from the origin to the destination node. Furthermore, we only account for shift costs on the outedges of origin nodes while all other edge weights are 0. Therefore, c_p^k is equal to the shift type cost c^k.

The coverage coefficients in constraints (7.8) will be denoted by g_{ip}^k for each $i \in I$ and $p \in \Omega^k$:

$$g_{ip}^k := \sum_{v\in V_i^k} \sum_{(v,w)\in\delta^+(v)} x_{vwp}^k$$

With p being a shift, this coefficient will be equal to 1 if p traverses a node belonging to task i and 0 otherwise. The coefficients g_{ip}^k thus indicate which work tasks are covered by a shift p of type k, i.e. $(g_{ip}^k \mid i \in I)$ is the characteristic vector of the set of covered work tasks.

Finally, we analyse the coefficient

$$\sum_{(v_o^k,w)\in\delta^+(v_o^k)} x_{v_o^k wp}$$

in the shift restrictions 7.9 and (7.10). This sum equals 1 if p is a shift of type k and will be 0 otherwise. Since the sums in (7.9) and (7.10) only run over $p \in \Omega^k$, the coefficient can be omitted.

With these observations and definitions, the *master program* reads as

$$\min \sum_{k\in K} \sum_{p\in\Omega^k} c_p^k \lambda_p^k \tag{7.12}$$

subject to

$$\sum_{k\in K} \sum_{p\in\Omega^k} g_{ip}^k \lambda_p^k \geq 1 \quad \forall i \in I \tag{7.13}$$

$$\sum_{k\in K_r} \sum_{p\in\Omega^k} \lambda_p^k \geq m_r \; \forall r \in R_{abs}^{min} \tag{7.14}$$

$$\sum_{k\in K_r} \sum_{p\in\Omega^k} \lambda_p^k \leq m_r \; \forall r \in R_{abs}^{max} \tag{7.15}$$

$$\lambda_p^k \geq 0 \quad \forall k \in K, \forall p \in \Omega^k \tag{7.16}$$

This linear program has a natural interpretation. The decision variable λ_p^k represents the "proportion" of a shift p used in the solution. Shift costs are derived from their shift types. In the constraint system, the column for a shift p indicates which tasks are covered. The shift restrictions sum up all shifts which are affected by the restrictions. Because the coefficients of the extreme rays were chosen to be binary, we will regain integrality if the variables λ_p^k are constrained to be binary. Then the program (7.12)-(7.16) corresponds to selecting a subset of all possible shifts such that all tasks are covered and the shift restrictions are obeyed.

While the original formulation (7.1)-(7.6) expresses the flow problem in terms of the flows on the edges, the Dantzig-Wolfe formulation (7.12)-(7.16) builds solutions by combinations of paths (unit flows) from origin to destination nodes. Dantzig-Wolfe decomposition on this problem thus corresponds to an application of the flow decomposition theorem, cf. Ahuja et al. [1993].

7.5. Column Generation

The solution of (7.12)-(7.16) by linear programming in principle requires a full enumeration of all possible shifts covering subsets of the tasks. In practice, this will only be possible on very small test cases. The idea of column generation is to maintain only a restricted subset $\Omega' \subset \Omega := \bigcup_{k \in K} \Omega^k$ of variables in the master program and assume $\lambda_p^k = 0$ for all $p \in \Omega \setminus \Omega'$. The restricted master program is solved to optimality on the restricted column set, using the simplex method. In the subproblems, the optimal dual values are then used to generate new shifts (extreme rays) which may improve the solution. The generated columns are added to the restricted master program which is then resolved. This process iterates until no more improving columns are found in the subproblems.

In the pricing problems, we generate extreme rays of the polyhedra defined by the constraint systems which are not represented in the master program. According to the simplex method, we should generate extreme rays (shifts) whose reduced costs are negative. Let π_i and μ_r be the dual values associated with the covering and shift restriction constraints in an optimal solution to the restricted master program. If the Dantzig rule is used for the choice of an entering variable, we seek for a column p such that the reduced cost

$$\bar{c}_p^k := c_p^k - \sum_{i \in I} g_{ip}^k \pi_i - \sum_{r \in R_{abs}^{min}} \rho_r(k)\mu_r - \sum_{r \in R_{abs}^{max}} \rho_r(k)\mu_r \tag{7.17}$$

is minimal. In this formulation, ρ_r is a function $\rho_r : K \to \{0,1\}$ which indicates if a given shift number restriction $r \in R_{abs}^{min} \cup R_{abs}^{max}$ refers to a shift type:

$$\rho_r(k) := \begin{cases} 1 \text{ if } k \in K_r \\ 0 \text{ else} \end{cases}$$

The dual values π_i and μ_r can be interpreted as shadow prices for covering a task i and using a shift type in K_r, respectively. For minimum shift restrictions, these prices will be nonnegative while for the maximum restrictions, the dual values will be nonpositive. If covering a task or obeying a shift restriction represents a bottleneck in a restricted master program, the dual values will be different from 0. Interpreting term (7.17), we thus seek for a shift of a priori costs $c_p^k = c^k$ which gets a reward for covering bottleneck tasks or minimum restrictions and which gets penalised for using shift types of bottleneck maximum restrictions. From the subproblem viewpoint, the coverage and shift restrictions are represented by a reward/penalty mechanism via dual values of the master program while the flow conservation constraints are represented explicitly.

As described above, the g_{ip}^k and the contributions to the shift restrictions are fixed for the master program. For the subproblems, they are calculated from the decision variables which are the edge flows X_{vw}^k. As described above, the flow formulation decomposes into individual models for each shift type k. As a consequence, the one pricing problem is solved for each shift type. The shift with minimum reduced costs can then be retrieved by minimisation over the results for each k. The pricing problem for shift type k is

$$\begin{aligned} \min \quad & \sum_{(v,w) \in E^k} c_{vw}^k X_{vw}^k - \sum_{i \in I} \pi_i \sum_{v \in V_i^k} \sum_{(v,w) \in \delta^+(v)} X_{vw}^k \\ & - \sum_{r \in R_{abs}^{min}} \rho_r(k)\mu_r - \sum_{r \in R_{abs}^{max}} \rho_r(k)\mu_r \end{aligned} \tag{7.18}$$

subject to

$$\sum_{(w,v) \in \delta^-(v)} X_{wv}^k - \sum_{(v,w) \in \delta^+(v)} X_{vw}^k = 0 \qquad \forall v \in V_b^k \cup \bigcup_{i \in I} V_i^k$$

$$X^k_{vw} \in \{0,1\}\ \forall (v,w) \in E^k$$

In the objective function (7.18), the term

$$\bar{c}^k := -\sum_{r \in R^{min}_{abs}} \rho_r(k)\mu_r - \sum_{r \in R^{max}_{abs}} \rho_r(k)\mu_r$$

for the shift restrictions is constant for a given k. The covering prices π_i can be attributed to the edges by defining new edge costs $\bar{c}^k_{vw}$:

$$\bar{c}^k_{vw} := \begin{cases} c^k_{vw} - \pi_i & \text{if } v \in V^k_i \\ c^k_{vw} & \text{else} \end{cases}$$

With these definitions, the pricing problem for shift type k can be reformulated as

$$\min \bar{c}^k + \sum_{(v,w) \in E^k} \bar{c}^k_{vw} X^k_{vw}$$

subject to

$$\sum_{(w,v) \in \delta^-(v)} X^k_{wv} - \sum_{(v,w) \in \delta^+(v)} X^k_{vw} = 0 \qquad \forall v \in V^k_b \cup \bigcup_{i \in I} V^k_i$$

$$X^k_{vw} \in \{0,1\}\ \forall (i,j) \in E^k$$

This is a shortest path problem on the modified edge costs $\bar{c}^k_{vw}$ which can e.g. be solved by the Dijkstra algorithm or by dynamic programming [Cormen et al., 2001]. Alternatively, it can be solved by the simplex method for linear programming since shortest path problems are naturally integer [Ahuja et al., 1993]. The lower bounds of the LP relaxations of the compact and the column generation formulation therefore coincide as described in Section 6.3.

As described above, we can obtain the best new shift by minimising over the results for all shift types. If the resulting shift has negative reduced costs, it is added as a new column to the restricted master program. The master program is then resolved, providing the pricing problems with new dual values in the next iteration. This process iterates until no more reduced cost columns can be found in which case the optimal solution to the relaxed problem has been found, see also Figure 6.1 in Section 6.1.

It should be noted that in the pricing problems, it suffices to find shifts with negative reduced costs instead of minimum cost solutions. Column generation schemes tend to converge faster if several negative reduced cost columns are added in each iteration [Desrosiers et al., 1984]. Note also that by subproblem pricing, all possible shifts are implicitly represented in the master program. Nevertheless, the simplex algorithm will only explore a small subset of the total number of shifts on its way to the optimum solution.

If several shifts of a shift type have equal reduced costs, we always choose a non-dominated shift, i.e. a shift into which no further tasks can be inserted without increasing reduced costs. Toth and Vigo [2001a] have called such columns *inclusion-maximal*. By this procedure, we include a maximum number of tasks with vanishing dual costs to the shift. This accounts for the fact that such tasks can be a bottleneck in later iterations. We can therefore expect that less columns must be generated if we always add non-dominated shifts, see also Lübbecke and Desrosiers [2004].

7.6. Branch-and-Price

Up to now, the integrality restrictions of the original problem were dropped and only the LP relaxation of the shift planning problem was solved. Set covering formulations are generally known to exhibit advantageous integer properties, see e.g. Bailey [1985] and Mehrotra et al. [2000]. In order to tackle fractionalities, we will devise an LP-based branch-and-bound algorithm. We will thus repeatedly branch on fractional values, imposing new integrality cuts on the branches. Branching decisions may render new variables advantageous, meaning that column generation should be performed throughout the search tree.

As pointed out in Section 6.3, branching should be applied to the variables of the compact formulation [Villeneuve et al., 2003], i.e. on the edge flow variables $X^k_{vw} = \sum_{p\in\Omega^k} x^k_{vwp}\lambda^k_p$ [Desrochers et al., 1992]. Since these flow variables are part of the pricing problems, this does not destroy the subproblem structure. Since we enforce integrality only on the original variables, the branch-and-price approach is based on the convexification of the solution space, see Section 6.6.

In the original formulation, the set partitioning ($= 1$) constraints were relaxed to set covering constraints (≥ 1). As a consequence, the edge values $\sum_{p\in\Omega^k} x^k_{vwp}\lambda^k_p$ can assume higher-order fractional values. Let $\sum_{p\in\Omega^k} x^k_{vwp}\lambda^k_p = y$ with $y > 1$ and fractional. We would then create two new branches, imposing $\sum_{p\in\Omega^k} x^k_{vwp}\lambda^k_p \leq \lfloor y\rfloor$ on one branch and $\sum_{p\in\Omega^k} x^k_{vwp}\lambda^k_p \geq \lceil y\rceil$ on the other. But on the $\leq$ branch, $\sum_{p\in\Omega^k} x^k_{vwp}\lambda^k_p$ may later have a fractional value $0 < y' < \lfloor y\rfloor$, and we would have to branch again.

This branching is clearly redundant because we can implicitly assume the set partitioning structure. Without loss of generality, the values $\sum_{p\in\Omega^k} x^k_{vwp}\lambda^k_p$ can be constrained to the range $[0, 1]$. If $\sum_{p\in\Omega^k} x^k_{vwp}\lambda^k_p$ assumes a fractional value y, we can therefore create two branches with constraints $\sum_{p\in\Omega^k} x^k_{vwp}\lambda^k_p = 0$ and $\sum_{p\in\Omega^k} x^k_{vwp}\lambda^k_p = 1$, respectively.

This corresponds to Ryan-Foster constraint branching, see Section 6.3. In the shift planning context, the branch $\sum_{p\in\Omega^k} x^k_{vwp}\lambda^k_p = 0$ means that we forbid the use of edge (v, w) in network G^k, i.e. the nodes v and w must not be covered by the same shift of type k. This can be implemented by forcing all shifts using (v, w) to 0 and removing edge (v, w) from G^k in the subproblems. The branch $\sum_{p\in\Omega^k} x^k_{vwp}\lambda^k_p = 1$ says that v and w must be covered once by the same shift of type k. This is realised by removing all shifts covering v or w but not both and removing all edges (v, w') with $w' \neq w$ from the pricing problems. Even if the λ^k_p will never exceed 1 in an optimal solution, the constraint $\sum_{p\in\Omega^k} x^k_{vwp}\lambda^k_p = 1$ must be explicitly added to the master program because it constrains the original variable X^p_{vw} to $[0, 1]$ which was not originally assumed. This constraint yields an additional dual value which must be subtracted from the edge cost in the pricing problem.

For the implementation described below, branching was applied on a more aggregate level. Instead of branching on the edges of the individual shift types, integrality is enforced on the level of consecutive tasks in shifts of arbitrary types, see also Desrochers and Soumis [1989]. We thus search for task pairs (i, j) such that the sum of the shifts covering i and j consecutively is fractional. The branching rule is then applied to all edges in $V^k_i \times V^k_j$ for all shift types $k \in K$. Clearly, the number of potential task pair branches is less than for edge level branching. Note however that task pair branching is not complete in general, i.e. fractionalities may remain even if all task pairs are integer. Nevertheless, it revealed to be sufficient on all test cases.

In order to avoid a destruction of the current master program solution, branching is applied to task pairs (i, j) with values less than 1 as long as possible. We always branch on the pair (i, j) whose value is closest to 1. Because such a solution suggests to cover tasks (i, j) consecutively and by the same shift, the $= 1$ branch is explored first. A similar branching scheme has been proposed by Ryan [1992]. It could be observed that this rule often allows for finding an optimal solution with a single descent in the search tree, i.e. the first integer solution is already optimal.

The overall branch-and-price scheme is given by Algorithm 4. The result of the shift planning construction heuristic described in Chapter 5 is used as a starting solution Σ^*. Σ^* thus provides the master program with a first column set and gives an initial upper bound z^*. It is important to note that the LP value z_{LP} does not provide a lower bound as long as improving columns can be found.

Algorithm 4 Branch-and-price

```
1:  initialisation with heuristic starting solution $\Sigma^*$ of value $z^*$
2:  while there are branch-and-bound nodes to be processed do
3:      take a node
4:      solve the restricted master problem
5:      solve the pricing problems on the current optimal dual values
6:      if there are columns with negative reduced cost then
7:          add columns to the master problem
8:      else if $z_{LP} \geq z^*$ then
9:          abandon node
10:     else if solution integer then
11:         new best solution: $\Sigma^* \leftarrow \Sigma_{LP}$, $z^* \leftarrow z_{LP}$
12:     else
13:         branching: create successor nodes
14:     end if
15: end while
```

The search tree is first traversed in depth-first order. Because we cannot give a new lower bound before column generation has terminated, a node initially inherits its parent's lower bound.

Additionally to the LP lower bound z_{LP}, a global lower bound is used which is equal to the minimum over the lower bounds on all nodes which are still to be processed. If this value is greater or equal to the value z^* of the incumbent solution, we can terminate the search because we will not find better solutions. In the shift planning problem, the best solution is frequently found very early with the global lower bound proving optimality.

We have initially conducted experiments with the generalised Lasdon lower bound of Section 6.4. This lower bound can be used for early termination of column generation and to prune the search space. For the calculation, we have used the number $|I|$ of tasks as an upper bound approximation to the number of final shifts. However, no performance improvements could be observed. On the one hand, the quality of the bound was often poor. On the other hand, the LP lower bound was very close to the optimum throughout large parts of the search tree. Additionally, the tailing-off effect of column generation was not very strong. We have therefore decided not to use the Lasdon bound for the experimental results presented below.

The pruning in Algorithm 4 was refined by some simple observations. If all shift type costs are integer, an improved shift plan must save costs of at least one, i.e. we can prune a node if $z_{LP} + 1 \geq z^*$. If there are non-integer shift type costs, determining the difference between the incumbent and an improving solution amounts to solving a subset sum problem [Martello and Toth, 1990]. While subset-sum is an $\mathcal{NP}$-hard problem, building all combinations of shift type costs can easily be accomplished by a pseudo-polynomial dynamic programming algorithm. In most cases, only fractions of a second were taken to determine the current stepwidth.

7.7. Problem Decomposition

For the above formulation, it was assumed that the shift planning problem decomposes into separate days. Consequently, day indices for shift types and shift restrictions were omitted. It should be clear that the model can easily be generalised to several days by replacing each shift types k by a shift type realisation (k, n) with n denoting the day. If K_r is the set of shift types and N_r

the set of days of a shift restriction $r \in R_{abs}^{min} \cup R_{abs}^{max}$, r refers to the set $K_r \times N_r$ of shift type realisations.

In the practice of airports, there are often times of little or no traffic (e.g. during night), i.e. operations are *discontinuous*, see also Section 1.2.3. Shift planning problems then indeed decompose into independent problems. Each of these problems typically covers one day and sometimes even spans over shorter time periods due to low workloads between peak times. In the sequel, it will be shown how given problems can be automatically decomposed into independent components. The individual problems are then solved separately and afterwards composed into a complete solution.

Decomposition is often crucial for making a scenario amenable to an exact solution approach, and even if a complete scenario could be solved by the above branch-and-price approach, shift planning by decomposition is more efficient. Note that this is generally not true for the local improvement algorithm of Chapter 5 because the complexity of local reoptimisation steps essentially depends on the neighbourhood sizes and not on the size of the overall problem. When problems could be decomposed by day, the daily problems were often very similar due to the resemblance of flight schedules and task generation rules.

We will use a *constraint graph* $G = (V, E)$ for decomposition. The node set V comprises all shift type realisations (k, n) on the different days, i.e. $V = \{(k, n) \mid k \in K, n \in N_k\}$ with N_k being the valid days of shift type k. We then check which shift type realisations may cover each task $i \in I$, taking breaks into account. For each task i, we introduce edges in G such that the nodes of shift type realisations covering i induce a complete subgraph (clique). Analogously, we introduce edges between all shift type realisations in $K_r \times N_r$ for each shift restriction $r \in R_{abs}^{min} \cup R_{abs}^{max}$ with reference shift types K_r and days N_r.

It should be clear that in the resulting graph, two nodes are connected if and only if decisions for the associated shift type realisations are interdependent. Consequently, the connected components of G represent independent optimisation problems. For each component, we build one subproblem consisting of the shift type realisations associated with its nodes and the tasks which can be covered by them. Each subproblem is then consecutively submitted to the above algorithm. Note that this assures that tasks which cannot be covered by the given shift types are never used as an input for the branch-and-price algorithm.

7.8. Experimental Results

The above algorithm has been implemented in Visual C++ 7.1, using libraries of the open software initiative COIN-OR (Common Optimisation Interface for Operations Research) [Ralphs and Ladányi, 2001] [Lougee-Heimer, 2003]. CLP of COIN-OR was used as LP solver, and BCP (Branch, Cut & Price) provided the basis for the branch-and-price implementation. OSI (Open Solver Interface) builds the bridge between BCP and CLP. CLP proved to be robust and sufficiently efficient on the test cases. By means of OSI, the implementation is independent of the LP solver which may be exchanged in the future. Originally, BCP did not provide a handling of global lower bounds as described in Section 7.6, so this feature was implemented as a part of this work. In fact, branch-and-bound could often be terminated prematurely using global lower bound checks.

For the tests, the shift planning construction heuristic described in Chapter 5 was used to provide the branch-and-price algorithm with a starting solution. An optimised dynamic programming algorithm was used for the subproblems. In every iteration, one shift is added to the master program for each shift type for which a reduced cost column exists. Integrality is achieved by aggregate branching on task pairs. Tests were carried out on a AMD-2000+ computer (1.67 GHz) with 512 MB main memory and operating system Windows XP SP1.

Table 7.1 summarises basic figures of the scenario set C (see also Chapter 5) which comprises

No.	days	tasks	task minutes	travel time range	shift types	abs. min. restrictions	abs. max. restrictions
C01	8	2934	108754	[0,0]	35	0	0
C02	8	2813	230257	[0,13]	198	0	0
C03	8	2128	57470	[0,26]	74	0	0
C04	8	1429	65035	[0,26]	67	0	0
C05	8	1027	46855	[0,19]	65	0	0
C06	8	1598	76005	[0,26]	74	0	0
C07	4	2588	176538	[0,0]	12	12	8
C08	4	1816	160206	[0,0]	12	8	8
C09	3	1718	123407	[0,0]	17	12	12
C10	3	860	96505	[0,0]	11	0	0
C11	3	327	21680	[0,0]	17	0	0
C12	3	572	33550	[0,0]	14	0	0
C13	8	3517	141030	[0,6]	9	35	14
C14	8	1297	25940	[0,13]	24	0	0
C15	8	149	3725	[0,1]	94	0	0
C16	8	255	17895	[0,28]	94	0	0
C17	7	2256	87770	[0,3]	5	0	0

Table 7.1.: Scenario data.

exactly the scenarios to which the branch-and-price algorithm applies. For the number of tasks and task minutes (sum of the task lengths) in Table 7.1, tasks which cannot be assigned to any of the shift types were omitted.

The original scenarios C01 through C06 and C14 through C16 define huge numbers of shift types (up to 270). This makes solution by the above algorithm difficult because all possible alternatives of covering tasks by different shift types would have to be evaluated in a huge number of subproblems. Note that the local improvement procedure of Chapter 5 does not suffer from this problem if the number of break rule days is low.

However, the use of many different shift types in a plan is often not desired, and the construction heuristic described in Section 5.1 effectively tends to employ only a small subset of the shift types. As described in Section 2.9, it is common practice to use only a *working subset* of the shift types for planning, see e.g. Easton and Rossin [1991], Bechtold and Brusco [1994b] and Brusco and Jacobs [2001]. Sørensen and Clausen [2002] have noted that considerable administrative burden is associated with using many shift types. For the experiments on test cases C01 through C06 and C14, the sets of shift types was restricted to those used in the initial heuristic solution; these numbers are given in the shift type column of Table 7.1. Note that this means that if a shift type is used on one of the days, it will be available for all days of the scenario. As a consequence, the numbers of shift types are partly still considerable (up to 198). However, it should be mentioned that the selection biases the solution procedure towards the outcome of the construction heuristic whose shift type decisions may not be optimal. The evaluation of alternative working subset methods may be subject of future research.

To all of the scenarios, the decomposition procedure of Section 7.7 was applied. Results are summarised in Table 7.2, giving the number of components and the average number of tasks and shift type realisations (shift types unfolded per day) per component. Comparing the number of components to the scenario days given in Table 7.1, it can be seen that scenarios often decompose by day or even into smaller units. However, three of the scenarios (C14, C15 and C17) do not

No.	components	average number of tasks	average number of shift type realisations
C01	9	326.0	31.6
C02	8	351.6	198.0
C03	9	236.4	65.9
C04	8	178.6	67.0
C05	8	128.4	65.0
C06	8	199.8	74.0
C07	4	647.0	12.0
C08	4	454.0	12.0
C09	3	572.7	17.0
C10	3	286.7	11.0
C11	3	109.0	17.0
C12	4	143.0	10.5
C13	8	439.6	9.0
C14	1	1297.0	191.0
C15	1	149.0	785.0
C16	9	28.3	83.3
C17	1	2256.0	36.0

Table 7.2.: Decomposition results.

decompose at all. Effectively, these scenarios are test cases from airports with 24×7 operations. While the initial scenarios were partly huge, the number of tasks and shift type realisations is usually moderate after decomposition. Scenarios C02, C14, C15 and C17 are still challenging due to the numbers of tasks (up to an average of 2256 tasks per component) and shift type realisations (up to an average of 785).

As described in Section 7.3, one flow model is built for each shift type realisation. Average graph sizes per shift type realisation are given in Table 7.3. Due to breaks, each task may be represented by several nodes with 1.18 nodes per task on average. Note that the number of edges per shift type graph can be considerable when a graph contains many nodes, with a maximum of 52041 edges in one of the components of scenario C13. Differences in the node number can be partly explained by different task lengths (e.g. tasks in scenario C13 have average lengths of 40 minutes, tasks in scenario C16 average lengths of more than 70 minutes) and shift types (e.g. shift type durations are 5804 minutes on average in scenario C13 and 479 minutes in scenarios C15 and C16).

Each component was separately submitted to the branch-and-price procedure with a runtime limit of two hours. Table 7.4 presents solution data, summing up over all components of each scenario while the runtime data of Table 7.5 is given as averages over the components. Running times are given in seconds.

Astonishingly, the values of the LP relaxation are often equal to the value of the optimum solution a maximum gap of 0.29% on one component of scenario C09 (Table 7.4). On all but four cases, optimality with regard to the working subset of shift types could be proven for the best solution found by the branch-and-price algorithm. The algorithm ran into the runtime limit on all components of scenario C02 and one component of scenarios C03, C13 and C14, respectively. As a comparison of the numbers of explored nodes and the search tree depths suggests (Table 7.5), optimal solutions were nearly always found with a single descent. The only exception among the scenarios for which optimality was proven is built by one component of each of the scenarios C01,

No.	average tasks	average task nodes	average edges
C01	106.6	120.6	6082.3
C02	103.6	135.7	4233.4
C03	78.4	113.2	3406.5
C04	58.1	80.5	1548.2
C05	43.0	59.2	920.5
C06	69.1	96.2	2013.7
C07	260.3	286.4	23463.0
C08	169.9	185.6	8969.7
C09	229.8	256.8	17619.9
C10	123.4	129.8	4224.8
C11	40.0	48.1	708.7
C12	76.4	89.1	2607.1
C13	252.9	296.5	28601.3
C14	53.2	53.2	1533.1
C15	5.3	5.3	24.0
C16	9.7	9.7	50.7
C17	126.5	142.8	8124.5

Table 7.3.: Graph construction.

C09 and C17. This shows the excellent quality of the branching rule which basically consists in rounding up edge flows close to 1.

On the components for which optimality could not be proven, best known results were often obtained by the first descent in the search tree. The main difficulty of the affected components of scenarios C02, C03, C13 and C14 lies in the gap between the value of the LP relaxation and the value of the encountered solution. While the gaps show that the best solutions are within 0.37% and 0.16% of the optimal value, it is possible that parts of these solutions are in fact optimal. Hoffman and Padberg [1993] have noted that larger set partitioning problems often exhibit worse integer properties and lower bounds than smaller problem instances; the same might be true for the above set covering model. It is interesting to note that test cases C02 and C14 are among the scenarios with the highest numbers of shift type realisations per component. The adverse effect high numbers of potential shifts can also be seen from the time for solving the LP relaxation (Table 7.5).

While the lower bound quality plays an essential role, we can also observe an influence of subproblem complexity on the runtimes (Table 7.5). Scenarios C07 and C13 which ask for the highest number of edges in the subproblem graphs are among the scenarios consuming most runtime. Note however that these times do not stem from the solution of the subproblems whose runtimes can nearly be neglected. Instead, computation time is mainly consumed by solving the potentially large linear programs which result from the problem complexity.

Especially near the optimum, it will not be possible to generate columns with negative reduced costs for all subproblems. As can be seen from Table 7.5, subproblems yield an average of 0.5 columns per call which are added to the restricted master program.

The highest runtime among the scenarios solved to optimality is consumed by scenario C17 which contains the highest number of tasks since it cannot be decomposed. This shows the importance of problem decomposition. Problems C11, C15 and C16 turned out to be very easy to solve with a maximum of 28 search nodes for one component of scenario C11. In total, runtimes

No.	initial solution	LP relaxation	optimum	best solution
C01	258484.5	252380.0	252380.0	252380.0
C02	73125.0	56327.3	–	56507.0
C03	25508.0	22841.5	–	22845.0
C04	24690.0	22121.0	22121.0	22121.0
C05	18622.0	16330.0	16330.0	16330.0
C06	29777.0	26329.0	26329.0	26329.0
C07	55680.0	51200.0	51200.0	51200.0
C08	45440.0	40960.0	40960.0	40960.0
C09	42000.0	39453.4	39520.0	39520.0
C10	25600.0	24800.0	24800.0	24800.0
C11	7280.0	7040.0	7040.0	7040.0
C12	13120.0	12400.0	12400.0	12400.0
C13	71168.0	56836.5	–	56840.0
C14	13190.0	12550.0	–	12560.0
C15	3260.0	3090.0	3090.0	3090.0
C16	8100.0	7020.0	7020.0	7020.0
C17	45768.5	44329.5	44329.5	44329.5

Table 7.4.: Branch-and-price results.

were rather moderate, and many real-world problem instances turn out to be within the reach of the optimal algorithm which was not expected before this work. This also shows that worst-case complexity results like in Section 7.2 do not always admit conclusions on the solution of practical problem instances.

Table 7.6 compares improvements achieved by the local improvement method of Chapter 5 and branch-and-price. Results of large neighbourhood search (LNS) are given as averages over five runs after 30 minutes runtime. It should be noted that branch-and-price sometimes necessitates much more runtime. It can be seen that while local improvement often yields moderate improvements, branch-and-price finds solutions which save average shift costs of 8.95%. Especially on scenarios with very few shift types (e.g. C13), LNS fails to find substantial improvement. In contrast, scenarios C15 and C16 involve multitudes of shifts, and results are nearly equivalent. This shows that while local search is an appropriate method for the handling of general large-scale scenarios, branch-and-price is a suitable technique for scenarios with limited constraints.

Further details for the comparison of initial solutions and branch-and-price results are given in Table 7.7. The number of shifts is effectively reduced by up to 30.8%. Furthermore, the shift utilisation (task minutes divided by shift minutes without breaks) which is rather low in some test cases could be increased by up to 16.9%.

In total, the branch-and-price algorithms has shown to be very efficient in solving real-world shift planning problems with restricted constraints. The decomposition procedure makes problems amenable to exact solution, and many scenarios can effectively be solved to optimality. This is mainly due to the quality of the LP lower bound and the success of the branching rule. The fact that the best solution was nearly always found within a single descent suggests that the algorithm could also be used heuristically by restricting the tree search to the left-most branch, see also Grönkvist [1998]. Results were very satisfying and show that integer column generation is an appropriate approach for restricted ground staff planning.

No.	total time	time to LP relaxation	column generator calls	number of generated variables	time for subproblems	search nodes	search depth
C01	408.31	35.73	15096.9	11247.6	0.08	247.7	220.1
C02	7200.00	425.73	326477.5	44568.6	2.98	540.6	169.9
C03	1323.53	57.42	50876.4	26045.2	0.37	405.6	157.6
C04	67.75	12.28	8746.5	4819.0	0.04	77.4	77.4
C05	30.17	6.30	6071.4	2808.3	0.03	52.6	52.6
C06	97.58	19.05	9324.5	4252.1	0.05	75.5	75.5
C07	608.62	53.97	8192.8	6420.5	0.05	326.8	326.8
C08	177.50	32.67	4513.3	3637.8	0.03	193.8	193.8
C09	370.19	37.20	10231.7	6354.0	0.04	227.7	159.7
C10	12.94	3.75	1284.3	751.3	0.00	53.3	53.3
C11	0.78	0.27	643.7	14.3	0.00	15.3	15.3
C12	9.24	1.00	1143.8	679.3	0.00	38.0	38.0
C13	2539.41	99.94	25895.5	20945.3	0.14	576.4	288.5
C14	7200.00	129.73	750248.0	333165.0	14.79	1624.0	833.0
C15	1.72	0.51	5495.0	885.0	0.02	2.0	2.0
C16	0.18	0.10	971.1	300.8	0.00	1.8	1.8
C17	4744.44	83.11	171270.0	82992.0	2.64	1849.0	947.0

Table 7.5.: Runtime data.

No.	initial solution	LNS		branch-and-price	
		solution	improvement	solution	improvement
C01	258484.5	254971.0	1.36%	252380.0	2.36%
C02	73125.0	71547.0	2.16%	56507.0	22.73%
C03	25508.0	25478.0	0.12%	22845.0	10.44%
C04	24690.0	24245.0	1.80%	22121.0	10.41%
C05	18622.0	18189.0	2.33%	16330.0	12.31%
C06	29777.0	29685.0	0.31%	26329.0	11.58%
C07	55680.0	55680.0	0.00%	51200.0	8.05%
C08	45440.0	45280.0	0.35%	40960.0	9.86%
C09	42000.0	42000.0	0.00%	39520.0	5.90%
C10	25600.0	25520.0	0.31%	24800.0	3.13%
C11	7280.0	7280.0	0.00%	7040.0	3.30%
C12	13120.0	12880.0	1.83%	12400.0	5.49%
C13	71168.0	71168.0	0.00%	56840.0	20.13%
C14	13190.0	12950.3	1.82%	12560.0	4.78%
C15	3260.0	3100.0	4.91%	3090.0	5.21%
C16	8100.0	7050.0	12.96%	7020.0	13.33%
C17	45768.5	45557.0	0.46%	44329.5	3.14%

Table 7.6.: Shift cost improvement by LNS and branch-and-price.

No.	initial solution				branch-and-price solution			
	number of shifts	shift minutes	shift minutes without breaks	shift utilisation	number of shifts	shift minutes	shift minutes without breaks	shift utilisation
C01	578	208320	199620	54.5%	560	198720	190680	57.0%
C02	1010	361710	323550	71.2%	699	292980	261540	88.0%
C03	361	124950	111120	51.7%	335	110940	98310	58.5%
C04	380	126600	111960	58.1%	342	111870	98970	65.7%
C05	284	97380	86130	54.4%	255	85050	75780	61.8%
C06	499	148020	131310	57.9%	448	129690	115080	66.0%
C07	696	336864	295104	59.8%	640	309760	271360	65.1%
C08	568	274912	240832	66.5%	512	247808	217088	73.8%
C09	525	251700	220200	56.0%	494	238396	208756	59.1%
C10	320	160320	141120	68.4%	310	155310	136710	70.6%
C11	91	42186	36726	59.0%	88	41133	35853	60.5%
C12	164	76074	66234	50.7%	155	72210	62910	53.3%
C13	716	435600	414120	34.1%	715	349620	328170	43.0%
C14	171	79140	79140	32.8%	161	75360	75360	34.4%
C15	45	19560	19560	19.0%	41	18540	18540	20.1%
C16	129	48600	48600	36.8%	99	42120	42120	42.5%
C17	469	247320	233250	37.6%	455	239400	225750	38.9%

Table 7.7.: Comparison of results.

7.9. Conclusions and Future Research

We have tackled a restricted shift planning problem, comprising tasks which are fixed in time, shifts with breaks and absolute shift number restrictions. A considerable number of practical planning instances is covered by this problem class, justifying the search for efficient algorithms exploiting specific problem structures.

Restricted shift planning is basically a combination of classical vehicle and shift scheduling problems. We have shown that covering fixed tasks by an efficient set of shift duties can be interpreted as a special multiple depot vehicle scheduling problem (MDVSP) or, more generally, as a multicommodity flow problem. We have proven that simplified shift planning is $\mathcal{NP}$-hard in the strong sense which also means that general task-level shift planning is computationally hard.

We have shown how the problem can be represented by a specialised flow model. The resulting block-structured constraint matrix can be exploited by Dantzig-Wolfe decomposition. We have therefore derived a column generation formulation in which columns can be naturally interpreted as valid shift duties. For achieving integrality, a branching rule was devised which is compatible with the shortest path subproblems.

On airports with night-flying restrictions or very low workloads within the days, shift planning problems often naturally decompose. This has been exploited by a decomposition procedure on the basis of a constraint graph that expresses interdependencies between decisions.

It was shown how most test cases can be solved to proven optimality by branch-and-price. In view of the $\mathcal{NP}$-hardness result, this is remarkable and was not expected before this work. Compared to an initial heuristic algorithm, shift costs can be decreased by up to 22.7%. Additionally, solution times are mostly moderate. The algorithm is part of a commercial staff scheduling system and has proven to be a very powerful tool in practice.

While results were very satifying, there are still possibilities for improvement. Dual values are known to oscillate heavily in column generation approaches with an adverse effect on convergence [Desrosiers and Lübbecke, 2003]. Preliminary tests with the stabilisation schemes of du Merle et al. [1999] and Neame [1999] have shown that parameter tuning is quite difficult in these methods, and stabilisation did not have the desired effect. Nonetheless, a systematic exploration or enhancements might reveal that computation times can still be improved.

For two problem instances, optimality of the solutions could not be proven. A possible remedy is the use of cutting planes in order to improve the lower bound still further, resulting in a branch-and-price-and-cut approach. While some separation procedures for strong cuts in set covering formulations have been proposed (see Cornuéjols and Sassano [1989], Sassano [1989] and Nobili and Sassano [1989]), these are computationally expensive, and little has been reported on their computational success. However, future progress may make this approach practical.

In an application to the cutting stock problem, Valério de Carvalho [2002] has shown that the introduction of additional dual cuts can speed up column generation. While the design of dual cuts is highly problem-dependent, the approach may also be interesting for routing and scheduling applications.

However, results were beyond expectations, and future directions mainly refer to enrichments of the model. When time windows are included into the model, we have to avoid negative reduced costs cycles in the subproblems [Cordeau et al., 2001a]. Each subproblem is then an elementary shortest path problem which is $\mathcal{NP}$-hard in the strong sense [Dror, 1994]. Different solution approaches have been proposed, see e.g. Kohl et al. [1999] and Irnich and Villeneuve [2003].

Additionally, we may include qualification restrictions by imposing resource constraints in the subproblems. The resulting resource-constrained shortest path problems are usually solved by pseudo-polynomial dynamic programming algorithms, see also Section 9.6. However, the representational complexity in maintaining aggregated qualification sets increases exponentially in the number of involved qualifications.

More flexibility for the incorporation of further constraints seems to be offered by methods combining constraint programming with integer programming. Different schemes have been proposed, including constraint-based column generation [Fahle et al., 2002] [Rousseau et al., 2002] and cost-based domain filtering [Focacci et al., 1998] [Focacci et al., 1999]. The basic idea of these techniques is to represent complex constraints in a CP model while the search is guided by linear programming. Clearly, this is also a promising approach for task-level shift planning.

8. Implicit Modelling of Flexible Break Placement

Tea breaks do not need to be scheduled.
— Adel Gaballa, Wayne Pearce,
Telephone Sales Manpower Planning at Qantas,
Interfaces Vol. 9, 1979

In Chapters 5 and 7, we have described models and algorithms for task-level shift planning, integrating aspects of shift scheduling and vehicle routing. We now turn to shift and tour scheduling models which aim at covering a demand curve of workloads given in discrete time periods. While task-level planning is more detailed, histogram-based scheduling is sufficient in many planning situations (see also Chapter 1). On the one hand, covering a demand curve yields a good approximation to task-level scheduling if tasks are little movable and few constraints are imposed on the task level. On the other hand, scheduling tasks may be over-detailed in long-term planning when information on flight events, passenger and load figures is less detailed.

The integer programming model developed in this chapter will provide the basis for an algorithm for cyclic roster generation. Cyclic rosters are usually designed several weeks or months ahead of time and are usually valid for a whole flight season. It is therefore appropriate to build solution algorithms upon a curve of aggregated workloads. As described in Dowling et al. [1997], the actual assignment of tasks can then be deferred to shortly before the day of operations. As cyclic rosters are rotating patterns such that each assigned employee works on any shift duty at a given time, workloads must also be homogeneous.

Demand-level scheduling is not only well-accepted and intuitive to staff planners, but also allows for additional planning flexibility. In practice, planners often do not fully cover workloads, but accept slight understaffing in periods of peak requirements [Gaballa and Pearce, 1979] [Dowling et al., 1997]. From a technical viewpoint, demand-level scheduling allows for efficient models and solution techniques and is often the only way to make shift scheduling and rostering problems computationally tractable [Ernst et al., 2004]. As described in Chapter 2, nearly all literature on shift and tour scheduling build upon demand curves of aggregate workloads.

We can suppose that levelled workforce requirements provide a more appropriate basis for demand-level shift planning and rostering. We will therefore assume that the following models and algorithms build upon the results of the levelling procedure of Chapter 4.

8.1. Introduction

In this chapter, we will develop a weekly shift scheduling formulation which serves as a basis for the cyclic rostering algorithm of Chapter 9. We therefore assume that given workforce requirements are cyclic, i.e. the first day follows the last day of the one-week scheduling horizon. Since rosters are usually created for a longer period in advance, planning should be based on a model week with typical flight events and passenger and baggage load figures. Our model will be designed for potentially continuous operations with shifts overlapping from one day to the other. The formulation will be based on the standard set covering formulation of Dantzig [1954] and solved by LP-based methods.

In task-level shift planning, breaks can be handled similarly to work tasks, see Section 5.2.1. For demand-based scheduling, different models have been proposed for the handling of flexible breaks (see Section 2.8). Since the publication of Bechtold and Jacobs [1990], flexible breaks have mostly been represented by implicit models which considerably reduce the size of LP formulations. The original model of Bechtold and Jacobs [1990] is restricted to discontinuous operations with one break per shift and equal break durations. Furthermore, break time windows must not exhibit a property named *extraordinary overlap* (EO). Aykin [1996] proposed an alternative implicit model which is more widely applicable and incorporates multiple breaks.

While the formulation of Bechtold and Jacobs [1990] generally necessitates less break variables, Aykin [2000] claimed that his formulation requires less nonzero constraint coefficients and showed its computational superiority on a set of cyclic shift scheduling problems. However, Aykin [2000] did not make use of a substitution idea of Bechtold and Jacobs [1990], reducing the number of nonzero constraint coefficients. In the tour scheduling experiments of Topaloglu and Ozkarahan [1998], the Bechtold/Jacobs formulation generally performed better, see also the annotations in Mehrotra et al. [2000]. Rekik et al. [2003] noted that the Bechtold/Jacobs model should be superior on problems in which many break variables can be shared among the shift types.

Most of the restrictive assumptions given in Bechtold and Jacobs [1990] have been relaxed by extensions of the basic model. Brusco and Jacobs [2000] and Aykin [2000] proposed to use wrap-around break variables to overcome the limitation to less-than-24h operations. Topaloglu and Ozkarahan [1998] and Aykin [2000] showed how more than one break can be incorporated in the formulation of Bechtold and Jacobs [1990]. Analogously, more than one break duration can be accounted for.

The extraordinary overlap condition has received far less attention which is possibly due to the fact that EO is not very frequent in real-world shift planning [Aykin, 2000]. In this chapter, we will show how the Bechtold/Jacobs model can be generalised to scenarios in which break windows exhibit extraordinary overlap. Furthermore, it will be described how the number of nonzero matrix elements can be further reduced by a partitioning of shift and break variables. We will show that the resulting formulation outperforms the Aykin [1996] model on a number of real-world airport test cases.

For the set covering formulation, the planning horizon is usually divided into periods of equal lengths, see e.g. Henderson and Berry [1976], Morris and Showalter [1983], Dowling et al. [1997], Brusco and Jacobs [1998b] and Rekik et al. [2003]. For computational reasons, the discretisation is often very coarse-grained with period lengths of 15, 30 or 60 minutes, see Thompson [1995]. We will show how workforce requirements can be represented more flexibly, adapting to the granularity of shift types and breaks of the scenario at hand. The proposed method allows for a more compact representation without sacrificing granularity at times requiring more representational detail.

The chapter will be structured as follows: In Section 8.2, we will review the basic set covering formulation for shift scheduling. Section 8.3 gives an illustrative introduction to implicit break modelling, leading to the integer programming formulation of Section 8.4. Section 8.5 contributes the new approach for avoiding extraordinary overlap to be incorporated in the extended model of Section 8.6 while Section 8.7 deals with the compact representation of labour requirements. Section 8.8 points out empirical advantages, and Section 8.9 concludes with a summary.

8.2. Basic Set Covering Formulation

We will start by introducing the generalised set covering formulation for shift scheduling, going back to Dantzig [1954]. Labour demands $d_t \in \mathbb{N}_0$ are given per discrete time period $t \in T = \{1, \ldots, |T|\}$ and cover one week. As before, the set of shift types is denoted by K. The set of days on which a shift type $k \in K$ can be realised is given by $N_k \subseteq N$ where $N := \{1, \ldots, 7\}$ is the set of all days of the one-week scheduling horizon. The costs of shift type $k \in K$ are given by c_k. S_{kn} is a decision variable denoting the number of shifts of shift type realisation (k, n). In the sequel, the term *shifts* will be used interchangeably with shift type realisations; note that shift instances are not distinguishable if we do not attribute tasks. The formulation will be cyclic, i.e. shifts starting on the last day can "wrap around" to the first day. We use a constraint matrix $[a_{tkn}]$ with $a_{tkn} = 1$ if shift (k, n) ($k \in K$, $n \in N_k$) covers demand period $t \in T$ and $a_{tkn} = 0$ otherwise.

Ground handling companies and airlines usually do not try to fully cover demands. In periods of high workloads, planners accept slight understaffing [Gaballa and Pearce, 1979] and rely on temporary employment companies to cover short-term shortages. In the cyclic roster algorithm which will be described in Chapter 9, we will introduce upper limits on the workforce size, meaning that the staff at hand may not be able to fully cover requirements. We therefore introduce shortage variables O_t, $t \in T$, which are penalised by costs of c^{sht} in the objective function, see e.g. Baker [1976], Keith [1979], Bailey [1985], Thompson [1993], Mason and Nielsen [1999] and Rekik et al. [2003]. These penalties should reflect actual costs of temporary staff as well as possible. We will also assume that shortage costs exceed average shift costs per time period.

With these definitions, the cyclic weekly shift scheduling problem reads as

$$\min \sum_{\substack{k \in K \\ n \in N_k}} c_k S_{kn} + \sum_{t \in T} c^{sht} O_t \tag{8.1}$$

subject to

$$\sum_{t \in T} a_{tkn} S_{kn} + O_t \geq d_t \qquad \forall\, t \in T \tag{8.2}$$

$$S_{kn} \geq 0 \text{ and integer } \forall\, k \in K, n \in N_k \tag{8.3}$$

$$O_t \geq 0 \text{ and integer } \forall\, t \in T \tag{8.4}$$

Objective function (8.1) seeks for a cost-minimal set of shift types, possibly accepting understaffing. S_{kn} gives the number of shifts of each realisation. The corresponding column in the constraint matrix indicates which periods are covered. If we imagine the right hand side d_t to be 1 for all t and the decision variables S_{kn} to be binary, the rows can be interpreted as a set which must be covered by subsets with the columns representing characteristic vectors of the subsets [Hromkovič, 2001]. Allowing d_t and S_{kn} to be greater than 1 corresponds to a generalisation to multisets. The resulting model is sometimes termed general (or generalised) set covering formulation, see e.g. Jacobs and Bechtold [1993].

8.3. Flexible Break Placement

Work regulations usually prescribe that shifts for full-time staff must contain at least one meal break and possibly further relief breaks. Clearly, workers are not available to cover workloads within the duration of their breaks. Union and legal regulations or company policies usually prescribe breaks to be taken within the limits of given time windows, specifying earliest and latest break start times (cf. Section 5.2.1). In the following presentation, we will first restrict ourselves to the case of exactly one break per shift. Furthermore, we will assume that breaks of different shift types have equal durations.

A first approach to handle break placement flexibility is to define one decision variable for each shift and each possible break placement, i.e. to replicate shift variables for each break placement [Dantzig, 1954]. The matrix entries in formulation (8.1)-(8.4) then contain a 0 for all time intervals

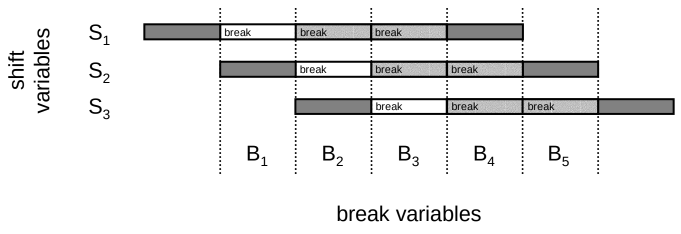


Figure 8.1.: Example for shifts with flexible breaks.

in which the corresponding break takes place. If breaks are not movable or break windows are tight, this is a feasible approach. In practice however, break windows frequently cover one or several hours, blowing up the LP formulation.

Gaballa and Pearce [1979], Bechtold and Jacobs [1990] and Aykin [1996] have proposed to use a distinct set of *break variables* to handle break placement flexibility. A break variable then indicates the number of breaks of a specific type. While Gaballa and Pearce [1979] and Aykin [1996] use one break variable for each break start time and shift, Bechtold and Jacobs [1990] share break variables among shifts and define a break variable B_l for each possible break start time.

Instead of encoding the attribution of breaks to shifts explicitly, Bechtold and Jacobs [1990] have shown how to formulate a set of *forward* and *backward constraints*. Under a number of conditions, these constraints ensure sufficient breaks to be available for all shifts. The actual attribution of breaks to shifts is then part of a downstream step. The (integer) linear program only encodes existence conditions on the attribution of breaks instead of incorporating complete information on break placement.

Instead of repeating the formal definition and proof of Bechtold and Jacobs [1990] and Bechtold and Jacobs [1996], we will give an intuitive introduction to implicit break handling. Fig. 8.1 shows three shifts, each consisting of five time periods, allowing for a break of one period in the second, third or fourth period of the shift. The number of breaks at each of the resulting five positions is represented by break variables B_1 through B_5. Note that for the moment being, we will omit the day indices from the shift variables S_1 through S_3.

If adequate numbers of shifts and breaks are given, the attribution of breaks to shifts can be interpreted as a transportation problem with restricted topology [Ford and Fulkerson, 1962] [Rekik et al., 2003]. For the above example, the corresponding bipartite graph is shown in Fig. 8.2: The nodes $V := \{v_1, v_2, v_3\}$ represent the shifts for which a "supply" equal to the number S_k of shifts is given. The nodes $W := \{w_1, \ldots, w_5\}$ represent breaks with "demands" corresponding to the break numbers B_l. Between each shift node $v \in V$, there is an edge toward break node $w \in W$ if and only if the break time window admits the corresponding break. The costs of "transportation" on these edges will be 0.

Transportation problems can be expressed as network flow algorithms, see e.g. Ahuja et al. [1993]. We therefore introduce an additional source node s and a sink (target) node t in the bipartite graph of Fig. 8.2. The source node is linked to all nodes in V by an edge of capacity equal to the number S_k of shifts associated with v. All nodes $w \in W$ are linked to the sink node by an edge of capacity B_l. The attribution edges between V and W have infinite capacities. The resulting graph for the above example is sketched in Fig. 8.3.

Our goal is the formulation of constraints between the variables which ensure that a feasible attribution of breaks to shifts exists. Let us first assume that we are given numbers S_k and B_l of shifts and breaks such that this attribution is possible. The numbers of shifts and breaks must clearly be equal, and we will set $N := \sum_k S_k = \sum_l B_l$. A feasible attribution of breaks to shifts

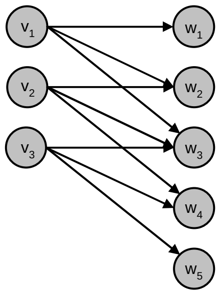


Figure 8.2.: Break attribution as a transportation problem.

then corresponds to a flow of cardinality N in the above network, i.e. a flow which saturates the edges in $\{s\} \times V$ and $W \times \{t\}$. We therefore have to ensure the existence of such a maximum flow. Çezik and Günlük [2002] call the underlying problem *bipartite flow feasibility problem*.

A *cut* is defined as a partition of the nodes $\{s,t\} \cup V \cup W$ into two parts C and $\bar{C}$. Note that a cut can also be identified with the arcs between C and $\bar{C}$ [Ahuja et al., 1993]. An *s-t cut* is a cut C such that $s \in C$ and $t \in \bar{C}$. According to the *min-cut max-flow theorem*, the maximum flow in a network corresponds to the capacity of the s-t cut of minimum capacity [Ford and Fulkerson, 1962]. In order to ensure that a flow of size N exists, we will require all s-t cuts to have capacities of at least N.

Clearly, any cut including edges between V and W has infinite capacity and will not limit the flow. Setting $C := \{s\}$, we cut all edges between s and V whose sum equals $\sum_k S_k$. The resulting cut capacity condition is $\sum_k S_k \geq N$ which is true by definition. Analogously, setting $C = \{s,t\} \cup V \cup W$ ($\bar{C} = \{t\}$) results in the trivial inequality $\sum_l B_l \geq N$.

Now imagine the non-trivial cut shown as red dotted line in Fig. 8.3a which corresponds to the cut set $C = \{v_1, w_1, w_2, w_3\}$. Comprising only edges from $\{s\} \times V$ and $W \times \{t\}$, its capacity is easily determined as $S_2 + S_3 + B_1 + B_2 + B_3$. As before, we require this sum to be at least N. Using $\sum_{i=1}^{3} S_i = N$, this can be formulated as

$$-S_1 + B_1 + B_2 + B_3 \geq 0 \tag{8.5}$$

in which N no longer appears. Including further nodes from W in the cut set or excluding nodes of V from C only leads to weaker constraints. As an example, take the cut of Fig. 8.4a, leading to

$$-S_1 + B_1 + B_2 + B_3 + B_4 \geq 0$$

which is dominated by (8.5).

As a second example, consider the cut of Fig. 8.3b. Enforcing the cut capacity $S_1 + S_2 + B_3 + B_4 + B_5$ to be larger or equal to N leads to the constraint

$$-S_3 + B_3 + B_4 + B_5 \geq 0 \tag{8.6}$$

As mentioned before, we must pay attention not to include any of the edges between V and W in a cut. As an example, imagine we would have included v_2 in the cut set of Fig. 8.3a, i.e. $C = \{v_1, v_2, w_1, w_2, w_3\}$. Since this cut includes the edge (v_2, w_4) of infinite capacity, the cut capacity is trivially greater than N.

We have already seen that the cut of Fig. 8.4a leads to a dominated constraint. This is equally true for cuts of the type given in Fig. 8.4b. In this example, the cut $C = \{v_1, w_1, w_2, w_3, w_4, w_5\}$ leads to a constraint

$$-S_1 + B_1 + B_2 + B_3 + B_4 + B_5 \geq 0$$

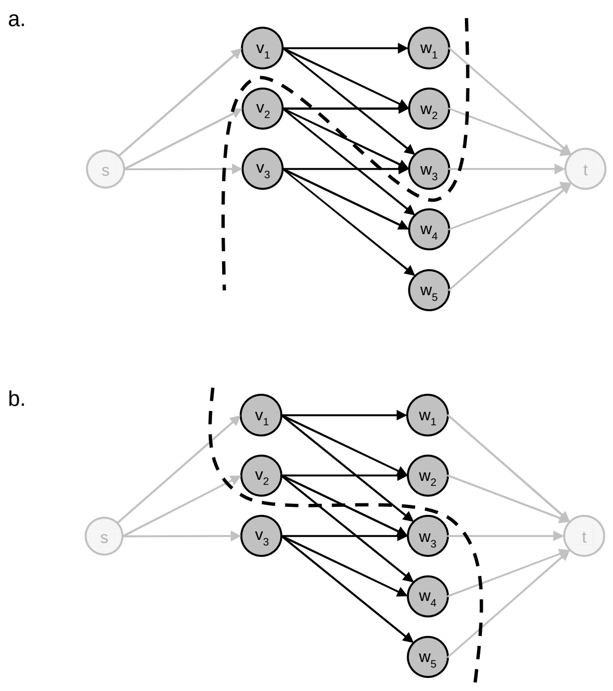


Figure 8.3.: Cuts in the break attribution network.

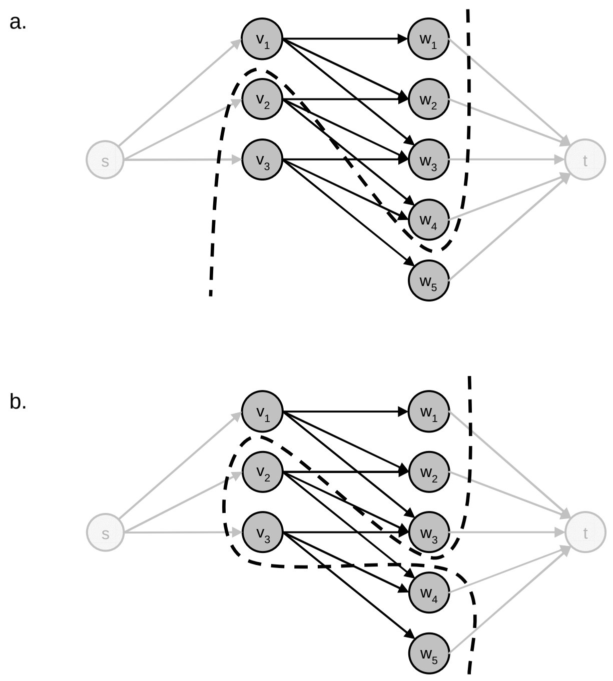


Figure 8.4.: Cuts leading to dominated constraints.

which is again dominated by (8.5). Effectively, only constraints of the types given in Fig. 8.3 lead to non-dominated constraints. Including more and more nodes from V and the corresponding nodes of W in cuts of Fig. 8.3a leads to a set of *forward constraints*. In the same way, we can gradually include more nodes in cut sets of Fig. 8.3b, starting from the last node of V, in order to to obtain a set of *backward constraints*. Adding the equality constraint on shifts and breaks, this leads to the following set of constraints for the above example:

$$\begin{array}{rrrrrrrrl}
-S_1 & & & +B_1 & +B_2 & +B_3 & & & \geq 0 \\
-S_1 & -S_2 & & +B_1 & +B_2 & +B_3 & +B_4 & & \geq 0 \\
& & -S_3 & & & +B_3 & +B_4 & +B_5 & \geq 0 \\
& -S_2 & -S_3 & & +B_2 & +B_3 & +B_4 & +B_5 & \geq 0 \\
-S_1 & -S_2 & -S_3 & +B_1 & +B_2 & +B_3 & +B_4 & +B_5 & = 0
\end{array}$$

8.4. Shift Scheduling with Breaks

With this illustration, we are now ready to give a formal definition of the implicit break model. Let $[est^b_{kn}, lst^b_{kn}]$ denote the break start time window of shift (k,n). Break windows will not be regarded as cyclic, i.e. if a shift starting on the seventh day admits breaks on the following day, we will assume that these breaks start on an imaginary eighth day. For each possible start time in $\bigcup_{\substack{k\in K\\ n\in N_k}} [est^b_{kn}, lst^b_{kn}]$, one break variable B_l is used, indexed by $l \in L$ in order of start times. Break indices associated with earliest break start times in the time windows of different shifts are denoted by L^{est}, and the set of breaks associated with latest start times L^{lst}. By $l^{est} := \min L^{est}$, we denote the overall earliest break and by $l^{lst} := \max L^{lst}$ the latest break.

For each break $l \in L$, we define the set $L^B(l) := \{l' \in L \,|\, l \leq l'\}$ of breaks starting no earlier than l. The corresponding set of shifts whose break time windows are subsets of $L^B(l)$ is $K^B(l) := \{(k,n) \,|\, k \in K, n \in N_k, [est^b_{kn}, lst^b_{kn}] \subseteq L^B(l)\}$. Analogously, we define a set $L^F(l) := \{l' \in L \,|\, l' \leq l\}$ of breaks starting no later than l and the set $K^F(l) := \{(k,n) \,|\, k \in K, n \in N_k, [est^b_{kn}, lst^b_{kn}] \subseteq L^F(l)\}$ of related shifts.

In the extended set covering formulation with breaks, the workload coverage by shifts is reduced by breaks. We therefore introduce a coefficient matrix $[b_{tl}]$ with $b_{tl} = 1$ if break l covers period t and $b_{tl} = 0$ otherwise. For these coverage coefficients, break times are considered as cyclic, i.e. a break on the eighth day reduces the coverage on the first day. Consequently, shifts starting on the first and last day of the one-week scheduling horizon use different break variables even if these breaks refer to the same actual start times. This idea of *wrap-around break variables* was independently proposed by Aykin [2000] and Brusco and Jacobs [2000].

Based on model (8.1)-(8.4), the set covering formulation including breaks reads as

$$\min \sum_{\substack{k\in K\\ n\in N_k}} c_k S_{kn} + c^{sht} O_t$$

subject to

$$\sum_{\substack{k\in K\\ n\in N_k}} a_{tkn} S_{kn} - \sum_{l\in L} b_{tl} B_l + O_t \geq d_t \qquad \forall\, t \in T \tag{8.7}$$

$$-\sum_{(k,n)\in K^F(l)} S_{kn} + \sum_{l'\in L^F(l)} B_{l'} \geq 0 \qquad \forall\, l \in L^{lst} \setminus \{l^{lst}\} \tag{8.8}$$

$$-\sum_{(k,n)\in K^B(l)} S_{kn} + \sum_{l'\in L^B(l)} B_{l'} \geq 0 \qquad \forall\, l \in L^{est} \setminus \{l^{est}\} \tag{8.9}$$

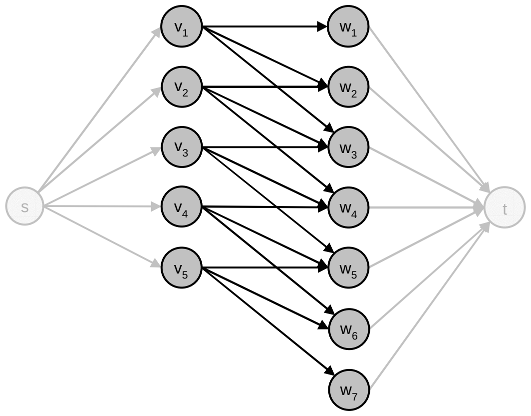


Figure 8.5.: Example network for equivalence proof of implicit break handling.

$$\sum_{\substack{k\in K\\ n\in N_k}} S_{kn} - \sum_{l\in L} B_l = 0 \tag{8.10}$$

$$S_{kn} \geq 0 \text{ and integer } \forall\, k \in K, n \in N_k \tag{8.11}$$

$$B_l \geq 0 \text{ and integer } \forall\, l \in L \tag{8.12}$$

$$O_t \geq 0 \text{ and integer } \forall\, t \in T \tag{8.13}$$

Inequalities (8.7) are the coverage constraints. (8.8) and (8.9) are sets of forward and backward constraints, spanning over successively more break and shift variables in forward and backward direction, respectively. Constraint (8.10) ensures that shift numbers match break numbers. All variables are nonnegative and integer ((8.11), (8.12), (8.13)).

Bechtold and Jacobs [1996] prove that forward and backward constraints and the equality constraint are sufficient conditions for the existence of a break attribution if no *extraordinary overlap* (EO) exists. Let (k_1, n_1) and (k_2, n_2) be two shifts. The break start time windows $[est^b_{k_1n_1}, lst^b_{k_1n_1}]$ and $[est^b_{k_2n_2}, lst^b_{k_2n_2}]$ of (k_1, n_1) and (k_2, n_2) exhibit extraordinary overlap if the break start time window of (k_1, n_1) completely lies in the interior of the break start time window of (k_2, n_2) (or vice-versa), i.e. $est^b_{k_2n_2} < est^b_{k_1n_1}$ and $lst^b_{k_1n_1} < lst^b_{k_2n_2}$.

We will shortly illustrate basic ideas of the proof in Bechtold and Jacobs [1996] which we will refer to in Section 9.4. We must show that sufficient numbers of breaks are available for each set of shifts. Any set of shifts can be divided into *associated sets* of shifts which compete for breaks which are contained in the break windows of both shifts. Since each set of shifts is made up of disjoint sets of associated shift which do not mutually compete for breaks, it is sufficient to show that for each associated set of shifts, sufficient numbers of breaks are available.

Related to a given set of associated shifts is a number of breaks. We will now extend the shift set by all shifts whose valid breaks are subsets of this set of breaks. As an example, take the associated set $\{S_1, S_3\}$ and the respective set $\{v_1, v_3\}$ of nodes in Fig. 8.5. These shifts compete for breaks $\{B_1, \ldots, B_5\}$ corresponding to the nodes $\{w_1, \ldots, w_5\}$. However, S_2 also competes for these breaks. Proving the sufficiency of breaks for the extended set of shifts effectively provides a stronger condition than using the original shift set.

If the enlarged associated set of shifts appears in a forward or backward constraint, nothing remains to be shown. As an example, the set $\{S_1, S_2, S_3\}$ in the above example makes part of a forward constraint. Let us therefore take the associated set $\{S_2, S_3, S_4\}$ corresponding to the nodes $\{v_2, v_3, v_4\}$ in Fig. 8.5. The corresponding break set is $\{B_2, B_3, B_4, B_5, B_6\}$. One forward and one backward constraint span around this set of breaks, namely the forward constraint

$$B_1 + B_2 + B_3 + B_4 + B_5 + B_6 \geq S_1 + S_2 + S_3 + S_4$$

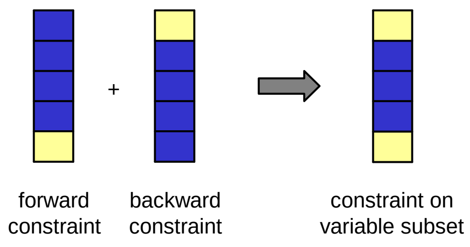


Figure 8.6.: Graphical illustration of variable sets in forward/backward constraints.

and the backward constraint

$$B_2 + B_3 + B_4 + B_5 + B_6 + B_7 \geq S_2 + S_3 + S_4 + S_5$$

Subtracting from the equality condition $\sum_{k=1}^{5} S_k = \sum_{l=1}^{7} B_l$ yields the inequalities $B_7 \leq S_5$ and $B_1 \leq S_1$.

Due to the construction of forward and backward constraints, the sets of break variables in both of these constraints must be disjoint. Furthermore, it can be shown that by the absence of EO, the sets of shift variables have to be disjoint. We can therefore add the latter constraints and substitute in the equality constraint, yielding

$$B_2 + B_3 + B_4 + B_5 + B_6 \geq S_2 + S_3 + S_4$$

This shows that sufficient numbers of breaks are available for the given set of shifts. The proof thus basically combines forward and backward constraints, making use of the fact that the complementary sets of shift and break variables are disjoint if EO is absent. This idea is illustrated in Fig. 8.6. Each box represents one variable, e.g. a break variable. Boxes which are covered by the respective constraints have blue colour. The complementary variable sets (shown in light yellow) are in fact disjoint, and we can combine a forward and backward constraint to generate a new implicit constraint on a variable subset (represented by the boxes on the right). We will get back to this illustration in Section 9.4 where we will show that this no longer holds when generalising implicit modelling to two dimensions.

To show the effect of extraordinary overlap, imagine that in the above example, shift S_3 only admits a break of type B_3, i.e. the edges (v_3, w_4) and (v_3, w_5) would not be part of the network in Fig. 8.5. This introduces EO since the break window $(\{B_3\})$ of S_3 is completely contained in the break window of S_2 $(\{B_2, B_3, B_4\})$. Then forward and backward constraint do not generally ensure a feasible assignment. As an example, it can be easily verified that setting $S_2 = 1$, $S_3 = 1$, $B_2 = 1$ and $B_4 = 1$ (and all other decision variables to 0) obeys all forward and backward constraints without admitting a feasible break assignment to S_3. In fact, the complementary variable sets of the forward constraint $B_1 + B_2 + B_3 \geq S_1 + S_3$ and the backward constraint $B_3 + B_4 + B_5 + B_6 + B_7 \geq S_3 + S_4 + S_5$ are no longer disjoint, and the necessary constraint $B_3 \geq S_3$ is not implicit.

An interesting alternative proof for the sufficiency of forward/backward constraints has been given by Rekik et al. [2003]. It is based on an explicit formulation of the transportation problem with variables denoting the number of breaks of given types assigned to a specific shift. Rekik et al. [2003] apply Benders' reformulation to this model and retrieve existence conditions for the break assignment as cutting constraints on the rays of the dual program. It is then shown that most inequalities are dominated by others when EO is absent while the remaining conditions build the forward/backward constraint set.

A number of researchers have used the aforementioned formulation to represent flexible break placement, including Jarrah et al. [1994], Thompson [1995] and Brusco and Jacobs [2000]. It allows for a formulation with only one variable for each break start time if all break durations are equal and if EO is absent. Furthermore, the number of forward and backward constraints is linear in the number of shifts.

Alternatively, Gaballa and Pearce [1979] have proposed to use one break variable for each start time and each shift. Aykin [1996] generalises this idea, using one set of break variables for each of three breaks in a shift. In order to match break and shift numbers, one equality constraint is created for each shift. As in the model of Bechtold and Jacobs [1990], the number of constraints is linear in the number of shifts. However, many more break variables are needed in general. A main advantage of the model of Gaballa and Pearce [1979] and Aykin [2000] is its generality since it naturally incorporates multiple breaks of different durations. Furthermore, it is not restricted to settings without extraordinary overlap.

Aykin [2000] shows that while his model uses more break variables, it often entails lower numbers of nonzeros in the constraint matrix. On a set of cyclic shift scheduling problems, he demonstrates the computational superiority of his approach. However, Topaloglu and Ozkarahan [1998] showed that on a set of tour scheduling problems, the Bechtold/Jacobs model generally performed better. Aykin [2000] does not make use of an idea given by Bechtold and Jacobs [1990], reducing the number of nonzero elements in the constraint matrix. To explain this transformation, let

$$-\sum_{(k,n)\in K^F(l)} S_{kn} + \sum_{l'\in L^F(l)} B_{l'} \geq 0$$

be the forward constraint for an $l \in L^{lst} \setminus \{l^{lst}\}$. Adding the equality constraint (8.10) (and multiplying by -1) yields the complementary constraint

$$-\sum_{\substack{(k,n)\in K\setminus K^F(l)\\ n\in N_k}} S_{kn} + \sum_{l'\in L\setminus L^F(l)} B_{l'} \leq 0$$

which may involve less nonzero coefficients. The analogous transformation can be applied to backward constraints. It is likely that this idea would have resulted in better results on the test cases considered by Aykin [2000].

It is worth mentioning that implicit modelling is not restricted to break representation. As summarised in Section 2.8, other applications include implicit models of flexible shift start times [Moondra, 1976] [Thompson, 1995], tour scheduling [Bailey, 1985] [Çezik et al., 2001] and start-time bands in tour scheduling [Jacobs and Brusco, 1996] [Brusco and Jacobs, 2000], limited employee availability [Thompson, 1990] and flexible placement of blocks of "controllable work" [Thompson, 1992]. A general methodology for implicit models has been given by Çezik and Günlük [2002], including an analysis of representational complexities under different assumptions.

8.5. Avoiding Extraordinary Overlap

Several publications have shown how to overcome restrictions of the Bechtold/Jacobs model. As described above, wrap-around break variables can be used to break cyclicity [Aykin, 2000] [Brusco and Jacobs, 2000]. If shifts contain more than one break, Topaloglu and Ozkarahan [1998] and Aykin [2000] propose to employ several sets of break variables and forward/backward constraints. Analogously, different break durations can be tackled by introducing separate variables and constraints for each duration [Rekik et al., 2003].

Surprisingly, the extraordinary overlap condition has received far less attention. This is probably due to the fact that EO is not very frequent in real-world scheduling problems. Aykin [2000] notes that break lengths and start time windows usually do not change with the shift types. Using the terminology of Chapter 5.2.1, real-world scenarios often use few break rules which define break durations and time windows relative to shift starting times. Bechtold and Jacobs [1996] note that even if break windows exhibit some extraordinary overlap, insufficient break availabilities are not very frequent even if forward/backward constraints do not guarantee the existence of an assignment.

Çezik and Günlük [2002] show that when EO is present, we can use a set of constraints which is quadratic in the number of shifts. Rekik et al. [2003] propose to add break constraints dynamically from Benders' reformulation of the transportation problem, ensuring feasibility even if EO is present. However, this solution is rather costly and potentially generates many dominated constraints. Another remedy consists in using the break model of Gaballa and Pearce [1979] and Aykin [1996] which does not suffer from problems with extraordinary overlap. However, this model generally requires many more break variables. Furthermore, we would not exploit the fact that EO is not very frequent in practice.

In the following, we will show how EO can be explicitly handled within the model of Bechtold and Jacobs [1990]. The basic idea is to partition the set of shifts such that break windows in each shift class do not exhibit extraordinary overlap. We can then use one set of break variables and forward/backward constraints (and one equality constraint) for each shift class. Let $G = (V, E)$ be a directed graph representing extraordinary overlap relations. We create one node v for each shift (k, n) and associate the break window $[est^b_{kn}, lst^b_{kn}] =: [est^b_v, lst^b_v]$ with v. G contains one edge for each pair (v_1, v_2) such that the break window associated with v_1 covers the break window of v_2 in the sense of extraordinary overlap, i.e.

$$E := \{(v_1, v_2) \mid est^b_{v_1} < est^b_{v_2}, lst^b_{v_2} < lst^b_{v_1}\}$$

The graph construction clearly necessitates $\mathcal{O}(|V|^2)$ operations.

We note that extraordinary overlap is a transitive relation, i.e. if $(v_1, v_2) \in E$ and $(v_2, v_3) \in E$, then also $(v_1, v_3) \in E$. If $(v_1, v_2) \in E$, the break window associated with v_2 is strictly tighter than the interval of valid break start times of v_1, meaning that G does not contain cycles. If the edges are interpreted as undirected, the EO graph is a *comparability graph*, and the directed edges in E represent a *transitive orientation* [Golumbic, 1980].

Our goal is to find a partition of V into pairwise disjoint subsets $V_0, \ldots, V_{m-1}$ such that

$$\bigcup_{i=0}^{m-1} V_i = V$$

and there is no edge between any two vertices in a subset, i.e. shift classes are *EO-free*. This problem is closely related to the *graph colouring problem* if we interprete edges as undirected: Partition the set V of vertices into classes $V_0, \ldots, V_{m-1}$ (nodes in V_i are assigned colour i) such that adjacent vertices are in different classes (have different colours). While it is well-known that the colouring of general graphs is an $\mathcal{NP}$-hard problem, Golumbic [1977] shows that comparability graphs can be coloured in $\mathcal{O}(|E|)$ time if a transitive orientation is known.

In fact, the transitive orientation induces a natural partial ordering on the vertices of a comparability graph. We can therefore assign a *height* $h(v)$ to each vertex v. If v is a sink (i.e. no edges emanate from v), we set $h(v) := 0$. Otherwise, we assign $h(v) := 1 + \max\{h(w) \mid (v, w) \in E\}$. All vertices of equal height are then assigned the same colour. Clearly, the resulting colouring is feasible since adjacent vertices are assigned different colours. Furthermore, there must be a directed path of length $\max_{v \in V} h(v)$ in G. Because E is transitive, the vertices of this longest path

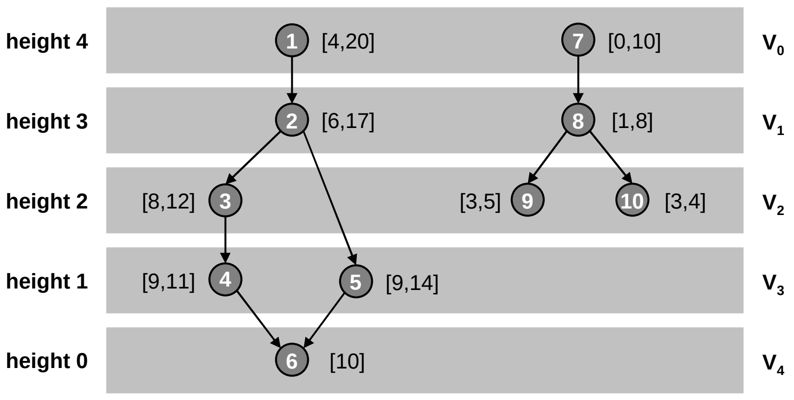


Figure 8.7.: Colouring of a comparability graph.

build a clique in the undirected graph corresponding to G, showing that the colouring is minimal. The height function $h : v \to \mathbb{N}_0$ can be determined in $\mathcal{O}(|E|)$ time by a recursive depth-first search [Golumbic, 1977].

As an example, take the graph of Fig. 8.7. The graph contains two connected components. Edges which follow from transitivity are not shown. Next to the nodes, the break windows $[est_v^b, lst_v^b]$ are given. Nodes are attributed to five layers corresponding to the colours. Due to transitivity, the path of vertices $\{1, 2, 3, 4, 6\}$ builds a clique in the undirected variant of the graph. Consequently, the chromatic number (minimum number of colours) of the graph equals five.

We now get back to our break representation problem. If the break windows of two shifts exhibit extraordinary overlap, these shifts share breaks of equal start times. Separating shifts and attributing a distinct set of breaks to each shift class therefore amounts to replicating variables for common breaks. As an example, vertices 2 and 5 in Fig. 8.7 share break start times 9 through 14. Since the nodes are linked by an edge, we would use separate break variables for all of these start times.

Consequently, we should try to place shifts which ask for common break start times in same shift classes. Minimising the number of required break variables, we look for an $m \in \mathbb{N}$ and a partition of V into classes $V_0, \ldots, V_{m-1}$ such that

$$\sum_{t \in \bigcup_{v \in V} [est_v^b, lst_v^b]} |\{i \mid \exists v \in V_i : t \in [est_v^b, lst_v^b]\}| \tag{8.14}$$

is minimised.

Clearly, the goal of graph colouring is advantageous toward this end. However, we can see from Fig. 8.7 that the height function always assigns vertices to the bottommost layer (node 5 is attributed a height of 1). It can furthermore be observed that nodes 1, 2, 3, and 5 ask for break start time 12. With the above colouring, this start time would be present on four layers. In contrast, attributing node 5 to layer (height) 2 results in only three break variables for start time 12. It thus seems to be advantageous to align nodes on the topmost layers.

This is exactly done by Algorithm 5. The EO graph $G = (V, E)$ is presented as input, and the partitioning of V into EO-free classes V_i, $i = 0, \ldots, m-1$, builds the output of the algorithm. In the description, $\delta^-(v) \subseteq E$ and $\delta^+(v) \subseteq E$ denote the in- and outedges of $v \in V$.

In comparison to the procedure for determining the height function, the order in which nodes are assigned to layers is reversed. The procedure basically amounts to imposing a topological order on the graph. Keeping track of the number of unvisited inedges $d[v]$ for each $v \in V$, the algorithm starts by assigning vertices without predecessors to layer 0. In each iteration, we visit

Algorithm 5 BREAKPARTITIONING

```
$i \leftarrow 0$
$d[v] \leftarrow |\delta^-(v)|$
$V_0 \leftarrow \{v \in V \mid d[v] = 0\}$
**repeat**
  **for all** $(v, w) \in \bigcup_{v' \in V_i} \delta^+(v')$ **do**
    $d[v] \leftarrow d[v] - 1$
    **if** $d[v] = 0$ **then**
      $V_{i+1} \leftarrow V_{i+1} \cup \{w\}$
    **end if**
  **end for**
  $i \leftarrow i + 1$
**until** $V_i = \emptyset$
```

the outedges of the nodes on the current layer and decrease the number of unvisited edges for adjacent nodes. In each iteration, the next layer is built by the vertices whose predecessors have completely been assigned to previous levels. Since each node either has an empty predecessor set or can be reached via a path from a node without predecessors, all nodes in V will be covered by exactly one of the sets V_i. Clearly, if the shift types of the original break partitioning problem do not exhibit extraordinary overlap, BREAKPARTITIONING degenerates, and V_0 will be equal to V.

Assuming an adjacency list representation of the graph, the initialisation phase of Algorithm 5 requires $\mathcal{O}(|V|)$ steps. The main loop handles each edge exactly once and thus requires $\mathcal{O}(|E|)$ operations. The overall BREAKPARTITIONING algorithm therefore runs in $\mathcal{O}(|V| + |E|) = \mathcal{O}(|E|)$ time.

It should be clear that BREAKPARTITIONING provides alternative colouring algorithm for comparability graphs. In contrast to the *height* procedure, vertex 5 in the above example (Fig. 8.7) is assigned to layer V_2 (height 2). It is interesting to note that the procedure does not make use of the break time windows associated with the nodes. Nevertheless, we can show that the resulting shift classes use the minimum possible number of breaks.

Theorem 4. BREAKPARTITIONING *solves the break partitioning problem to optimality.*

Proof. We have to prove that

$$\sum_{t \in \bigcup_{v \in V} [est_v^b, lst_v^b]} |\{i \mid \exists v \in V_i : t \in [est_v^b, lst_v^b]\}|$$

is minimal in the solution of BREAKPARTITIONING. Toward this end, we prove the stronger property that $|\{i \mid \exists v \in V_i : t \in [est_v^b, lst_v^b]\}|$ is minimal for each t.

For a given t, let $v \in V$ be a node whose break window covers t (i.e. $t \in [est_v^b, lst_v^b]$). Let v be on level i in the result of BREAKPARTITIONING ($v \in V_i$). Then there is a path $v_0, v_1, \dots, v_i = v$ such that $v_{i'} \in V_{i'}\ \forall i' = 0, \dots, i$, i.e. each level contains exactly one of the nodes. If v is the only node of its component on level V_i (e.g. node 6 in Fig. 8.7), this is obvious. If level V_i comprises several nodes "in parallel" (as for node 5), the claim follows from the fact that BREAKPARTITIONING places v on the uppermost possible level. Because the time windows of all predecessors in the path must be supersets of $[est_v^b, lst_v^b]$, all levels $V_0, \dots, V_i$ comprise start time t.

Let i_{max} be the last level containing a node v with $t \in [est_v^b, lst_v^b]$. By transitivity, the path $v_0, \dots, v_{i_{max}} = v$ with $v_{i'} \in V_{i'}$ is a clique if the graph is interpreted as undirected. Therefore,

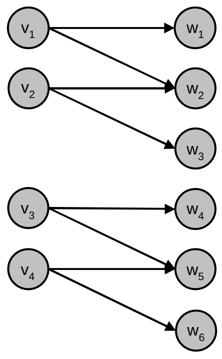


Figure 8.8.: Example for further separation of shift classes.

we need at least $i_{max} + 1$ EO-free classes to cover t. But this is exactly the number of partitions used by BREAKPARTITIONING. □

For each shift class, we will use one set of break variables and constraints. Since there is no extraordinary overlap between shifts in a class, forward and backward constraints ensure assignment feasibility for breaks and shifts. Using Algorithm 5, the resulting number of break variables is minimised. Since we still share break variables between different shifts, the number of variables will generally be lower than in the approach of Gaballa and Pearce [1979] and Aykin [1996]. However, as the experiments of Aykin [2000] indicate, the sparsity of the constraint matrix also seems to play an important role in solution efficiency.

Additionally to complementing forward and backward constraints as described in Section 8.4, we will make use of a further idea to reduce the number of nonzero constraint coefficients. Shift classes determined by BREAKPARTITIONING will usually be made up of subsets of shifts which do not share any breaks. This is especially true when using a one-week scheduling horizon since break windows of shifts starting on different days are often disjoint. As an example, imagine that a shift class exhibits the structure shown in Fig. 8.8. The shifts associated with v_1 and v_2 only compete for breaks w_1 through w_3 while shifts relating to v_3 and v_4 ask for breaks from $\{w_4, w_5, w_6\}$.

It is clear that such shift classes can be broken up, creating one constraint system for each set of associated shifts. This not only reduces the number of nonzero coefficients in the constraint matrix, but also reduces the number of constraints. In the example, the combined system would necessitate seven constraints (three forward and backward constraints, one equality constraint) while after separation, we only need six constraints (one forward, backward and equality constraint for each subsystem).

8.6. General Shift Scheduling with Multiple Breaks

Up to now, we have assumed that each shift contains exactly one break and that all breaks have equal durations. If a shift contains more than one break, we can simply use one set of break variables and constraints for each break type (e.g. early, main and late break), see Topaloglu and Ozkarahan [1998] and Aykin [2000]. Analogously, if several durations are used for the same break type (e.g. 30 and 60 minute main breaks), the break system can be split up further [Rekik et al., 2003]. We will therefore solve one break partitioning problem for each break type and duration. If there are subclasses of shifts which do not mutually compete for break variables, the resulting shift classes can be decomposed.

The result of this decomposition procedure is a set R of *break classes*. In order to illustrate the complete set covering model, we will extend the above identifiers by a break class index r. The set of all breaks for class $r \in R$ will be denoted by L_r while K_r are the shifts associated with r. Note that a shift can be part of several classes if it contains more than one break. The sets of earliest and latest break start times in the time windows of shifts in K_r are denoted by L_r^{est} and L_r^{lst}, respectively. $l_r^{est} := \min L_r^{est}$ and $l_r^{lst} := \max L_r^{lst}$ are the earliest and latest overall break start times in class $r \in R$. For a break $l \in L_r$, $L_r^B := \{l' \in L_r \mid l \leq l'\}$ and $L_r^F(l) := \{l' \in L_r \mid l' \leq l\}$ are the sets of breaks starting no earlier and no later than l, respectively. $K_r^B(l)$ and $K_r^F(l)$ are the corresponding sets of shifts whose break windows are completely covered by L_r^B and L_r^F.

B_{rl} is the decision variable for break l in class r. The coverage reduction coefficients are given by b_{trl} which equal 1 if t is a period covered by break l in class r and 0 otherwise. For reasons which will become clear in the sequel, the following set covering model for the weekly shift scheduling problem uses distinct shortage costs c_t^{sht} for each time interval $t \in T$.

$$\min \sum_{\substack{k \in K \\ n \in N_k}} c_k S_{kn} + \sum_{t \in T} c_t^{sht} O_t \tag{8.15}$$

subject to

$$\sum_{\substack{k \in K \\ n \in N_k}} a_{tkn} S_{kn} - \sum_{r \in R} \sum_{l \in L} b_{trl} B_{rl} + O_t \geq d_t \qquad \forall\, t \in T \tag{8.16}$$

$$- \sum_{(k,n) \in K_r^F(l)} S_{kn} + \sum_{l' \in L_r^F(l)} B_{rl'} \geq 0 \qquad \forall\, r \in R,\ \forall l \in L_r^{lst} \setminus \{l_r^{lst}\} \tag{8.17}$$

$$- \sum_{(k,n) \in K_r^B(l)} S_{kn} + \sum_{l' \in L_r^B(l)} B_{rl'} \geq 0 \qquad \forall\, r \in R,\ \forall l \in L_r^{est} \setminus \{l_r^{est}\} \tag{8.18}$$

$$\sum_{(k,n) \in K_r} S_{kn} - \sum_{l \in L_r} B_{rl} = 0 \qquad \forall\, r \in R \tag{8.19}$$

$$S_{kn} \geq 0 \text{ and integer } \forall\, k \in K, n \in N_k \tag{8.20}$$

$$B_{rl} \geq 0 \text{ and integer } \forall\, r \in R, l \in L_r \tag{8.21}$$

$$O_t \geq 0 \text{ and integer } \forall\, t \in T \tag{8.22}$$

As described before, the break constraints (8.17) through (8.19) only guarantee sufficient numbers of breaks for all shifts. To generate an actual shift plan, the break assignment problem must be solved. One possibility is to use the transportation problem associated with the shifts and breaks of each break class, e.g. by the network flow formulation used in Section 8.3. Alternatively, Bechtold and Jacobs [1990] describe a simple single-pass procedure.

8.7. Action Time Transformation

Up to now, we have tacitly assumed that the one-week scheduling horizon is divided into demand periods of equal lengths. While the workload levelling experiments in Chapter 4 were based on a minute discretisation, it will generally be sufficient to represent aggregated workloads on a more coarse-grained level, e.g. 15 minutes, reducing the number of coverage constraints. This is in accordance with the workforce scheduling literature proposing 15, 30 or even 60 minute discretisations, see e.g. Henderson and Berry [1976], Morris and Showalter [1983], Thompson [1995], Dowling et al. [1997], Brusco and Jacobs [2000] and Rekik et al. [2003].

While a 15- or 30-minute discretisation will generally be sufficient throughout times of low activity, we might opt for a finer structure in periods in which shifts or breaks start and end. In fact, shift start and end times may not coincide at all with given interval boundaries. In models

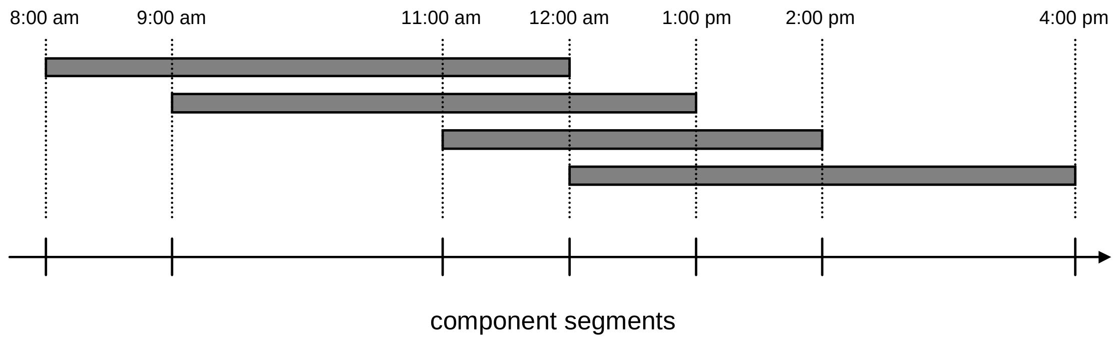


Figure 8.9.: Component segments of shifts and demand periods.

without breaks, Mason and Smith [1998] and Jaumard et al. [1998] independently proposed to divide the planning horizon into *component segments* (Jaumard et al. [1998]: *modified demand periods*) in which no shifts start and end. In such segments, the shift coverage will be uniform. Instead of using fixed-length time periods, the scheduling horizon is then segmented into intervals of variable durations, see Fig. 8.9.

Not only shift start and end times, but also starting and ending times of breaks defined *action times* [Thompson, 1996a]. If breaks can start within given time windows, we can assume a suitable discretisation for break start times, e.g. five minutes. Each interval between action times will then represent a demand period t in model (8.15)-(8.21). In general, there will be many times within the day without any shift or break start and end times, and a formulation based on modified demand periods will usually be more compact than a model with fixed length intervals. Furthermore, an action time transformation flexibly adapts to the given scenario, using more detail in times of many action times and less granularity in times of low activity.

We define the workforce demand of a modified demand period to be the maximum over the corresponding one minute intervals of the original demand curve. If no understaffing is used, covering the modified demand periods is equivalent to covering the original workload histogram (assuming a one minute discretisation for breaks). If shortage is admitted, we will define the understaffing cost c_t^{sht} of the modified demand period t as $c_t^{sht} := \kappa c^{sht}$ if κ is the length of the demand period. However, the transformed model will only approximate shortage costs if the original demands within the segment are not uniform. In fact, the model assumes all original workloads to be equal to the maximum over the interval.

One remedy consists in defining additional action times for each change in workload. This introduces many additional demand intervals while the error by a coarser-grained model is not important if segments are not too long. We will therefore adopt a mixed approach. First, the original time horizon is divided into segments in which action times are only defined by shift and break start and end times. Break start times are discretised and aligned to even five minute intervals to increase the chance of several action times to coincide. Only if a break time window does not admit a placement at an even five minute time (if e.g. start times between 10:02 and 10:04 are admitted), we let its (uneven) earliest start induce an action time.

In a second step, segments are subdivided if they do not exhibit uniform workforce demands. Each 15-minute subsegment defines a separate demand period if it contains a change in labour requirements. If the demand does not change over a period of more than 15 minute, this larger interval defines a segment. By this approach, we avoid large declines in approximation quality for shortage costs.

No.	shift types	shifts	shifts with breaks	different break durations	average break window width
B03	20	140	140	1	1.0
B05	11	77	0	0	–
C01	20	140	77	1	301.0
C13	5	35	35	1	277.0
C14	15	105	0	0	–
C16	8	56	0	0	–
C17	5	35	35	1	403.0
D09	8	56	56	1	391.0
D10	22	154	154	1	1.0
D11	18	126	126	1	1.0
D12	13	91	91	1	1.0
D15	20	140	21	3	301.0
D16	22	154	0	0	–
D25	20	68	68	1	295.3
D26	16	52	52	1	301.0
D27	20	140	0	0	–

Table 8.1.: Scenario data.

8.8. Experimental Results

We now evaluate the above model on real-world airport planning scenarios. A set of 16 test cases was selected which will also be used for the tests of the cyclic roster algorithm presented in the following chapter. Further criteria for choosing these scenarios will be given in Section 9.10. We first analyse the impact of the action time transformation on the number of demand periods. Afterwards, we compare the above break model which generalises the implicit formulation of Bechtold and Jacobs [1990] to the approach of Gaballa and Pearce [1979] and Aykin [1996] who used one set of break variables for each shift. For the tests, only main breaks have been considered since the supplementary relief breaks (usually of 15 minutes lengths) have turned out to have only minor impact (see also Schindler and Semmel [1993]). All scenarios cover a one week horizon, starting on Monday.

Basic properties of the test cases are summarised in Table 8.1. With exceptions in scenarios D25 and D26, the shift types of all scenarios can be realised on all days of the week. While on some scenarios, all shifts include a main break, five scenarios do not contain any breaks at all, and we will use these scenarios only for an evaluation of the temporal transformation. The break window widths indicate the average number of admissible break start times over all shifts. It can be seen that scenarios B03 and D10 through D12 only comprise fixed breaks while other scenarios incorporate substantial break placement flexibility with a largest break window of 496 minutes in scenario D25.

As described in Section 1.1, workloads on airports usually exhibit strong variations over the day and over the week. The different requirement profiles are described by Table 8.2. While average demands over the one week planning horizon are often moderate, peak demands raise up to 59 workers in scenarios C13 and D15. The standard deviation[1] reflects the demand fluctuations. As an example, Fig. 8.10 shows the demand curve of scenario C13. It can be seen that there are five peak times within each day with slightly lower demands on the weekend.

[1] $\sqrt{\sum_{t\in T}(d_t - \bar{d})^2}$ when $\bar{d}$ denotes the average workload.

No.	average demand	maximum demand	standard deviation
B03	6.4	20	4.7
B05	3.0	11	1.6
C01	9.5	48	9.4
C13	13.2	59	15.1
C14	3.0	15	2.2
C16	1.5	6	1.9
C17	8.9	43	8.1
D09	2.3	14	2.8
D10	8.4	25	6.5
D11	5.5	22	4.7
D12	5.7	10	3.1
D15	18.7	59	15.2
D16	16.3	36	6.6
D25	9.7	34	8.5
D26	1.8	9	2.0
D27	8.9	31	7.6

Table 8.2.: Demand curve characteristics.

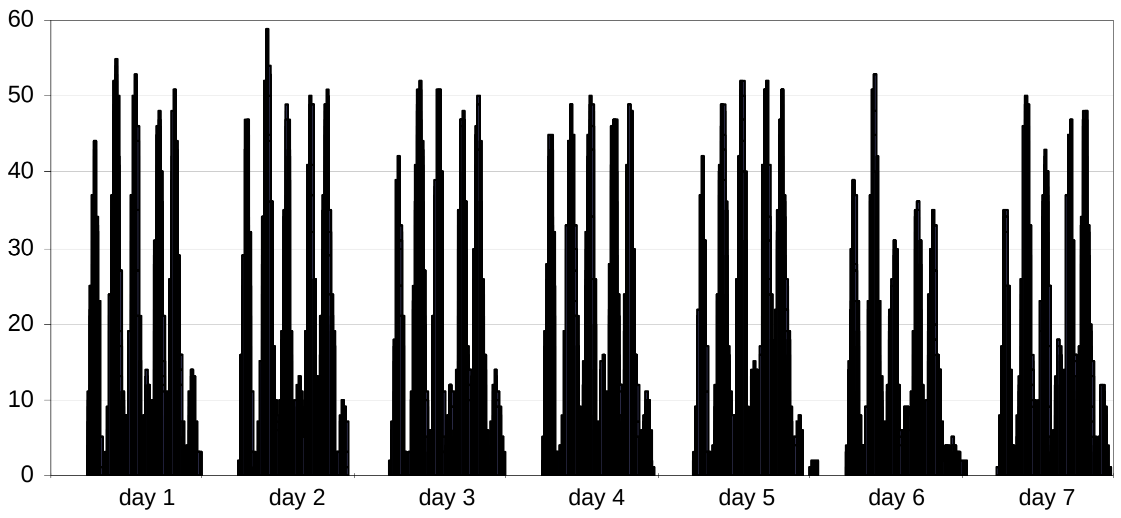


Figure 8.10.: Demand curve of scenario C13.

No.	after shifts and breaks	after histogram
B03	238	568
B05	119	656
C01	1274	1335
C13	1134	1267
C14	91	668
C16	70	198
C17	1302	1439
D09	1442	1448
D10	294	614
D11	252	575
D12	175	175
D15	182	515
D16	98	672
D25	624	809
D26	660	817
D27	133	529

Table 8.3.: Number of demand periods after action time transformation.

Results of the action time transformation are given in Table 8.3. The second column shows the number of demand periods after defining action times for shift and break start times. As explained above, breaks are defined in five minute steps if they can be flexibly placed. It can be seen that especially on scenarios with high break flexibility, the number of demand periods is already considerable. Scenario D09 exhibits the highest number of 1442 demand periods after this first step, corresponding to an average demand period length of 7 minutes.

For the results given in the third column, additional action times were defined in 15 minute steps if periods comprise demand changes. It can be seen that histogram action times add only few additional demand periods if the first step already entails a high resolution. However, on scenarios which do not comprise breaks (B05, C14, C16, D16 and D27), histogram action times are the most relevant factor for the number of demand periods. On average, the resulting demand periods have lengths of little more than 18 minutes. As a consequence, the adaptive action time transformation of Section 8.7 on average yields less demand periods than the most fine-grained models in the literature, using 15-minute demand intervals. At the same time, the transformed time basis is able to incorporate shifts which do not start and end in even 15-minute intervals and makes use of a finer-grained break resolution.

As we use only one break, the number of BreakPartitioning problems for the above problems is equal to the number of break durations given in Table 8.1. Results for the scenarios incorporating breaks are given in Table 8.4. The second column gives the sums of edges for the different break partitioning problems, i.e. the number of break windows exhibiting extraordinary overlap. Conforming with statements of Bechtold and Jacobs [1990] and Aykin [2000], extraordinary overlap is quite rare in these real-world test cases, and only scenarios D09 and D26 exhibit EO at all.

The third column gives the number of shift (and break) classes (layers) as a sum over the different BreakPartitioning runs. On scenarios without extraordinary overlap, this number is equal to the number of BreakPartitioning problems solved. On scenarios D09 and D26 in which EO is present, we only need two layers, meaning that extraordinary overlap does not entail

No.	edges	layers	break classes
B03	0	1	140
C01	0	1	14
C13	0	1	14
C17	0	1	35
D09	21	2	28
D10	0	1	154
D11	0	1	126
D12	0	1	91
D15	0	3	21
D25	0	1	46
D26	6	2	26

Table 8.4.: Results of BREAKPARTITIONING and additional splitup.

No.	without splitup			with splitup			breaks per shift type		
	variables	constraints	nonzeros	variables	constraints	nonzeros	variables	constraints	nonzeros
B03	140	279	19880	140	140	280	140	140	280
C01	1106	153	46771	1106	140	4732	2387	77	2464
C13	1022	69	19530	1022	56	2296	1379	35	1414
C17	1085	69	20704	1085	35	1120	1085	35	1120
D09	2282	103	34249	2282	77	3241	3976	56	4032
D10	154	307	24024	154	154	308	154	154	308
D11	126	251	16128	126	126	252	126	126	252
D12	91	181	8462	91	91	182	91	91	182
D15	21	39	186	21	21	42	21	21	42
D25	475	91	13143	475	46	543	1004	68	1072
D26	518	50	6215	518	26	570	1480	52	1532

Table 8.5.: Sizes of LP representations for different break formulations.

too many variable replications.

The last column of Table 8.4 indicates the number of break classes after splitup by associated shifts and breaks. It can be seen that the increase upon the number of classes after BREAKPARTITIONING is considerable. This means that the formulation after splitup uses many more systems of break variables and constraints. On scenarios admitting only one break start time per shift, each shift class is only made up of one shift.

We now compare the extended variant of the Bechtold/Jacobs break model with the approach of Gaballa and Pearce [1979] and Aykin [1996] (Table 8.5). The first two sets of columns in Table 8.5 indicate key figures of the Bechtold/Jacobs approach. For the second set of columns, shifts and breaks were split up as described before. The figures relate to the above action time transformation, i.e. variables for flexible breaks are created with a five minute discretisation.

While the number of variables with and without splitup is equal, the number of constraints is reduced by an average of 40.4%. Especially if shifts comprise only one break, the model without splitup uses sets of forward *and* backward constraints in an order of the number of shifts. In contrast, the advanced model uses only one equality constraint. In such cases, the break formulation is trivial if shifts allow for only one break start time. However, when looking for a generic approach for a wide range of realistic scenarios, the superiority of the splitup approach is important.

The advantage of the splitup idea becomes especially clear from the number of nonzero coefficients in the constraint matrix. In both cases, the constraint substitution of Bechtold and Jacobs

No.	extended Bechtold/Jacobs model	Gaballa/Pearce/Aykin model
C01	41.63	31.17
C13	2.42	2.63
C17	1.83	1.94
D09	4.22	5.80
D26	11.34	12.14
D27	2.36	2.97

Table 8.6.: Runtimes for LP relaxation of tour scheduling model.

[1990] for reducing the number of nonzeros has been applied (see Section 8.4). The splitup procedure effectively reduces the number of nonzeros by an average of 92.8%.

Clearly, using one set of break variables for each shift type (third set of columns in Table 8.5) results in more break variables than in the aforementioned model; on average, 47.5% more variables are used. On scenarios containing only fixed breaks, the number of variables is equal. With regard to the number of constraints, none of the models offers a clear advantage. While the model of Gaballa and Pearce [1979] and Aykin [1996] uses exactly one equality constraint for each shift which includes a break, the above model (after splitup) sometimes uses more constraints while on other scenarios, sharing break variables between shifts results in a lower number of constraints. Note that even if the numbers of constraints (or even the number of nonzeros) are equal, the types of constraints can be different.

With regard to the number of nonzeros, the extended Bechtold/Jacobs model (with splitup) seems to be slightly superior. Comparing the approaches on the seven scenarios allowing for flexible breaks, the model with one set of break variables per shift necessitates an average surplus of 29.2% nonzeros. However, none of the formulations strictly dominates the other: While on scenario D26, the latter model uses 168.8% more nonzeros, it entails a reduction of 47.9% in the number of nonzero matrix elements on scenario C01. However, the assumption of Aykin [2000] that his model is generally superior with regard to the number of nonzeros is not valid if the above advanced model is applied to typical airport planning scenarios.

On the scenarios on which the two models yield different constraint systems (six scenarios), we have additionally evaluated LP solution times. Running times for model (8.15)-(8.21) have been generally too low to yield expressive results. As the above model is meant as a first step towards a cyclic roster algorithm, the solution times for the LP relaxation of the cyclic roster algorithm of Chapter 9 were taken for comparison. The exact test conditions and roster constraints will be given in Section 9.10. The algorithm has been implemented in Visual C++ 7.1, using BCP of COIN-OR and XPRESS-MP Release 2004 of Dash Optimization as LP solver [Dash, 2004]. The experiments were carried out on an AMD Athlon 3000+ computer with 2.16 GHz, 1 GB main memory and operating system Windows XP SP1.

Computation times are given in Table 8.6 (in seconds). It can be seen that on five of the six scenarios, the proposed break model yields better runtimes than the model using one set of variables and constraints per shift. On average, the Gaballa/Pearce/Aykin model consumes 5.7% more runtime. Only on scenario C01, the latter model yields a better result. In our experiments, we could observe that the performance of the different models also depends on the roster parameters (see also Section 9.10). In practice, the extended Bechtold/Jacobs model nearly always outperformed the Gaballa/Pearce/Aykin model, making it the method of choice as a generic approach for modelling break placement flexibility.

8.9. Conclusions and Future Research

In this chapter, we have introduced implicit models for the handling of flexible breaks in a weekly shift scheduling problem. We have shown that the attribution of breaks to shifts can be represented as a transportation problem. While the resulting formulation already incorporated generalisations of the original shift scheduling model of Bechtold and Jacobs [1990], we have pointed out that an elementary assumption named *extraordinary overlap* has only rudimentally been tackled up to now.

We have developed a polynomial-time algorithm which separates the sets of shifts and breaks such that each resulting class does not incorporate extraordinary overlap. It has been shown that by an additional splitup idea, the number of constraints and nonzero matrix elements can be significantly reduced. Using these considerations, the weekly shift scheduling formulation has been generalised to the case of multiple breaks with different durations. We have shown that by taking shift and break start and end times into account, the time scale can be transformed in an intelligent way. This transformation yields a reduction in the number of demand periods without sacrificing modelling detail at times of high scheduling flexibility, e.g. due to breaks.

By empirical analysis on real-world planning scenarios, we have demonstrated the efficacy of the action time transformation. In an evaluation of the break model, we have shown that splitting up shifts and breaks by associated sets is crucial in developing efficient representations. The extended break formulation has been shown to be superior to an alternative formulation not only in terms of the number of variables, but also with regard to the sparsity of the constraint matrix. Computationally, the new procedure mostly performs better than the alternative approach. While the test cases incorporated either no break, a fixed break or rather large break windows, the scenarios are representative of airport staff planning problems. On such scenarios, the extended Bechtold/Jacobs model is not only theoretically superior by sharing break variables, but also seems to be appropriate as a generic approach in real-world ground staff planning.

The weekly shift scheduling model could be extended by further constraints, e.g. by the shift restrictions of Section 5.2.6. While in practice, the model often naturally yields integer solutions, we have not described a systematic approach for arriving at integer solutions. Instead of enhancing the isolated shift scheduling formulation, we will use the model as a basis for an integrated cyclic rostering algorithm in the following chapter.

Acknowledgements

I would like to thank Torsten Fahle for the discussions and ideas on the efficiency of the BREAK-PARITITIONING algorithm.

9. Cyclic Roster Generation by Branch-and-Price

Tower: Lufthansa 893, number one,
check for workers on the taxiway.
Pilot: Roger... (after a short while)
We've checked the workers,
they are all working.
— from a radio message

In the previous chapter, we have introduced a basic model for demand-level scheduling including flexible breaks. Even if we have considered a cyclic weekly scheduling horizon, we have not explicitly built roster lines for the individual employees. The setting was therefore a *shift scheduling* problem which additionally considers shift overlapping between the days. In contrast to shift scheduling, *rostering* takes constraints on valid sequences of shifts into account (see Chapter 2). In the following, we will build upon the formulation of Chapter 8 and develop a novel model and algorithm for integrated cyclic rostering.

9.1. Introduction

Cyclic rosters (equivalently, *shift patterns* or *rotating schedules*) represent sequences of shifts designed for a group of employees. One worker starts on each week on the roster, switching cyclically from one week to the next. The basic principle is most easily described by an example. In the roster of Fig. 9.1, two shift types A and B are used; "–" signifies a day off. Four employees work on this schedule. After finishing one week, each worker switches to the subsequent week while worker four turns to the first week. All workers therefore rotate over the pattern for a given period of time, e.g. a flight season.

The lower part of Fig. 9.1 shows the shift coverage, i.e. the number of shifts of types A and B for each day of the week. Clearly, the coverage is constant for the whole season for which the roster is unrolled. Naturally, all staff working on a roster must have equal contract types with regard to weekly work hours and identical skills. Cyclic rosters are frequently used to schedule

	Mon	Tue	Wed	Thu	Fri	Sat	Sun
1	A	A	A	A	–	–	A
2	A	A	B	B	B	–	–
3	B	B	–	A	A	A	–
4	B	B	B	B	B	–	–
A	2	2	1	2	1	1	1
B	2	2	2	2	2	0	0
Σ	4	4	3	4	3	1	1

Figure 9.1.: Cyclic roster example.

full-time staff at airlines and airports, but also in call centres, hospitals, emergency services and in public transport, see e.g. Bennett and Potts [1968], Khoong et al. [1994] and Çezik et al. [2001]. Many of these organisations are facing continuous labour requirements on seven days a week, 24 hours a day.

On the one hand, cyclic rosters are quite rigid and inable to adapt to changes in labour requirements. Restricted availabilities of employees, e.g. due to trainings, illnesses or vacation, cannot be acounted for. However, they offer a maximum degree of fairness in the distribution of shifts [Millar and Kiragu, 1998]. Furthermore, workers can foresee their duties for a whole planning season in advance. Airports are frequently obliged to use cyclic rosters due to union regulations or company policies. Short-term demand peaks or staff absences are usually compensated by shift swaps or part-time staff.

For planners, cyclic roster generation is a very demanding task. Two classes of constraints have to be considered [Ernst et al., 2004]:

- Demand constraints: The roster should cover the workforce demands given per time period or per shift type.
- Union and legal regulations: These typically specify minimum and maximum consecutive workdays, the number of weekends off, if single days off can be given, etc.

The objective in roster construction can be the size of the workforce, labour costs, measures relating to roster quality (like weekends off, consecutive workdays, etc.) or combinations of these. Due to the large scale of airport rostering problems and the multitude of restrictions, manual roster creation is burdensome and error-prone. Glover and McMillan [1986] mention an example of planners taking between 8 to 15 hours to create a schedule for 70 to 100 employees. On airports, it is frequent practice to reuse rosters from one season to the next with only slight modifications for changes in labour requirements.

As in the preceding chapter, we will assume requirements to be given by a demand curve. As cyclic rosters are not able to adapt to fluctuating demands, requirements should be based on a model week with typical workloads [Gaballa and Pearce, 1979]. Clearly, demands must be cyclic. For several reasons, demand-level scheduling is particularly appropriate for cyclic roster generation. On the one hand, rotating schedules are created for a whole planning season in advance, and actual task-level workloads are generally not exactly known. On the other hand, all workers must cover the qualification requirements of all work task, i.e. workloads naturally have to be homogeneous.

The creation of cyclic rosters involves different linked aspects, including the determination of appropriate daily shifts, the sequencing of shifts and days off and the cyclic closure. As shown in Chapter 2, many authors have taken sequential approaches, first solving shift scheduling problems for each day and subsequently assigning days off and shifts, see e.g. Bennett and Potts [1968], Emmons [1985] and Lau [1994]. While nowadays, non-cyclic rostering problems are usually solved in an integrated way (e.g. Jarrah et al. [1994], Brusco and Jacobs [2000], Rekik et al. [2003]), there has not been any publication on integrated cyclic tour scheduling to the knowledge of the author. In fact, all references for cyclic rosters build upon demand figures per shift type, see e.g. Balakrishnan and Wong [1990], Mason [1999], Muslija et al. [2000] and Felici and Gentile [2004].

Cyclic roster approaches have regularly taken special assumptions. While some models (e.g. Laporte et al. [1980]) only allow for equal shifts on subsequent workdays, other approaches provide no control over shift successions [Çezik et al., 2001]. Models frequently allow for only one shift type [Rosenbloom and Goertzen, 1987] or one shift length, meaning that constraints on weekly working hours are implicit in the number of shifts, see e.g. Mason [1999]. Most publications do not incorporate meal and rest breaks and do not represent weekend-off constraints

[Balakrishnan and Wong, 1990]. Other approaches are very heuristic in nature (e.g. Khoong and Lau [1992], Khoong et al. [1994]) or apply only to special restrictions (e.g. Felici and Gentile [2004]).

In this work, we propose a more generic model and algorithm for cyclic roster generation which applies to a large variety of scenarios. Mason and Nielsen [1999] note that it is very difficult to "develop a single truly generic package that can be used for all rostering problems". As this is also true for planning practice on airports, we have tried to define a setting which covers most typical restrictions. The resulting model is still more general and covers more constraints than any approach before. The formulation will be integrated in determining daily shifts and shift and day-off positions simultaneously. It will apply to continuous as well as discontinuous operations and will incorporate one or multiple flexible breaks per shift. Furthermore, it will allow for operative planning with given workforce sizes as well as what-if analyses with flexible staff sizes.

Especially in recent years, most solution approaches for cyclic and non-cyclic staff scheduling problems have built upon linear or integer programming [Chew, 1991] [Ernst et al., 2004]. We will devise a branch-and-price approach for cyclic rostering which builds upon the generalised set covering formulation of the preceding chapter. The solution method will based upon previous propositions but use a new representation of rosters in order to overcome limitations of alternative approaches. We will furthermore give some annotations on the use of implicit models in roster generation.

The chapter will be structured as follows: In the next section, we describe the constraints for our cyclic roster approach. Different modelling approaches will be reviewed in Section 9.3 while Section 9.4 describes how a week-based representation of cyclic rosters can be extended for our complex setting. The full model is presented in Section 9.5, using column generation to implicit represent variables (Section 9.6). In Sections 9.7 and 9.8, we will show how connected integer solutions are obtained by suitable branching strategies. Section 9.9 will describe some generalisations of the model. Finally, we will give experimental results (Section 9.10) and a summary (Section 9.11).

9.2. Problem Description

The cyclic roster problem consists in determining a shift pattern of several weeks subject to restrictions on the validity of shift and day-off sequences, covering given workforce requirements as well as possible. Two different planning scenarios are imaginable. In most cases, the planner will create an operational roster for the next season, implying that the staff size is fixed while and understaffing may be acceptable. In other cases, the planner may be interested in a what-if analysis of minimum workforce sizes to cover given demands. A possible scenario is the estimation of staff costs due to the acquisition of additional clients. To provide maximum flexibility, we will assume minimum and maximum limits w_{min} and w_{max} for the number of roster weeks to be given. Fixed staff size planning then corresponds to the case $w_{min} = w_{max}$.

As in the previous chapter, shift types will be denoted by K. Each shift type can be realised on a set $N_k \subseteq N$ of days of the one-week planning horizon $N := \{1, \ldots, 7\}$. Shift type realisations (k, n) ($k \in K, n \in N_k$) will also be referred to as *shifts* and entail costs of c_k. Shifts are attributed to the day on which they start.

While we allow for different shift types on consecutive days, bad shift transitions should generally be avoided. As an example, a night shift should not be followed by an early shift. Such transitions can be avoided by prescribing minimum rest times between shifts. Additionally, shifts on consecutive days are often constrained to start at the same time or to follow a "forward creeping" pattern, meaning that start times on subsequent days are nondecreasing. We will therefore restrict shift starting times to a given interval around the start time on the preceding day. Minimum rest times and start-time offsets can be aggregated by defining $G_k \subseteq K$ to be the set of valid predecessor shift types for $k \in K$. Inconvenient transitions from a shift type k_1 to k_2 will be penalised by a nonnegative value τ_{k_1,k_2}. We will assume that $k \in G_k$ for each $k \in K$. τ_{k_1,k_2} will be calculated as a linear measure of the start time difference between k_1 and k_2, i.e. $\tau_{k,k} = 0\,\forall k \in K$.

A similar constraint refers to shift successions over days off. In general, a sequence of night shift, day off and early shift is not admissible since the time off would be too short. A day off is generally defined as an off-duty period of at least $24 + h$ hours where h is usually between 0 and 12 hours. More generally, a day-off stretch of κ days must ensure a period of $24\kappa + h$ hours off duty. We will aggregate this information, defining $H_k \subseteq K$ to be the set of valid day off predecessors of shift type $k \in K$. No penalties will be imposed for day-off transitions.

Consecutive days on build so-called *workstretches* which will be restricted to minimum and maximums numbers of m^{on}_{min} and m^{on}_{max} shifts, respectively. Analoguously, day-off stretches will be limited to a minimum and maximum of m^{off}_{min} and m^{off}_{max} days off, respectively. Single days off can e.g. be avoided by setting $m^{off}_{min} > 1$. We will assume that $m^{on}_{min} \leq 7$ and $m^{off}_{max} \leq 6$ which is sufficient in all realistic cases.

In practice, shift types usually have lengths which are multiples of e.g. 30 or 60 minutes. To provide control over weekly working hours, we assume that a shift type $k \in K$ covers a number of $u_k \in \mathbb{N}$ *work units* each of which e.g. corresponds to 30-minute time period. We will restrict each roster week to comprise between u_{min} and u_{max} work units. Since labour contracts prescribe weekly working hours, we will ensure that a roster is made up of an average number of u^{avg} weekly work units up to a given tolerance of u_{tol} units.

The user can furthermore specify an A-out-of-B weekends-off constraint. Since blocked weekends are virtually always undesirable, this will automatically imply an even spreading of weekends off.

In accordance with most workforce scheduling publications (see e.g. Koop [1988], Thompson [1993], Ernst et al. [2004]), we will follow a multiobjective approach. Our primary goal will be the minimisation of labour shortages. Among solutions entailing equal shortages, we will minimise shift costs. We will therefore define shortage penalties which exceed average shift type costs per time period. Note that when the number of roster weeks is not fixed, less roster weeks entail lower shift costs if shift costs are reasonably defined. Additionally, shift costs provide means for avoiding e.g. night shifts. Inconvenience costs for shift type transitions are regarded a subordinate objective. We will therefore assume $\max_{k_1,k_2} \tau_{k_1,k_2} \ll \min_k c_k$.

The aforementioned cyclic rostering problem is $\mathcal{NP}$-hard which follows from the complexity results of Bartholdi III [1981] for cyclic staffing problems with intermittently available resources. Furthermore, the changing shift assignment problem analysed by Lau [1994] and the continuous shift assignment problem of van den Berg and Panton [1994] build $\mathcal{NP}$-hard subclasses of the above setting.

9.3. Modelling Approaches

In the following, we will review different column generation models for the cyclic roster problem. The formulations will be analysed with regard to their suitability to cover the above restrictions as well as constraints which may come up in the future. For ease of exposition, we will assume demand figures to be given per shift. The cyclic roster formulation will later be based on the set covering formulation from the preceding chapter, meaning that labour requirements will be given per time period. A set of equality constraints will be used to couple shift and roster models.

In integer programming formulations of the cyclic roster problem, columns (variables) represent subsequences (building blocks) of shifts. In the different formulations, these building blocks represent either workstretches, weeks or single days. Different constraints are imposed in the lin-

		X_1	X_2	X_3	X_4	X_5		
Mon	A	1	1				$\geq$	2
	B			1		1	$\geq$	2
Tue	A	1	1				$\geq$	2
	B			1		1	$\geq$	2
Wed	A	1					$\geq$	1
	B		1			1	$\geq$	2
Thu	A	1			1		$\geq$	2
	B		1			1	$\geq$	2
Fri	A				1		$\geq$	1
	B		1			1	$\geq$	2
Sat	A				1		$\geq$	1
	B						$\geq$	0
Sun	A		1				$\geq$	1
	B						$\geq$	0
Mon		1	1	1		1	$=$	4
Tue		1	1	1		1	$=$	4
Wed		1	1	1		1	$=$	4
Thu		1	1		1	1	$=$	4
Fri		1	1		1	1	$=$	4
Sat		1	1		1	1	$=$	4
Sun			2		1	1	$=$	4

Figure 9.2.: Workstretch encoding.

ear program in order to match these basic blocks and to form a valid roster. For illustration, we will show how the roster of Fig. 9.1 is represented in the different models. Two shift types A and B with demand figures as given in the lower part of the table will be used. The shifts are planned in a cyclic roster of four weeks length with every other weekend off.

Laporte et al. [1980], Chew [1991], Lau [1996b] and Mason [1999] have proposed to divide rosters into *workstretches*[1]. For the purpose of this model, workstretches are defined as sequences of consecutive shifts along with the following block of days off. Constraint coefficients represent the shifts and days off which are covered by the workstretch, and integer variables denote the number of chosen workstretches of different types. Cost coefficients typically represent labour costs or penalties for inconvenient workstretches. The integer program therefore chooses a cost-minimal set of compatible workstretches to cover workloads.

Fig. 9.2 shows a restricted integer program for the workstretch scheme. As an example, the first column in program represents a workstretch starting on Monday, consisting of four shifts of type A and two subsequent days off. The upper part of Fig. 9.2 represents shift coverage constraints. Additionally, we must ensure that each day of the roster is covered w times if w is a given number of roster weeks [Mason, 1999]. This is accomplished by the constraint system in the lower part. Note that workstretches contribute to the day equalities with days on and days off. If there are no restrictions with regard to shifts surrounding days off, rosters naturally decompose by workstretches, see e.g. Koop [1988].

Alternatively, variables and columns of the integer program may represent roster weeks, see Fig. 9.3. While shift transitions within the week are incorporated in the given week patterns,

[1]Workstretches have also been denoted as *basic patterns* [Chew, 1991], *stints* [Millar and Kiragu, 1998] or *work blocks* [Muslija et al., 2000].

		X_1	X_2	X_3	X_4		
Mon	A	1	1			$\geq$	2
	B			1	1	$\geq$	2
Tue	A	1	1			$\geq$	2
	B			1	1	$\geq$	2
Wed	A	1				$\geq$	1
	B		1		1	$\geq$	2
Thu	A	1		1		$\geq$	2
	B		1		1	$\geq$	2
Fri	A			1		$\geq$	1
	B		1		1	$\geq$	2
Sat	A			1		$\geq$	1
	B					$\geq$	0
Sun	A	1				$\geq$	1
	B					$\geq$	0
weeks		1	1	1	1	$=$	4

Figure 9.3.: Week encoding.

special attention must be given to the validity of transitions between the weeks. Rosenbloom and Goertzen [1987] introduce special transition variables to ensure validity with regard to minimum and maximum days on and off at week boundaries.

We could also use smaller building blocks. As an example, Çezik et al. [2001] and Çezik and Günlük [2002] solve seven daily shift scheduling models and combine these into a tour scheduling formulation. Feasibility with regard to start time differences on consecutive days is ensured by explicit link variables or, projecting out the link variables, by a constraint system similar to the implicit break constraints described in Section 8.3.

Larger building blocks are conceivable as well. When we have to grant A out of B weekends off, it can be advantageous to build units of B weeks within the column generator. This idea relates to Emmons [1985] and Burns and Koop [1987] who propose to build rosters by combination of blocks of B weeks. However, this approach entails the largest potential number of variables. Furthermore, the optimal or given roster length may not always be a multiple of B, or there may be no weekend-off constraint at all. For these reasons, we will not further pursue this approach.

While it may seem natural to solve the above models by column generation, specific settings may allow for a complete enumeration of building blocks, see e.g. Laporte et al. [1980] and Rosenbloom and Goertzen [1987]. In more general settings however, complete enumeration becomes prohibitive. In the following, we will assume that problems are decomposed when workstretch or week encoding are used, adding columns dynamically via a column generator subproblem (cf. e.g. Easton and Rossin [1991], Gamache and Soumis [1998] and Mason and Smith [1998]).

Clearly, the workstretch representation easily incorporates minimum and maximum bounds on the number of workdays and days off. Furthermore, minimum rest times between shifts are easily handled within the column generator. Minimum temporal distances between shifts over days off are difficult to take into account and would require special constraint systems like in Çezik and Günlük [2002]. While average working hours can be represented by an additional constraint in the master program (and a larger search space in the subproblem), minimum and maximum weekly working hours cannot be taken into account.

In contrast, this task is easy if weeks are taken as building blocks. But while observing bounds on consecutive days on and off is easy within weeks, special care has to be given to transitions

	X_1	X_2	X_3	X_4		
Mon	1		1	1	$\geq$	3
Tue	1		1	1	$\geq$	3
Wed	1	1			$\geq$	2
Thu	1	1	1		$\geq$	3
Fri	1	1	1		$\geq$	3
Sat	1		1	1	$\geq$	3
Sun			1	1	$\geq$	2
Mon	2		1	1	$=$	4
Tue	2		1	1	$=$	4
Wed	1	1	1	1	$=$	4
Thu	1	1	2		$=$	4
Fri	1	1	2		$=$	4
Sat	1	1	1	1	$=$	4
Sun	1	1	1	1	$=$	4

Figure 9.4.: Roster containing disconnected subcycles.

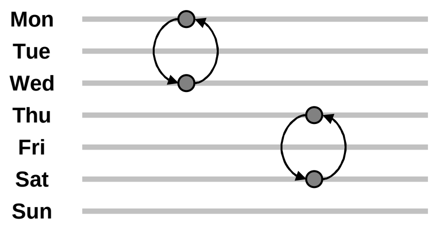


Figure 9.5.: Disconnected subcycles.

between the weeks. The same is true for minimum rest times between shifts and over days off at the week boundary, creating a feasibility problem for week transitions in two dimensions.

Çezik et al. [2001] only show how weeks of five workdays can be extracted from an implicit model combining daily set covering formulations. In contrast, bounds on workdays and days off cannot be easily incorporated. Minimum times between shifts and over days off could be respected via an implicit constraint system. While average working hours could be covered via an additional constraint, there is no easy way to guarantee adherence to minimum and maximum working hours per week.

In week and workstretch formulations, minimum numbers of weekends off over the complete roster can be easily respected via an additional constraint. Nevertheless, weekends off generally have to be equally distributed which is not easy to represent.

Another problem in all formulations arises from disconnected subcycles [Laporte et al., 1980] [Mason, 1999]. Figure 9.4 gives an example for a workstretch formulation. For ease of exposition, only one shift type is used. The roster comprises four workstretches *Mon→Tue, Wed→Sun, Thu→Fri, Sat→Wed*. As can be easily verified, all constraints are observed, but the workstretches cannot be composed into a single rotation. Figure 9.5 shows workstretches as arcs, leading from their first day to the day after their last day; weekday nodes are shown as lines. From this graph, it becomes clear that *Mon→Tue*/*Wed→Sun* and *Thu→Fri*/*Sat→Wed* form disconnected subcycles.

The problem of disconnected subcycles is closely related to the subtour problem in IP representations of vehicle routing and travelling salesman problems, see e.g. Dantzig et al. [1954], Fisher

	single shifts	workstretches	weeks
Consecutive days on/days off	difficult	easy	easy within weeks, difficult between weeks
Minimum rest times between shifts	difficult	easy	easy within week, difficult between weeks
Minimum rest time over day off	difficult	difficult	easy within week, difficult between weeks
Minimum and maximum working hours per week	difficult	difficult	easy
Average working hours per week	easy	easy	easy
Distance between weekends off	difficult	difficult	difficult
Disconnected subcycles	may exist	may exist	may exist

Table 9.1.: Comparison of different cyclic roster formulations.

and Jaikumar [1981] and Lawler et al. [1985]. Disconnected subcycles may arise in all of the above models, but are less likely to occur in week and single-day formulations. Laporte et al. [1980] and Mason [1999] show how to eliminate disconnected subcycles by means of cuts and branching, respectively. Other authors build several cyclic rosters from the subcycles (e.g. Burns and Koop [1987]) or implicitly tolerate their existence (Rosenbloom and Goertzen [1987], Çezik et al. [2001]). We will assume that subcycles are undesired since attributing employees to separate rosters sacrifices fairness of shift attributions.

Table 9.1 summarises advantages and disadvantages of workstretch, week and single-shift encoding. Obviously, the Çezik et al. [2001] model of combining daily set covering formulations into a tour scheduling formulation is not appropriate for the setting described above. The main flaw of the workstretch model is its difficulty to represent minimum and maximum working hours per week. If the above setting is enriched by further constraints in the future, these are likely to refer to weeks; Rosenbloom and Goertzen [1987] even state that “virtually all labour restrictions concern weekly schedules”.

We will therefore adopt the week formulation and show how all aforementioned constraints can be represented. We will assume that a work week starts on Monday and ends on Sunday. While the column generation approach of Chapter 7 was developed from a compact formulation, the compact model for the above problem would amount to defining one variable per roster position. However, weeks in cyclic rosters are isomorphic, and attributing variables to specific weeks entails symmetry problems [Felici and Gentile, 2004]. In contrast, the column generation formulation does not attribute weeks to specific positions and therefore naturally avoids symmetry. We will thus only make reference to the compact formulation in integer branching (Section 9.7).

9.4. Encoding Week Transitions

In the preceding section, we have seen that many roster constraints are easy to obey in the week formulation. However, it becomes more difficult to ensure validity between roster weeks. We therefore have to ensure that the integer program chooses weeks such that the shift sequence

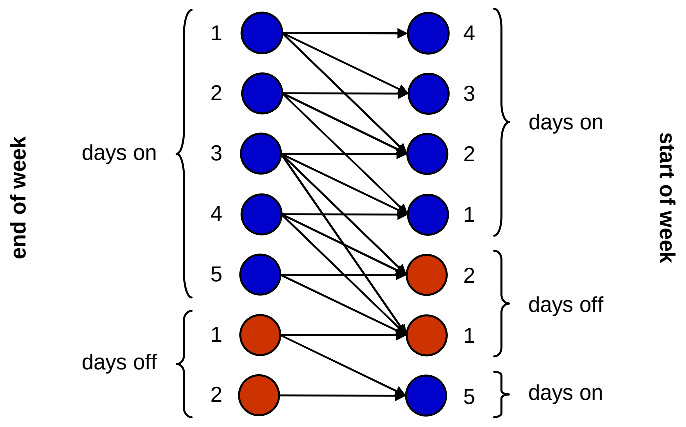


Figure 9.6.: Transportation problem for numbers of days on/off.

at the end of one week matches at least one other week start, i.e. weeks can be linked without violating constraints. Before describing the generation of weeks in more detail, we will analyse the restrictions governing validity of week transitions. For the moment being, we will neglect weekend-off constraints.

First, we must observe minimum and maximum limits on the number of consecutive days on and off. As an example, let $[m^{on}_{min}, m^{on}_{max}] = [3,5]$ and $[m^{off}_{min}, m^{off}_{max}] = [1,2]$. This means that if e.g. a week ends on a single day on, there must be another week starting with at least two and at most four days on. If a week ends on e.g. three days on, it can be linked to either a week starting on a maximum of two days on or to a week commencing on one or two days off. Analoguously, only weeks obeying certain starting conditions can follow a week ending on days off. Sequences of days on and off at week boundaries represent feasibility conditions which must be suitably encoded.

Figure 9.6 shows the basic situation. Nodes on the left-hand side represent week endings on different numbers of days on (blue) and off (red). Nodes for week starts are shown on the right-hand side. Valid week transitions are represented by edges. The attribution of matching numbers of days on and off can be interpreted as a transportation problem, matching supplies of week endings to demands of week starts. It is easy to observe that the problem of ensuring matching numbers of weeks is equivalent to the shift-break attribution problem of Section 8.3. By construction, it is obvious that the network does not exhibit extraordinary overlap (EO) as defined in Section 8.4.

However, there are additional restrictions in our approach. Shift types on consecutive days on must be chosen such that start times only differ by a limited time. The basic situation is illustrated in Fig. 9.7. Nodes on the left-hand side correspond to start times of Sunday shifts while right-hand side nodes represent start times of Monday shifts. In the example, start times are allowed to deviate by at most one hour. Again, matching conditions could be encoded by implicit constraint. In this way, Çezik et al. [2001] couple seven daily shift scheduling formulation to a tour scheduling model. Using this idea only at week boundaries would be a means of ensuring start time validity.

While the above example can be seen to obey the extraordinary overlap condition, this is generally not true if additional minimum rest times must be observed between shifts. Analoguously, limiting shift transitions over days off can entail extraordinary overlap. We therefore cannot use the linear-size constraint system of Bechtold and Jacobs [1990]. Instead, we would have to resort to a quadratic number of constraints (see Çezik and Günlük [2002]) or use the separation logic of Section 8.5. Furthermore, shift transitions not only imply feasibility conditions, but can also entail inconvenience costs. However, as opposed to shift successions within weeks, transitions at

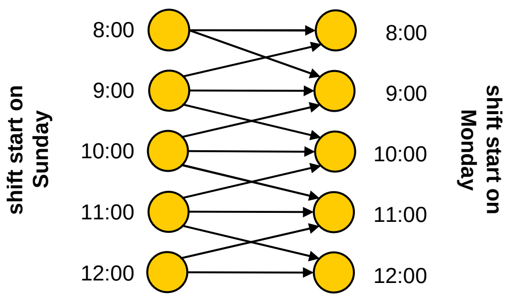


Figure 9.7.: Transportation problem for shift start times on successive days.

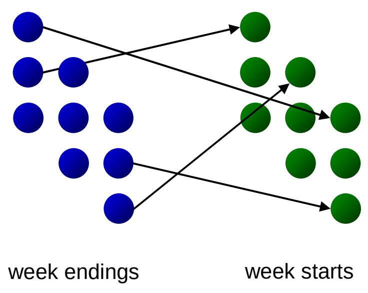


Figure 9.8.: Two-dimensional transportation problem for week transitions.

week boundaries may not be too critical, and it may be acceptable to neglect transition penalties between weeks.

To ensure matching numbers of weeks, we must obey both limits on the numbers of days on and off *and* restrict shift transitions, creating a matching problem in two dimensions (Fig. 9.8). In analogy to Fig. 8.6, Figure 9.9 visualises the sets of variables which are covered by forward and backward constraints in two dimensions (blue boxes).

In Section 8.4, we have sketched the proof of Bechtold and Jacobs [1996] for the sufficiency of forward and backward constraints to ensure assignment feasibility. The basic idea of the proof consists in combining forward and backward constraints whose complementary variable sets are disjoint if extraordinary overlap is absent. However, we see that in Fig. 9.9, the variables represented by the lower left and the upper right (yellow) boxes are present in the complementary sets of both forward and backward constraint. This means that even if there is no extraordinary overlap, complementary variable sets will not be disjoint in two dimensions. The proof of Bechtold and Jacobs [1996] therefore does not apply, and forward and backward constraints will generally not be sufficient to encode matching conditions between weeks.

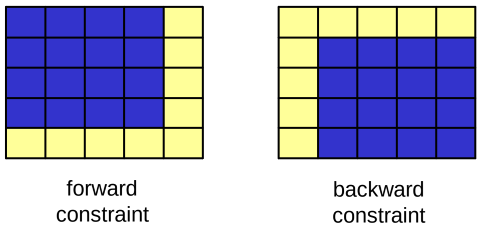


Figure 9.9.: Forward and backward constraints in two dimensions.

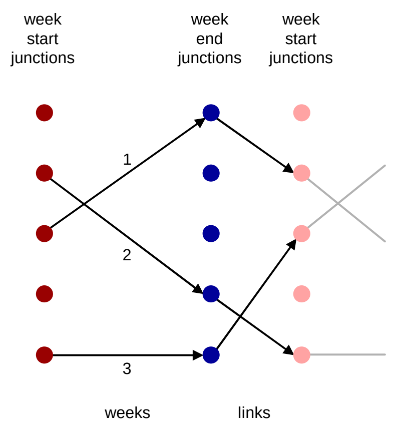


Figure 9.10.: Week link encoding.

Using a quadratic number of constraints in each dimension (according to Çezik and Günlük [2002]) would be a remedy. The resulting number of constraints could be implicitly dealt with, generating transition cuts dynamically; Çezik and Günlük [2002] describe how cuts can be separated by determining minimum flow capacities. However, preliminary experiments have shown that even if only non-dominated cuts were generated, the overall performance of the approach was poor.

Dealing explicitly with succession feasibility between weeks amounts to introducing additional variables representing the numbers of week transitions of each type. Each such *link variable* is related to given numbers of days on or off and shift types at week starts and endings. We will refer to the characteristics of week starts or endings as *junctions*. A junction can be represented by a pair $(k, m) \in K \times \mathbb{N}$ and an additional indication of the junction type (day-on/day-off). As a example, let (k, m) be a week start junction. If it is a day-on junction, k refers to the shift on Monday and m to the number of days on at the week start. If (k, m) represents a week start off-day junction, k denotes the first shift in the week and m the number of days off at the week start.

The validity of week transitions can be checked using only information on the junctions. If e.g. (k_1, m_1) is a week end day-on junction and (k_2, m_2) a week start day-on junction, the two corresponding weeks follow each other if $k_1 \in G_{k_1}$ and $m_1 + m_2 \in [m_{min}^{on}, m_{max}^{on}]$. Weeks can then be interpreted as connecting week start and end junctions while links build bridges between week end and start junctions. Shift type transition penalties τ_{k_1,k_2} can be accounted for in the costs of links.

The basic idea is sketched in Fig. 9.10. Red nodes represent week start (equivalently, link end) junctions, blue nodes week end (equivalently, link start) junctions; week start junctions are replicated on the right side. The order of weeks in the example is given on the week edges.

Link variables enable us to explicitly represent feasible week transitions in a linear program by enforcing the numbers of week start junctions to match the number of target junctions of links and week endings to equal the numbers of links starting on the respective junctions. Rosenbloom and Goertzen [1987] use similar transition variables in a simple setting admitting only few feasible weeks and links encoding feasibility with regard to bounds on days on and off. In our setting, we will not only generate weeks dynamically, but also defer link variable generation to a column generator. This allows for dealing implicitly with link variables whose potential number is biquadratic in the number of days on/off and shift types.

Because we assumed $m_{max}^{off} \leq 6$, each week will comprise at least one shift. Consequently, validity of shift transitions can always be ensured at the interface between subsequent week. Since furthermore $m^{on}_{max} \leq 7$, a workstretch cannot span over more than one week. Consequently, a week with seven days on must be preceded and followed by days off, and encoding feasibility with regard to the length of workstretches does not involve more than two adjacent weeks.

It is interesting to note that the implicit formulation described above can be obtained from an explicit model by projecting out link variables, see Çezik and Günlük [2002]. Clearly, the resulting model is only equivalent if all link variables have costs of 0 which is not the case in the above model. Theoretically, it is also possible to use half-implicit approaches, e.g. by using a separate implicit constraint system to encode shift type transitions for each combination of days on/off at week starts/ending. This potentially results in a quadratic number of linear-complexity constraint systems. It remains to be investigated if mixed-implicit approaches are competitive with pure implicit or explicit approaches.

Regardless of the model used, we may face the subcycle problem described in Section 9.3. The explicit formulation will generally be more susceptible to disconnected solutions since it leaves lower degrees of freedom for matching week starts and endings. We will defer this problem to Section 9.8 where we will describe how disconnected solutions can be cut off by specialised branching rules.

9.5. Integer Programming Model

The following column generation formulation for the cyclic roster problem is based on the weekly shift scheduling model of Section 8.6. Additional week and link variables will be coupled to the shift variables and represent the sequencing aspect of rostering. While coverage and break constraints are considered as before, additional roster constraints apply to the level of weeks and links. By combining these models, we arrive at an integrated model for cyclic tour scheduling.

As before, the model incorporates decision variables for shifts (S_{kn} for $k \in K$, $n \in N_k$), breaks (B_{rl} for break class $r \in R$ and break index l) and shortages (O_t for time period $t \in T$). Shifts of type k entail costs of c_k while shortages are penalised by period-dependent costs of c^{sht}_t. a_{tkn} and b_{trl} are shift and break coverage coefficients for period t. Break index sets for forward and backward constraints are denoted by L^{lst}_r and L^{est}_r while l^{lst}_r and l^{est}_r are the latest and earliest breaks per break class.

We newly introduce index sets I and J for valid weeks and links, respectively. A week variable ϑ_i indicates how many weeks of type $i \in I$ are used in a solution. Analoguously, λ_j denotes the number of links of type $j \in J$. Shift transition costs (τ_{k_1,k_2}) within each week are aggregated in a cost coefficient c^w_i for each week $i \in I$. Analoguously, link costs c^l_j, $j \in J$, incorporate transition costs if j represents a day-on link; note that day-off transitions do not entail penalties. In order to respect bounds on the number of roster weeks and to account for average working hours, we introduce an additional variable W denoting the number of roster weeks.

Each dynamically generated variable contributes a column to the constraint matrix. For week variables, we define shift coefficients s^{kn}_i with $s^{kn}_i = 1$ if week i covers shift (k, n) and 0 otherwise. By u_i, we will denote the number of work units aggregated over the week ($u_i = \sum_{k \in K} \sum_{n \in N_k} s^{kn}_i u_{kn}$).

Furthermore, we must specify the start and end junctions for each week $i \in I$. We define $ws^{km,on}_i = 1$ if i starts with m days on and incorporates shift type k on Monday ($ws^{km,on}_i = 0$ otherwise). If i starts with m days off and shift type k on the first workday of the week, we set $ws^{km,off}_i = 1$ ($ws^{km,off}_i = 0$ otherwise). Symmetrically, we define coefficients $we^{km,on}_i$ and $we^{km,off}_i$ for week endings on days on and off, respectively.

As described above, links are counterparts of weeks, connecting week end (link start) junctions to week start (link end) junctions. We therefore define $ls^{km,on}_j = 1$ if link $j \in J$ matches a week end junction with m days on and shift type k on Sunday ($ls^{km,on}_j = 0$ otherwise). Analoguously,

$ls_j^{km,off}$ equals 1 if j starts on a junction specifying m days off and shift type k on the last workday of a week. Link end (week start) junction coefficients $le_j^{km,on}$ and $le_j^{km,off}$ are defined symmetrically.

The integrated cyclic roster model reads as

$$\min \sum_{i \in I} c_i^w \vartheta_i + \sum_{j \in J} c_j^l \lambda_j + \sum_{\substack{k \in K \\ n \in N_k}} c_k S_{kn} + \sum_{t \in T} c_t^{sht} O_t \tag{9.1}$$

subject to

$$\sum_{\substack{k \in K \\ n \in N_k}} a_{tkn} S_{kn} - \sum_{r \in R} \sum_{l \in L} b_{trl} B_{rl} + O_t \geq d_t \; \forall \, t \in T \tag{9.2}$$

$$- \sum_{(k,n) \in K_r^F(l)} S_{kn} + \sum_{l' \in L_r^F(l)} B_{rl'} \geq 0 \;\; \forall \, r \in R, l \in L_r^{lst} \setminus \{l_r^{lst}\} \tag{9.3}$$

$$- \sum_{(k,n) \in K_r^B(l)} S_{kn} + \sum_{l' \in L_r^B(l)} B_{rl'} \geq 0 \;\; \forall \, r \in R, l \in L_r^{est} \setminus \{l_r^{est}\} \tag{9.4}$$

$$\sum_{(k,n) \in K_r} S_{kn} - \sum_{l \in L_r} B_{rl} = 0 \;\; \forall \, r \in R \tag{9.5}$$

$$-S_{kn} + \sum_{i \in I} s_i^{kn} \vartheta_i = 0 \;\; \forall \, k \in K, n \in N_k \tag{9.6}$$

$$\sum_{i \in I} ws_i^{km,on} \vartheta_i - \sum_{j \in J} le_j^{km,on} \lambda_j = 0 \;\; \forall \, k \in K : 1 \in N_k, \; m \leq m_{max}^{on} \tag{9.7}$$

$$\sum_{i \in I} ws_i^{km,off} \vartheta_i - \sum_{j \in J} le_j^{km,off} \lambda_j = 0 \;\; \forall \, k \in K : m+1 \in N_k, \; m \leq m_{max}^{off} \tag{9.8}$$

$$\sum_{i \in I} we_i^{km,on} \vartheta_i - \sum_{j \in J} ls_j^{km,on} \lambda_j = 0 \;\; \forall \, k \in K : 7 \in N_k, \; m \leq m_{max}^{on} \tag{9.9}$$

$$\sum_{i \in I} we_i^{km,off} \vartheta_i - \sum_{j \in J} ls_j^{km,off} \lambda_j = 0 \;\; \forall \, k \in K : 7-m \in N_k, \; m \leq m_{max}^{off} \tag{9.10}$$

$$\sum_{i \in I} \vartheta_i - W = 0 \tag{9.11}$$

$$\sum_{i \in I} u_i \vartheta_i - u_{avg} W \geq -u_{tol} \tag{9.12}$$

$$\sum_{i \in I} u_i \vartheta_i - u_{avg} W \leq u_{tol} \tag{9.13}$$

$$w_{min} \leq W \leq w_{max} \text{ and integer} \tag{9.14}$$

$$\vartheta_i \geq 0 \text{ and integer } \; \forall \, i \in I \tag{9.15}$$

$$\lambda_j \geq 0 \text{ and integer } \; \forall \, j \in J \tag{9.16}$$

$$S_{kn} \geq 0 \text{ and integer } \; \forall \, k \in K, n \in N_k \tag{9.17}$$

$$B_{rl} \geq 0 \text{ and integer } \; \forall \, r \in R, l \in L_r \tag{9.18}$$

$$O_t \geq 0 \text{ and integer } \; \forall \, t \in T \tag{9.19}$$

Objective function (9.1) vises at minimising a mix of shift costs, shift transition penalties (given week and link coefficients) and understaffing penalties. As in the model of Section 8.6, (9.2) is a set of coverage constraints while (9.3), (9.4) and (9.5) are forward, backward and equality constraints for breaks. Constraint set (9.6) matches shift variables shifts used in the weeks. While (9.7) and (9.8) match the numbers of week start/link end junctions, (9.9) and (9.10) enforce equality of junctions for week starts and link endings. (9.11) determines the number of roster weeks which are restricted by constraints (9.14). The roster week variable is additionally employed in inequalities (9.12) and (9.13), enforcing average working hours to be obeyed up to a given tolerance. (9.15) through (9.19) are nonnegativity and integrality constraints for week, link, shift, break and shortage variables.

While parts of the work rules are explicitly represent in model (9.1)-(9.19), other restrictions (e.g. minimum and maximum weekly working hours) are implicit in the definition of the index sets I and J. We will thus only include valid weeks and links.

The column generation formulation of the cyclic roster problem is based on representing only subsets $I' \subseteq I$ and $J' \subseteq J$ of the variables in a restricted master program corresponding to (9.1)-(9.19). Valid weeks and links will then be generated dynamically and added to the master program. With regard to the classification of Section 6.2, (9.1)-(9.19) is a column generation formulation with identical restrictions on the subsets.

We will shortly indicate relations to models proposed in the literature. Model (9.1)-(9.19) uses an implicit representation of weeks, using shifts as given by a distinct set of shift variables. This strongly relates to the implicit model for non-cyclic tour scheduling of Bailey [1985]. However, the above model specifies shifts explicitly, providing more control over shift sucessions than in the Bailey model. Work hour constraints like (9.12) and (9.13) have been incorporated in the non-cyclic tour scheduling models of Jaumard et al. [1998], Mason and Smith [1998] and Mason and Nielsen [1999]. Two-dimensional link encoding in a column generation approach is a novel proposition.

An illustration of the block-angular structure of model (9.1)-(9.19) is given in Fig. 9.11. For clarity reasons, nonnegativity and integrality constraints and bounds on roster weeks are not represented.

9.6. Week and Link Generation

Since the number of week and link variables in model (9.1)-(9.19) is very large, we only represent subsets $I' \subseteq I$ and $J' \subseteq J$ of the weeks and links in a restricted master program. New weeks and links with negative reduced costs are then generated on demand. In order to solve the LP relaxation, we will first drop the integrality restrictions in constraints (9.14) through (9.19). Section 9.7 will describe how integer solutions can be obtained by using compatible branching rules.

Generating link variables with negative reduced costs is fairly easy and basically amounts to enumerating feasible combinations of day-on/day-off stretches and shift types at the interface between weeks. Let $\gamma^{on}_{k,m}$ and $\gamma^{off}_{k,m}$ be the dual prices associated with constraints (9.7) and (9.8) for week start (link end) junctions and $\eta^{on}_{k,m}$ and $\eta^{off}_{k,m}$ be the dual prices of equalities (9.9) and (9.10) for week end (link start) junctions.

As an example, let (k_1, m_1) be the junction for a week ending on m_1 days on with shift type k_1 on Sunday. Analoguously, let (k_2, m_2) be the junction of a week starting on m_2 days on and shift type k_2 on Monday. If $k_1 \in G_{k_2}$ and $m_1 + m_2 \in [m^{on}_{min}, m^{on}_{max}]$, the week starting on (k_2, m_2) is a valid successor of the week ending on (k_1, m_1). The reduced cost $\bar{c}^{on,on}_j$ of the corresponding link j is calculated by subtracting the inner product of dual prices and constraint coefficients from the actual (penalty) costs. For the example, we obtain

$$\bar{c}^{on,on}_j := \tau_{k_1,k_2} + \eta^{on}_{k_1,m_1} + \gamma^{on}_{k_2,m_2}$$

Analoguously, we can calculate reduced costs for valid day-off/day-off, day-on/day-off and day-off/day-on links. Note however that no penalty costs are incurred for transitions over days off. Calculating the minimum cost link therefore amounts to determining

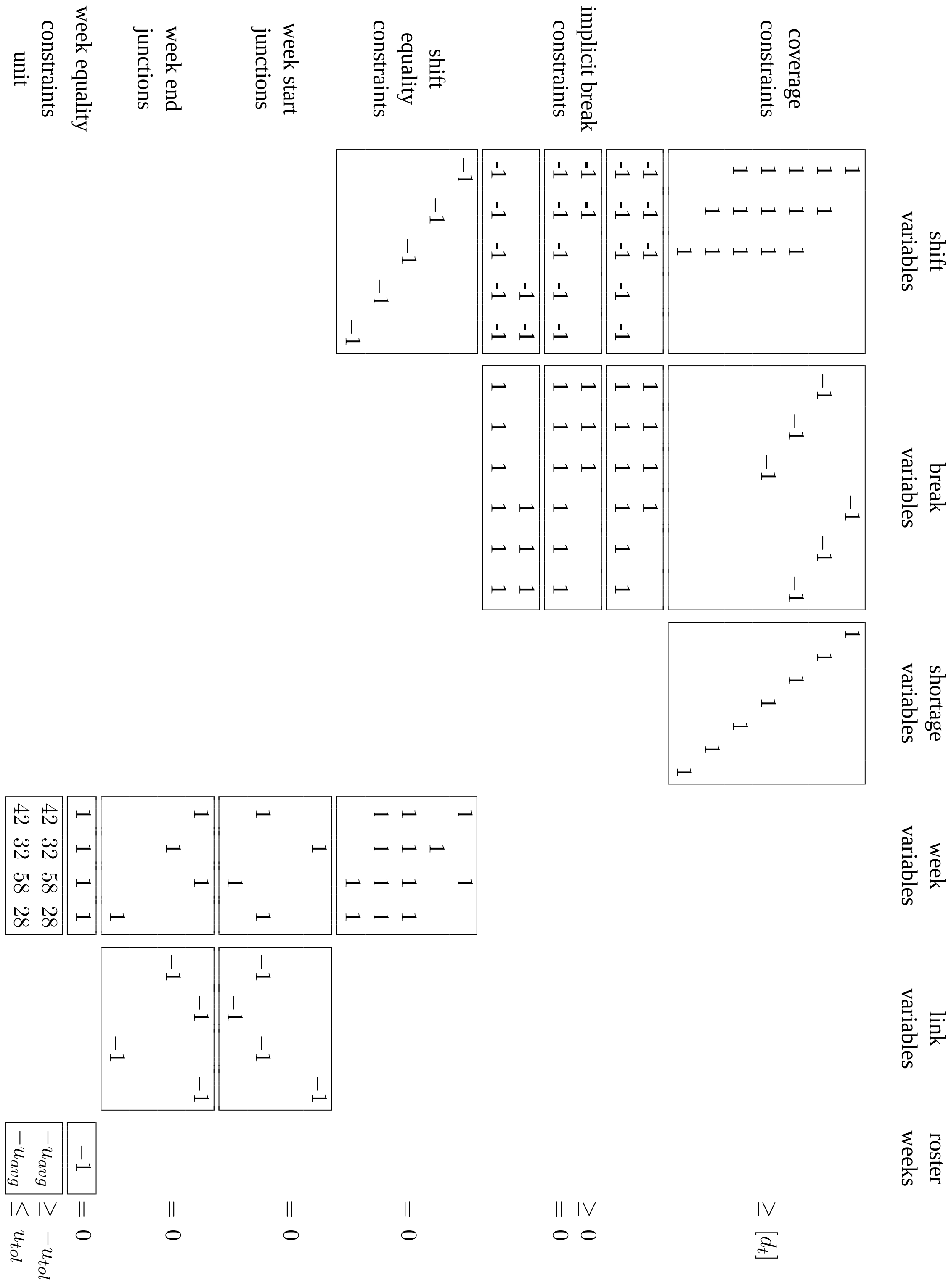


Figure 9.11.: Structure of the linear program.

$$\min \left\{ \begin{array}{ll} \min\limits_{\substack{(k_1,m_1),(k_2,m_2),k_1\in G_{k_2}\\ m_1+m_2\in[m_{min}^{on},m_{max}^{on}]}} & \tau_{k_1,k_2} + \eta_{k_1,m_1}^{on} + \gamma_{k_2,m_2}^{on}, \\ \min\limits_{\substack{(k_1,m_1),(k_2,m_2),k_1\in H_{k_2}\\ m_1+m_2\in[m_{min}^{off},m_{max}^{off}]}} & \eta_{k_1,m_1}^{off} + \gamma_{k_2,m_2}^{off}, \\ \min\limits_{\substack{(k_1,m_1),(k_2,m_2),k_1\in H_{k_2}\\ m_1\in[m_{min}^{on},m_{max}^{on}]\\ m_2\in[m_{min}^{off},m_{min}^{off}]}} & \eta_{k_1,m_1}^{on} + \gamma_{k_2,m_2}^{off}, \\ \min\limits_{\substack{(k_1,m_1),(k_2,m_2),k_1\in H_{k_2}\\ m_1\in[m_{min}^{off},m_{max}^{off}]\\ m_2\in[m_{min}^{on},m_{max}^{on}]}} & \eta_{k_1,m_1}^{off} + \gamma_{k_2,m_2}^{on} \end{array} \right\} \tag{9.20}$$

The calculation can be accelerated by predetermining minimum dual values over start and end junctions. If e.g. a link start junction cannot entail negative reduced cost even if the minimum end junction cost is taken, we do not need to evaluate any of the combinations involving the start junction. By solving (9.20), we determine one or several links of minimum reduced costs which are added to the restricted master program. The corresponding cost coefficient c_j^l is set to τ_{k_1,k_2} for day-on transitions and 0 otherwise. The calculation of (9.20) is linear in the number of shift types and the widths of the day-on and day-off windows ($[m_{min}^{on}, m_{max}^{on}]$ and $[m_{min}^{off}, m_{max}^{off}]$).

The problem of finding valid weeks with negative reduced costs and corresponds to solving a special resource-constrained shortest path problem (RCSP). Resources represent the number of consecutive days on and off and aggregated work units over the shifts. The RCSP is $\mathcal{NP}$-hard [Dror, 1994] and is usually solved by pseudo-polynomial dynamic programming (DP) algorithms, see e.g. Desrosiers et al. [1993], Jaumard et al. [1998]. For the generation of weeks, we basically define DP states for each day, shift type and given resource consumptions. States are unfolded in the order of weekdays, respecting valid shift transitions and minimum/maximum bounds on the number of days on and off.

However, encoding week transitions by links and junctions introduces an aspect which distinguishes the generation of weeks from standard resource-constrained shortest path problems. In our model, we are given dual prices for week start and end junctions, specifying the first and last shift type of the week and the number of initial or final days on or off. Week start and end junctions define starting *and* ending conditions for weeks. In contrast, there are no starting conditions in standard RCSPs, i.e. resource counters are initialised e.g. to 0 at the start of the dynamic program and aggregated in the course of unfolding the underlying network. In our setting, we also have to provide control over the number of days on/off and shift type at the start of the week in order to consider dual prices of start junctions. Together with the ending conditions as given by end junction prices, we are given a two-sided resource-constrained shortest path problem.

We will therefore solve the RCSP in two steps: Day-on start junction prices will be handled by a first dynamic program termed *backward net*. As day-on start junctions specify the number of initial days on in the week, the backward net will include a counter for the number of remaining days on. After creating this first stretch of days on, states are expanded via a day-off transition into a *forward net*. While remaining days on decrease in backward net expansions, states in the forward net comprise a counter for the number of days on used so far. While the backward net only incorporates day-on transitions, the forward net includes day-on and day-off state expansions. While day-on start junctions provide starting conditions for the backward net, day-off junctions represent weeks starting on days off. Day-off start junction prices are therefore directly considered in the forward net.

The basic idea is sketched in Fig. 9.12. The backward net is responsible for creating the first

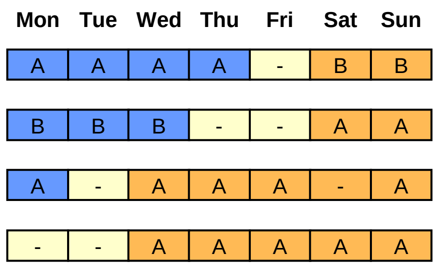


Figure 9.12.: Partition of week into backward net part and forward net part.

workstretch of each week (marked in blue) while further stretches of days on and off are handled by the forward net (shown in orange colour).

Additionally to the prices of (week) start and end junctions (denoted as above), we have to incorporate further dual values. The dual value $\pi_{k,n}$ associated with each of the constraints (9.6) gives a price for shift type k on day n. The dual prices for constraints (9.12) and (9.13) indicate the cost of a work unit; their sum will be denoted by ω. Finally, each week contributes to the number of roster weeks, and ϕ will denote the dual price associated with the week equality constraint (9.11).

We start with the description of the backward net. Let $P(k,n,m,u)$ represent the accumulated costs for a partial shift sequence with shift type k on day n, remaining workdays m and aggregated work units u, including the work units of shift (k,n). As mentioned before, the backward net only covers day-on transitions for the first stretch of workdays. In the following formula, we will assume that prices $\gamma^{on}_{k,m}$ and $\gamma^{off}_{k,m}$ equal $-\infty$ for invalid start junctions, e.g. if $1 \notin N_k$ for a day-on start junction. Furthermore, we will set $P(k,n,m,u) = \infty$ for invalid states. The DP recurrence for the backward net is then given by

$$P(k,n,m,u) := \begin{cases} -\gamma^{on}_{k,m+1} - \pi_{k,n} & n = 1, u = u_k \\ \min\limits_{\substack{k' \in G_k: \\ n-1 \in N_{k'}}} \left[P(k', n-1, m+1, u-u_k) + \tau_{k',k}\right] - \pi_{k,n} & n > 1, u > u_k \end{cases} \tag{9.21}$$

The first term considers day-on junction prices for states on the first day (Monday) and subtracts the dual price of the corresponding shift. While the counter u for the work units equals u_k after the first shift, m denotes the remaining days on and is decremented for each day. The second term refers to all further days, assuming that DP states are evaluated in order of weekdays. For a given state (k,n,m,u), minimisation runs over all states in the preceding layer $(n-1)$ if remaining days and units match. Additionally to considering dual costs for the current shift $(\pi_{k,n})$, the shift transition penalty $\tau_{k',k}$ is accounted for.

The recurrence equation for the forward net is similar, but includes additional day-off transitions and uses m as a counter for workdays accomplished so far (instead of counting remaining days down). $Q(k,n,m,u)$ represents the accumulated costs of a state with shift k on day n as the m'th consecutive day on and aggregated work units u. The forward net covers only shifts after an initial stretch of workdays (from the backward net) or days off (using start junctions) and therefore starts on the second day $(n = 2)$. Again, we will assume $Q(k,n,m,u) = \infty$ for "unknown" states, e.g. if $n < 2$.

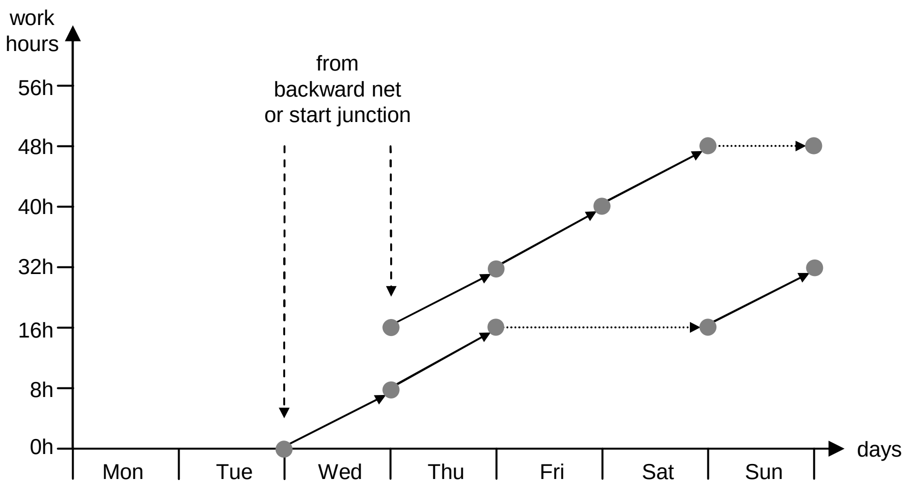


Figure 9.13.: Exemplary expansions in the forward net.

$$
Q(k,n,m,u) :=
\begin{cases}
-\gamma^{off}_{k,n-1} - \pi_{k,n} & m = 1,\ u = u_k \\[1em]
\min\Bigg[\min\limits_{\substack{k' \in H_k \\ n-n'-1 \in [m^{off}_{min}, m^{off}_{max}]}} P(k', n', 0, u - u_k), \\
\qquad \min\limits_{\substack{k' \in H_k, n-n'-1 \in [m^{off}_{min}, m^{off}_{max}] \\ m' \in [m^{on}_{min}, m^{on}_{max}]}} Q(k', n', m', u - u_k) \Bigg] - \pi_{k,n} & m = 1,\ u > u_k \\[1em]
\min\limits_{k' \in G_k} \left[Q(k', n-1, m-1, u-u_k) + \tau_{k',k} \right] - \pi_{k,n} & m > 1,\ u > u_k
\end{cases}
\tag{9.22}
$$

Again, some words of explanation are in place. The first line inserts day-off junction values into the forward net, using shift k on day n and accounting for a first day on ($m = 1$) and the work units of shift type k ($u = u_k$). Lines two and three refer to day-off transitions ($m = 1$), expanding states from the backward and forward net. Backward net expansions are restricted to states in which no further workdays have to be taken ($P(k', n', 0, u - u_k)$) while for forward net states, the number of days on in the workstretch must be in $[m^{on}_{min}, m^{on}_{max}]$. In both cases, the number of days off must be within the given limits $[m^{off}_{min}, m^{off}_{max}]$. Additionally to summing up work units, dual shift costs $\pi_{k,n}$ are subtracted. The last line refers to transitions between consecutive days on, taking shift costs as well as transition penalties ($\tau_{k',k}$) into account.

In the definitions of $P(k, n, m, u)$ and $Q(k, n, m, u)$, we have tacitly assumed that only states are evaluated which can result in a valid solution. A state (k, n, m, u) is e.g. invalid if $n \notin N_k$, $u > u_{max}$ or $m \leq m^{on}_{max}$ in the forward net. An illustration for the evaluation of forward net states can be found in Fig. 9.13. The state space is only displayed with regard to working hours (work units) and days of the week while shift types and the day-on counter were left out.

The above recurrence formulae evaluate states by referring back to previous states ("pulling"). However, (9.21) and (9.22) will be typically implemented by expanding states in forward direction ("reaching"). By the reaching strategy, we naturally evaluate only valid states whose number is quite moderate in practice. The dynamic programs will thus be evaluated in order of increasing weekdays, expanding each state to the next layer(s). Dual values of week start junctions are initially inserted into the backward net (for day-on junctions) or forward net (for day-off junctions).

Additionally, final backward net states (for $m = 0$) will be expanded to the forward net.

Analysing the complexity of the above operations, we first note that the workstretches created by backward and forward net are disjoint. We can therefore give bounds on the common complexity of both dynamic programs. The number of states is in the order of $|K| \cdot m^{on}_{max} \cdot u_{max}$ for each weekday ($|N| = 7$), but their exact number varies between the days. On the one hand, the possible values of m depend on the weekday. As an example, the forward net only contains states for $m = 1$ on Tuesday, for $m \in \{1, 2\}$ on Wednesday etc. On the other hand, only a limited number of different sums of work units are valid on each day. On Monday, the minimum work units amount to $\min_{k \in K} u_k$, and Saturday states which may lead to a valid week will have between $\max(u_{min} - \max_{k \in K} u_k, 0)$ and $u_{max} - \min_{k \in K} u_k$ work units. If we additionally take restrictions on workstretch lengths into account, this can be used to additionally prune the search space. In realistic test cases, we can furthermore assume that only very few sums of work units will be generated. In the extreme case of equal work units for all shift types, the unit dimension in the above dynamic programs degenerates, and aggregated units will reflect the number of workdays so far.

The number of operations for each valid state depends on its type. Each node will either allow for day-on transitions (backward net states and forward net states with $m > 1$) or for day-off transitions (forward net states with $m = 1$). Day-on transitions necessitate $|G_k|$ operations if k is the shift type of the state. Day-off transitions entail operations in the order of $|H_k| \cdot (m^{off}_{max} - m^{off}_{min} + 1)$, but the exact complexity again depends on the weekday. Additionally, we have to take start junction prices into account, namely at backward net states for $n = 1$ and forward net states for $m = 1$ and $n - 1 \leq m^{off}_{max}$.

It remains to be shown how weeks with negative reduced costs can be obtained from the backward and forward net calculations. Nodes of both dynamic programs (9.21) and (9.22) may represent final states of a week. Additionally to subtracting dual values for week end junctions, we must consider costs of work units and roster weeks. Finding a minimum cost week amounts to evaluating

$$\min_{\substack{k \in K \\ u \in [u_{min}, u_{max}]}} \left\{ \min \left(\min_{n \in \{7 - m^{off}_{max}, \dots, 6\}} [P(k, n, 0, u) - \eta^{off}_{k, 7-n}], \right. \right.$$
$$P(k, 7, 0, u) - \eta^{on}_{k,7},$$
$$\min_{\substack{n \in \{\max(2, 7 - m^{off}_{max}), \dots, 6\} \\ m \in [m^{on}_{min}, m^{on}_{max}]}} Q(k, n, m, u) - \eta^{off}_{k, 7-n}, \tag{9.23}$$
$$\left. \left. \min_{m \in [1, m^{on}_{max}]} Q(k, 7, m, u) - \eta^{on}_{k,m} \right) - u\omega \right\} - \phi$$

where the second line ($P(k, 7, 0, u) - \eta^{on}_{k,7}$) is only taken if the setting allows for seven days on ($m^{on}_{max} = 7$), meaning that the backward net contains workstretches of seven days. The first two lines retrieve final states from the backward net, assuming that all prescribed workdays from the start junctions have been carried out ($m = 0$). For $n \leq 6$, the week will end on days off, and we will subtract the corresponding day-off junction value. For $n = 7$ (if valid), we retrieve a week which is only made up of workdays and ends on a day-on junction. Analoguously, we retrieve weeks ending on days off from the forward net (third line) if the number of workdays is in the valid range. The fourth line corresponds to forward net states corresponding to week endings on days on. In every case, dual costs of week units ($u\omega$) and roster weeks (ϕ) are considered.

Let us shortly analyse the complexity of the above operations. Day-off end junctions from the backward net (first line of (9.23)) are retrieved by minimisation over shift types, units and final days off in the order of $|K| \cdot (u_{max} - u_{min} + 1) \cdot m^{off}_{max}$. For day-on endings from the backward net (second line), we minimise over shift types and units (complexity $|K| \cdot (u_{max} - u_{min} + 1)$). Day-off endings from the forward net (third line) additionally require a minimisation over valid numbers of days on ($|K|\cdot(u_{max}-u_{min}+1)\cdot(m^{on}_{max}-m^{on}_{min}+1)\cdot m^{off}_{max}$). Finally, day-on end junctions from the forward net (fourth line of (9.23)) are retrieved in the order of $|K| \cdot (u_{max} - u_{min} + 1) \cdot m^{on}_{max}$ operations.

Actual reduced cost weeks are retrieved by tracing back over the states which have led to the respective final state. The week is then added to the restricted master program (9.1)-(9.19). The cost coefficient c^w_j is calculated as the sum of hift transition penalties accounted for in (9.21) and (9.22). Note that actual shift costs are coefficients of the shift variables S_{kn} and are only transferred to the column generation subproblem via dual costs $\pi_{k,n}$.

Some loosely related approaches have been described in the literature. Mason and Smith [1998] consider weekly working hours in column generation for named rostering. In the solution of an RCSP for nurse scheduling, Jaumard et al. [1998] include matching conditions with regard to the previous planning period. However, there is no need for a two-sided dynamic program in their case. A similar dynamic programming formulation can be found in Çezik et al. [2001] who however use explicit nodes for days off. The two-sided backward/forward DP scheme is a novel contribution which becomes necessary by the special encoding of week links.

The above subproblems for generating links and weeks frequently yield multiple optimal solutions. In such cases, all resulting columns are added to the master program. It should be noted that the LP relaxation of (9.1)-(9.19) is initially infeasible if no starting solution can be provided. The violated constraints (9.12) and (9.14) for units and roster weeks are therefore relaxed by introducing slack variables. By attributing high costs to the slack variables, we quickly reach feasibility with regard to the original problem which is retained afterwards.

9.7. Integer Branching

After describing the subproblems, we now return to the master program which is basically a set covering formulation. Many authors have noted that set covering models exhibit advantageous integer properties and provide tight lower bounds, see e.g. Bailey [1985], Jacobs and Bechtold [1993], Aykin [1996] and Mehrotra et al. [2000]. As a consequence, they lend themselves very well for solution by LP-based branch-and-bound procedures.

If a solution is fractional, we first check if the value f of the roster week variable W is integer. If not, we apply a branch and impose $W \geq \lceil f \rceil$ on the first branch and $W \leq \lfloor f \rfloor$ on the second branch. This kind of branching turned out to have strong integerising effects, see also Smith and Wren [1988] and Gamache and Soumis [1998] for similar applications in crew scheduling. Note that under most realistic cost structures, shift costs will increase monotonically with the number of roster weeks since weekly working hours are fixed (up to a given tolerance). Conversely, if in the LP relaxation at a search node, the value of the roster week variable is greater than $\lceil f \rceil$, it is reasonable to assume that the search node will lead to a roster of at least $\lfloor f \rfloor$ weeks. This justifies following the roundup branch first (see also Section 9.10).

Additionally, we can branch on the shift variables S_{kn}. If a given variable S_{kn} has a fractional value f, we enforce $S_{kn} \geq \lceil f \rceil$ on one of the branches and $S_{kn} \leq \lfloor f \rfloor$ on the other. Branching on variables which are closest to an integer value turned out to be advantageous in order to quickly find solutions of high quality. However, rounddown branches turned out to entail rather slow convergence. We therefore search for the shift whose value is closest to the next higher integer value and first follow the roundup branch. Clearly, rounding up is more constructive and therefore allows to obtain integer solutions more quickly.

When devising branching rules for remaining fractionalities, we must respect compatibility with the column generation procedures described before. In the shift planning problem of Chapter 7, branching rules were based on variables of the compact formulation. Here we can follow a similar approach. In analogy to the task pair branching of Section 7.6, we branch on fractional shift transitions within weeks. For each used shift transition $(k_1, n_1) \rightarrow (k_2, n_2)$ between consecutive days on or over days off, we sum up the values ϑ_i of all weeks incorporating this transition. If the sum for any shift transition has a fractional value, say f, we branch and enforce the sum to be $\geq \lceil f \rceil$ on one branch and $\leq \lfloor f \rfloor$ on the other ("constraint branching"). Clearly, this can be easily incorporated in the structure of the week subproblem by attributing the corresponding dual value to the shift transitions in the backward and forward net. As for shift branching, we prefer roundup branches.

If all week values are integral, links will also be integer because the related transportation problem is naturally integer, see e.g. Ahuja et al. [1993]. However, branching on link variables is still helpful if solutions are still fractional. Similarly to shift transition branching within the weeks, we search for fractional transitions described by links, preferring roundup branches. Since link column generation basically corresponds to complete enumeration, dual values of link transition cuts can be easily incorporated if some attention is given to the speedup idea of Section 9.6.

After applying these branching rules, week and link variables were always integer. However, the break variables can still be fractional. Aykin [1996] notes that break fractionalities are quite rare which is confirmed by our experiments. If fractional values remain, these can be easily cut off by branching according to the aforementioned rules.

Note that these branch types do not give an absolute guarantee for finding integer solutions. In contrast to the approach of Chapter 7, our integer programming model is based on a discretisation of the search space. We therefore aim at rendering the derived variables ϑ_i, $i \in I$, and λ_j, $j \in J$, integer. However, branching basically applies to the original flow variables. In practice, it is common to devise branching rules which are empirically sufficient for obtaining integer solutions, see e.g. Mehrotra et al. [2000]. It should be clear that the above branching rules leave only little room for fractionalities. In practice, the above branch types were always sufficient for obtaining integer solutions.

9.8. Subcycle Branching

As described in Section 9.3, resulting integer solutions can still contain disconnected subcycles. An example is given in Fig. 9.14. It can be seen that the used links match the weeks, but still no contiguous cyclic roster can be built since weeks $\{1, 2\}$ and $\{3\}$ are disconnected. Under typical roster restrictions, subcycles are even likely to occur: Typical basic patterns in rosters are e.g. made up five workdays with equal shift types and two days off. It is clear that week endings of such patterns will often be valid successors of their own starts.

Laporte et al. [1980] propose to cut off exactly one integer solution if subcycles arise. Clearly, this is the weakest possible remedy for disconnectedness. Mason [1999] provides a review on subcycle problems and describes a more involved three-way branch for a formulation based on a workstretch representation. In the following, we will analyse the connectedness of week-link solutions and derive a novel branching scheme for the above solution approach.

We first recall some notions of graph theory on directed graphs (*digraphs*). A *path* $v_1 \rightsquigarrow v_2$ in a digraph $G = (V, E)$ is a sequence $v_1, v_2, \ldots, v_n$ of vertices such that $(v_i, v_{i+1}) \in E$ for all $i = 1, \ldots, n-1$ and no edge is used more than once. A path $v_1 \rightsquigarrow v_n$ that starts and ends at the same vertex ($v_1 = v_n$) is called *circuit*. A digraph is *strongly connected* if for any two vertices $v_1, v_2 \in V$, there is a path $v_1 \rightsquigarrow v_2$ and a path $v_2 \rightsquigarrow v_1$. If a digraph is not strongly connected, it contains more than one connected *component*. A path which contains every edge of a digraph exactly once is called an *Euler path*, and an Euler path starting and ending at the same vertex is an *Euler circuit*. As in previous chapters, edges with origin (destination) node v are called *outedges* (*inedges*) of v. The number of outedges (inedges) of v is called the *indegree* (*outdegree*) of v.

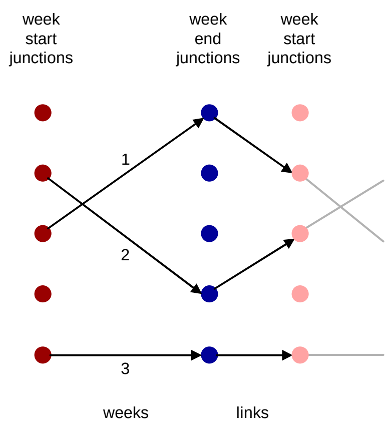


Figure 9.14.: Weeks and links generating disconnected subcycles.

Now imagine an integer solution to the cyclic roster problem. From the given weeks and links, we must build a closed cyclic roster. We define V_1 and V_2 to be sets of vertices representing week start and end junctions, respectively. In the example of Fig. 9.14, V_1 corresponds to the red nodes and V_2 to the blue nodes. For each week $i \in I$ which is part of the solution, we create ϑ_i edges between the junctions corresponding to its start and end, i.e. weeks are represented by edges in $V_1 \times V_2$. Analoguously, we insert λ_j edges between the week end and week start node corresponding to the origin and destination of the link, respectively. Links thus correspond to edges in $V_2 \times V_1$. The result is a bipartite *roster graph*. A cyclic roster now corresponds to an Euler circuit in the roster graph, i.e. a circuit which uses each edge exactly once.

According to a prominent result of graph theory, a digraph contains an Euler circuit if and only if

1. the digraph is strongly connected and

2. the indegree of each vertex equals its outdegree,

see e.g. Wilson [1996]. Condition 2 exactly corresponds to constraints (9.7) through (9.10) of the cyclic roster problem, enforcing equality of week and link junctions. We therefore only have to ensure that the roster graph is strongly connected.

Subcycle problems are also treated in integer programming models of the vehicle routing and travelling salesman problems. When using LP-based solution methods, subtour elimination constraints are often added dynamically to the model, see Lawler et al. [1985, p. 26]. The main difference in cyclic roster generation lies in the fact that the “cities” (junctions) to be visited are not fixed in advance. In fact, junctions which are used by a given LP solution do not need to be present in an optimal solution. The necessary subcycle breaking restrictions cannot be formulated as linear constraints. We will therefore resort to a branching scheme.

Assume that an integer solution to model (9.1)-(9.19) is given. We can then determine strongly connected components of the associated roster graph, e.g. by Tarjan's algorithm [Tarjan, 1972]. If the graph is disconnected, each strongly connected component represents one subcycle. Let $V_1' \subseteq V_1$, $V_2' \subseteq V_2$ be the week start and end junctions involved in the largest subcycle. By $I_{V_1' \to \overline{V_2'}} \subseteq I$, we will denote the set of weeks starting on a junction associated with one of the nodes in V_1' and ending on a junction which is not associated with one of the nodes in V_2'. Analoguously, the set $J_{V_2' \to \overline{V_2'}} \subseteq J$ will denote the all links starting in V_2' and ending on a junction which is not in V_2'. $I_{V_1' \to V_2}$ ($J_{V_2' \to V_1}$) represents all weeks (links) starting on a junction in V_1' (V_2'). Finally, $J_{V_2 \to \overline{V_1'}}$ and $J_{\overline{V_2'} \to V_1}$ are the sets of junctions which do not end or start in V_1' or V_2', respectively.

We can now create a four-way branch with the following restrictions:

1. $\sum_{j \in J_{V_2' \to \overline{V_1'}}} \lambda_j \geq 1$

2. $\sum_{i \in I_{V_1' \to \overline{V_2'}}} \vartheta_i \geq 1$

3. $\sum_{j \in J_{V_2 \to \overline{V_1'}}} \lambda_j = 0$

4. $\sum_{j \in J_{\overline{V_2'} \to V_1}} \lambda_j = 0$

Branch 1 and 2 enforce at least one link or week to emanate from the given subcycle. Clearly, a final solution might not at all use junctions from V_1' or V_2'. These cases are therefore covered by branch 3 and 4, forbidding links using the corresponding junctions. Clearly these conditions could also be imposed on the week variables since the junctions of links and weeks are synchronised by constraints (9.7) through (9.10). It should be clear that the subcycle between V_1' and V_2' is not valid on any of the branches.

Let us analyse the impact of the above constraints on the structure of the subproblems. The constraints on branches 1, 3, and 4 refer to links. As links are basically generated by complete enumeration, these restrictions are easily incorporated if some care is taken with regard to the speedup procedure of Section 9.6.

The second branch is somewhat more complicated as it refers to the generation of weeks by the backward/forward net approach. The constraint implies that we must be able to price out combinations of start and end junctions. Costs of weeks from V_1' to $\overline{V_2'}$ must therefore be reduced by the dual value of the branching constraint. This however introduces interdependencies between week start and end junction decisions. These are resolved by the following approach: We first execute the backward/forward dynamic programs as described in Section 9.6. If a resulting solution links V_1' to $\overline{V_2'}$, we subtract the dual value of the branching constraint. As this dual value may render further reduced costs negative, we run the two-sided RCSP algorithm for a second time, allowing only for start junctions in V_1' and end junctions in $\overline{V_2'}$. This clearly means that following the second branch entails n additional subproblem executions on the n'th search tree level. Because the constraints on branches 1, 3, and 4 are much easier to incorporate, we heuristically prefer these link branches in the branch-and-bound procedure.

Up to now, we have assumed that subcycle branches are only applied to integer solutions. In practice, it will be very difficult to arrive at connected solutions if many integrality conditions have been imposed before. In the above exposition, we have used a roster graph, using one edge for each single week and link. However, we can equivalently check connectivity on a graph which uses only one edge for each week or link which is part of the solution, i.e. we collapse multiple edges to single ones. Clearly, this *connectivity graph* is connected if and only if the roster graph is connected. If we are given a fractional solution, we can build the connectivity graph by introducing an edge for each positive week and link variable. Clearly, this is an optimistic assumption for connectivity since we cannot expect each of these weeks and links to be part of a final solution. However, this enables us to render fractional solutions connected. In the implementation, we apply integer branches only to connected solutions, i.e. we generate subcycle branches as long as solutions are not connected.

It is worth mentioning that other branches of lower order are conceivable[2]. However, the above four-way branch offers two distinct advantages: On the one hand, it separates the complicating week branch 2 from the easier link branches. On the other hand, it exhibits advantageous integer properties because the right-hand sides of branches 1 and 2 equal 1. This means that the subcycle branch will often save us from an additional integer branch on fractional links. In first experiments, the above four-way branch has shown to outperform alternative binary or three-way branches.

Imagine that we have arrived at a connected integer solution. Finding an actual cyclic roster amounts to building an Euler circuit in the roster graph. Algorithms for this task date back to the 19th century and were proposed by Hierholzer [1873] and Fleury [1883]. Hierholzer's algorithm is not only more efficient, but is also very intuitive. It basically consists in building edge-disjoint circuits, each time starting at a node whose outedges are not yet completely covered. Because we assume indegrees to match outdegrees of each node, all edges can be covered by such circuits. Due to connectedness, all of these circuits can be linked into a single Euler circuit covering all edges. The computational complexity of Hierholzer's algorithm is in $\Theta(|E|)$, see also Jungnickel [2002].

For the overall problem of building a roster, we additionally have to assign breaks as given by the break variables. As described in Section 8.6, this can be done by a simple single-pass procedure proposed by Bechtold and Jacobs [1990].

9.9. Generalisations

In the following, we will describe some possible extensions to the above model which refer to additional constraints which may be required in some rostering settings.

Roster scenarios often specify an A-out-of-B weekends-off constraint. Since blocked weekends are virtually always undesirable, this automatically implies that weekends off must be evenly spread over the roster. We will only consider the case $A = 1$, i.e. each B'th weekend is a weekend off; generalising to $A > 1$ is straightforward. In order to evenly spread weekends off, it does not suffice to introduce an additional constraint in the master program. Instead, we decompose rosters into blocks of B weeks at maximum, see Emmons [1985], Burns and Koop [1987] and Panton [1991] for related approaches. Roster generation will then assign each week to a *week position* from $\{1, \ldots, B\}$ within these blocks. Each week in the B'th position is enforced to include a weekend off. Note that regardless of its position, each week covers shift demands of the unique model week. Additionally to assigning weeks to positions, we replicate the junctions. Junctions will generally connect weeks of subsequent positions in cyclic order.

The basic idea is sketched in Fig. 9.9, assuming $B = 2$, i.e. each second weekend is a weekend off. The setting includes two week positions, entailing two systems of start and end junctions; the second set of end junctions is replicated on the left-hand side. Links mostly connect weeks of subsequent week positions. However, the length of a roster may not be divisible by 2, meaning that we also have to allow for shorter stretches between weekends off. This is done by admitting

[2]The reader may verify that the binary branch with conditions

$$\sum_{j \in J_{V_2' \to \overline{V_1'}}} \lambda_j + \sum_{i \in I_{V_1' \to \overline{V_2'}}} \vartheta_i \geq \frac{1}{2w_{max}} \left(\sum_{j \in J_{V_2' \to V_1}} \lambda_j + \sum_{i \in I_{V_1' \to V_2}} \vartheta_i \right)$$

and

$$\sum_{j \in J_{V_2 \to \overline{V_1'}}} \lambda_j = 0, \sum_{j \in J_{\overline{V_2'} \to V_1}} \lambda_j = 0$$

also invalidates the current subcycle without cutting off valid solutions.

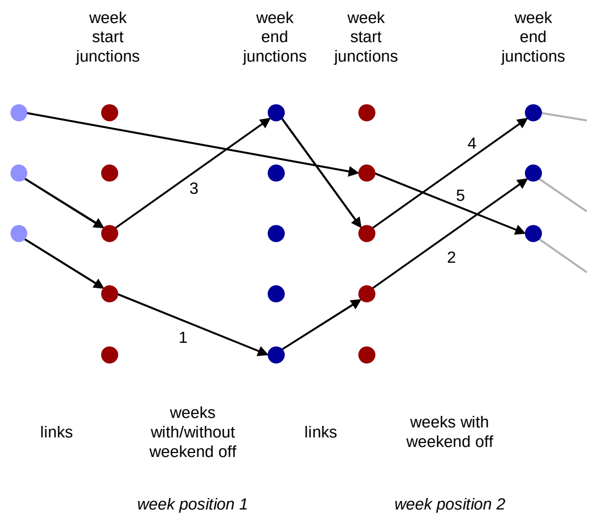


Figure 9.15.: Generalised week link encoding for weekends off.

links between each system of week end junctions and the start junctions of the last week position. As an example, Fig. 9.9 shows a roster of five weeks, including a link between an end junction of position 2 and another start junction of a week in position 2. Consequently, the consecutive roster weeks 4 and 5 both include a weekend off.

Introducing weekend-off constraints not only entails a replication of junction systems and constraints, but also results in a linear growth of the number of week and link subproblems to be solved. Additionally, subcycle branches have to be generalised.

Another possible generalisation regards the simultaneous creation of several rosters. Rosters then relate to different worker categories, and different restrictions can apply to each roster. Workloads are covered by all rosters in common, accounting for interdependencies between the rosters. Different subsets of shift types may be valid for the different worker categories [Alvarez-Valdez et al., 1999]. When generating multiple rosters, we must take care with the roundup cut for the number W of roster weeks described in Section 9.7. In general, we will not be able to impose lower limits on the weeks of single rosters. In practice however, roundup cuts may still be helpful in obtaining integer solutions if we keep in mind that they render the approach heuristic.

Up to now, we have assumed that one worker is assigned to each roster week. However, if employees work in crews, more than one worker may be attributed to each roster week, meaning that demands are covered in multiples of the crew size [Laporte et al., 1980] [Chew, 1991]. The number of workers on each week is also known as the *blocking factor* of the roster. If all rosters which are planned simultaneously have identical blocking factors, this can be easily incorporated by dividing (and rounding up) demand figures. If blocking factors are different, we can still divide demands by their greatest common divisor. Remaining factors are considered by letting each week contribute more than one shift coverage in the equality constraints (9.6).

9.10. Experimental Results

The cyclic roster algorithm has been implemented in Visual C++ 7.1, using BCP of COIN-OR for the implementation of branch-and-price [Ralphs and Ladányi, 2001]. XPRESS-MP Release 2004 was used as LP solver [Dash, 2004]. Basic data for the test cases was already given in the evaluation of the different break models in Chapter 8. The scenarios were chosen by different criteria. First, it was checked by visual observation if demands were typical of a model week. Scenarios

No.	original shift types	used shift types in shift scheduling LP relaxation
B03	33	20
B05	17	11
C01	176	33
C13	9	5
C14	41	15
C16	94	8
C17	5	5
D09	8	8
D10	47	22
D12	31	18
D13	31	13
D16	280	51
D17	41	22
D26	71	30
D27	29	16
D28	32	20

Table 9.2.: Working subset reduction.

had to span over at least seven days, and scenarios in which workloads differed substantially from day to day were left out. Furthermore, we have not included scenarios with crew tasks because it would not have been clear if we should use cyclic rosters with blocking factors (see Section 9.9) to cover such workloads. Additionally, we have excluded scenarios which obviously ask for a mixed workforce with different qualifications. These selection rules resulted in the choice of 16 test cases.

The scenarios were then restricted to seven days, reaching from Monday to Sunday. Scenarios including movable tasks were submitted to the levelling procedure of Chapter 4. Since convergence of the branch-and-price algorithm proved to be sensitive especially to the number of shift types, we have restricted the shift types to working subsets. While in Chapter 7, we have used the initial heuristic solution for the choice of shift types, this did not seem to be appropriate for demand-level planning. We have therefore basically used the approach of Mabert and Watts [1982] and Brusco and Jacobs [1998b] who propose to generate working subsets by solving the analoguous shift scheduling formulation. The shift types employed by the resulting shift plan are then used for the solution of the tour scheduling problem.

In contrast to Brusco and Jacobs [1998b], we have not projected the weekly problem on a single day, but solved the weekly shift scheduling formulation as described in Chapter 8. By this approach, we account for the fact that the days of the week may require different shift types in order to efficiently cover demands. Table 9.2 shows the original numbers of shift types and the numbers of shift types used in the shift scheduling solution.

It can be seen that the number of effectively used shift types is generally quite moderate. It is worth mentioning that the shift scheduling solutions were nearly always integer, i.e. the shift type selection is often based on actual demand-level shift plans. Scenarios C01, D16 and D26 still asked for 33, 51 and 30 shift types, respectively. In order to obtain moderately sized working subsets, we have heuristically restricted these scenarios to the 20 most frequent shift types in the shift scheduling solution.

Clearly, it is not evident that working subsets from a shift scheduling model are also appropriate

No.	shift types	shifts	paid shift durations			shift costs			action times	demands		
			min.	max.	avg.	min.	max.	avg.		avg.	max.	std. dev.
B03	20	140	360	480	462.0	70.0	131.0	92.8	568	6.4	20	4.7
B05	11	77	480	480	480.0	80.0	80.0	80.0	656	3.0	11	1.6
C01	20	140	240	480	372.0	43.0	1566.5	411.2	1335	9.5	48	9.4
C13	5	35	330	690	516.0	58.0	118.0	89.0	1267	13.2	59	15.1
C14	15	105	360	600	452.0	60.0	100.0	75.3	668	3.0	15	2.2
C16	8	56	360	540	435.0	60.0	90.0	72.5	198	1.5	6	1.9
C17	5	35	450	570	522.0	85.0	143.5	106.9	1439	8.9	43	8.1
D09	8	56	360	480	420.0	70.0	92.0	80.5	1448	2.3	14	2.8
D10	22	154	480	480	480.0	90.0	126.5	95.9	614	8.4	25	6.5
D12	18	126	360	480	460.0	70.0	129.0	93.8	575	5.5	22	4.7
D13	13	91	360	480	461.5	71.0	129.0	90.7	175	5.7	10	3.1
D16	20	140	360	720	504.0	70.0	130.0	95.2	515	18.7	59	15.2
D17	22	154	360	720	490.9	60.0	120.0	81.8	672	16.3	36	6.6
D26	20	68	240	720	503.4	50.0	130.0	94.5	809	9.7	34	8.5
D27	16	52	270	540	426.1	55.0	101.0	82.1	817	1.8	9	2.0
D28	20	140	360	540	460.5	70.0	129.0	89.2	529	8.9	31	7.6

Table 9.3.: Scenario data.

for tour scheduling and cyclic rostering problems. In practice however, it has frequently been observed that working subset reductions do not severely affect solution quality if the reduced number of shift types allows for sufficiently scheduling flexibility, see e.g. Easton and Rossin [1991], Bechtold and Brusco [1994b] and Brusco and Jacobs [1998b]. Brusco and Jacobs [2001] show that as few as three to five shift types can be sufficient if their choice is taken carefully. One possible explanation for these observations is that scheduling formulations often allow for several alternative optimal solutions, see e.g. Segal [1972], Thompson [1993] and Brusco and Johns [1995].

Working subset methods not only make scenarios amenable to complex roster algorithms. Limiting the number of used shift types is often also desired by planners, and many approaches have been described which treat shift type restrictions as implicit or explicit objectives in rostering, see e.g. Gaballa and Pearce [1979], Holloran and Byrn [1986], Schindler and Semmel [1993] and Brusco et al. [1995]. Another reason for using limited shift type sets is the administrative burden and the difficulty of managing briefing sessions when many shift start times are allowed [Brusco and Jacobs, 2001]. Since organisations using cyclic rosters usually face strong union regulations, it is natural to assume that only a limited set of shift types can be exploited. Working subset methods then represent efficient means of determining efficient sets of shift types before entering into the solution of the roster problem, see also Section 2.9.

Table 9.3 summarises data on the resulting test cases. Besides the numbers of shift types and shifts, we indicate minimum, maximum and average shift costs and lengths. Note that these entities refer to *paid* durations, i.e. unpaid breaks are subtracted from the shift lengths. Furthermore, the table gives the numbers of action intervals (as defined in Section 8.7) and information on workloads throughout the week (average and maximum demands as well as standard deviation). These scenarios were also used for the evaluation of the break model in Chapter 8; further information on the numbers of breaks and action times can be found in Section 8.8.

We have used the cyclic roster algorithm to generate cyclic rosters of variable lengths which cover labour requirements as completely as possible. We have therefore set w_{min} to 1 and w_{max} to 150 which was sufficient for covering workloads in all test cases. Some scenarios contain workloads in periods which are not covered by any shift type, meaning that shortage penalties cannot be avoided. For all test cases, we have set the limits $[m^{on}_{min}, m^{on}_{max}]$ on the number of

No.	average number of day on successors	average number of day off successors
B03	5.9	17.8
B05	3.0	10.1
C01	7.3	19.5
C13	1.8	4.8
C14	3.8	12.9
C16	2.0	7.0
C17	1.0	4.2
D09	3.5	8.0
D10	6.0	19.4
D12	4.8	15.9
D13	5.0	12.1
D16	6.7	19.5
D17	5.5	18.4
D26	10.3	20.0
D27	6.9	16.0
D28	9.7	19.4

Table 9.4.: Valid successor shift types on days on and off.

consecutive days on to $[2, 6]$ and $[m^{off}_{min}, m^{off}_{max}]$ to $[2, 4]$. Shift type lengths were subjected to a discretisation of 30 minutes, i.e. work units are counted in multiples of half hours. Each roster week was constrained to comprise between 28 (i.e. $u_{min} = 56$) and 52 (i.e. $u_{max} = 104$) work hours, the average work time was set to 40 hours (i.e. $u_{avg} = 80$) with a tolerance of 4 hours over the whole roster ($u_{tol} = 8$). Between two shifts, we have imposed a minimum rest time of eight hours; the additional rest time over days off (see Section 9.2) was chosen as six hours.

Shift start time differences between consecutive days were limited to 120 minutes. As explained in Section 9.2, we impose transition penalties for consecutive shifts with different start times. These penalties were chosen to be linear in the start time differences. To ensure that penalties are treated as a subordinate objective, we normalise them with the minimum shift costs. For the tests, we have limited transition penalties for backward-creeping shift start times (e.g. start time 10h on Monday, 8h on Tuesday) to 20% of the minimum shift costs. For forward-creeping start times (e.g. 8h on Monday, 10h on Tuesday), we have used maximum relative penalties of 5%. Period shortages were penalised by a factor of 1000 for each uncovered minute, meaning that shortages are avoided whenever possible.

Table 9.4 shows the numbers of successor shift types as averages over all shifts. It can be seen that the above parameter choices generally lead to considerable scheduling flexibility. Table 9.5 shows the number of states in the forward and backward dynamic programs as described in Section 9.6. In the right column, the potential number of link variables is given. While on some scenarios (e.g. C13, C17), the number of links is quite moderate, scenarios C01, D10, D17 and D28 ask for more than 15000 potential link variables. Clearly, representing these variables statically in the master program would considerably slow down the simplex pricing.

To speed up convergence of column generation, we have slightly perturbed the costs of weeks and links by attributing marginal costs to day-on and day-off stretches of different lengths, see also Mason and Nielsen [1999]. In the above setting, we have added 0.01 to the costs of weeks and links involving stretches of five days on, 0.02 to four-day stretches and so on. Analoguously, we have attributed increasing costs to shorter day-off stretches. Note that this not only imposes

No.	number of states in forward/backward net	potential number of links
B03	1204	13518
B05	341	4158
C01	1403	15027
C13	491	927
C14	2694	7257
C16	771	2088
C17	196	768
D09	632	2532
D10	682	16038
D12	1049	10761
D13	826	6156
D16	8424	14880
D17	5080	15162
D26	3871	3785
D27	1434	2078
D28	5919	15681

Table 9.5.: Number of forward/backward net states and potential links.

cost perturbations, but should also have a slight effect on preferring longer stretches of days on and off whenever possible.

Branching decisions were taken as follows: If at the root node, the roster week variable W has a fractional value, say f, then we apply the cut $W \geq \lceil f \rceil$. If W is fractional at a deeper search node, we apply a branch on the number of roster weeks as described in Section 9.7. If the roster weeks are integer, we check for subcycles and apply the connectivity branch of Section 9.8 if necessary. For the remaining branches, preliminary experiments have shown that shift transition branches (for week variables) and link branches more quickly yield integer solutions than branches on the shift variables S_{kn}. However, imposing shift branches only after all other branches did not seem to be advantageous either. We have therefore adopted a mixed approach: If any fractionality of at least 0.9 (e.g. 3.9) is found among the shift transitions or links, we create the corresponding roundup branch. Otherwise, we check for shift fractionalities of 0.9 and more before falling back to 0.8 fractionalities. Branches on fractional break variables (see Section 9.7) are only imposed if necessary.

The search tree is traversed in depth-first order until a first connected integer solution is found. Afterwards, we switch to best-first search, i.e. we always choose the search node with the least lower bound value for processing.

We sometimes encountered convergence problems with column generation after applying some branches. We therefore terminate column generation prematurely as soon as the difference between the current objective function value and the exponentially smoothed objective function (with coefficient 0.5) falls below 10^{-6}. In this case, we directly apply the next branch ("early branching", see also Desaulniers et al. [2001]). In order to obtain lower bounds even if early branching is applied, we have used the generalised Lasdon lower bound of Section 6.4. Note that the parameter w_{max} provides a predefined limit on the number of roster weeks and on the number of links as required by the lower bound calculation.

Table 9.6 shows LP relaxation results for test runs on an AMD Athlon 3000+ computer with 2.16 GHz, 1 GB main memory and operating system Windows XP SP1. The left set of columns re-

No.	first LP relaxation			LP relaxation after week cut			gap
	obj. fct.	roster weeks	runtime (secs.)	obj. fct.	roster weeks	runtime (secs.)	
B03	19680.7	43.0	30.88	19680.7	43	30.88	0.00%
B05	12560.8	31.5	4.06	12760.8	32	4.95	1.59%
C01	277742.9	122.7	41.63	277757.8	123	41.64	0.01%
C13	220916.3	122.7	2.42	221049.5	123	2.48	0.06%
C14	12401.1	30.9	17.48	12407.6	31	18.09	0.05%
C16	6404.8	16.1	0.73	6760.2	17	1.22	5.55%
C17	47173.3	99.8	1.83	47199.1	100	1.86	0.05%
D09	102909.6	28.7	4.22	103059.6	29	4.58	0.15%
D10	25600.6	55.9	45.06	25612.0	56	45.11	0.04%
D12	19812.0	43.3	19.22	20052.5	44	25.53	1.21%
D13	11473.9	24.8	13.28	11583.9	25	14.39	0.96%
D16	193318.0	95.6	57.42	193396.8	96	70.63	0.04%
D17	36235.3	90.7	87.78	36361.3	91	109.77	0.35%
D26	31898.4	70.6	11.34	31974.4	71	11.75	0.24%
D27	19296.0	22.7	2.36	19427.5	23	2.72	0.68%
D28	32523.7	71.0	17.17	32523.7	71	17.17	0.00%

Table 9.6.: LP relaxations before and after week cut.

fer to the original LP relaxation, the right set gives information on the LP relaxation after applying the first $\geq$ cut on the number of roster weeks. Note that the original LP relaxation exactly corresponds to the experiments conducted for comparing the different break models (in Section 8.8). It can be seen that solution times for the LP relaxations are generally quite moderate. As expected, scenarios involving more complex column generators (see Table 9.5) generally result in higher runtimes. The gap between the solution values before and after applying the week cut is mostly quite tight.

We have terminated the subsequent branch-and-price search as soon as the gap between the latter lower bound and the best known solution is less than 0.5%. In addition, we have imposed a three hour time limit on all runs. From Table 9.7, we can see that this runtime limit has never been reached. In an average runtime of less than 572 secs., we could find solutions which are within an average of 0.1% of the second lower bound (after applying the week cut). On scenario B03, the first lower bound proves the optimality of the found solution (with regard to the working subset of shift types). The numbers of weeks of the resulting rosters were mostly equal to the supposed optimal values indicated in Table 9.6. The best solution for scenario D09 required one more roster week, scenario C01 even five more roster weeks. Nevertheless, the gap to the first LP relaxation is still very low on these scenarios. This can be explained by the shift cost structure; scenarios C01 and D09 in fact contain some costly night shifts. This means that we may be able to avoid the use of night shifts if more roster weeks are employed. Clearly, this also implies that applying the roster week cut in the root node (instead of branching on fractional week values) has to be regarded a heuristic measure.

While we have used the second lower bound (after applying the week cut) as a termination criterion, we must assess the quality of the generated rosters by comparing the final objective function values to the original lower bounds. From Table 9.7, we can see that these gaps are generally very low and amount to only 0.79% on average. However, the gaps on scenarios B05, D12, D13 and especially C16 are somewhat higher. On these scenarios, we have additionally

No.	obj. fct.	roster weeks	runtime (secs.)	gap first relaxation	gap relaxation after week cut
B03	19680.7	43	36.38	0.00%	0.00%
B05	12800.8	32	7.14	1.91%	0.31%
C01	277989.0	128	82.67	0.09%	0.08%
C13	221052.0	123	3.56	0.06%	0.00%
C14	12408.8	31	2185.86	0.06%	0.01%
C16	6766.3	17	599.52	5.65%	0.09%
C17	47262.5	100	144.97	0.19%	0.13%
D09	103515.0	30	12.20	0.59%	0.44%
D10	25679.3	56	138.00	0.31%	0.26%
D12	20061.1	44	42.33	1.26%	0.04%
D13	11598.4	25	4864.28	1.08%	0.13%
D16	193399.0	96	168.95	0.04%	0.00%
D17	36371.5	91	644.28	0.38%	0.03%
D26	31976.6	71	17.91	0.25%	0.01%
D27	19437.1	23	118.36	0.73%	0.05%
D28	32525.5	71	81.73	0.01%	0.01%

Table 9.7.: Integer solution data.

conducted tests with a full branching strategy, i.e. we have replaced the week roundup cut in the root node by a full branch. Within six hours runtime, we could prove that on all of these scenarios, the lower bound after week roundup is indeed a valid global lower bound. This means that these problems in fact exhibit worse LP lower bounds, and the best solution of these scenarios is in fact within less than 0.5% of the optimal value. This shows that the second lower bound is a good measure for the potential quality of an optimal roster, justifying its use as a termination criterion.

Table 9.8 gives additional details on the experiments. From the comparison of search nodes and the maximum search depth, it can be seen that on ten test cases, the best solutions could be found with a single descent in the search tree. In contrast, we require considerable search effort for scenarios C16, C17 and D13. Table 9.8 also indicates the number of column generation iterations. As the numbers of generated week and link variables show, we often find several variables with identical reduced costs in each iteration which are all added to the restricted master program. In comparison to overall runtimes, the solution times for the subproblems are very moderate.

Table 9.9 indicates the numbers of different branches which are required to arrive at the best known solution. Note that the test cases using roster week branches on the way to the best solution (C01, D09) are exactly the scenarios in which the final number of roster weeks exceeds the value of the second LP relaxation. As can be seen, we have never required break variable branching. The number of subcycle branches gives an indication of the difficulty in arriving at connected solutions. While on many scenarios, only few connectivity branches were needed, scenarios D10 and D13 required as many as 14 and 20 subcycle branches, respectively.

For the evaluation of the weekend-off model of Section 9.9, we have enforced every third weekend off on the above scenarios. Results are given in Table 9.10. As expected, the weekend-off constraint substantially increases the number of required roster weeks. Note that the weekend-off constraint not only cuts the workload coverage on weekends, but also restricts the set of feasible workstretches between two weekends off.

While on most scenarios, the runtime limit of three hours has been the limiting factor, we can see that the algorithm yields results of very high quality. Gaps toward the second lower

No.	search nodes	max. search depth	iterations	week variable generator		link variable generator	
				generated variables	runtime (secs.)	generated variables	runtime (secs.)
B03	40	40	514	1276	1.31	3120	0.30
B05	17	17	382	681	0.22	938	0.08
C01	92	92	1092	2419	3.72	4317	0.31
C13	39	39	189	204	0.11	293	0.02
C14	989	327	13534	44218	57.29	24423	22.73
C16	2692	392	9926	15594	12.45	12135	4.33
C17	3305	92	7717	5659	2.52	9631	0.41
D09	40	39	419	804	0.38	1390	0.06
D10	76	76	1420	2652	2.08	7083	1.36
D12	60	60	732	2117	1.48	4509	0.31
D13	4481	575	37009	72143	57.04	61725	37.17
D16	142	142	1020	3408	18.29	6984	0.81
D17	211	211	1382	5073	14.53	7664	1.17
D26	78	78	504	1076	3.19	1703	0.11
D27	515	167	3999	10173	8.67	6260	0.89
D28	130	130	633	3198	9.57	5313	0.39

Table 9.8.: Runtime data.

No.	integer branches					subcycle branches
	roster weeks	transitions	links	shifts	breaks	
B03	0	27	6	4	0	2
B05	0	13	0	1	0	2
C01	1	75	11	4	0	0
C13	0	31	6	1	0	0
C14	0	28	3	1	0	5
C16	0	23	2	4	0	5
C17	0	18	7	3	0	1
D09	1	24	9	2	0	2
D10	0	51	9	1	0	14
D12	0	47	11	1	0	0
D13	0	62	8	6	0	20
D16	0	115	23	0	0	3
D17	0	138	54	12	0	6
D26	0	68	8	0	0	1
D27	0	21	7	1	0	4
D28	0	86	30	7	0	6

Table 9.9.: Number of branches of different types.

No.	first LP relaxation			LP relaxation after week cut			total	best solution			gap	gap
	obj. fct.	roster weeks	runtime (secs.)	obj. fct.	roster weeks	runtime (secs.)	runtime (secs.)	obj. fct.	roster weeks	runtime (secs.)	first relaxation	relaxation after week cut
B03	601822.7	61.7	67.63	601971.7	62	72.17	10800.00	608281.0	65	6683.53	1.07%	1.05%
B05	460304.5	40.3	9.30	460561.2	41	10.24	10800.00	465921.0	42	2311.56	1.22%	1.16%
C01	1662260.7	124.0	63.75	1662260.7	124	63.75	63.84	1662282.2	124	63.84	0.00%	0.00%
C13	226301.1	136.1	4.31	226650.9	137	4.63	11.45	227069.0	138	11.45	0.34%	0.18%
C14	13760.9	34.5	21.00	13960.9	35	31.42	10800.00	14377.0	36	7616.13	4.48%	2.98%
C16	12053.2	30.2	0.84	12360.9	31	1.02	2112.30	12360.9	31	2112.30	2.55%	0.00%
C17	50980.5	107.1	2.86	51042.2	108	3.00	179.97	51062.5	108	179.97	0.16%	0.04%
D09	420885.4	36.7	4.52	421035.4	37	4.70	10800.00	431440.0	38	59.67	2.51%	2.47%
D10	812019.6	85.0	40.53	812019.6	85	40.53	10800.00	817879.0	87	6625.97	0.72%	0.72%
D12	561514.7	60.7	32.44	561664.3	61	34.27	10800.00	567093.0	62	592.34	0.99%	0.97%
D13	376877.4	38.0	10.39	376877.4	38	10.39	10800.00	399126.0	43	9379.06	5.90%	5.90%
D16	198142.0	108.0	95.27	198142.0	108	95.27	222.39	198142.0	108	222.39	0.00%	0.00%
D17	36483.2	91.0	159.94	36483.4	91	160.73	10800.00	37278.5	93	1724.41	2.18%	2.18%
D26	32643.5	72.8	19.64	32725.2	73	20.19	1301.27	32734.5	73	1301.27	0.28%	0.03%
D27	28558.4	43.2	3.69	28889.6	44	3.84	10800.00	29783.4	46	4046.58	4.29%	3.09%
D28	38667.1	85.4	51.72	38906.4	86	53.05	10800.00	39354.4	87	299.88	1.78%	1.15%

Table 9.10.: Results of runs with weekend-off constraint.

bound (after applying the roster week cut) amount to an average of 1.37% with a maximum of 5.9% on scenario D13. According to our experiments, the quality of the lower bounds seems to be somewhat worse when weekend-off constraints are imposed. While the above results are already very satisfactory, investigations on improving the lower bound quality are part of ongoing research.

9.11. Contributions and Future Research

We have proposed a novel approach for integrated cyclic rostering. While all literature builds upon demand figures given per shift type, the new model integrates the daily shift scheduling problems with the determination of shift positions and days off. The model covers more constraints than any cyclic roster approach before. Reviewing different representations, we have shown that the given setting is most appropriately tackled by a formulation in which decision variables represent roster weeks. The resulting problems of encoding feasibility of week successions was analysed with regard to implicit and explicit approaches. We have proposed a novel integer programming formulation which is based on encoding week transitions by explicit link variables. Both roster weeks and links are generated dynamically within a column generation approach.

While the generation of reduced cost links basically corresponds to enumerating feasible transitions, weeks are generated by solving a two-sided resource-constrained shortest path problem. We have shown how integral solutions can be obtained by constraint branching on fractional shift transitions and static variables of the master program. Special attention has been given to the problem of disconnected solutions. We have developed a specialised subcycle breaking strategy for the week-link encoding scheme. Furthermore, we have demonstrated how final rosters can be built by determining Euler circuits in a roster graph. Different generalisations of the basic model have been described, including the incorporation of weekend-off constraints and blocking factors as well as the creation of multiple rosters.

In an experimental evaluation, we have shown that the algorithm is effective and efficient on real-world planning scenarios. Shift types have been restricted to working subset which not only speeds up calculations, but is also frequently desired by staff planners. For these working subsets, we have shown that the algorithm finds near-optimal solutions in very moderate runtimes. Additionally, we have conducted experiments with weekend-off constraints. While computation times were somewhat higher, solutions could again be proven to be very close to optimality.

Different refinements could be subject of future research. On some problem instances, we have encountered numerical difficulties with column generation. We already employ different methods to tackle these problems, including cost perturbation and early branching. Experiments with the stabilisation method of du Merle et al. [1999] have not yet yielded competitive results, but further investigations may provide more insight into promising techniques.

Brusco [1998] and Brusco and Jacobs [2001] have successfully used the Gomory all-integer cutting plane for the solution of compact integer programming formulations of discontinuous tour scheduling problems. While it is not clear if this method generalises well to column generation, the use of cutting planes may be successful in improving the quality of lower bounds.

Cordeau et al. [2001c] and Cordeau et al. [2001b] have successfully employed Benders' decomposition for models in which two types of variables are generated dynamically. The generation of link variables could therefore be submitted to a reformulation which might render the overall approach still more efficient.

The model itself could also be further generalised. Alternatively to limiting weekly work units, we may use the unit counter to impose restrictions on the number of weekly shift duties. Additionally to creating several cyclic rosters in parallel, we may allow for the creation of parallel non-cyclic rosters or even use a pure shift scheduling model for some worker categories. The idea of junction systems could be generalised in order to encode contiguity of subsequent planning periods in linear rostering.

We believe that further research on implicit modelling is very promising. In this work, we have already described possible applications for representing flexible breaks and for encoding week transitions. A possible future application is cyclic roster generation based on a given shift plan, i.e. with demand figures given per shift type. A more flexible integrative approach should exploit the freedom of prolonging shifts if this makes shifts easier to cover by a roster (e.g. due to working hour constraints). The resulting problem of generating roster shifts which subsume a given set of planned shifts again exhibits the typical transportation structure of implicit models.

10. Summary and Outlook

Takeoffs are optional.
Landings are mandatory.
— saying among pilots

In this thesis, we have conceived models and designed efficient algorithms for ground staff scheduling on airports. It was shown that contributions in the literature only partially fulfil requirements of airport staff planners. Up to now, shift scheduling and rostering models have built upon labour demands given in discrete time periods (demand-level planning). However, workloads in ground handling are basically made up of work tasks at different locations on the airport, introducing a vehicle routing aspect to workforce scheduling (task-level planning). We have argued that depending on the planning context and the detail of available flight information, either task-level or demand-level planning are appropriate. New models and interfaces have been proposed to provide a unifying view on airport staff planning.

Contributions of this work are threefold:

- We have proposed *more integrated models and solution methods*, including two algorithms for task-level shift planning (Chapters 5 and 7), combining shift scheduling with vehicle routing, and the cyclic roster algorithm, solving shift scheduling and rostering problems simultaneously (Chapter 9).
- We have *provided a missing link* between task-level scheduling and demand-based planning when work tasks can be carried out within given intervals, see the workload levelling algorithm of Chapter 4.
- We have designed more complex models and algorithms than in previous contributions *to accommodate for the complexity of real-world ground staff planning*, see e.g. the shift planning application of Chapter 5, the implicit break model of Chapter 8 and the cyclic roster algorithm of Chapter 9.

Apart from investigating novel problems and contributing new models and algorithms, we have provided real-world applications of modern constraint programming and branch-and-price solution techniques. Most recent contributions have been considered and enhanced for the given optimisation problems. On the theoretical side, we have provided complexity results for workload levelling and task-level shift planning.

In Chapter 4, we have tackled the *workload levelling problem* which arises in demand planning of airport ground staff and equipment. Its goal is to avoid unnecessary peaks in workforce demands by placing movable tasks at times of low workloads. Covering work tasks by tours, travel times and time windows are taken into account. While the basis of workload levelling is a vehicle routing problem with time windows, we have shown that the problem is closely related to resource levelling in project scheduling. We have given an $\mathcal{NP}$-hardness proof and conceived a CP-based local improvement algorithm building upon the results of an initial tour minimisation algorithm. The solution method was shown to be effective and efficient on a wide range of real-world planning scenarios. The resulting demand curves provide a suitable basis for demand-level shift planning and rostering and have been used for the tests of Chapters 8 and 9.

Complex task-level shift planning was the subject of Chapter 5. We have developed a mathematical formulation for the shift planning problem, including shift type and break restrictions, qualification and crew constraints and preemptive tasks, to name just a few. It was shown how this complex setting can be incorporated in a constraint programming model. On this basis, we have designed an improvement algorithm which provides a remedy for flaws of an initial construction heuristic. The method was shown to be efficient and robust even on large-scale test cases and strongly constrained scenarios. Especially if shift types allow for sufficient flexibility, CP-based local search yields major improvements which translate into considerable cost savings for ground handling companies.

Both of the algorithms of Chapters 4 and 5 have used a constraint programming based solution technique termed *large neighbourhood search* (LNS). However, motivations for using LNS were quite different. While the workload levelling procedure naturally builds upon the outcome of a tour minimisation algorithm, the local improvement algorithm for shift planning was designed to account for deficiencies of a given construction algorithm.

We have then turned to linear programming based algorithms. As we have introduced basic concepts of constraint programming in Chapter 3, Chapter 6 has given an overview of recent developments in column generation and branch-and-price techniques.

In Chapter 7, we have focused on an important *subclass of task-level shift planning*. It has been shown that even basic shift planning is $\mathcal{NP}$-hard in the strong sense. Using Dantzig-Wolfe decomposition, we have reformulated an initial flow formulation as a column generation model. A problem decomposition method has been proposed to make real-world airport planning scenarios amenable to solution by branch-and-price. The algorithm has been shown to provide optimal or near-optimal results on a set of real-world planning scenarios. It has turned out to be very successful especially on scenarios on which the local search algorithm of Chapter 5 fails to find substantial improvement, making it a complementary solution technique.

In Chapter 8, we have introduced *implicit models* for the consideration of *flexible breaks* in integer programming models for demand-level scheduling. We have shown how an efficient approach from the literature can be extended to be applicable to arbitrary problems. A partitioning algorithm has been proposed to account for so-called extraordinary overlap of break time windows. By empirical comparison with an alternative approach, we have demonstrated that the new formulation can generally be solved more efficiently.

Based on this formulation, we have developed a model and an algorithm for *cyclic rostering* (Chapter 9). Reviewing different modelling alternatives, we have shown that a formulation in which decision variables represent roster weeks is most appropriate for the given setting. The roster-level model has been coupled to the shift scheduling model of Chapter 8, resulting in a formulation for simultaneous shift scheduling and rostering. A novel branch-and-price formulation has been developed, and appropriate column generators and branching strategies have been designed. Tackling more constraints than any approach before, the algorithm has been shown to be effective and efficient on different test cases with quite different characteristics.

All algorithms have been implemented as a part of a commercial software package for airport ground staff scheduling. The contributions of this thesis provide solutions to new scheduling problems and allow for a more integrated view on different planning stages. Being important new elements of the planning system, they have proven to be effective and efficient on real-world settings and have considerably increased customer satisfaction.

Reviewing the solution techniques, constraint programming has shown to be appropriate for the incorporation of complex constraints and non-linear objective functions. In practice, it is also important that constraint-based solution methods are easily extendible to new constraints. Local search has taken advantage of the search space reduction by constraint propagation, using large neighbourhoods to find improvements in strongly constrained problems. Nevertheless, we have also pointed out some limitations of large neighbourhood search (Chapter 5) which are mainly due to the heuristic nature of neighbourhood selection rules. Experiments with LP-based release strategies have been conducted, but pure integer programming approaches have yielded better performance on the problems considered.

Our experiments have confirmed the success of Dantzig-Wolfe decomposition and column generation techniques for scheduling problems. Linear programming frequently yields very tight lower bounds, making formulations particularly appropriate for branch-and-bound solution methods. Integer programming models have shown to provide a very good view on global optimality. Clearly, integer programming models are more limited with regard to the incorporation of many constraints. Furthermore, large problems sometimes have to be heuristically reduced in order to reduce the search space. Even if this can frequently be done without sacrificing solution quality, the proposed integer programming algorithms are more sensitive to characteristics of the given problems and require careful parameter tuning.

One of the most promising subjects of future research is the application of hybrid constraint programming/integer programming methods to the aforementioned problems. First contributions have e.g. been provided by Focacci et al. [1998] and Focacci et al. [1999] ("cost-based domain filtering") as well as Fahle et al. [2002] and Rousseau et al. [2002] ("constraint-based column generation"). The basic idea of these techniques is to represent complex constraint in a constraint programming model while linear programming guides the search. Especially task-level shift planning should provide an interesting field of application.

Further investigations could aim at analysing interactions between different planning stages. As an example, we may examine the relationship between demand-level and task-level scheduling in more detail. This may include models and algorithms for creating demand curves which explicitly aim at approximating task-level planning problems or providing lower bounds.

However, one of the most important fields of future research should be the robustness of plans with regard to flight delays and other interruptions. Statistics on flight delays are e.g. available from the Association of European Airlines (AEA) and the Central Office for Delay Analysis (CODA) of Eurocontrol. As an example, more than 20% of the intra-European flight departures were delayed by more than 15 minutes in the third quarter of 2004 [AEA, 2004]. Other sources of uncertainty may also be important to consider, e.g. employee illness [Dowling et al., 1997]. In the practice of ground staff planning, such problems are usually tackled by ad-hoc and best-practice methods on the day of operations, e.g. by using overtime and standby staff [Stern and Hersh, 1980] [Dowling et al., 1997]. Research on explicitly considering uncertainty in the planning stage is still at its beginning, see e.g. Bolat [2000] and Yan et al. [2002] for applications in aircraft stand assignment.

Depending on the causes of uncertainty and the type of available information, different theories will be suitable for modelling uncertainty [Zimmermann, 2000]. We are currently investigating the robustness of given task-level shift plans by simulation analysis, using statistical flight delay data [Sørensen and Clausen, 2002]. Furthermore, we have conducted first experiments with a specialisation of the local improvement algorithm of Chapter 5, distributing gaps between the tasks evenly over the shift plan. In general, this will make conflicts due to flight delays less likely [Bolat, 2000]. To make demand-level plans more robust, it may be helpful to consider minimum and maximum workforce demand profiles under different realisations of task start times, see the *resource envelope* approach of Muscettola [2002].

A. Mathematical Symbols

Sets

$\mathbb{N}$	natural numbers
$\mathbb{N}_0$	natural numbers including 0
$\mathbb{R}$	real numbers
$\mathbb{R}_+$	nonnegative real numbers
$\mathbb{Z}$	integer numbers

Tasks and shifts

$i \in I$	tasks
$t \in T$	time periods of the planning horizon
$n \in N$	days of the planning horizon
$[a_i, b_i] \subseteq T$	start time window of task i
$l_i \in \mathbb{N}$	length of task i
$d_{i,j} \in \mathbb{N}$	travel time between tasks i and j
$s \in S$	shifts

Shift types and breaks

$k \in K$	shift types
$n \in N_k$	valid days of shift type k
$st_{kn}, et_{kn} \in \mathbb{N}$	start/end time of the realisation of shift type k on day n
$c_k \in \mathbb{R}^+$	cost of shift type k
$br_k^{eb}, br_k^{mb}, br_k^{lb} \in BR$	early, main and late break rule of shift type k
$l_{br} \in \mathbb{N}$	length of break defined by break rule br
$bb_{br}^{eb}, bb_{br}^{lb} \in \mathbb{N}$	early/late buffer defined by break rule br
$[est_{kn}^b, lst_{kn}^b] \subseteq T$	break start time window of shift (k, n)

Shift number restrictions

$r \in R_{abs}^{min} \cup R_{abs}^{max}$	absolute minimum/maximum shift number restrictions
$r \in R_{rel}^{min} \cup R_{rel}^{max}$	relative minimum/maximum shift number restrictions
$K_r \subseteq K$	reference shift type set for shift number restriction r
$N_r \subseteq N$	reference days of shift number restriction r
$m_r \in \mathbb{N}^0$	limit for absolute shift number restriction r
$p_r \in \mathbb{R}^+$	limit for relative shift number restriction r

Graphs

$G = (V, E)$	graph
$v, w \in V$	vertices
E	edge set
$\delta^-(v) \subseteq E$	inedges of v
$\delta^+(v) \subseteq E$	outedges of v

Workload levelling

$r \in R$	tours
$C \in \mathcal{C}$	crews
I^o, I^d	origin and destination tasks
$d_i^{min} \in \mathbb{N}$	minimum travel time before task i
$R_i \in R$	tour of task i
$P_i \in I$	predecessor of task i
$S_i \in I$	successor of task i
$T_i \in T$	start time of task i
$W(t) \in \mathbb{N}$	workload at time t
$I^{rel} \subseteq I$	set of relaxed tasks
$\Theta(i) \in I \times I$	insertion positions of task i
$\delta_i^{start} = [\alpha_i, \beta_i]$	start time domain of task i
δ_i^{ins}	boolean “inserted” domain of task i
δ_i^{pred}	predecessor domain of task i
δ_i^{ip}	insertion position domain of task i
δ_i^{tour}	tour domain of task i
π_i	current predecessor of task i
σ_i	current successor of task i
$D \in \mathbb{N}^0$	discrepancy
$L_D \in \mathbb{N}^0$	discrepancy limit

Shift planning local improvement

$al(i) \in \mathbb{N}$	actual length of task i
$i_s^o \in I^o$	origin tasks for the shifts
$i_s^d \in I^d$	destination tasks for the shifts
$i_s^{eb}, i_s^{mb}, i_s^{lb} \in I^b$	early, main, late break task of shift s
$C \in \mathcal{C}$	task crews
$B \in \mathcal{B}$	subcrews
$cs \in \mathbb{N}$	crew size
$H \in \mathcal{H}$	shift crews
$P \in \mathcal{P}$	split tasks
$l_P \in \mathbb{N}$	total length of split task P
$msl_i, msl_P \in \mathbb{N}$	minimum split length for task i/split task P
$P_i \in \mathcal{P}$	split task of split part i
i_{pseudo}^P	pseudo split part of split task P
$spred_i, ssucc_i \in I$	split part predecessor/successor of split part i
$tol_i^{max} \in \mathbb{N}$	maximum overlap for (non-split) task i
$ol_i^{max} \in \mathbb{N}$	derived maximum overlap for task i
Q	set of all qualifications
$Q_i \subseteq Q$	qualification requirements of task i
$Q_s \subseteq Q$	qualification requirements of shift s
$q^{max} \in \mathbb{N}$	maximum number of qualifications per shift
$qp(Q') \in \mathbb{R}^+$	qualification preference for set Q'

$qp_{min} \in \mathbb{R}^+ \setminus \{0\}$	minimum positive qualification preference
$qr \in QR$	qualification restrictions
$q_{qr} \in Q$	reference qualification for restriction qr
$m_{qr} \in \mathbb{N}$	limit for restriction qr
$T_i \in T$	start time of task i
$S_i \in S$	shift of task i
$pred_i, succ_i$	predecessor/successor of task i
$K_s \in K$	shift type of shift s
$N_s \in N$	day of shift s
$w^{QP} \in \mathbb{R}^+$	qualification penalty weight
$M^{QR} \in \mathbb{R}^+$	penalty for violating qualification restrictions
$M^{SNR} \in \mathbb{R}^+$	penalty for violating shift number restrictions
$M^{UT} \in \mathbb{R}^+$	penalty for unassigned task minutes
$brd \in BRD$	break rule days
$n_{brd} \in N$	day of break rule day brd
$BR_{brd} \subseteq BR$	set of break rules of break rule day brd
$K_{brd} \in K$	shift type of break rule day brd
s_i^{dummy}	dummy shift for unassigned task i
k_l^{dummy}	pseudo shift type for task length l
brd^{dummy}	pseudo break rule day
$\delta_i^{start} = [\alpha_i, \beta_i]$	start time domain for task i
δ_i^{sl}	split part length domain for split part i
π_i, σ_i	predecessor, successor of task i
δ_i^{shift}	shift domain of task i
δ_i^{pred}	predecessor domain of task i
δ_i^{ins}	“inserted” domain of task i
δ_s^{K}	shift type domain of task i
δ_s^{used}	“used” domain of task i
δ_s^{Q}	qualificaton domain of task i
δ_s^{qp}	qualification penalty domain of task i
$I^{rel} \subseteq I$	tasks to be relaxed
$I^{dep} \subseteq I$	dependent tasks
$S^{dep} \subseteq S$	dependent shifts

Shift planning by branch-and-price

$q \in \mathbb{N}$	number of variables
$p \in \mathbb{N}$	number of clauses
$U = \{u_0, \ldots, u_{q-1}\}$	boolean variables
$\mathcal{C} = \{C_0, \ldots, C_{p-1}\}$	clauses
$tr : U \rightarrow \{\text{false}, \text{true}\}$	truth assignment
I^{lit}	literal tasks
I^{fill}	filling tasks
I^{cl}	clause tasks
K^T	T (true) shift types
K^F	F (false) shift types

K^{fill}	filling shift types
i_o^k	origin task of shift type k
i_d^k	destination task of shift type k
i_b^k	break task of shift type k
$G^k = (V^k, E^k)$	flow graph for shift type k
v_o^k	origin node of shift type k
v_d^k	destination node of shift type k
V_b^k	break nodes of shift type k
c_{vw}^k	cost of edge (v, w) of graph G^k
X_{vw}^k	flow on edge (v, w) of graph G^k
$p \in \Omega^k$	extreme rays for subproblem k
Ω	set of all extreme rays
x_{vwp}^k	edge flow on (v, w) for ray p of subproblem k
λ_p^k	decision variable for ray p of subproblem k
$c_p^k \in \mathbb{R}^+$	cost coefficient of ray p of subproblem k
$g_{ip}^k \in \mathbb{N}$	coverage coefficient for constraint i of ray p of subproblem k
π_i	dual value for coverage constraint i
μ_r	dual value for shift number restriction r
$\bar{c}_p^k$	reduced cost of ray p of subproblem k
$\rho_r(k)$	indicator function for shift number restriction r
$\bar{c}^k$	derived reduced cost associated with shift type k
$\bar{c}_{vw}^k$	derived reduced cost of edge (v, w) of subproblem k

Implicit break modelling

$l \in L$	break indices
$r \in R$	shift/break classes
L^{est}, L_r^{est}	set of earliest start times over all break windows (in class r)
L^{lst}, L_r^{lst}	set of latest start times over all break windows (in class r)
l^{est}, l_r^{est}	earliest overall break (in class r)
l^{lst}, l_r^{lst}	latest overall break (in class r)
$L^B(l)$, $L_r^B(l)$	set of breaks (in class r) starting no earlier than l
$K^B(l)$, $K_r^B(l)$	set of shifts (in class r) associated with $L^B(l)$
$L^F(l)$, $L_r^F(l)$	set of breaks (in class r) starting no later than l
$K^F(l)$, $K_r^F(l)$	set of shifts (in class r) associated with $L^F(l)$
$d_t \in \mathbb{N}^0$	workforce demand in period t
$a_{tkn} \in \mathbb{N}^0$	coverage coefficient for shift (k, n) in period t
$b_{tl} \in \mathbb{N}^0$	coefficient for break l in period t
$c^{sht}, c_t^{sht} \in \mathbb{R}^+$	shortage cost (for demand period t)

Cyclic rosters

$[w_{min}, w_{max}] \subseteq \mathbb{N}^0$	limits on the number of roster weeks
$G_k \in K$	valid day-on predecessor shift types of shift type k
$H_k \in K$	valid day-off predecessor shift types of shift type k
$\tau_{k_1,k_2} \in \mathbb{R}^+$	shift transition penalty between k_1 and k_2
$[m_{min}^{on}, m_{max}^{on}] \subseteq \mathbb{N}$	limits on the number of consecutive days on
$[m_{min}^{off}, m_{max}^{off}] \subseteq \mathbb{N}$	limits on the number of consecutive days off
$u_k \in \mathbb{N}$	work units of shift type k

$[u_{min}, u_{max}] \subseteq \mathbb{N}^0$	limits on the number of work units per week
$u_{avg} \in \mathbb{N}$	average work units per week
$u_{tol} \in \mathbb{N}$	tolerance for average work units per week
I	index set for week variables
J	index set for link variables
ϑ_i	number of weeks of type i
λ_j	number of links of type j
$c_i^w \in \mathbb{R}^+$	cost of week i
$c_j^l \in \mathbb{R}^+$	cost of link j
$u_i \in \mathbb{N}$	work units of week i
$s_i^{kn} \in \{0, 1\}$	shift coverage coefficient of week i for shift (k, n)
$m \in \mathbb{N}$	number of consecutive days on/off
$ws_i^{km,on}, ws_i^{km,off}$	coefficients for start junction (k, m) of week i
$we_i^{km,on}, we_i^{km,off}$	coefficients for end junction (k, m) of week i
$ls_j^{km,on}, ls_j^{km,off}$	coefficients for start junction (k, m) of link j
$le_j^{km,on}, le_j^{km,off}$	coefficients for end junction (k, m) of link j
$\gamma_{k,m}^{on}, \gamma_{k,m}^{off}$	dual prices associated with week start/link end constraints
$\eta_{k,m}^{on}, \eta_{k,m}^{off}$	dual prices associated with week end/link start constraints
$\pi_{k,n}$	dual price associated with shift equality constraints
ω	dual price associated with work units
ϕ	dual price associated with roster week constraint
$P(k, n, m, u)$	value of backward net state for shift (k, n), m remaining days on and u work units
$Q(k, n, m, u)$	value of forward net state for shift (k, n), m days on and u work units

B. Acronyms and Abbreviations

B.1. Acronyms

BCP	branch, cut and price
COIN-OR	common optimisation interface for Operations Research
COP	constraint optimisation problem
CP	constraint programming
CSP	constraint satisfaction problem
CVSP	capacitated vehicle scheduling problem
DAG	directed acyclic graph
DP	dynamic programming
EO	extraordinary overlap
GF	greedy fixing
IP	integer programming
LDS	limited discrepancy search
LNS	large neighbourhood search
LP	linear programming
MDVSP	multiple-depot vehicle scheduling problem
OSI	open solver interface
RCSP	resource-constrained shortest path problem
RLDP	repeated local dynamic programming
RMP	restricted master program
SA	simulated annealing
SFDP	successive fixing by dynamic programming
TSP	travelling salesman problem
TSPTW	travelling salesman problem with time windows
VRP	vehicle routing problem
VRPTW	vehicle routing problem with time windows
VSP	vehicle scheduling problem
WLP	workload levelling problem

B.2. Abbreviations

abs.	absolute
avg.	average
cf.	confer
e.g.	for example (exempli gratia)
etc.	et cetera
fig.	figure
i.e.	that is (id est)
iff	if and only if
min.	minimum

max.	maximum
No.	number
obj. fct.	objective function
p.	page
pref.	preference
qual.	qualification
rel.	relative
secs.	seconds
std. dev.	standard deviation
w.l.o.g.	without loss of generality

C. Overview of Workforce Scheduling Literature

In the following, an overview of the relevant literature cited in Chapter 2 will be given. Publications will be categorised by model types, solution techniques, named and anonymous models as well as application areas.

C.1. Overview by Models

Table C.1 summarises references with regard to different models, including shift, day-off and tour scheduling, shift assignment and cyclic rostering. Special nurse scheduling models which often do not fall into these categories are mentioned separately. Within each section, publications are listed chronologically.

It should be noted that not all publications can be categorised without ambiguities. As an example, some of the day-off approaches refer to cyclic rosters, e.g. Emmons [1985]. Furthermore, Bennett and Potts [1968] which is a classical reference on cyclic rosters effectively uses a sequential day-off and shift assignment approach. Similarly, all shift assignment references except for Jackson et al. [1997] refer to cyclic rosters. If publications explicitly refer to more than one model type, these are listed more than once, e.g. Millar and Kiragu [1998]. Bartholdi et al. [1980], Bartholdi III [1981], Karp and Orlin [1981] and Vohra [1988] treat a class of cyclic staffing problems. While these also cover day-off scheduling models, their focus is on shift scheduling to which we have attributed these publications.

C.2. Overview by Solution Methods

In Table C.2, we categorise workforce scheduling publications by solution approaches. Overviews of parts of these references can also be found in Bechtold et al. [1991], Thompson [1992], Brusco and Jacobs [1993b], Thompson [1993] and Thompson [1995]. Ernst et al. [2004] mention that the literature is heavily skewed towards mathematical programming and metaheuristic approaches. However, the employed techniques also dependend on the application and setting. While e.g. no constraint programming approaches have been proposed for anonymous scheduling, CP seems to be appropriate for more constrained models in named nurse scheduling.

While Table C.2 separately lists greedy optimal algorithms, it should be noted that combinatorial approaches also belong to the class of polynomial-time optimal algorithms, see Section 2.3. Integer programming models comprise some column generation (Easton and Rossin [1991], Panton and Ryan [1999]) and branch-and-price approaches (Mason and Smith [1998], Mason [1999], Mason and Nielsen [1999], Mehrotra et al. [2000]). References for graph algorithms comprise publications using several or mixed approaches (Lagrangian relaxation in Balakrishnan and Wong [1990], heuristics in Panton [1991] and van den Berg and Panton [1994], and simulation in Mason et al. [1998]). “Other solution approaches” in Table C.2 refer to the additive algorithm of Balas [Musa and Saxena, 1984], goal programming [Ozkarahan and Bailey, 1988] and a half-automatic constraint-based approach [Muslija et al., 2000].

shift scheduling	**day-off scheduling**	**tour scheduling**
Dantzig [1954]	Tibrewala et al. [1972]	Ritzman et al. [1976]
Bennett and Potts [1968]	Baker [1974]	McGinnis et al. [1978]
Segal [1972]	Brownell and Lowerre [1976]	Mabert and Watts [1982]
Buffa et al. [1976]	Miller et al. [1976]	Morris and Showalter [1983]
Henderson and Berry [1976]	Baker and Magazine [1977]	Glover et al. [1984]
Moondra [1976]	Lowerre [1977]	Bailey [1985]
Gaballa and Pearce [1979]	Mabert and Raedels [1977]	Glover and McMillan [1986]
Keith [1979]	Baker et al. [1979]	Holloran and Byrn [1986]
Bartholdi et al. [1980]	Burns and Carter [1985]	Bechtold and Showalter [1987]
Stern and Hersh [1980]	Emmons [1985]	Showalter and Mabert [1988]
Bartholdi III [1981]	Burns and Koop [1987]	Andrews and Parsons [1989]
Karp and Orlin [1981]	Bechtold [1988]	Taylor and Huxley [1989]
Bailey and Field [1985]	Koop [1988]	Bechtold et al. [1991]
Vohra [1988]	Emmons and Burns [1991]	Easton and Rossin [1991]
Bechtold and Jacobs [1990]	Hung [1993]	Li et al. [1991]
Love and Hoey [1990]	Hung [1994a]	Loucks and Jacobs [1991]
Thompson [1990]	Hung [1994b]	Brusco and Jacobs [1993b]
Thompson [1992]	Emmons and Fuh [1997]	Brusco and Jacobs [1993a]
Schindler and Semmel [1993]	Burns and Narasimhan [1999]	Jacobs and Bechtold [1993]
Thompson [1995]		Thompson [1993]
Thompson [1996a]	**shift assignment**	Bechtold and Brusco [1994a]
Thompson [1996b]	Lau [1994]	Brusco et al. [1995]
Aykin [1998]	van den Berg and Panton [1994]	Brusco and Jacobs [1995]
Nobert and Roy [1998]	Lau [1996a]	Brusco and Johns [1995]
Panton and Ryan [1999]	Lau [1996b]	Brusco and Johns [1996]
Aykin [2000]	Jackson et al. [1997]	Jacobs and Brusco [1996]
Mehrotra et al. [2000]		Dowling et al. [1997]
Çezik and Günlük [2002]	**cyclic rosters**	Brusco [1998]
Sørensen and Clausen [2002]	Bennett and Potts [1968]	Brusco and Jacobs [1998a]
Musliu et al. [2004]	Laporte et al. [1980]	Brusco and Jacobs [1998b]
	Rosenbloom and Goertzen [1987]	Henderson and Mason [1998]
named scheduling for nurses/physicians	Balakrishnan and Wong [1990]	Jaumard et al. [1998]
	Chew [1991]	Mason and Smith [1998]
Miller et al. [1976]	Panton [1991]	Millar and Kiragu [1998]
Warner [1976]	Khoong and Lau [1992]	Topaloglu and Ozkarahan [1998]
Arthur and Ravindran [1981]	Mason et al. [1998]	Alvarez-Valdez et al. [1999]
Musa and Saxena [1984]	Millar and Kiragu [1998]	Mason and Nielsen [1999]
Ozkarahan and Bailey [1988]	Mason [1999]	Brusco and Jacobs [2000]
Weil et al. [1995]	Muslija et al. [2000]	Brusco and Jacobs [2001]
Berrada et al. [1996]	Çezik et al. [2001]	Çezik et al. [2001]
Cheng et al. [1997]	Felici and Gentile [2004]	Çezik and Günlük [2002]
Meisels et al. [1997]		Rekik et al. [2003]
Dowsland [1998]		
Schaerf and Meisels [1999]		
Carter and Lapierre [2001]		
Bellanti et al. [2004]		

Table C.1.: Overview of publications on different models.

integer programming	**LP/rounding**	**combinatorial approaches**
Bennett and Potts [1968]	Dantzig [1954]	Tibrewala et al. [1972]
Warner [1976]	Henderson and Berry [1976]	Baker [1974]
Mabert and Raedels [1977]	Moondra [1976]	Brownell and Lowerre [1976]
Gaballa and Pearce [1979]	Keith [1979]	Baker and Magazine [1977]
Laporte et al. [1980]	Bartholdi et al. [1980]	Lowerre [1977]
Stern and Hersh [1980]	Bartholdi III [1981]	Burns [1978]
Bailey [1985]	Morris and Showalter [1983]	Baker et al. [1979]
Rosenbloom and Goertzen [1987]	Bailey and Field [1985]	Bechtold [1981]
Bechtold and Jacobs [1990]	Holloran and Byrn [1986]	Burns and Carter [1985]
Easton and Rossin [1991]	Li et al. [1991]	Emmons [1985]
Thompson [1992]	Thompson [1990]	Burns and Koop [1987]
Jacobs and Bechtold [1993]	Bechtold et al. [1991]	Bechtold [1988]
Schindler and Semmel [1993]	Thompson [1993]	Koop [1988]
Thompson [1995]	Bechtold and Brusco [1994a]	Emmons and Burns [1991]
Brusco and Johns [1996]		Hung [1993]
Jacobs and Brusco [1996]	**simulated annealing**	Hung [1994a]
Thompson [1996a]	Brusco and Jacobs [1993b]	Hung [1994b]
Aykin [1998]	Brusco and Jacobs [1993a]	Emmons and Fuh [1997]
Brusco [1998]	Brusco et al. [1995]	Burns and Narasimhan [1999]
Brusco and Jacobs [1998a]	Brusco and Jacobs [1995]	
Henderson and Mason [1998]	Brusco and Johns [1995]	**other greedy**
Mason and Smith [1998]	Thompson [1996b]	**optimal approaches**
Nobert and Roy [1998]	Dowling et al. [1997]	Vohra [1988]
Topaloglu and Ozkarahan [1998]	Brusco and Jacobs [1998b]	Chew [1991]
Mason [1999]	Sørensen and Clausen [2002]	
Mason and Nielsen [1999]		**heuristics**
Panton and Ryan [1999]	**tabu search**	Buffa et al. [1976]
Aykin [2000]	Glover et al. [1984]	Miller et al. [1976]
Brusco and Jacobs [2000]	Glover and McMillan [1986]	Ritzman et al. [1976]
Mehrotra et al. [2000]	Berrada et al. [1996]	McGinnis et al. [1978]
Brusco and Jacobs [2001]	Jackson et al. [1997]	Arthur and Ravindran [1981]
Çezik et al. [2001]	Dowsland [1998]	Bechtold and Showalter [1987]
Çezik and Günlük [2002]	Alvarez-Valdez et al. [1999]	Taylor and Huxley [1989]
Rekik et al. [2003]	Carter and Lapierre [2001]	Loucks and Jacobs [1991]
Felici and Gentile [2004]	Bellanti et al. [2004]	Khoong and Lau [1992]
	Musliu et al. [2004]	Lau [1994]
graph algorithms,		Lau [1996a]
network flows	**other local search approaches**	Lau [1996b]
Segal [1972]	Miller et al. [1976]	
Bartholdi et al. [1980]	Schaerf and Meisels [1999]	**other approaches**
Karp and Orlin [1981]		Musa and Saxena [1984]
Balakrishnan and Wong [1990]	**constraint programming**	Ozkarahan and Bailey [1988]
Love and Hoey [1990]	Weil et al. [1995]	Muslija et al. [2000]
Panton [1991]	Cheng et al. [1997]	
van den Berg and Panton [1994]	Meisels et al. [1997]	
Mason et al. [1998]		

Table C.2.: Overview of solution techniques.

C.3. Overview by Anonymous and Named Models

While anonymous planning refers to staff scheduling on an aggregate level, named models attribute shifts and tours to employees. Named models can incorporate skills, availabilities as well as preferences. Cyclic models as well as combinatorial approaches belong to the class of anonymous models. An overview can be found in Table C.3.

C.4. Overview by Application Areas

Table C.4 lists references by application areas if these are explicitly mentioned. Again, there are interdependencies with the aforementioned categories. As an example, fast-food and nurse scheduling models generally belong to the class of named models.

anonymous scheduling	**(continued)**	**named scheduling**
Dantzig [1954]	Brusco and Jacobs [1993b]	Miller et al. [1976]
Bennett and Potts [1968]	Hung [1993]	Warner [1976]
Segal [1972]	Jacobs and Bechtold [1993]	Arthur and Ravindran [1981]
Tibrewala et al. [1972]	Schindler and Semmel [1993]	Glover et al. [1984]
Baker [1974]	Thompson [1993]	Glover and McMillan [1986]
Baker [1976]	Bechtold and Brusco [1994b]	Thompson [1990]
Brownell and Lowerre [1976]	Bechtold and Brusco [1994a]	Weil et al. [1995]
Henderson and Berry [1976]	Hung [1994a]	Berrada et al. [1996]
Moondra [1976]	Hung [1994b]	Cheng et al. [1997]
Ritzman et al. [1976]	Jarrah et al. [1994]	Jackson et al. [1997]
Baker and Magazine [1977]	Lau [1994]	Meisels et al. [1997]
Lowerre [1977]	van den Berg and Panton [1994]	Dowsland [1998]
Mabert and Raedels [1977]	Brusco et al. [1995]	Jaumard et al. [1998]
McGinnis et al. [1978]	Brusco and Jacobs [1995]	Mason and Smith [1998]
Baker et al. [1979]	Brusco and Johns [1995]	Mason and Nielsen [1999]
Gaballa and Pearce [1979]	Thompson [1995]	Schaerf and Meisels [1999]
Keith [1979]	Aykin [1996]	Carter and Lapierre [2001]
Bartholdi et al. [1980]	Bechtold and Jacobs [1996]	Bellanti et al. [2004]
Laporte et al. [1980]	Brusco and Johns [1996]	
Stern and Hersh [1980]	Jacobs and Brusco [1996]	**anonymous scheduling and**
Bartholdi III [1981]	Lau [1996a]	**subsequent assignment**
Mabert and Watts [1982]	Lau [1996b]	Buffa et al. [1976]
Tien and Kamiyama [1982]	Thompson [1996a]	Bailey and Field [1985]
Morris and Showalter [1983]	Thompson [1996b]	Love and Hoey [1990]
Musa and Saxena [1984]	Dowling et al. [1997]	Alvarez-Valdez et al. [1999]
Bailey [1985]	Emmons and Fuh [1997]	
Burns and Carter [1985]	Brusco [1998]	**named and**
Emmons [1985]	Brusco and Jacobs [1998a]	**anonymous scheduling**
Holloran and Byrn [1986]	Brusco and Jacobs [1998b]	Khoong and Lau [1992]
Burns and Koop [1987]	Henderson and Mason [1998]	Khoong et al. [1994]
Bechtold and Showalter [1987]	Mason et al. [1998]	
Rosenbloom and Goertzen [1987]	Millar and Kiragu [1998]	
Bechtold [1988]	Nobert and Roy [1998]	
Koop [1988]	Topaloglu and Ozkarahan [1998]	
Ozkarahan and Bailey [1988]	Burns and Narasimhan [1999]	
Vohra [1988]	Mason [1999]	
Andrews and Parsons [1989]	Panton and Ryan [1999]	
Taylor and Huxley [1989]	Aykin [2000]	
Balakrishnan and Wong [1990]	Brusco and Jacobs [2000]	
Bechtold and Jacobs [1990]	Mehrotra et al. [2000]	
Bechtold et al. [1991]	Muslija et al. [2000]	
Chew [1991]	Brusco and Jacobs [2001]	
Easton and Rossin [1991]	Çezik et al. [2001]	
Emmons and Burns [1991]	Çezik and Günlük [2002]	
Li et al. [1991]	Sørensen and Clausen [2002]	
Loucks and Jacobs [1991]	Rekik et al. [2003]	
Panton [1991]	Felici and Gentile [2004]	
Thompson [1992]	Musliu et al. [2004]	
Brusco and Jacobs [1993b]		

Table C.3.: Overview of publications on anonymous and named scheduling.

airport ground handling	**bus drivers**	**bank applications**
Chew [1991]	Bennett and Potts [1968]	Moondra [1976]
Khoong and Lau [1992]		Mabert and Raedels [1977]
Schindler and Semmel [1993]	**call centres**	Mabert and Watts [1982]
Khoong et al. [1994]	Andrews and Parsons [1989]	Li et al. [1991]
Brusco et al. [1995]	Henderson and Mason [1998]	
Dowling et al. [1997]	Mason and Nielsen [1999]	**physician scheduling**
Brusco and Jacobs [1998a]	Brusco and Jacobs [2000]	Carter and Lapierre [2001]
Brusco and Jacobs [1998b]	Brusco and Jacobs [2001]	
Sørensen and Clausen [2002]	Çezik et al. [2001]	**nurse scheduling**
Felici and Gentile [2004]		Miller et al. [1976]
	telephone operators	Warner [1976]
aircraft cleaning	Segal [1972]	Arthur and Ravindran [1981]
Stern and Hersh [1980]	Buffa et al. [1976]	Musa and Saxena [1984]
	Henderson and Berry [1976]	Rosenbloom and Goertzen [1987]
aircraft refuelling	McGinnis et al. [1978]	Ozkarahan and Bailey [1988]
Alvarez-Valdez et al. [1999]	Keith [1979]	Khoong and Lau [1992]
		Khoong et al. [1994]
airport passenger services	**toll collectors**	Weil et al. [1995]
Schindler and Semmel [1993]	Dantzig [1954]	Berrada et al. [1996]
Brusco et al. [1995]	Bennett and Potts [1968]	Cheng et al. [1997]
	Jacobs and Brusco [1996]	Meisels et al. [1997]
airline sales reservation offices		Dowsland [1998]
Gaballa and Pearce [1979]	**fast-food restaurants**	Jaumard et al. [1998]
Holloran and Byrn [1986]	Glover and McMillan [1986]	Mason and Smith [1998]
	Love and Hoey [1990]	Millar and Kiragu [1998]
airport Customs/immigration authority	Loucks and Jacobs [1991]	Mason and Nielsen [1999]
Mason et al. [1998]	Thompson [1996a]	Schaerf and Meisels [1999]
Mason and Nielsen [1999]		Bellanti et al. [2004]
	mail facility	
air cargo terminal	Ritzman et al. [1976]	**police patrol officers**
Nobert and Roy [1998]	Jarrah et al. [1994]	Taylor and Huxley [1989]
		Casino security officers
		Panton [1991]

Table C.4.: Overview of applications.

Bibliography

J. Adams, E. Balas and D. Zawack. The shifting bottleneck procedure for job shop scheduling. *Management Science*, 34:391–401, 1988.

AEA, 2004. *European Airline Delays in 3rd Quarter 2004*. Association of European Airlines, 2004. http://www.aea.be.

H. N. Ahuja. *Construction Performance Control by Networks*. John Wiley & Sons, 1976.

R. K. Ahuja, T. L. Magnanti and J. B. Orlin. *Network Flows – Theory, Algorithms, and Applications*. Prentice Hall, 1993.

A. Allahverdi, J. N. D. Gupta and T. Aldowaisan. A review of scheduling research involving setup considerations. *Omega*, 27:219–239, 1999.

R. Alvarez-Valdez, E. Crespo and J. Tamarit. Labour scheduling at an airport refuelling installation. *Journal of the Operational Research Society*, 50:211–218, 1999.

R. Anbil, R. Tanga and E. L. Johnson. A global approach to crew-pairing optimization. *IBM Systems Journal*, 31:71–78, 1992.

B. H. Andrews and H. L. Parsons. L. L. Bean chooses a telephone agent scheduling system. *Interfaces*, 19:1–9, 1989.

L. H. Appelgren. A column generation algorithm for a ship scheduling problem. *Transportation Science*, 3:53–68, 1969.

D. Applegate and W. Cook. A computational study of the job-shop scheduling problem. *ORSA Journal on Computing*, 3:149–156, 1991.

J. L. Arthur and A. Ravindran. A multiple objective nurse scheduling model. *AIIE Transactions*, 13:55–60, 1981.

N. Ashford, H. P. M. Stanton and C. A. Moore. *Airport Operations, 2nd edition*. McGraw-Hill, 1997.

A. A. Assad. Multicommodity network flows: A survey. *Networks*, 8:37–91, 1978.

T. Aykin. Optimal shift scheduling with multiple break windows. *Management Science*, 42:591–602, 1996.

T. Aykin. A composite branch and cut algorithm for optimal shift scheduling with multiple breaks and break windows. *Journal of the Operational Research Society*, 49:603–615, 1998.

T. Aykin. A comparative evaluation of modeling approaches to the labor shift scheduling problem. *European Journal of Operational Research*, 125:381–397, 2000.

J. Bailey. Integrated days off and shift personnel scheduling. *Computer and Industrial Engineering*, 9:395–404, 1985.

J. Bailey, H. Alfares and W. Y. Lin. Optimization and heuristic models to integrate project task and manpower scheduling. *Computers and Industrial Engineering*, 29:473–476, 1995.

J. Bailey and J. Field. Personnel scheduling with flexshift models. *Journal of Operations Management*, 5:327–338, 1985.

K. R. Baker. Scheduling full-time and part-time staff to meet cyclic requirements. *Operational Research Quarterly*, 25:65–76, 1974.

K. R. Baker. Workforce allocation in cyclical scheduling problems: A survey. *Operational Research Quarterly*, 27:155–167, 1976.

K. R. Baker, R. N. Burns and M. Carter. Staff scheduling with day-off and workstretch constraints. *AIIE Transactions*, 11:286–292, 1979.

K. R. Baker and M. J. Magazine. Workforce scheduling with cyclic demands and day-off constraints. *Management Science*, 24:161–167, 1977.

N. Balakrishnan and R. T. Wong. A network model for the rotating workforce scheduling problem. *Networks*, 20:25–42, 1990.

E. Balas. An additive algorithm for solving linear programs with zero-one variables. *Operations Research*, 13:517–546, 1965.

M. Bandelloni, M. Tucci and R. Rinaldi. Optimal resource leveling using non-serial dynamic programming. *European Journal of Operational Research*, 78:162–177, 1994.

P. Baptiste, C. Le Pape and W. Nuijten. Incorporating efficient Operations Research algorithms in constraint-based scheduling. In *Proceedings of the First International Joint Workshop on Artificial Intelligence and Operations Research*. 1995.

P. Baptiste, C. Le Pape and W. Nuijten. *Constraint-Based Scheduling*. Kluwer, 2001.

C. Barnhart, C. A. Hane and P. H. Vance. Using branch-and-price-and-cut to solve origin-destination integer multicommodity flow problems. *Operations Research*, 48:318–326, 2000.

C. Barnhart, E. L. Johnson, G. L. Nemhauser, M. W. P. Savelsbergh and P. H. Vance. Branch-and-price: Column generation for huge integer programs. *Operations Research*, 46:316–329, 1998.

J. J. Bartholdi, J. B. Orlin and H. D. Ratliff. Cyclic scheduling via integer programs with circular ones. *Operations Research*, 28:1074–1085, 1980.

J. J. Bartholdi III. A guaranteed-accuracy round-off algorithm for cyclic scheduling and set covering. *Operations Research*, 29:501–510, 1981.

S. E. Bechtold. Work force scheduling for arbitrary cyclic demands. *Journal of Operations Management*, 1:205–214, 1981.

S. E. Bechtold. Implicit optimal and heuristic labor staffing in a multiobjective, multilocation environment. *Decision Sciences*, 19:353–373, 1988.

S. E. Bechtold and M. J. Brusco. A microcomputer-based heuristic for tour scheduling of a mixed workforce. *Computer and Operations Research*, 21:1001–1009, 1994a.

S. E. Bechtold and M. J. Brusco. Working set generation methods for labor tour scheduling. *European Journal of Operational Research*, 74:540–551, 1994b.

S. E. Bechtold, M. J. Brusco and M. J. Showalter. A comparative evaluation of labor tour scheduling methods. *Decision Sciences*, 22:683–699, 1991.

S. E. Bechtold and L. W. Jacobs. Implicit modeling of flexible break assignments in optimal shift scheduling. *Management Science*, 36:1339–1351, 1990.

S. E. Bechtold and L. W. Jacobs. The equivalence of general set-covering and implicit integer programming formulations for shift scheduling. *Naval Research Logistics*, 43:233–249, 1996.

S. E. Bechtold and M. J. Showalter. A methodology for labor scheduling in a service operating system. *Decision Sciences*, 18:89–107, 1987.

J. C. Beck, P. Prosser and E. Selensky. On the reformulation of vehicle routing problems and scheduling problems. In *Proceedings of the 5th International Symposium on Abstraction, Reformulation and Approximation, LNCS 2371*, pages 282–289. Springer Verlag, 2002.

F. Bellanti, G. Carello, F. D. Croce and R. Tadei. A greedy-based neighborhood search approach to a nurse rostering problem. *European Journal of Operational Research*, 153:28–40, 2004.

B. T. Bennett and R. B. Potts. Rotating roster for a transit system. *Transportation Science*, 2:14–34, 1968.

O. Berman, R. C. Larson and E. Pinker. Scheduling workforce and workflow in a high volume factory. *Management Science*, 43:158–172, 1997.

I. Berrada, J. A. Ferland and P. Michelon. A multi-objective approach to nurse scheduling with both hard and soft constraints. *Socio-Economic Planning Science*, 30:183–193, 1996.

A. A. Bertossi, P. Carraresi and G. Gallo. On some matching problems arising in vehicle scheduling models. *Networks*, 17:271–281, 1987.

J.-Y. Blais and J.-M. Rousseau. Overview of HASTUS current and future versions. In J. R. Daduna and A. Wren, editors, *Computer-Aided Transit Scheduling*, pages 175–187. 1988.

L. Bodin and B. Golden. Classification in vehicle routing and scheduling. *Networks*, 11:97–108, 1981.

L. Bodin, B. Golden, A. Assad and M. Ball. Routing and scheduling of vehicles and crews – The state of the art. *Computer and Operations Research*, 10:63–211, 1983.

A. Bolat. Procedures for providing robust gate assignments for arriving aircrafts. *European Journal of Operational Research*, 120:63–80, 2000.

W. S. Brownell and J. M. Lowerre. Scheduling of work forces required in continuous operations under alternative labor policies. *Management Science*, 22:597–605, 1976.

P. Brucker, A. Drexl, R. Möhring, K. Neumann and E. Pesch. Resource-constrained project scheduling: Notation, classification, models, and methods. *European Journal of Operational Research*, 112:3–41, 1999.

M. J. Brusco. Solving personnel tour scheduling problems with the dual all-integer cutting plane. *IIE Transactions*, 30:835–844, 1998.

M. J. Brusco and L. W. Jacobs. A simulated annealing approach to the cyclic staff-scheduling problem. *Naval Research Logistics*, 40:69–84, 1993a.

M. J. Brusco and L. W. Jacobs. A simulated annealing approach to the solution of flexible labour scheduling problems. *Journal of the Operational Research Society*, 44:1191–1200, 1993b.

M. J. Brusco and L. W. Jacobs. Cost analysis of alternative formulations for personnel scheduling in continuously operating organizations. *European Journal of Operational Research*, 86:249–261, 1995.

M. J. Brusco and L. W. Jacobs. Eliminating redundant columns in continuous tour scheduling problems. *European Journal of Operational Research*, 111:518–525, 1998a.

M. J. Brusco and L. W. Jacobs. Personnel tour scheduling when starting-time restrictions are present. *Management Science*, 44:534–547, 1998b.

M. J. Brusco and L. W. Jacobs. Optimal models for meal-break and start-time flexibility in continuous tour scheduling. *Management Science*, 46:1630–1641, 2000.

M. J. Brusco and L. W. Jacobs. Starting-time decisions in labor tour scheduling: An experimental analysis and case study. *European Journal of Operational Research*, 131:459–475, 2001.

M. J. Brusco, L. W. Jacobs, R. J. Bongiorno, D. V. Lyons and B. Tang. Improving personnel scheduling at airline stations. *Operations Research*, 43:741–751, 1995.

M. J. Brusco and T. R. Johns. Improving the dispersion of surplus labor in personnel scheduling solutions. *Computers and Industrial Engineering*, 28:745–754, 1995.

M. J. Brusco and T. R. Johns. A sequential integer programming method for discontinuous labor tour scheduling. *European Journal of Operational Research*, 95:537–548, 1996.

E. S. Buffa, M. J. Cosgrove and B. J. Luce. An integrated work shift scheduling system. *Decision Sciences*, 7:620–630, 1976.

R. Burns and R. Narasimhan. Multiple shift scheduling of workforce on four-day workweeks. *Journal of the Operational Research Society*, 50:979–981, 1999.

R. N. Burns. Manpower scheduling with variable demands and alternate weekends off. *INFOR*, 16:101–111, 1978.

R. N. Burns and M. W. Carter. Work force size and single shift schedules with variable demands. *Management Science*, 31:599–607, 1985.

R. N. Burns and G. J. Koop. A modular approach to optimal multiple-shift manpower scheduling. *Operations Research*, 35:100–110, 1987.

A. M. Campbell and M. Savelsbergh. Efficient insertion heuristics for vehicle routing and scheduling problems. *Transportation Science*, 38:369–378, 2004.

A. Caprara, M. Fischetti, P. Toth, D. Bigo and P. L. Guida. Algorithms for railway crew management. *Mathematical Programming*, 79:125–141, 1997.

A. Caprara, P. Toth, D. Vigo and M. Fischetti. Modeling and solving the crew rostering problem. *Operations Research*, 46:820–830, 1998.

J. Carlier and E. Pinson. An algorithm for solving the job-shop problem. *Management Science*, 35:164–176, 1989.

M. W. Carter and S. D. Lapierre. Scheduling emergency room physicians. *Health Care Management Science*, 4:347–360, 2001.

Y. Caseau and F. Laburthe. Improved CLP scheduling with task intervals. In P. van Hentenryck, editor, *Proceedings of the 11th International Conference on Logic Programming*. MIT press, 1994.

Y. Caseau and F. Laburthe. Disjunctive scheduling with task intervals. Technical report, Laboratoire d'Informatique de l'Ecole normale Supérieure, 1995.

Y. Caseau and F. Laburthe. Solving small TSPs with constraints. In *Proceedings of ICLP '97*. 1997.

Y. Caseau and F. Laburthe. Heuristics for large constrained vehicle routing problems. *Journal of Heuristics*, 5:281–303, 1999.

T. Çezik and O. Günlük. Reformulating linear programs with transportation constraints – with applications to workforce scheduling. *Optimization Online*, 2002. http://www.optimization-online.org.

T. Çezik, O. Günlük and H. Luss. An integer programming model for the weekly tour scheduling problem. *Naval Research Logistics*, 48:607–624, 2001.

B. Cheng, J. Lee and J. Wu. A nurse rostering system using constraint programming and redundant modeling. *IEEE Transactions on Information Technology in Biomedicine*, 1:44–54, 1997.

K. L. Chew. Cyclic schedule for apron services. *Journal of the Operational Research Society*, 42:1061–1069, 1991.

N. Christofides. Vehicle routing. In E. L. Lawler, J. K. Lenstra, A. H. G. R. Kan and D. B. Shmoys, editors, *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*. Wiley, 1985.

I. T. Christou, A. Zakarian, J.-M. Liu and H. Carter. A two-phase genetic algorithm for large-scale bidline-generation problems at Delta air lines. *Interfaces*, 29:51–65, 1999.

V. Chvátal. *Linear Programming*. W. H. Freeman & Company, 1983.

G. Clarke and G. W. Wright. Scheduling of vehicles from a central depot to a number of delivery points. *Operations Research*, 12:568–581, 1964.

M. B. Clowes. On seeing things. *Artificial Intelligence*, 2:179–185, 1971.

J. Cordeau, G. Desaulniers, J. Desrosiers, M. M. Solomon and F. Soumis. VRP with time windows. In P. Toth and D. Vigo, editors, *The Vehicle Routing Problem*, pages 155–194. Society for Industrial and Applied Mathematics, 2001a.

J.-F. Cordeau, F. Soumis and J. Desrosiers. Simultaneous assignment of locomotives and cars to passenger trains. *Operations Research*, 49:531–548, 2001b.

J.-F. Cordeau, G. Stojković, F. Soumis and J. Desrosiers. Benders decomposition for simultaneous aircraft routing and crew scheduling. *Transportation Science*, 35:375–388, 2001c.

T. H. Cormen, C. E. Leiserson, R. L. Rivest and C. Stein. *Introduction to Algorithms, 2nd edition*. McGraw-Hill, 2001.

G. Cornuéjols and A. Sassano. On the 0,1 facets of the set covering polytope. *Mathematical Programming*, 43:45–55, 1989.

J. R. Daduna, I. Branco and J. P. Paixão, editors. *Computer-Aided Transit Scheduling*. Springer Verlag, 1995.

J. R. Daduna and M. Mojsilovic. Computer-aided vehicle and duty scheduling using the HOT programme system. In J. R. Daduna and A. Wren, editors, *Computer-Aided Transit Scheduling*, pages 133–146. Springer Verlag, 1988.

J. R. Daduna and A. Wren, editors. *Computer-Aided Transit Scheduling*. Springer Verlag, 1988.

G. Dantzig, D. Fulkerson and S. Johnson. Solutions of large-scale traveling-salesman problems. *Operations Research*, 2:393–410, 1954.

G. B. Dantzig. A comment on Edie's "Traffic delays at toll booths". *Journal of the Operations Research Society of America*, 2:339–341, 1954.

G. B. Dantzig and J. H. Ramser. The truck dispatching problem. *Management Science*, 6:80–91, 1959.

G. B. Dantzig and P. Wolfe. Decomposition principle for linear programs. *Operations Research*, 8:101–111, 1960.

Dash, 2004. *Xpress-Optimizer Reference Manual Release 15*. Dash Optimization, 2002.

P. R. Day and D. M. Ryan. Flight attendant rostering for short-haul airline operations. *Operations Research*, 45:649–661, 1997.

R. Dechter. *Constraint Processing*. Morgan Kaufmann, 2003.

G. Desaulniers, J. Desrosiers and M. M. Solomon. Accelerating strategies in column generation methods for vehicle routing and crew scheduling problems. In C. C. Ribeiro and P. Hansen, editors, *Essays and Surveys in Metaheuristics*, pages 309–324. Kluwer, 2001.

G. Desaulniers, J. Lavigne and F. Soumis. Multi-depot vehicle scheduling problems with time windows and waiting costs. *European Journal of Operational Research*, 111:479–494, 1998.

M. Desrochers, J. Desrosiers and M. M. Solomon. Using column generation to solve the vehicle routing problem with time windows. In *Operational Research '90 – Selected Papers from the Twelfth IFORS International Conference on Operational Research*, pages 411–419. Pergamon Press, 1990.

M. Desrochers, J. Desrosiers and M. M. Solomon. A new optimization algorithm for the vehicle routing problem with time windows. *Operations Research*, 40:342–354, 1992.

M. Desrochers and J.-M. Rousseau, editors. *Computer-Aided Transit Scheduling*. Springer Verlag, 1992.

M. Desrochers and F. Soumis. A column generation approach to the urban transit crew scheduling problem. *Transportation Science*, 23:1–13, 1989.

J. Desrosiers and M. E. Lübbecke. A primer in column generation. Technical report, Technische Universität Berlin, Germany, 2003.

J. Desrosiers, M. M. Solomon and F. Soumis. Time constrained routing and scheduling. In *Handbooks of Operations Research and Management Science*, volume 8, pages 35–139. North-Holland, 1993.

J. Desrosiers, F. Soumis and M. Desrochers. Routing with time windows by column generation. *Networks*, 14:545–565, 1984.

U. Dorndorf, E. Pesch and T. Phan-Huy. Constraint propagation techniques for the disjunctive scheduling problem. *Artificial Intelligence*, 122:189–240, 2000.

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

K. A. Dowsland. Nurse scheduling with tabu search and strategic oscillation. *European Journal of Operational Research*, 106:393–407, 1998.

M. Dror. Note on the complexity of the shortest path models for column generation in VRPTW. *Operations Research*, 42:977–979, 1994.

O. du Merle, D. Villeneuve, J. Desrosiers and P. Hansen. Stabilized column generation. *Discrete Mathematics*, 194:229–237, 1999.

Y. Dumas, J. Desrosiers and F. Soumis. The pickup and delivery problem with time windows. *European Journal of Operational Research*, 54:7–22, 1991.

S. M. Easa. Resource leveling in construction by optimization. *Journal of Construction Engineering and Management*, 115:302–316, 1989.

F. F. Easton and D. F. Rossin. Sufficient working subsets for the tour scheduling problem. *Management Science*, 37:1441–1451, 1991.

H. Emmons. Work-force scheduling with cyclic requirements and constraints on days off, weekends off, and work stretch. *IIE Transactions*, 17:8–16, 1985.

H. Emmons and R. N. Burns. Off-day scheduling with hierarchical worker categories. *Operations Research*, 39:484–495, 1991.

H. Emmons and D.-S. Fuh. Sizing and scheduling a full-time and part-time workforce with off-day and off-weekend constraints. *Annals of Operations Research*, 70:473–492, 1997.

A. T. Ernst, H. Jiang, M. Krishnamoorthy, H. Nott and D. Sier. Rail crew scheduling and rostering: Optimisation algorithms. In *Computer-Aided Scheduling of Public Transport, 8th International Conference, Berlin, Germany*. 2000.

A. T. Ernst, H. Jiang, M. Krishnamoorthy and D. Sier. Staff scheduling and rostering: A review of applications, methods and models. *European Journal of Operational Research*, 153:3–27, 2004.

EU, 1993. *Council Directive 93/104/EC*. Council of the European Union, 1993.

EU, 2000a. *Council Directive 2000/79/EC*. Council of the European Union, 2000.

EU, 2000b. *Directive 2000/34/EC of the European Parliament and of the Council*. European Parliament and Council of the European Union, 2000.

EU, 2002. *Directive 2002/15/EC of the European Parliament and of the Council*. European Parliament and Council of the European Union, 2002.

T. Fahle, U. Junker, S. E. Karisch, N. Kohl, M. Sellmann and B. Vaaben. Constraint programming based column generation for crew assignment. *Journal of Heuristics*, 8:59–81, 2002.

J. C. Falkner and D. M. Ryan. EXPRESS: Set partitioning for bus crew scheduling in Christchurch. In M. Desrochers and J.-M. Rousseau, editors, *Computer-Aided Transit Scheduling*, pages 359–378. Springer Verlag, 1992.

A. A. Farley. A note on bounding a class of linear programming problems, including cutting stock problems. *Operations Research*, 38:922–923, 1990.

G. Felici and C. Gentile. A polyhedral approach for the staff rostering problem. *Management Science*, 50:381–393, 2004.

M. L. Fisher and R. Jaikumar. A generalized assignment heuristic for vehicle routing. *Networks*, 11:109–124, 1981.

M. Fleury. Deux problèmes de géométrie de situation. *Journal de mathématiques élémentaires*, pages 257–261, 1883.

F. Focacci, A. Lodi and M. Milano. Cost-based domain filtering. In J. Jaffar, editor, *Principles and Practice of Constraint Programming*, pages 189–203. Springer Verlag, 1999.

F. Focacci, A. Lodi and M. Milano. A hybrid exact algorithm for the TSPTW. *INFORMS Journal on Computing*, 14:403–417, 2002.

F. Focacci, A. Lodi, M. Milano and D. Vigo. Solving TSP through the integration of OR and CP techniques. In *Proceedings CP98 Workshop on Large Scale Combinatorial Optimisation and Constraints*. 1998.

F. Focacci and M. Milano. Connections and integrations of dynamic programming and constraint programming. In *Proceedings of the International Workshop on Integration of AI and OR techniques in Constraint Programming for Combinatorial Optimization Problems*. 2001.

L. R. Ford and D. R. Fulkerson. A suggested computation for maximal multi-commodity network flows. *Management Science*, 5:97–101, 1958.

L. R. Ford and D. R. Fulkerson. *Flows in Networks*. Princeton University Press, 1962.

E. C. Freuder. Synthesizing constraint expressions. *Communications of the ACM*, 21:958–966, 1978.

A. Gaballa and W. Pearce. Telephone sales manpower planning at Qantas. *Interfaces*, 9:1–9, 1979.

M. Gamache and F. Soumis. A method for optimally solving the rostering problem. In G. Yu, editor, *Operations Research in the Airline Industry*, pages 124–157. Kluwer, 1998.

M. Gamache, F. Soumis, G. Marquis and J. Desrosiers. A column generation approach for large-scale aircrew rostering problems. *Operations Research*, 47:247–263, 1999.

M. Gamache, F. Soumis, D. Villeneuve, J. Desrosiers and É. Gélinas. The preferential bidding system at Air Canada. *Transportation Science*, 32:246–255, 1998.

M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, 1979.

M. Gendreau, A. Hertz and G. Laporte. A tabu search heuristic for the vehicle routing problem. *Management Science*, 40:1276–1290, 1994.

A. M. Geoffrion. Lagrangean relaxation for integer programming. *Mathematical Programming Study*, 2:82–114, 1974.

P. C. Gilmore and R. E. Gomory. A linear programming approach to the cutting-stock problem. *Operations Research*, 9:849–859, 1961.

F. Glover and C. McMillan. The general employee scheduling problem: An integration of MS and AI. *Computers and Operations Research*, 13:563–573, 1986.

F. Glover, C. McMillan and R. Glover. A heuristic programming approach to the employee scheduling problem and some thoughts on “managerial robots”. *Journal of Operations Management*, 4:113–128, 1984.

J.-L. Goffin and J.-P. Vial. Convex nondifferentiable optimization: A survey focussed on the analytic center cutting plane method. *Optimization Methods and Software*, 17:805–867, 2002.

B. L. Golden and A. A. Assad. *Vehicle Routing: Methods and Studies*. North-Holland, 1988.

M. C. Golumbic. The complexity of comparability graph recognition and coloring. *Computing*, 18:199–208, 1977.

M. C. Golumbic. *Algorithmic Graph Theory and Perfect Graphs*. Academic Press, 1980.

M. Grönkvist. *Structure in Airline Crew Optimization Problems*. Master’s thesis, Chalmers University of Technology, Göteborg, Sweden, 1998.

M. Grönkvist. Using constraint propagation to accelerate column generation in aircraft scheduling. In *Proceedings of the 8th International Conference on Principles and Practice of Constraint Programming, LNCS 2470*. Springer Verlag, 2002.

R. M. Haralick and G. L. Elliott. Increasing tree search efficiency for constraint satisfaction problems. *Artificial Intelligence*, 14:263–313, 1980.

R. B. Harris. Packing method for resource leveling (PACK). *Journal of Construction Engineering and Management*, 116:331–350, 1990.

W. D. Harvey and M. L. Ginsberg. Limited discrepancy search. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pages 607–613. Morgan Kaufmann, 1995.

S. Heipcke. Comparing constraint programming and mathematical programming approaches to discrete optimisation – the change problem. *Journal of the Operational Research Society*, pages 581–595, 1999.

S. G. Henderson and A. J. Mason. Rostering by iterating integer programming and simulation. In D. Medeiros and E. Watson, editors, *Proceedings of the 1998 Winter Simulation Conference*. 1998.

W. B. Henderson and W. L. Berry. Heuristic methods for telephone operator shift scheduling: An experimental analysis. *Management Science*, 22:1372–1380, 1976.

P. V. Hentenryck, Y. Deville and C.-M. Teng. A generic arc-consistency algorithm and its specializations. *Artificial Intelligence*, 57:291–321, 1992.

C. Hierholzer. Über die Möglichkeit, einen Linienzug ohne Wiederholung und Unterbrechung zu umfahren. *Mathematische Annalen*, 6:30–42, 1873.

J.-C. Hiriart-Urruty and C. Lemaréchal, editors. *Convex analysis and minimization algorithms, part 2: Advanced theory and bundle methods*. Springer Verlag, 1993.

K. L. Hoffman and M. Padberg. Solving airline crew scheduling problems by branch-and-cut. *Management Science*, 39:657–682, 1993.

T. J. Holloran and J. E. Byrn. United Airlines station manpower planning system. *Interfaces*, 16:39–50, 1986.

J. Hromkovič. *Algorithmics for Hard Problems*. Springer Verlag, 2001.

D. A. Huffman. Impossible objects as nonsense sentences. In B. Meltzer and D. Michie, editors, *Machine Intelligence 6*, pages 295–323. Edinburgh University Press, 1971.

R. Hung. A three-day workweek multiple-shift scheduling model. *Journal of the Operational Research Society*, 44:141–146, 1993.

R. Hung. A multiple-shift workforce scheduling model under the 4-day workweek with weekday and weekend labour demands. *Journal of the Operational Research Society*, 45:1088–1092, 1994a.

R. Hung. Multiple-shift workforce scheduling under the 3–4 workweek with different weekday and weekend labor requirements. *Management Science*, 40:280–284, 1994b.

IATA, 2004a. *IATA International Industry Statistics September 2004*. International Air Transport Association, 2004. http://www.iata.org.

IATA, 2004b. *IATA Annual Report 2004*. International Air Transport Association, 2004. http://www.iata.org.

S. Irnich. *Netzwerk-Design für zweistufige Transportsysteme und ein Branch-and-Price-Verfahren für das gemischte Direkt- und Hubflugproblem*. Ph.D. thesis, Fakultät für Wirtschaftswissenschaften, Rheinisch-Westfälische Technische Hochschule Aachen, Germany, 2002.

S. Irnich and D. Villeneuve. The shortest path problem with resource constraints and k-cycle elimination for $k \geq 3$. Technical report, Les Cahiers du GERAD, HEC Montréal, Canada, 2003.

W. K. Jackson, W. S. Havens and H. Dollard. Staff scheduling: A simple approach that worked. Technical Report CMPT97-23, Simon Fraser University, Burnaby, Canada, 1997.

L. W. Jacobs and S. E. Bechtold. Labor utilization effects of labor scheduling flexibility alternatives in a tour scheduling environment. *Decision Sciences*, 24:148–166, 1993.

L. W. Jacobs and M. J. Brusco. Overlapping start-time bands in implicit tour scheduling. *Management Science*, 42:1247–1259, 1996.

A. I. Z. Jarrah, J. F. Bard and A. H. deSilva. Solving large-scale tour scheduling problems. *Management Science*, 40:1124–1144, 1994.

B. Jaumard, F. Semet and T. Vovor. A generalized linear programming model for nurse scheduling. *European Journal of Operational Research*, 107:1–18, 1998.

E. L. Johnson, G. L. Nemhauser and M. W. P. Savelsbergh. Progress in linear programming-based algorithms for integer programming: An exposition. *INFORMS Journal on Computing*, 12:2–23, 2000.

D. Jungnickel. *Graphs, Networks and Algorithms*. Springer Verlag, 2002.

L. Kantorovich. Mathematical methods of organising and planning production. *Management Science*, 6:366–422, 1960.

R. M. Karp and J. B. Orlin. Parametric shortest path algorithms with an application to cyclic staffing. *Discrete Applied Mathematics*, 3:37–45, 1981.

E. G. Keith. Operator scheduling. *AIIE Transactions*, 11:37–41, 1979.

J. E. Kelley. The cutting-plane method for solving convex programs. *Journal of the Society for Industrial and Applied Mathematics*, 8:703–712, 1960.

C. M. Khoong and H. C. Lau. ROMAN: An integrated approach to manpower planning and scheduling. In O. Balci, R. Sharda and S. Zenios, editors, *Computer Science and Operations Research: New Developments in Their Interfaces*, pages 383–396. Pergamon Press, 1992.

C. M. Khoong, H. C. Lau and L. W. Chew. Automated manpower rostering: Techniques and experience. *International Transactions in Operational Research*, 1:353–361, 1994.

P. Kilby, P. Prosser and P. Shaw. Implementation of LNS for constrained VRPs. Technical report, APES Group, Department of Computer Science, University of Strathclyde, Glasgow, Scotland, UK, 1998.

P. Kilby, P. Prosser and P. Shaw. A comparison of traditional and constraint-based heuristic methods on vehicle routing problems with side constraints. *Journal of Constraints*, 5:389–414, 2000.

N. Kohl, J. Desrosiers, O. B. G. Madsen, M. M. Solomon and F. Soumis. 2-path cuts for the vehicle routing problem with time windows. *Transportation Science*, 33:101–116, 1999.

N. Kohl and S. E. Karisch. Airline crew rostering: Problem types, modeling, and optimization. *Annals of Operations Research*, 127:223–257, 2004.

G. J. Koop. Multiple shift workforce lower bounds. *Management Science*, 34:1221–1230, 1988.

R. E. Korf. Improved limited discrepancy search. In *Proceedings of the 13th National Conference on Artificial Intelligence*, pages 288–291. 1996.

G. Kortsarz and W. Slany. The minimum shift design problem and its relation to the minimum edge-cost flow problem. Technical report, Institut für Informationssysteme, Technische Universität Wien, Austria, 2001.

V. Kumar. Algorithms for constraint satisfaction problems. *AI Magazine*, 13:32–44, 1992.

T. Kwasniok. *Entwurf eines Modells zur Ermittlung von Bedarfszahlen zur langfristigen Planung von Personal und Geräten in der Flugabfertigung*. Master's thesis, Rheinisch-Westfälische Technische Hochschule Aachen, Germany, 1994.

A. Langevin, M. Desrochers, J. Desrosiers, S. Gélinas and F. Soumis. A two-commodity flow formulation for the traveling salesman and makespan problem with time windows. *Networks*, 23:631–640, 1993.

G. Laporte, Y. Nobert and J. Biron. Rotating schedules. *European Journal of Operational Research*, 4:24–30, 1980.

G. Laporte and I. H. Osman. Routing problems: A bibliography. *Annals of Operations Research*, 61:227–262, 1995.

L. S. Lasdon. *Optimization Theory for Large Systems*. MacMillan, 1970.

H. C. Lau. Manpower scheduling with shift change constraints. In *Proceedings of the 5th International Symposium on Algorithms and Computation, LNCS 834*, pages 616–624. Springer Verlag, 1994.

H. C. Lau. Combinatorial approaches for hard problems in manpower scheduling. *Journal of the Operations Research Society of Japan*, 39:88–98, 1996a.

H. C. Lau. On the complexity of manpower shift scheduling. *Computers and Operations Research*, 23:93–102, 1996b.

E. L. Lawler, J. K. Lenstra, A. H. G. R. Kan and D. B. Shmoys, editors. *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*. Wiley, 1985.

C. Lemaréchal. Lagrangian relaxation. In M. Jünger and D. Naddef, editors, *Computational Combinatorial Optimization*, pages 112–156. Springer Verlag, 2001.

J. K. Lenstra and A. H. G. R. Kan. Complexity of vehicle routing and scheduling problems. *Networks*, 11:221–227, 1981.

C. Li, E. P. Robinson and V. A. Mabert. An evaluation of tour scheduling heuristics with differences in employee productivity and cost. *Decision Sciences*, 22:700–718, 1991.

S. Lin. Computer solutions of the traveling salesman problem. *Bell Systems Technical Journal*, 44:2245–2269, 1965.

A. Löbel. *Optimal Vehicle Scheduling in Public Transit*. Ph.D. thesis, Technische Universität Berlin, Germany, 1997.

J. S. Loucks and F. R. Jacobs. Tour scheduling and task assignment of a heterogeneous work force: A heuristic approach. *Decision Sciences*, 22:719–738, 1991.

R. Lougee-Heimer. The Common Optimization INterface for Operations Research. *IBM Journal of Research and Development*, 47:57–66, 2003.

R. R. Love and J. M. Hoey. Management science improves fast-food operations. *Interfaces*, 20:21–29, 1990.

J. M. Lowerre. Work stretch properties for the scheduling of continuous operations under alternative labor policies. *Management Science*, 23:963–971, 1977.

M. Lübbecke. *Engine Scheduling by Column Generation*. Ph.D. thesis, Fachbereich für Mathematik und Informatik, Technische Universität Braunschweig, Germany, 2001.

M. E. Lübbecke and J. Desrosiers. Selected topics in column generation. Technical report, Technische Universität Berlin, Germany, 2004.

I. J. Lustig and J.-F. Puget. Program does not equal program: Constraint programming and its relationship to mathematical programming. *Interfaces*, pages 29–53, 2001.

V. A. Mabert and A. R. Raedels. The detail scheduling of a part-time work force: A case study of teller staffing. *Decision Sciences*, 8:109–120, 1977.

V. A. Mabert and C. A. Watts. A simulation analysis of tour-shift construction procedures. *Management Science*, 28:520–532, 1982.

H. Marchand, A. Martin, R. Weismantel and L. Wolsey. Cutting planes in integer and mixed integer programming. *Discrete Applied Mathematics*, 123:397–446, 2002.

R. E. Marsten, W. W. Hogan and J. W. Blankenship. The BOXSTEP method for large-scale optimization. *Operations Research*, 23:389–337, 1975.

S. Martello and P. Toth. *Knapsack Problems: Algorithms and Computer Implementation*. Wiley, 1990.

A. Martin. General mixed integer programming: Computation issues for branch-and-cut algorithms. In M. Jünger and D. Naddef, editors, *Computational Combinatorial Optimization*, pages 1–25. Springer Verlag, 2001.

A. Mason and D. Nielsen. PETRA: A programmable optimisation engine and toolbox for personnel rostering applications. Technical report, Department of Engineering Science, University of Auckland, New Zealand, 1999.

A. J. Mason. Solution methods for cyclic roster construction. Technical report, University of Auckland, New Zealand, 1999.

A. J. Mason, D. M. Ryan and D. M. Panton. Integrated simulation, heuristic and optimisation approaches to staff scheduling. *Operations Research*, 46:161–175, 1998.

A. J. Mason and M. C. Smith. A nested column generator for solving rostering problems with integer programming. In L. Caccetta, K. L. Teo, P. F. Siew, Y. H. Leung, L. S. Jennings and V. Rehbock, editors, *International Conference on Optimisation: Techniques and Applications*. 1998.

L. F. McGinnis, W. D. Culver and R. H. Deane. One- and two-phase heuristics for workforce scheduling. *Computers and Industrial Engineering*, 2:7–15, 1978.

A. Mehrotra, K. E. Murphy and M. A. Trick. Optimal shift scheduling: A branch-and-price approach. *Naval Research Logistics*, 47:185–200, 2000.

A. Meisels, E. Gudes and G. Solotorevsky. Combining rules and constraints for employee timetabling. *International Journal of Intelligent Systems*, 12:419–439, 1997.

P. Meseguer. Constraint satisfaction problems: An overview. *Artificial Intelligence Communications*, 2:3–17, 1989.

P. Meseguer. Interleaved depth-first search. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence*, pages 1382–1387. 1997.

H. H. Millar and M. Kiragu. Cyclic and non-cyclic scheduling of 12h shift nurses by network programming. *European Journal of Operational Research*, 104:582–592, 1998.

H. E. Miller, W. P. Pierskalla and G. J. Rath. Nurse scheduling using mathematical programming. *Operations Research*, 24:857–870, 1976.

U. Montanari. Networks of constraints: Fundamental properties and applications to picture processing. *Information Sciences*, 7:95–132, 1974.

S. L. Moondra. An L.P. model for work force scheduling in banks. *Journal of Bank Research*, 6:299–301, 1976.

J. G. Morris and M. J. Showalter. Simple approaches to shift, days-off and tour scheduling problems. *Management Science*, 29:942–950, 1983.

A. A. Musa and U. Saxena. Scheduling nurses using goal-programming techniques. *IIE Transactions*, 16:216–221, 1984.

N. Muscettola. Computing the envelope for stepwise-constant resource allocations. In *Proceedings of the 8th International Conference on Principles and Practice of Constraint Programming, LNCS 2470*, pages 139–154. 2002.

N. Muslija, J. Gärtner and W. Slany. Efficient generation of rotating workforce schedules. Technical report, Institut für Informationssysteme, Technische Universität Wien, Austria, 2000.

N. Musliu, A. Schaerf and W. Slany. Local search for shift design. *European Journal of Operational Research*, 153:51–64, 2004.

P. Neame. *Nonsmooth dual problems in integer programming*. Ph.D. thesis, University of Melbourne, Australia, 1999.

P. Neame. *Nonsmooth Dual Methods in Integer Programming*. Ph.D. thesis, Department of Mathematics and Statistics, University of Melbourne, Australia, 2000.

G. L. Nemhauser and S. Park. A polyhedral approach to edge coloring. *Operations Research Letters*, 10:315–322, 1991.

G. L. Nemhauser and L. A. Wolsey. *Integer and Combinatorial Optimization*. John Wiley and Sons, 1988.

K. Neumann and J. Zimmermann. Resource levelling for projects with schedule-dependent time windows. *European Journal of Operational Research*, 117:591–605, 1999.

Y. Nobert and J. Roy. Freight handling personnel scheduling at air cargo terminals. *Transportation Science*, 32:295–301, 1998.

P. Nobili and A. Sassano. Facets and lifting procedures for the set covering polytope. *Mathematical Programming*, 45:111–137, 1989.

I. Or. *Travelling Salesman-Type Combinatorial Problems and Their Relation to the Logistics of Blood-Banking*. Ph.D. thesis, Department of Industrial Engineering and Management Sciences, Northwest University, 1976.

W. Orchard-Hays. *Advanced Linear-Programming Computing Techniques*. McGraw-Hill, 1968.

I. Ozkarahan and J. E. Bailey. Goal programming model subsystem of a flexible nurse scheduling support system. *IIE Transactions*, 20:306–316, 1988.

M. Padberg and G. Rinaldi. A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. *SIAM Review*, 33:60–100, 1991.

D. M. Panton. On the creation of multiple shift continuous operation cyclic rosters under general workforce conditions. *Asia-Pacific Journal of Operational Research*, 8:189–201, 1991.

D. M. Panton and D. M. Ryan. Column generation models for optimal workforce allocation with multiple breaks. Technical report, Centre for Industrial and Applicable Mathematics, University of South Australia, Mawon Lakes, Australia, 1999.

G. Pesant and M. Gendreau. A view of local search in constraint programming. In E. C. Freuder, editor, *Proceedings of the Second International Conference on Principles and Practice of Constraint Programming, LNCS 1118*, pages 353–366. Springer Verlag, 1996.

G. Pesant and M. Gendreau. A constraint programming framework for local search methods. *Journal of Heuristics*, 5:255–279, 1999.

G. Pesant, M. Gendreau, J.-Y. Potvin and J.-M. Rousseau. An exact constraint logic programming algorithm for the traveling salesman problem with time windows. *Transportation Science*, 32:12–29, 1998.

J.-Y. Potvin and S. Bengio. The vehicle routing problem with time windows – part II: Genetic search. *INFORMS Journal on Computing*, 8:165–172, 1996.

J.-Y. Potvin, T. Kervahut, B. L. Garcia and J.-M. Rousseau. The vehicle routing problem with time windows – part I: Tabu search. *INFORMS Journal on Computing*, 8:158–164, 1996.

J.-Y. Potvin and J.-M. Rousseau. A parallel route building algorithm for the vehicle routing and scheduling problem with time windows. *European Journal of Operational Research*, 66:331–340, 1993.

J.-Y. Potvin and J.-M. Rousseau. An exchange heuristic for routing problems with time windows. *Journal of the Operational Research Society*, 46:1433–1446, 1995.

T. K. Ralphs and L. Ladányi. *COIN/BCP User's Manual*, 2001. http://www-124.ibm.com/developerworks/opensource/coin/Presentations/bcp-man.pdf.

J.-C. Régin. A filtering algorithm for constraints of difference in CSPs. In *Proceedings of the 12th National Conference on Artificial Intelligence*, pages 362–367. American Association for Artificial Intelligence, 1994.

J.-C. Régin. Newsgroup posting on the complexity of the *alldifferent* constraint, sci.op-research. 2000. http://mat.gsia.cmu.edu/ORCS/archive/0038.html.

M. Rekik, J.-F. Cordeau and F. Soumis. Implicit shift scheduling with multiple breaks and pre- and post-break restrictions. Technical report, Ecole Polytechnique de Montréal and GERAD, Montréal, Canada, 2003.

C. Ribeiro and F. Soumis. A column generation approach to the multiple depot vehicle scheduling problem. *Operations Research*, 42:41–53, 1994.

L. P. Ritzman, L. J. Krajewski and M. J. Showalter. The disaggregation of aggregate manpower plans. *Management Science*, 22:1204–1214, 1976.

Y. Rochat and É. Taillard. Probabilistic diversification and intensification in local search for vehicle routing. *Journal of Heuristics*, 1:147–167, 1995.

E. S. Rosenbloom and N. F. Goertzen. Cyclic nurse scheduling. *European Journal of Operational Research*, 31:19–23, 1987.

J.-M. Rousseau, editor. *Computer Scheduling of Public Transport 2*. North-Holland, 1985.

L.-M. Rousseau, M. Gendreau and G. Pesant. Solving small VRPTWs with constraint programming based column generation. In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems CPAIOR 02*. 2002.

B. Roy and B. Sussman. Les problémes d'ordonnancement avec contraintes disjonctives. Notes DS No. 9bis, SEMA, Paris, 1964.

D. M. Ryan. The solution of massive generalized set partitioning problems in aircrew rostering. *Journal of the Operational Research Society*, 43:459–467, 1992.

D. M. Ryan and B. A. Foster. An integer programming approach to scheduling. In A. Wren, editor, *Computer Scheduling of Public Transport*, pages 269–280. North-Holland, 1981.

A. Sassano. On the facial structure of the set covering polytope. *Mathematical Programming*, 44:181–202, 1989.

M. W. P. Savelsbergh. A branch-and-price algorithm for the generalized assignment problem. *Operations Research*, 45:831–841, 1997.

A. Schaerf and A. Meisels. Solving employee timetabling problems by generalized local search. In *Proceedings of the 6th Italian Conference on Artificial Intelligence*, pages 493–502. 1999.

S. Schindler and T. Semmel. Station staffing at Pan American world airways. *Interfaces*, 23:91–98, 1993.

M. Segal. The operator-scheduling problem: A network-flow approach. *Operations Research*, 22:808–823, 1972.

J. E. Seibert and G. W. Evans. Time-constrained resource leveling. *Journal of Construction Engineering and Management*, 117:503–520, 1991.

M. Sellmann, K. Zervoudakis, P. Stamatopoulos and T. Fahle. Crew assignment via constraint programming: Integrating column generation and heuristic tree search. *Annals of Operations Research*, 115:207–226, 2002.

P. Shaw. Using constraint programming and local search methods to solve vehicle routing problems. In M. Mahler and J.-F. Puget, editors, *Proceedings of the Fourth International Conference on Principles and Practice of Constraint Programming, LNCS 1520*, pages 417–431. Springer Verlag, 1998.

H. D. Sherali. Equivalent weights for lexicographic multi-objective programs: Characterizations and computations. *European Journal of Operational Research*, 11:367–379, 1982.

M. J. Showalter and V. A. Mabert. An evaluation of a full-/part-time tour scheduling methodology. *International Journal of Operations and Production Management*, 8:54–71, 1988.

B. M. Smith and A. Wren. A bus crew scheduling system using a set covering formulation. *Transportation Research*, 22A:97–108, 1988.

M. M. Solomon. Algorithms for the vehicle routing and scheduling problem with time window constraints. *Operations Research*, 35:254–265, 1987.

M. D. Sørensen and J. Clausen. Decentralized ground staff scheduling. Technical report, Informatics and Mathematical Modelling, Technical University of Denmark, Kongens Lyngby, Denmark, 2002.

H. I. Stern and M. Hersh. Scheduling aircraft cleaning crews. *Transportation Science*, 14:277–291, 1980.

E. Taillard. Parallel iterative search methods for vehicle routing problems. *Networks*, 23:661–673, 1993.

R. Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1:146–160, 1972.

P. E. Taylor and S. J. Huxley. A break from tradition for the San Francisco Police: Patrol officer scheduling using an optimization-based decision support system. *Interfaces*, 19:4–24, 1989.

D. Teodorović. *Airline Operations Research*. Gordon and Breach, 1989.

G. M. Thompson. Shift scheduling when employees have limited availability: An L.P. approach. *Journal of Operations Management*, 9:352–370, 1990.

G. M. Thompson. Improving the utilization of front-line service delivery system personnel. *Decision Sciences*, 23:1072–1098, 1992.

G. M. Thompson. Representing employee requirements in labour tour scheduling. *Omega*, 21:657–671, 1993.

G. M. Thompson. Improved implicit optimal modeling of the labor shift scheduling problem. *Management Science*, 41:595–607, 1995.

G. M. Thompson. Controlling action times in daily workforce scheduling. *Cornell Hotel and Restaurant Administration Quarterly*, 37:82–96, 1996a.

G. M. Thompson. A simulated-annealing heuristic for shift scheduling using non-continuously available employees. *Computers and Operations Research*, 23:275–288, 1996b.

R. Tibrewala, D. Philippe and J. Browne. Optimal scheduling of two consecutive idle periods. *Management Science*, 19:71–75, 1972.

J. M. Tien and A. Kamiyama. On manpower scheduling algorithms. *SIAM Review*, 24:275–287, 1982.

S. A. Topaloglu and I. Ozkarahan. A research on optimization based modeling for tour scheduling problems with flexible break assignments. Technical report, Department of Industrial Engineering, Dokuz Eylul University, Izmir, Turkey, 1998.

P. Toth and D. Vigo. An overview of vehicle routing problems. In P. Toth and D. Vigo, editors, *The Vehicle Routing Problem*, pages 1–26. Society for Industrial and Applied Mathematics, 2001a.

P. Toth and D. Vigo, editors. *The Vehicle Routing Problem*. Society for Industrial and Applied Mathematics, 2001b.

E. Tsang, editor. *Foundations of Constraint Satisfaction*. Academic Press, 1993.

J. M. Valério de Carvalho. LP models for bin packing and cutting stock problems. *European Journal of Operational Research*, 141:253–273, 2002.

M. van den Akker, H. Hoogeveen and S. van de Velde. Combining column generation and Lagrangean relaxation to solve a single-machine common due date problem. *INFORMS Journal on Computing*, 14:37–51, 2002.

Y. van den Berg and D. M. Panton. Personnel shift assignment: Existence conditions and network models. *Networks*, 24:385–394, 1994.

P. H. Vance, C. Barnhart, E. L. Johnson and G. L. Nemhauser. Solving binary cutting stock problems by column generation and branch-and-bound. *Computational Optimization and Applications*, 3:111–130, 1994.

P. H. Vance, C. Barnhart, E. L. Johnson and G. L. Nemhauser. Airline crew scheduling: A new formulation and decomposition algorithm. *Operations Research*, 45:188–200, 1997.

F. Vanderbeck. On Dantzig-Wolfe decomposition in integer programming and ways to perform branching in a branch-and-price algorithm. *Operations Research*, 48:111–128, 2000.

F. Vanderbeck and L. A. Wolsey. An exact algorithm for IP column generation. *Operations Research Letters*, 19:151–159, 1996.

D. Villeneuve, J. Desrosiers, M. Lübbecke and F. Soumis. On compact formulations for integer programs solved by column generation. Technical report, Les Cahiers du GERAD, 2003.

R. V. Vohra. A quick heuristic for some cyclic staffing problems with breaks. *Journal of the Operations Research Society*, 39:1057–1061, 1988.

T. Walsh. Depth-bounded discrepancy search. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence*, pages 1388–1395. 1997.

D. L. Waltz. Understanding line drawings of scenes with shadows. In P. H. Winston, editor, *The Psychology of Computer Vision*, pages 19–91. McGraw-Hill, 1975.

D. M. Warner. Scheduling nursing personnel according to nursing preference: A mathematical programming approach. *Operations Research*, 24:842–856, 1976.

G. Weil, K. Heus, P. François and M. Poujade. Constraint programming for nurse scheduling. *IEEE Engineering in Medicine and Biology Magazine*, 14:417–422, 1995.

P. Wentges. Weighted Dantzig-Wolfe decomposition for linear mixed-integer programming. *International Transactions in Operational Research*, 4:151–162, 1997.

N. H. M. Wilson, editor. *Computer-Aided Transit Scheduling*. Springer Verlag, 1999.

R. J. Wilson. *Introduction to Graph Theory*. Addison-Wesley, 1996.

A. Wren, editor. *Computer Scheduling of Public Transport*. North-Holland, 1981.

A. Wren and J.-M. Rousseau. Bus driver scheduling – an overview. In J. R. Daduna, I. Branco and J. P. Paixão, editors, *Computer-Aided Transit Scheduling*, pages 173–187. Springer Verlag, 1995.

S. Yan, C.-Y. Shieh and M. Chen. A simulation framework for evaluating airport gate assignments. *Transportation Research Part A: Policy and Practice*, 36:885–898, 2002.

G. Yu, editor. *Operations Research in the Airline Industry*. Kluwer, 1998.

H.-J. Zimmermann. An application-oriented view of modeling uncertainty. *European Journal of Operational Research*, 122:190–198, 2000.

H.-J. Zimmermann. *Operations Research – Methoden und Modelle*. Vieweg, 2005.

J. Zimmermann and H. Engelhardt. Lower bounds and exact algorithms for resource levelling problems. Technical report, Universität Karlsruhe, 1998.

Index

Curriculum Vitae

Angaben zur Person

Name	Jörg Herbers
Geburtsdatum	16.11.1975
Geburtsort	Haselünne

Schul- und Hochschulausbildung

06/2001	Springorum-Denkmünze, RWTH Aachen
05/2000	Diplom mit Auszeichnung in Informatik, RWTH Aachen
10/1999-05/2000	Diplomarbeit, DaimlerChrysler Forschungszentrum Berlin
04/1998-05/2000	Stipendium der DaimlerChrysler Studienförderung Forschung & Technologie
09/1997-06/1998	Auslandsstudium an der Ecole Polytechnique Fédérale de Lausanne, Schweiz
07/1997	Vordiplom in Informatik, RWTH Aachen
02/1996-05/2000	Stipendium der Studienstiftung des deutschen Volkes
10/1995-05/2000	Informatik-Studium, RWTH Aachen
06/1995	Abitur, Gymnasium Marianum Meppen

Beruflicher Werdegang

seit 06/2000	Operations Research Modellentwickler bei INFORM Institut für Operations Research und Management GmbH, Aachen
10/1999-05/2000	Forschungstätigkeit am DaimlerChrysler Forschungszentrum Berlin
11/1998-07/1999	Studentische Hilfskraft am Lehrstuhl für Informatik VI, RWTH Aachen
10/1996-04/1997	Studentische Hilfskraft am Institut für Geometrie und Praktische Mathematik, RWTH Aachen

Berufsverbände

seit 01/1997	Gesellschaft für Informatik e.V.