MIP-based approaches for complex planning problems

MIP-based Approaches for Complex Planning Problems

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de
Technische Universiteit Eindhoven, op gezag van de
rector magnificus, prof.dr.ir. C.J. van Duijn, voor een
commissie aangewezen door het College voor
Promoties in het openbaar te verdedigen
op donderdag 10 december 2009 om 16.00 uur

door

John Johannes Jacobus van den Broek

geboren te Berghem

Dit proefschrift is goedgekeurd door de promotor:

prof.dr. G.J. Woeginger

Copromotor:
dr.ir. C.A.J. Hurkens

A catalogue record is available from the
Eindhoven University of Technology Library

ISBN: 978-90-386-2060-2

Acknowledgements

The work that resulted in this thesis started September 2004. Leen Stougie arranged a position as a PhD student at the Technical University of Eindhoven for me. For this I owe him many thanks. I enjoyed the period as a PhD student very much. That's why I would like to acknowledge those people who contributed to this and directly or indirectly to the thesis.

One person deserves to be mentioned first. I would like to express my gratitude to my supervisor Cor Hurkens. His great enthusiasm made the work challenging and he helped me out numerous times. We had many discussions and probably I was stubborn sometimes, but Cor seemingly had an endless patience. He read this thesis over and over again. Next to this he was always willing to help and advise me and takes care for a good sense of humor at the time.

Working together with Gerard Woeginger was a great pleasure to me. I am also grateful to him for letting me part of his group and being my promotor. I also want to thank Judith Keisper and Murat Firat for proofreading the introduction. It clearly improved this chapter. I always went with great pleasure to the TU Eindhoven. Next to the exciting work, this was due to the nice colleagues and the many coffee breaks with the other PhD students. Therefore, I want to thank al my (former) colleagues of the combinatorial optimization group. Our secretaries Harma and Kora deserve a special word of thanks. They assisted me with an enormous number of things.

During the PhD project I have been working for one day a week at the Netherlands Railways, which I enjoyed a lot. For me this was always a nice distraction from the work as a PhD student. Therefore, I would like to thank all my colleagues from the Netherlands Railways and the students that passed by in the last years.

Finally, I want to express my gratitude to my friends and family for the necessary relaxation and their support.

John van den Broek,
September 2009

Contents

Chapter 1

Introduction

Plans, schedules and timetables can be found everywhere around us in our daily lives. Examples are the Dutch railway timetable, the schedule for the Dutch soccer league and the roster of nurses in hospitals or home care. Creation of those plans, schedules or timetables is called *planning*, *scheduling* or *timetabling*. Planning, scheduling and timetabling problems are closely related problem classes and it is difficult to make a clear distinction between them. Sometimes there is even confusion to which class a problem belongs. We try to explain the main features that differentiate between planning, scheduling and timetabling.

Planning. In a planning problem a plan has to be found to achieve a certain goal. A plan is a sequence of actions specified with a certain time frame that transfers the initial state into one in which the desired goal is achieved. Planning defines what is done and how it is done. An example of a plan is the drawing of a house that has to be built. Another example is the plan listing on which track of the shunt yard train units have to stay during the night.

Scheduling. Scheduling is concerned with the optimal allocation of scarce resources (machines, processors, robots, operators, etc.) to activities over time, with the objective of optimizing one or several performance measures. Scheduling is deciding when and by whom a job is performed. Constraints like precedences, duration, capacity, and incompatibility have to be respected. The main difference between scheduling and planning is that in scheduling we know the set of activities in advance while in planning we have to generate the activities. This difference also explains the interaction between planning and scheduling: first we generate a set of activities and then we assign these activities to resources. An example of a scheduling problem is the assignment of train drivers to trains.

Timetabling. Timetabling is the allocation in space and time of given resources to events. Timetabling problems usually have more than one objective and often the objectives are at odds. In comparison with scheduling the importance of the resource allocation to events is small, the time at which an event is placed is the main issue. An example is a school timetable where a course is allocated to a classroom at a certain time. Another example is determining at what time and by which route a train unit has to be brought to its track on the shunt yard.

The difference between timetabling and scheduling is very subtle. It is difficult

to categorize a problem as one of the two. This is also not a main issue. What is important is that currently we are able to solve certain real-world timetabling and scheduling problems. Together with the increase in computing power, solution techniques for solving such real-world optimization problems improved a lot. Until recently most real-world timetabling and scheduling problems could only be handled by applying heuristics, but currently we are also able to prove optimality of a solution in a computation time that is acceptable for applications.

The rest of this chapter provides a more extensive introduction of the central problems and it introduces some concepts that are used throughout the whole thesis. The real-world optimization problems that we consider are all *combinatorial optimization* problems. Therefore, the next section contains a short introduction into the field of combinatorial optimization. Many combinatorial optimization problems can be formulated as a *Mixed Integer Program (MIP)*. In this thesis, solution techniques that use a MIP formulation of a problem are applied to solve complex planning, scheduling and timetabling problems. In Section 1.2 we introduce MIP problems. Section 1.3 is especially devoted to branch & bound, a solution approach for combinatorial optimization problems that returns throughout the whole thesis.

Section 1.4 provides an introduction into the area of educational timetabling. It marks two timetabling problems arising at universities that return in later chapters. Section 1.5 describes the shunt planning process for the larger railway stations in the Netherlands and introduces the problem that has our main interest. A simplified model of this problem is a well-studied scheduling problem. Therefore, we provide a short introduction into the field of scheduling in Section 1.6. An overview of the rest of the thesis is presented in Section 1.7.

1.1 Combinatorial optimization

An *optimization* problem is determining the minimum or maximum of a function $f(x)$, in which the domain of x forms a set of alternative values. The function f is called the *objective function* of this problem. The problem becomes more difficult when f is a function of more variables, like $f(x_1, \ldots, x_n)$. Now the goal is to find permitted values for the variables x_1 to x_n such that f is maximized or minimized. The variables $x_1, \ldots, x_n$ are called *decision variables*. An example of an optimization problem is the traveling salesman problem. A salesman lives in city X. On a certain day he has to visit a set of other cities and at the end of the day he wants to be back in city X. He wants to know in what order he should visit the cities? The objective that he wants to achieve is traveling the minimum total distance.

The values of decision variables can be restricted by *constraints*. An example of a constraint is that the sum of the decision variables x_1 and x_2 has to be equal to one. The set of *feasible solutions* of an optimization problem is the set that contains all combinations of the values in the domains of the decision variables that obey all constraints. Solving an optimization problem is finding a feasible solution that provides the maximum or minimum objective function value. Such a solution is called an *optimal solution*.

If the decision variables can take real values we call the problem a *continuous optimization* problem. If the domains of the variables form a countable or countably infinite set, then the problem is called a *combinatorial optimization* problem. An example of such a set is the set $\mathbb{N} = \{0, 1, 2, \ldots\}$ of integer numbers.

Most optimization problems have corresponding decision versions. For a decision problem there is given a constant k and then the following question has to be answered: "Does there exist a feasible solution with objective value less than or equal to k?". So decision problems always have a "yes" or "no" answer. The decision variant of the traveling salesman problem answers the question whether there exists a tour that visits all the cities and has length at most L.

The complexity of a problem and the available computation time determine what kind of solution approach fits the best. To help making such decisions, *computational complexity* has been introduced. Very small instances of a problem can always be solved to optimality by a computer reasonably fast. But how do the computation time and memory usage increase if the size of the input increases? This is the main question in computational complexity.

To classify the complexity of problems, complexity classes have been introduced. An algorithm is said to run in polynomial time if its running time is bounded by a polynomial function of the input size. The class P is the class of problems that can be solved in polynomial time. For problems in this class an algorithm exists that is able to solve any problem instance within polynomial time. Examples are the shortest path problem and the greatest common divisor problem. The complexity class NP contains all decision problems that for a yes-instance have a certificate that is verifiable in polynomial time. A certificate for a yes-instance of the traveling salesman problem is a tour of length at most L. Note that the complexity class P is contained in NP. If a decision problem is contained in NP, then also its optimization variant is contained in NP.

No polynomial time algorithm is known for decision problems that are in the complexity class NP-complete. A decision problem is NP-complete if it belongs to NP and if it is at least as difficult as any other problem in NP. A proof that a decision problem p is NP-complete consists of two steps:

1. A proof that it belongs to NP.
2. A polynomial time reduction from another problem, which is known to be NP-complete, to problem p.

The optimization variant of an NP-complete problem is said to be NP-hard. For an extensive introduction in complexity theory we refer to Garey and Johnson (1979).

If a combinatorial optimization problem arises in practice, the first question to answer is what kind of solution approach is required. The possible solution approaches can be divided into three classes:

1. Exact algorithms: provide a global optimal solution, but may require a high computation time.
2. Approximation algorithms: provide a solution with a worst case performance guarantee within acceptable time.
3. Heuristics: provide a solution in an acceptable time, but nothing can be said about their quality.

An *exact algorithm* always finds an optimal solution and also guarantees that no feasible solution exists with a better objective value. Those algorithms generate feasible solutions one by one and use clever elimination rules to prevent the need to search the complete space of feasible solutions. The computation time of an exact algorithm can be too large for practical use. Examples of exact algorithms are Dijkstra's algorithm and branch & bound.

If an exact algorithm takes too much computation time or solving a problem to optimality is not required, then an *approximation algorithm* or *heuristic* can be developed. An approximation algorithm provides a solution with an objective value that is never worse than the *performance guarantee* ρ times the optimal solution value within a short time. In the case of a minimization problem $\rho \geq 1$ and for a maximization problem $\rho \leq 1$. An approximation algorithm with a performance guarantee equal to one is an exact algorithm.

A heuristic also finds a solution within a small amount of time, but without a worst case performance guarantee. The solution found could be arbitrarily bad, but in reality a good heuristic often results in a good solution. The quality of a heuristic is evaluated by its performance in practice or by comparing its computational results with the computational results of other heuristics on the same instances. Examples of heuristics are greedy heuristics and metaheuristics like local search and genetic algorithms. For the interested reader we refer to Aarts and Lenstra (1997) for a more in depth discussion on local search methods and to Talbi (2009) for a detailed discussion on metaheuristics.

Most real-world problems that we consider in this thesis are NP-complete. They have in common that they can be formulated as mixed integer programs. For each of the problems an exact or heuristic algorithm has been developed that provides a good or even optimal solution in a computation time that is short enough for practical use. Each of the algorithms is based on a mixed integer programming formulation of the problem. Therefore, we first provide an introduction in mixed integer programming.

1.2 Mixed integer programming

We first introduce the class of **linear programming (LP)** problems. An LP is a problem that has a linear objective function, subject to linear equality and/or linear inequality constraints. The decision variant is defined as follows:

> **Instance:** $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$, $c \in \mathbb{R}^n$ and a constant k.
>
> **Question:** Does there exist a vector $x \in \mathbb{R}^n$ such that $Ax \geq b$ and $c^T x \leq k$?

Any linear program can be solved in polynomial time, Khachiyan (1979). Hence, the class of LP problems is an element of the complexity class P.

If the decision variables are all required to be integers, then the problem is called an **integer program (IP)**. The special case that the objective function and all constraints are linear is called an **integer linear program (ILP)**. Often the word linear is omitted. A **mixed integer linear program (MIP)** is a linear program

with a set of linear constraints and a linear objective function, but at least one of its decision variables is required to be integer. The class of MIP problems belong to the complexity class NP-complete.

All real-world problems in this thesis have been formulated as MIP problems. The main question for these problems was,"Is it possible to find an optimal solution and also prove that it is optimal within an acceptable computation time using techniques for solving MIP problems". A solution approach that comes back in a few chapters of this thesis is branch & bound. Therefore we devote a separate section to this solution procedure.

1.3 Branch & bound

In a branch & bound algorithm an implicit search through the space of all possible feasible solutions is done. The solution process of a branch & bound algorithm can be represented with a search tree. A node of the search tree corresponds to a subset of the set of feasible solutions of the problem. Initially, this tree consists of one node which is called the *root* of the tree. This root node represents all feasible solutions. A leaf of the enumeration tree contains at most one not necessarily feasible solution.

A *branching rule* specifies how the set of feasible solutions of a node is partitioned into subsets, each corresponding with a descendant node of the current node. An example of a branching rule is rounding up, respectively rounding down a non-integer decision variable of the LP-relaxation of the MIP problem corresponding with the node. The LP-relaxation of a MIP is the linear program that results if the integrality constraints are removed. If an integer variable x_i in the optimal solution of the LP-relaxation has a value $\hat{x_i} = 2.5$, then one branch of the tree contains the feasible solutions with $x_i \leq 2$ and the other branch of the tree the feasible solutions with $x_i \geq 3$ when this branching rule is applied.

In each node of the search tree a *lower bound LB* and *upper bound UB* are determined. Let us assume for the rest of this section that we want to solve a minimization problem. Then an upper bound *UB* is the objective value of the best known feasible solution of the original problem. This upper bound is updated each time a new feasible solution is found with objective value less than the current *UB*. A lower bound *LB* bounds the objective values of all feasible solutions in the node. The feasible solutions in the resulting tree can not obtain an objective value less than *LB*. If no feasible solution exists in the node, then $LB = +\infty$. Lower bounds are most often obtained by solving a relaxed problem. A common lower bound is the optimal objective value of the LP-relaxation. The LP-relaxation is less restrictive than the corresponding MIP, hence the set of feasible solutions of the LP contains the set of feasible solutions of the MIP. Therefore, the optimal solution value of the LP-relaxation is never worse than the value of an optimal solution of the MIP and can be used as a lower bound.

If a node has a *LB* that is greater or equal than the currently best known solution value *UB*, then there is no need to continue the search any further in this part of the tree as it can never improve the existing upper bound. Then this node is *fathomed*, this means that all descendant nodes are disregarded. If a node is fathomed or a leaf

node is found, then the search backtracks to the deepest unfathomed node in the tree.

At any time during the search there is a set of nodes in the current search tree representing subspaces to be investigated. These nodes are called the active nodes. From the set of active nodes we have to choose a node from which we are expanding the search tree. A *search strategy* determines the order in which the nodes of the search tree are selected for branching. Examples of search strategies are depth-first and best-bound. With a depth-first search strategy the deepest node in the search tree is always selected. The node with the lowest lower bound is selected if a best-bound search strategy is applied.

A powerful tool for branch & bound algorithms are *dominance rules*. Those rules attempt to eliminate nodes prior to the computation of the lower bounds. Let us consider as example a scheduling problem with one machine. If in the current node of the search tree job A is scheduled before job B and job B is scheduled before job C, this implies that job A has to be scheduled before job C in all nodes below this one.

Crucial in a branch & bound algorithm is the computational effort done in each node. A good balance between the number of nodes investigated and the computational effort in each node determines the success of the algorithm.

A possibility for solving mixed integer programming problems is the use of a general purpose solver. A highly sophisticated general purpose solver that we will use throughout the thesis is CPLEX. If the input of CPLEX is an IP or MIP, then CPLEX applies branch & bound to solve it. The method used for finding lower bounds is always solving the LP-relaxation of the MIP. A lot of combinatorial optimization problems can be solved in a reasonable amount of time by formulating it as a MIP and using CPLEX to solve the MIP. Whether CPLEX is able to solve the MIP in a reasonable amount of time depends on the complexity of the problem, but also on the type of constraints and the MIP formulation that is applied. A disadvantage of a general purpose solver like CPLEX is that it is not able to use problem specific information.

1.4 Educational timetabling

The literature contains an enormous list of real-world applications where timetables are very important. For example: aviation, public transport, operating rooms, schools, universities and so on. Our focus was on timetabling problems that arise at educational institutions, mainly universities. Two timetabling problems have been studied and the results of this research are presented in the first part of this thesis. The goal of this section is to position the two timetabling problems within the huge amount of literature on educational timetabling problems.

Typical for most educational timetabling problems is that a constructed timetable has to satisfy a set of hard constraints to be feasible and should violate as few soft constraints as possible. A typical hard constraint is that a student can only do one course at a time. An example of a soft constraint is that students preferably do not have only one class a day. The goal of such a timetabling problem is usually to minimize the number of violated soft constraints. Individual institutions can weigh

these soft constraints according to their own preference. Most educational timetabling variants differ from each other by the type of institution involved (university or high school) and by the type of constraints.

The annotated bibliography of timetable construction by Schmidt and Ströhlein (1980) lists many papers that appeared before 1980. Surveys of timetabling methods and applications are De Werra (1985), Burke et al. (1997), Carter and Laporte (1997) and Burke and Petrovic (2002). Schaerf (1999) gives a survey of various formulations of timetabling problems and classifies educational timetabling problems into the following three main classes:

1. School timetabling

2. Examination timetabling

3. Course timetabling.

Of course this classification is crude, and there are many real-world timetabling problems that fall in between two of these classes. We will now briefly explain the three classes.

The basic *school timetabling* problem is also known as the *class-teacher model*. A class is a group of students taking an identical set of courses and typically remaining together throughout the week. The simplest version of the problem consists in assigning lectures to periods in such a way that no teacher or class is involved in more than one lecture at a time. Even et al. (1976) proved that there always exists a solution of this simplest version of the school timetabling problem, unless a teacher or class is involved in more lectures than there are time slots. Alternative formulations of the school timetabling problem with more constraints can be found for example in Even et al. (1976), Garey and Johnson (1979) and De Werra (1985). Daskalaki and Birbas (2005) provide an integer programming formulation of the class-teacher problem and solve it with a two-stage relaxation procedure.

Most common from a practical perspective is the model of *examination timetabling* in which students are considered to be enrolled on particular courses, McCollum et al. (2007). Exams have to be scheduled into a number of periods within an examination session while satisfying all hard constraints. Examples of hard constraints are: a student can do at most one exam at a time and the capacity of a room can not be exceeded. Examples for additional soft constraints are: students can do at most one exam per day; students may not have too many consecutive exams; exams that share a room should have the same duration. Schaerf (1999) gives an ILP formulation of the examination timetabling problem and describes some alternative variants of the problem. An overview on examination timetabling has been provided by Carter and Laporte (1995). McCollum et al. (2007) present the examination timetabling problem that was track 1 of the second international timetabling competition (ITC2), see McCollum et al. (2009) and ITC (2007). The introduced model and datasets in this track are based on real world instances.

Course timetabling is the weekly scheduling of a set of lectures for a set of university courses within a set of rooms and timeslots, minimizing the overlaps of lectures

having common students, Schaerf (1999). De Werra (1985) gives a binary integer programming formulation of the basic course timetabling problem. An overview on practical course timetabling problems has been given by Carter and Laporte (1997). Schaerf (1999) discusses some of the most common variants of the basic formulation. There are two different types of course timetabling problems, namely *curriculum-based course timetabling* and *post enrolment course timetabling*. Both were a separate track within ITC2.

In the curriculum-based timetabling problem conflicts arise according to the curricula published by the university and not on the basis of enrolment data. Hard constraints in the curriculum-based timetabling problem are for example: all lectures of a course must be scheduled to different time slots, no two lectures can take place in the same room on the same time slot, lectures in the same curriculum or with the same teacher have to be scheduled in different time slots. Also teacher availability is usually taken into account. Typical soft constraints are: capacity of rooms can not be exceeded, lectures of the same course should be spread over the week and lectures of the same course have to be assigned to the same room. Track 3 of ITC2 was a curriculum-based course timetabling problem. The datasets have been composed by real-world instances provided by the University of Udine, Gaspero et al. (2007).

In a post enrolment course timetabling problem students select the lectures that they wish to attend. Then a timetable has to be constructed according to these choices and no overlap of courses containing the same students is allowed and no two courses can be assigned to the same room. The post enrolment course timetabling problem in ITC2 has also been used in the first International Timetabling Competition (ITC1), ITC (2002). The timetabling problem of ITC1 is well described by Lewis and Paechter (2007). The model that was used in track 2 of ITC2 is an extension of this model, Lewis et al. (2007). This model is the central topic of chapter 2. For a detailed description of this post enrolment course timetabling problem we refer to that chapter.

Course timetabling and examination timetabling can both be classified as university timetabling. Petrovic and Burke (2004) discuss university timetabling problem statements and give an overview on recent research results. The main differences between course timetabling and examination timetabling are that examination timetabling has only one exam for each course, that the time conflict condition for a student is strict, and that several exams can share a room.

McCollum (2006) explains that for university timetabling there is still a gap between successful research projects and what is needed in practice. He tries to bridge this gap between research and practice by providing up-to-date information from practice which can be used by researchers. Burke et al. (2006) and Zampieri and Schaerf (2006) remark that many of the search methodologies described in the literature are not applicable in most educational institutions, because they are simplified too much. Carter and Laporte (1997) note that they were "somewhat surprised to discover that there are very few course timetabling papers that actually report that the (research) methods have been implemented and used in institution". McCollum (2006) explains the situation has hardly changed in the last decade. In Chapter 3, we describe a MIP model for a specific course timetabling problem that came up at the department of

Industrial Design of the Technical University of Eindhoven. The implementation was successfully used to obtain a good timetable.

1.5 Planning problems arising at railway stations

The main operator of trains in the Netherlands is Netherlands Railways (NS). The company transports more than 1.1 million passengers in the Netherlands each day. All together, these passengers traveled 16.2 billion kilometers in 2008. This is done by almost 5,000 scheduled train services a day on lines arranging in length between 15 and over 200 kilometers. The Dutch railway network consists of approximately 3,000 kilometers of railway track and almost 400 stations. This makes the Dutch railway network one of the most heavily utilized networks of the world.

Each year the number of train services accomplished by NS grows while the capacity of the infrastructure hardly increases. Especially the infrastructure at railway stations usually does not change, because they are in or around the city center where space is lacking. As a consequence, more trains have to stop at a station or pass a station while the capacity of the stations does not increase. The capacity of the railway stations has become one of the main bottlenecks in the logistic planning proces of NS. Constructing an executable plan for all train movements at a railway station has become an impractical job.

Next to all the timetabled passenger and cargo trains, many shunting movements between platform tracks and shunting areas are carried out in and around the large stations in a railway network. Shunting movements have to provide passenger trains with the right composition of rolling stock. The rolling stock for trains that start their duty sometimes has to be picked from the shunt yard and the rolling stock of ending trains sometimes has to be brought to a shunt yard. During rush hours, passenger trains are usually operated at full capacity. However, outside rush hours an operator of passenger trains usually has a surplus of rolling stock. This surplus has to be parked at a shunting area in order to be able to fully exploit the main railway infrastructure. So especially just before and just after the peak hours a lot of shunting movements are necessary.

Shunting processes are highly dependent on the timetable and on the rolling stock circulation of a railway operator. First, the shunting movements share the capacity of the railway infrastructure with the timetabled passenger and cargo trains. Moreover, as soon as the planned timetable or the planned rolling stock circulation is modified, the number, the order, and the compositions of the shunting movements usually change as well. Therefore, also the shunting plans have to be modified in such cases. Currently this requires a lot of manual planning work. Because of this dependency on the rolling stock circulation and the timetable, the shunt planning is performed shortly before execution which results in a high time pressure. Advanced decision support systems would help shunt planners a lot in speeding up this shunt planning.

Typical in the Netherlands is that most train services are operated by train units that can move in both directions without a locomotive. Only train units of the same type can be combined to form one longer train. Mostly, train units of the same type

and the same number of carriages can be used interchangeably.

The shunt planning problem at NS is split into three subproblems:

1. A matching problem for arriving and departing train units
2. A parking problem for storing the train units at the shunting tracks
3. A routing problem for routing train units to the shunting tracks

Those subproblems have been introduced by Freling et al. (2005) who aimed at the development of planning tools that support planners to generate detailed shunting plans from scratch. More details on the shunt planning process at Netherlands railways can be found in Lentink (2006).

A fourth element of the shunt process is the availability of shunting crew. Coupling, decoupling and shunting movements are examples of tasks that have to be done by shunting crew. These tasks are usually performed by local shunting crews, but drivers operating timetable trains can also perform some of these tasks. More information on the shunting crew scheduling problem can be found in Hoekert (2001). In comparison with the infrastructure, crew is a relatively cheap resource and there is usually a sufficient number available to carry out the shunting movements. Therefore, the crew planning problem is not taken into account in this thesis. We will now briefly introduce the three subproblems.

Given a timetable with arrival and departure times and compositions of the arriving and departing trains, a planner has to match arriving and departing train units. A large part of this matching is prescribed by the timetable, e.g. the matching of the arriving and departing train units of through trains, which continue passenger service after a short dwell time. The main goal in this matching subproblem is minimizing the number of times that train units within the same train service have to be split.

In the parking subproblem positions and compositions of the train units at the shunt yard have to be determined, such that the operations in the next morning can start up as smoothly as possible. The order of train units on a shunt track is especially important if there are train units of different types or of different number of carriages on the same track. Not all tracks have catenary installed, which implies that the train units can not be positioned there. Another restriction is that some tracks are less desired if they are used frequently for other purposes, e.g. for through train services or for temporary parking of train units.

Kroon et al. (2008) present a model that is capable of solving the matching and parking subproblem in an integrated manner. Their model incorporates that trains are composed of several train units and tracks can be approached from two sides. Their model minimizes the number of train units from the same train service that have to be split and the number of tracks with multiple train units of different type or length that are parked at it.

After solving the matching and parking subproblem, routes and time instants for the necessary shunting movements have to be established. This routing subproblem forms the basis of the second part of the thesis and will be explained in detail in Chapter 4.

1.6 Machine scheduling

The study of machine scheduling started more than fifty years ago and was initiated by seminal papers of Johnson (1954) and Bellman (1956). Since then machine scheduling has received a lot of attention. As a result, a great diversity of scheduling models and optimization techniques have been developed with applications in industry. The interested reader can find many nice readings on scheduling theory in textbooks, monographs and handbooks. Books have been written by Conway et al. (1967), Baker (1974), French (1982), Pinedo (2005), Brucker (2007) and Blazewicz et al. (2007). An extensive review paper has been presented by Chen et al. (1998).

In this section we introduce some terminology and notation that is common for machine scheduling problems. A *processor* or *machine* is a resource that can perform at most one activity at a time. These activities are commonly referred to as *operations*. Each operation j has a process time p_j, the minimum time that it needs to spend on a machine. Every *job* or *task* consists of a set of operations that have to be performed, possibly with *precedence constraints* between the operations. A precedence constraint between two operations means that an operation cannot start as long as its predecessor operation has not been completed. A *schedule* is an allocation of each operation to one or more machines during a time interval. Such a schedule is feasible if no two time intervals on the same machine overlap and other requirements concerning the machine environment and operation characteristics are met.

Machine scheduling problems are commonly classified by the three-field notation: $\alpha|\beta|\gamma$, where α represents the machine environment, β the job characteristics and γ the optimality criterion. This three-field notation has been introduced by Graham et al. (1979).

There are a lot of different *machine environments*. We distinguish between single stage and multi stage scheduling problems. In a single stage problem every job consists of one operation and the machine environment is denoted by $\alpha \in \{1, P, Q, R\}$. If each job has to be operated on the same machine, then it is a single machine environment ($\alpha = 1$). In the case of a parallel machine environment a job can be operated on any one of the machines available. If $\alpha = P$ all machines are identical. Machines have different speeds if $\alpha = Q$ and in the case of unrelated machines ($\alpha = R$) each job has a different process time on each machine.

In a multi stage scheduling problem a job can have more than one operation and the machine environment is denoted by $\alpha \in \{F, O, J\}$. In a *flow shop* ($\alpha = F$) all jobs follow the same machine sequence and each job has exactly one operation on each machine. Whenever a job has been completed at one machine it joins the queue at the next machine, so the job sequence on machines may vary. An *open shop* ($\alpha = O$) problem is similar to flow shop, except that operations of a job may be performed in any order, so the order of the operations in a job is chosen by the scheduler. In a *job shop* ($\alpha = J$) jobs follow different prescribed machine routes and may use the same machine more than once. Note that a flow shop is a job shop in which every job has the same route.

The second field of the three-field notation contains the *job characteristics*, like existence of precedence constraints, restrictions on process times and whether pre-

emption is allowed. Preemption is interrupting a job to complete it at a later time. Also possible properties of an operation j are added to the second field. Examples of such properties are the release date r_j and deadline $\overline{d}_j$. The release date r_j is the first possible start time of operation j and the deadline $\overline{d}_j$ is the last completion time allowed for operation j.

The *objective* of scheduling problems is put in the third field. One of the most used objectives is minimizing the completion time of the last completed job, commonly referred to as the makespan C_{max}. Fixed data that is often used is the due date d_j of an operation j. If an operation j is completed later than the due date ($C_j > d_j$), then a certain penalty has to be paid. Dynamic data in a machine scheduling problem depend on the schedule and are not fixed in advance. The most commonly used dynamic data are:

- $L_j = C_j - d_j$: the lateness of operation j.
- $T_j = \max\{L_j, 0\}$: the tardiness of operation j.
- $U_j = \begin{cases} 1 & \text{if } C_j > d_j, \\ 0 & \text{otherwise} \end{cases}$

Next to minimizing the makespan, other possible objectives are minimizing maximum lateness, minimizing the number of late jobs or minimizing the total tardiness. A lot more objectives are possible and also their weighted equivalents are used.

One of the most popular models in scheduling theory is that of the job shop $(J||C^{max})$. It has earned a reputation for being difficult to solve and is probably the most studied model in deterministic scheduling theory. It serves as a comparative problem for different solution techniques. Assuming an instance has m machines and n jobs with m operations that have to be performed on a different machine, then there are $(n!)^m$ possible solutions. For a problem with 10 jobs and 10 machines this gives $3.959.10^{65}$ possible solutions. Even though many solutions are not possible, complete enumeration is not practical. In Chapter 5 job shop problems that have special types of precedence constraints are studied.

1.7 Outline of the thesis

Until recently most real-world planning, scheduling and timetabling problems are taken on by heuristics. Disadvantage of those heuristics is that nothing can be said about the quality of the solutions. Due to the increase in computing power and the improvement in solution techniques much more real-world problems can be solved by exact algorithms. Our goal in this thesis is to explore whether certain real-world planning problems can be solved to optimality with sophisticated solution approaches, mainly based on MIP-based techniques. If a problem can not be solved to optimality in a reasonable amount of time, then a heuristic is developed that also applies MIP-based techniques.

Three real-world planning problems have our main concern. Two post enrolment course timetabling problems are the central topic of Chapters 2 and 3. The first

problem is the post enrolment course timetabling problem of the second international timetabling competition, ITC (2007). We have formulated the problem as a MIP, but the general purpose solver CPLEX was even not able to solve the LP-relaxation in the allowed computation time. Therefore, we choose to develop a heuristic based on the extended MIP formulation of this problem. The second problem is a real-world timetabling problem originating from the department of Industrial Design at the TU Eindhoven. We show that it is NP-complete, but fortunately it can be solved by modeling it as a MIP formulation and solve it with CPLEX.

The third real-world problem is the routing problem of the shunt planning proces that arises at larger railway stations. This problem is described in detail in Chapter 4. We studied a restricted and a full version of the problem and formulated both as a MIP. The restricted version assumes that the route of each train is fixed. CPLEX was able to solve this problem for all railway stations in a very small computation time. In the full version of the problem, also a route has to be chosen. For some railway stations CPLEX was not able to solve the problem in a practical computation time. Therefore, we also studied a simplified model of this shunting problem, namely the no-wait and blocking job shop scheduling models. These models are the topic of Chapter 5. In this chapter we present a MIP-based heuristic and branch & bound algorithms that we developed for those problems. Each chapter is intended to be self-contained, such that it can be read separately. The contents of the chapters in more detail is the following.

Track 2 of the second International Timetabling Competition, ITC (2007), was a post enrolment course timetabling problem. A set of events had to be assigned to a timeslot and to a room such that all students were able to attend their requested events. The assignment had to satisfy certain hard constraints and should violate as few soft constraints as possible. Chapter 2 describes the problem in detail and a heuristic is presented that assigns events to timeslots based on an LP solution constructed with column generation. We get an integer solution by fixing columns one at a time. This heuristic finds a complete solution for all 24 instances of the competition in a small amount of computation time. The generated solution is improved by selecting a set of events that are reassigned by solving a MIP. This MIP minimizes the number of soft constraint violations. In comparison with the five finalists of the competition, our heuristic is more successful in finding a solution with all events assigned to a room and timeslot.

Chapter 3 describes a successfully implemented solution approach for a post enrolment course timetabling problem of the Department of Industrial Design at the TU Eindhoven. The students of this department are allowed to design a part of their curriculum themselves by selecting courses from a huge course pool. They do this by handing in a list of preferred courses for the forthcoming time period. Based on this information (and taking into account many other constraints), the department assigns courses to students. Until the end of 2005, this assignment was computed by human schedulers who used a quite straightforward greedy approach. In 2005, however, the number of students increased substantially, and as a consequence the greedy approach did not yield acceptable results anymore.

This chapter discusses the solution of this real-world timetabling problem. We

present a complete mathematical formulation and explain all the constraints resulting from the situation in Eindhoven. The problem has been solved using lexicographical optimization with four subproblems. For all four subproblems, a MIP formulation is given which can be put into CPLEX. Finally, we report on our computational experiments and results around the Eindhoven real-world data.

Chapter 4 presents two MIP formulations for the routing problem that has been introduced in Section 1.5. Both models try to schedule the necessary shunting movements in between the passenger and cargo trains, such that the number of shunting movements that can not be planned because of lack of capacity of the infrastructure is minimized. In the first model the routes of the trains are fixed beforehand and in the second model the routes are determined by the model. The computation times for solving this last MIP model with CPLEX were too large for practical use.

Because of those large computation times we tried to get a better understanding of the problem by looking at a simplified model of the shunting problem. This simplified model is the job shop scheduling problem with blocking and no-wait precedence constraints. The traditional job shop scheduling problem assumes that there are buffers with infinite capacity available to hold jobs between the finishing on one machine and the start of processing on the next machine. However, in many production processes and also in our routing problem these intermediate buffers are not available. The class of machine scheduling problems without intermediate buffers can be characterized by *no-wait* and/or *blocking* precedence constraints. In a no-wait environment an operation has to start immediately after its predecessor operation has been completed. So a job must be processed from start to completion without any interruption either on or between machines. Blocking occurs when a job, whose processing has been completed on a machine, remains on the machine until the successor machine becomes available for processing. This implies that the actual time an operation keeps its machine occupied is not known in advance. Several branch & bound algorithms and the corresponding computational results on benchmark instances of the traditional job shop are presented in Chapter 5.

Chapter 2

Post enrolment course timetabling problem of the ITC2007

One of the major difficulties in the field of university timetabling is that the proposed algorithms are mostly tested on data of the university of the researchers. In addition to this, almost all universities have different timetabling problems, mainly different types of constraints. This makes a fair comparison between different algorithms very difficult. Therefore, the second International Timetabling Competition (ITC2) has been organized, McCollum et al. (2009), ITC (2007). The competition contained three tracks. The second track was the post enrolment course timetabling problem, which is the subject of this chapter.

Given is a set of students that have selected lectures that they wish to attend. Within the competition lectures are called events. The events have to be assigned to timeslots and rooms such that there are no events that are requested by a common student are assigned to the same timeslot. The post enrolment course timetabling problem in ITC2 has also been used in the first International Timetabling Competition, ITC (2002). In ITC2 two extra hard constraints have been introduced to make the problem more realistic and more difficult to find a timetable in which all events have been assigned to a room and timeslot.

Especially the last decades local search techniques became popular for solving course timetabling problems, see for example Kostuch (2004) and Rossi-Doria et al. (2002). Several of the finalists of track 2 of the ITC2 applied a local search technique. Cambazard et al. (2008) use several techniques, in particular simulated annealing, constraint programming and hybrids of those in a large neighborhood search scheme. With their simulated annealing augmented with a large neighborhood search move at the low temperatures they won the competition. Atsuta et al. (2008) use a general CSP solver. This solver is a hybrid algorithm of tabu search and iterated local search. A stochastic local search approach consisting of several heuristic modules in different phases of the solution process is applied by Chiarandini et al. (2008). All modules were tuned with the automated algorithm configuration procedure ParamILS, Hutter et al. (2007). The fourth finalist of track 2 was Mayer et al. (2008). They developed a heuristic based on ant colony optimization. Muller (2008) has been the fifth finalist. He developed a constraint-based framework containing a series of local search algo-

rithms that operate over feasible, not necessarily complete, solutions. His solver won the tracks 1 and 3 of ITC2 and ended number 5 in track 2.

Our main contribution is a totally different solution approach for the post enrolment course timetabling problem of track 2. We use integer programming techniques and introduce a new deterministic construction heuristic using column generation.

The next section provides a detailed description of the problem and introduces our notation. Section 2.2 describes a construction heuristic that assigns events to timeslots based on a solution of the LP-relaxation of an extended ILP formulation of the problem. This solution is found by applying column generation. After construction of a solution we try to improve the solution by solving small subproblems with a compact ILP formulation. Our improvement procedure is presented in Section 2.3. Section 2.4 gives our results and compares them with the results of the five finalists.

2.1 Problem description

Given is a set E of events that have to be assigned to a timeslot from the set T of timeslots and to a room from the set R of rooms. Each weekday contains nine possible timeslots, therefore $T = \{0, \ldots, 44\}$. Every room $r \in R$ has a set of features F_r and a specific seating capacity C_r. In addition to this we have a set S of students that all have a set E_s of events that they are attending. Every event e has a set T_e of timeslots to which it can be assigned, a set F_e of features that it requires and N_e students attending the event. An event e can be assigned to room r if room r satisfies all features that event e requires ($F_e \subseteq F_r$) and the seating capacity of room r is sufficient for the number of students attending ($N_e \leq C_r$). We define R_e as the set of rooms to which event e can be assigned. Also precedence constraints are given that state that certain events should be scheduled earlier in the week than other events. Precedences between events are modeled with the parameter p_{ef}. It has value one if e is a predecessor of f.

In summary: The input consists of:

- set T of 45 timeslots $\rightarrow T = \{0, \ldots, 44\}$.
  - $\forall t \in T$: set E_t of events that can be assigned to timeslot t.
- set R of rooms.
  - $\forall r \in R$: set E_r of events that can be assigned to room r.
- set E of events.
  - $\forall e, f \in E$: $p_{ef} = 1$ if event e has to be scheduled before event f.
  - $\forall e \in E$: N_e = number of students attending event e.
  - $\forall e \in E$: set T_e of timeslots to which event e can be assigned.
  - $\forall e \in E$: set R_e of rooms to which event e can be assigned.
- set S of students.
  - $\forall s \in S$: set E_s of events that student s attends.

Two events *conflict* if they can not be assigned to the same timeslot. This arises if they have a student in common, if both have only one possible room which is the same or if there is a precedence relation between the two events. We define C_e as the set of events that conflict with event e.

In addition to the fact that a timetable has to be created such that all students can attend the events they request, a timetable should be of a good quality. Therefore, the constraints in the timetabling problem can be divided into soft and hard constraints. The hard constraints need to be satisfied in order for the timetable to be feasible. Soft constraints have to be satisfied as much as possible to provide a timetable of a better quality.

The five hard constraints that have to be satisfied are:

1. No student can attend more than one event at a time.
2. An event e can only be assigned to a room $r \in R_e$.
3. At most one event is assigned to each room in any time slot.
4. An event e can only be assigned to a time slot $t \in T_e$.
5. Events have to be scheduled in the prescribed order during the week.

The soft constraints that have to be satisfied are:

- Events should not be assigned to the last timeslot of a day (that is: $t = 8, 17, 26, 35$ and 44).
- Students should not have to attend three or more events in successive timeslots on the same day.
- Students should not be required to attend only one event a day.

Because feasibility could be difficult the organizers made the distinction between a *valid timetable* and a *feasible timetable*. For both timetables hard constraint violations are not permitted. For a valid timetable it is allowed that events are left out of the timetable. A feasible timetable is one in which there are no hard constraint violations and all events are assigned. All solutions have to be valid. The quality of valid solutions is evaluated with two measures:

1. Distance to feasibility (dtf)
2. Total number of violated soft constraints.

The *distance to feasibility* is equal to the number of students that want to attend an unplaced event. A feasible timetable always has a distance to feasibility of zero. The valid solution with the lowest distance to feasibility is the better solution. If two valid solutions have the same distance to feasibility, then the solution with the minimum number of violated soft constraints is preferred.

Lewis et al. (2007) provide an exact description of the problem of track 2 of ITC2.

2.2 Construction heuristic based on LP solution

A compact MIP formulation of the problem, including the soft constraints, will be presented in Section 2.3. Unfortunately, solving this compact MIP formulation is not possible within the available computation time, even not without the soft constraints.

Even worse, CPLEX did not start branching within the available computation time. Therefore, we decided to develop heuristics.

We formulate the problem without the soft constraints as an extended ILP formulation. The LP-relaxation of this extended ILP is solved using column generation. After solving this LP-relaxation we fix one column. This corresponds with definitely assigning a set of events to a certain timeslot. Then the LP-relaxation is solved again. By solving the LP-relaxation every iteration, we guarantee that the events are spread over the week.

An introduction to column generation is provided by Desrosiers and Lübbecke (2005) and Lübbecke and Desrosiers (2005). They also give an overview of recent papers on this topic. Numerous applications in which column generation is used are described in the literature. Desaulniers et al. (2001) apply column generation for vehicle routing problems. Crew scheduling is another hot topic for which column generation is used a lot, see for example Desaulniers et al. (2001) and Borndörfer et al. (2003). Many more examples of applications can be found in Desrosiers and Lübbecke (2005).

We are aware of only one paper that applied a column generation approach on a course timetabling problem, namely Qualizza and Serafini (2004). It considers a curriculum-based course timetabling problem where a weekly schedule of courses has to be made with constraints like "not two courses in one classroom at the same time" and "no overlap in timeslots of certain groups of courses". In the extended ILP formulation each column corresponds with a weekly timetable of a single course.

Our main goal of the heuristic described in this section is the construction of a feasible solution. Therefore, our focus is on assigning all events to a room and timeslot without violating one of the hard constraints. In the first subsection the extended LP formulation is introduced. The column generation procedure to solve this LP is described in the second subsection. How new columns for the LP are generated is presented in Subsection 2.2.3. Subsection 2.2.4 covers the heuristic that constructs an integer feasible solution using the feasible solution of the LP.

2.2.1 The extended LP formulation

The LP formulation described in this subsection is a formulation of the post enrolment course timetabling problem without the soft constraints. We first introduce the set K of *slot-schedules*. A *slot-schedule* $k \in K$ is characterized by a timeslot t_k, a set E_k of events and a matching $\rho_k : E_k \to R$. The matching ρ_k describes a feasible room assignment for all events in the slot-schedule k. A slot-schedule k is feasible if:

- $\forall e, f \in E_k : e, f$ do not conflict.
- $\forall e \in E_k : t_k \in T_e$.
- $\forall e \in E_k : \rho_k(e) \in R_e$
- $\forall e \neq f \in E_k : \rho_k(e) \neq \rho_k(f)$

For all slot-schedules $k \in K$ and all events $e \in E$ we introduce the parameter a_{ek}, which is one if $e \in E_k$ and zero otherwise.

The LP formulation has three different types of decision variables. These are:

- $\forall k \in K : x_k = 1$ if slot-schedule k is selected, $x_k = 0$ otherwise.
- $\forall e \in E : y_e = 0$ if event e is assigned to a timeslot, $y_e = 1$ otherwise.
- $\forall e, f \in E, p_{ef} = 1 : z_{ef} \in \mathbb{Z}_+$ is the violation of the precedence constraint between e and f.

Now the decision variables and parameters have been introduced, hence we are able to present the LP:

$$\min \sum_{e \in E} y_e + \sum_{e,f \in E | p_{ef}=1} z_{ef}$$

$$y_e + \sum_{k \in K} a_{ek} x_k \geq 1 \qquad \forall e \in E \tag{2.1}$$

$$\sum_{k \in K | t_k = t} x_k \leq 1 \qquad \forall t \in T \tag{2.2}$$

$$z_{ef} + \sum_{k \in K} t_k (a_{fk} - a_{ek}) x_k \geq 1 \qquad \forall e, f \in E | p_{ef} = 1 \tag{2.3}$$

$$x_k \geq 0 \qquad \forall k \in K \tag{2.4}$$

$$y_e \geq 0 \qquad \forall e \in E \tag{2.5}$$

$$z_{ef} \geq 0 \qquad \forall e, f \in E | p_{ef} = 1 \tag{2.6}$$

A feasible solution has y_e and z_{ef} equal to zero. The constraints together with the objective function ensure that decision variables x and y are not set to higher than one. Constraint (2.1) together with the first term of the objective function ensure that all events are assigned at least once. If the decision variable y_e is set to zero this implies that a slot-schedule containing event e is selected. For every timeslot at most one slot-schedule can be selected, which is enforced by constraint (2.2). A precedence relation between two events is taken care of by constraint (2.3) together with the second part of the objective function.

2.2.2 The column generation procedure

There are instances with a lot of precedence constraints. These precedence constraints elicit that solving the LP takes a large computation time. This is mainly caused by the large number of rows in the constraint matrix. Therefore we choose to relax the LP-formulation by removing the precedence constraints (2.3). They are taken into account by the procedure described in Subsection 2.2.4. The master problem (MP) is the LP-formulation without the precedence constraints.

We could consider the complete set of possible slot-schedules, but we cannot handle such a big LP and it would already cost too much computation time to generate these slot-schedules. Therefore, we try to solve MP using column generation. The restricted master problem (RMP) is MP with a restricted set K^r of slot-schedules. The main

idea of the column generation approach is to generate only the slot-schedules that with high probability are in the basis of an optimal feasible solution of MP.

Solving RMP provides *shadow prices* for all constraints. The shadow price of a particular constraint of an LP is the value of the optimal dual solution variable associated with that constraint. It also represents the change in the value of the objective function per unit increase or decrease of the righthand-side value of that constraint. We will illustrate this with two examples.

Let us assume $\forall k \in K^r : E_k = \emptyset$. This means that in an optimal solution of RMP all variables y_e are set to one. Subtracting one from the righthand side of constraint (2.1) of event e', gives for variable $y_{e'}$ a value of zero in the optimal solution of RMP. So also the objective function decreases with one and this implies that the shadow price for constraint (2.1) of event e' is equal to one.

Now assume there exists a slot-schedule $k \in K^r$ with $e^* \in E_k$ and RMP has an optimal solution with $x_k = 1$ and $y_{e^*} = 0$. Subtracting one from the righthand side of constraint (2.1) of event e^* does not change the value of y_{e^*}. Hence, the objective function does not change and the shadow price of the constraint for event e^* is zero.

To conclude: Shadow prices provide us useful information that we use to generate promising feasible slot-schedules. We denote the shadow prices of constraint (2.1) by α_e and of constraint (2.2) by β_t.

After solving RMP, the shadow prices are used to generate ("to price") new columns. These columns are generated by the *column generator* that we describe in detail in Subsection 2.2.3. The column generator generates feasible slot-schedules for a certain timeslot t. The input of the column generator consists of the set $E^{cg} \subseteq E_t$ of events with $\alpha_e \geq 0$. For a slot-schedule k that is generated by the column generator, the *reduced cost* c^k is:

$$c^k = -(\textstyle\sum_{e \in E_k} \alpha_e + \beta_t)$$

We proceed generating new columns until for no timeslot a new slot-schedule is found or until a certain amount of columns has been generated. Note that the column generation procedure does not always generate an optimal solution of MP.

The column generator generates a set K_t^{cg} of feasible slot-schedules for a timeslot t. We are only interested in slot-schedules that are with high probability in an optimal solution of MP. Therefore, a slot-schedule $k \in K_t^{cg}$ is only added to K^r if the reduced cost c^k of the column is less than a certain fraction of the average reduced costs $\overline{c}$ of the most recently added slot-schedules.

To start a column generation procedure a number of variables equal to the number of constraints are needed to find an initial basis. The y_e variables are added to the initial formulation of RMP and for each timeslot t we generate a slot-schedule k_t. The set E_{k_t} of events for slot-schedule k on timeslot t is determined by using the column generator with as input all events that can be scheduled on timeslot t with α_e set to one for each event e. The generated slot-schedule $k_t \in K_t^{cg}$ with the largest value of $\sum_{e \in E_{k_t}} |C_e|$ is taken for the initial basis. For the last timeslots of the day slot-schedules are added with $E_{k_t} = \emptyset$.

We now present our column generation procedure in more detail:

1. Initialize K^r, set $t = 0$ and $factor = 0.75$.
2. Solve RMP $\rightarrow \alpha_e, \beta_t$ and obj^{rmp}.
3. If $obj^{rmp} \leq 0$, then quit.
4. Generate set K_t^{cg} of feasible slot-schedules for timeslot t.
5. Add column $k \in K_t^{cg}$ to K^r if $c^k < factor * \bar{c}$. Add at most $\lfloor \frac{|E|+|T|}{100} \rfloor$ columns.
6. $t = t + 1$
7. If $|K^r| > 2(|E| + |T|)$ or $\forall t \in T$ no column has been added, then quit.
8. Go to step 2.

In step 3 we quit if the objective value of RMP is zero, because then an optimal solution of RMP has been found. Step 4 generates feasible slot-schedules for a certain timeslot. At most 120 feasible slot-schedules are generated as is described in the next subsection. To take care of enough variation between the columns in RMP, only the best $\lfloor \frac{|E|+|T|}{100} \rfloor$ columns are added. In step 7 the column generation procedure is terminated if the number of columns is twice the number of rows of RMP or when no promising column has been found for each timeslot.

2.2.3 The column generator

A set K_t^{cg} of feasible slot-schedules for a certain timeslot t are generated by the column generator. The input of the column generator consists of a set $E^{cg} \subseteq E$ of events with for each event $e \in E^{cg}$ a shadow price α_e. The goal is to generate feasible slot-schedules with a large sum of the shadow prices of the assigned events.

It is possible to determine a feasible slot-schedule with the maximum sum of the shadow prices of the events in this slot-schedule. This can be done by solving an IP that uses the decision variable $y'_e = 1 - y_e$ that is one if event e is assigned to a room, otherwise it is zero. Also the following decision variable is defined $\forall r \in R_e$:

$$y'_{er} := \begin{cases} 1 & \text{if event } e \text{ is assigned to room } r \\ 0 & \text{otherwise} \end{cases}$$

The IP problem is:

$$\max \sum_{e \in E^{cg}} \alpha_e y'_e$$

$$\sum_{r \in R_e} y'_{er} = y'_e \qquad \forall e \in E^{cg} \tag{2.7}$$

$$\sum_{e \in E^{cg} \cap E_s} y'_e \leq 1 \qquad \forall s \in S \tag{2.8}$$

$$\sum_{e \in E^{cg} \cap E_r} y'_{er} \leq 1 \qquad \forall r \in R \tag{2.9}$$

$$y'_e + y'_f \leq 1 \qquad \forall e, f \in E^{cg} | p_{ef} = 1 \tag{2.10}$$

$$y'_{er} \in \{0, 1\} \qquad \forall e \in E^{cg}, \forall r \in R_e \tag{2.11}$$

$$y'_e \in \{0, 1\} \qquad \forall e \in E^{cg} \tag{2.12}$$

Constraint (2.7) takes care that an event is assigned to at most one room or it is not assigned. Students with more than one event assigned are not allowed, this is fulfilled by constraint (2.8). Constraint (2.9) imposes that only one event is assigned to each room. If there is a precedence relation between two events they can not be assigned both, this is enforced by constraint (2.10).

Solving this IP to optimality and adding the generated column to RMP would guarantee that we find an optimal solution of RMP, Lübbecke and Desrosiers (2005). But unfortunately solving the IP takes too much computation time. Therefore, we developed a procedure that generates quickly a set of feasible slot-schedules by a greedy assignment procedure. This procedure is the central topic of the rest of this section.

The procedure starts with sorting all $e \in E^{cg}$ according to the following criteria:

1. Decreasing order of α_e.
2. Decreasing order of $|C_e|$.
3. Decreasing order of N_e.
4. Increasing order of $|R_e|$.
5. Increasing order of $|T_e|$.

We denote the sorted set of events by E_s^{cg}. The procedure generates a number of different columns by selecting sets of three non-conflicting events on top of the sorted list for which a room assignment is possible. After those three events are assigned we try to greedily fill up the slot-schedule. The full procedure to generate columns will now be presented in pseudo code:

$K_t^{cg} = \emptyset$.
for (first n_1 events e_1 of E_s^{cg}) **do**
  for (first n_2 events e_2 of E_s^{cg} after e_1, not conflicting with e_1) **do**
    Assign e_1 and e_2 to rooms.
    for (first n_3 events e_3 of E_s^{cg} after e_2, not conflicting with e_1 and e_2 and with possible room assignment) **do**
      Assign e_3 to a room.
      repeat
        e is next event of E_s^{cg}.
        if (e does not conflict with an already assigned event) and (e can be assigned to a room)) **then**
          Assign e to a room.
        end if
      until (no event left in E_s^{cg})
      Add generated slot-schedule to K_t^{cg}.
    end for
  end for
end for

After tuning the values of n_1, n_2 and n_3 are set to $n_1 = 20, n_2 = 3$ and $n_3 = 2$. This means that the procedure generates at most 120 different columns.

Every time an event e is a candidate for adding it to a slot-schedule it has to be verified whether there is a room assignment for e together with the already assigned events. Assigning events to rooms is a bipartite matching with the set of rooms and set of events as the vertices of the graph. Two vertices are connected if an event can be assigned to a room. Given the already assigned events to rooms it costs at most one augmenting path step to determine whether e can be added to the given matching. This matching algorithm is also applied by for example Chiarandini et al. (2008) and Cambazard et al. (2008).

2.2.4 Fix generated columns

The solution approach described in the last subsections results in a solution that is most likely not feasible for the LP-formulation, because some of the precedence constraints are probably violated. The generated solution consists of non-integer x-variables. To get a feasible integer solution we developed a heuristic that fixes one column at a time, which is the same as setting one x-variable to one. Fixing the column of slot-schedule k means assigning events $e \in E_k$ definitely to timeslot t_k.

We define the set E_{nf} of events that have not been fixed and the set T_{nf} of timeslots that do not have a fixed slot-schedule. During the column generation procedure a lot of feasible slot-schedules are generated. Of all generated slot-schedules we fix the generated slot-schedule k with the highest value of $obj_k = \sum_{e \in E_{nf}} a_{ek} c_e - penalties$ with c_e the number of events $f \in E_{nf}$ conflicting with event e. The goal is to fix a slot-schedule that results in a low number of remaining conflicts between events. A penalty of $|T_{nf}|$ is subtracted for each event $e \in E_{nf}$ that has only one possible timeslot left after fixing column k and a penalty of one is subtracted for every violation of a three-in-a-row constraint after fixing slot-schedule k.

After fixing a column we delete some slot-schedules from the set K^r. Columns of slot-schedules containing events that were in the column that has been fixed and columns with the same timeslot are deleted. Also slot-schedules that are not feasible anymore, because of violated precedence constraints are deleted. Next to those, we also delete columns that most likely will not be in the basis of an optimal feasible solution of MP. An example of such a column: If an event e has to be assigned earlier in the week than f, we delete a slot-schedule with timeslot on the first day if it contains f. Deletion of all these columns gives an enormous speed up in solving MP.

The heuristic to construct a valid solution goes as follows:

1. Initialize $T_{nf} = T \backslash \{8, 17, 26, 35, 44\}$ and $E_{nf} = E$.
2. Apply column generation procedure $\rightarrow K^r$.
3. Fix slot-schedule $k' = \max_{k \in K^r} obj_k$.
4. $T_{nf} = T_{nf} \backslash t_{k'}$, $E_{nf} = E_{nf} \backslash E_{k'}$ and delete columns.
5. If $|T_{nf}| > 5$ and $|E_{nf}| > 0$, then go to step 2.
6. If $|E_{nf}| > 0$, then solve the compact IP presented in Section 2.3 to assign all $e \in E_{nf}$ to a timeslot $t \in T_{nf} \cup \{8, 17, 26, 35, 44\}$.

To take into account the soft constraint of no events on the last timeslot of the day, these timeslots are not added to the set T_{nf}. Because CPLEX is able to find a very good feasible solution quickly for the last 10 timeslots we quit in step 5 when $|T_{nf}| = 5$. In step 6 we solve the compact IP formulation of the problem with the events that are left. This IP is introduced in Section 2.3. Mostly the optimal solution of the IP is found in less than a second. To prohibit a large computation time, CPLEX has been given at most thirty seconds in which it always found a very good feasible solution.

2.3 Improvement heuristic by integer programming

In Section 2.2 a heuristic has been introduced that constructs a valid solution. The heuristic focuses on assigning the events to timeslots and hardly takes into account the soft constraints. Constructing this valid solution costs only a small part of the computation time that was available within the competition. Therefore, we have developed a procedure that improves the constructed solution. The first subsection presents a compact MIP formulation of the timetabling problem including all soft constraints. This MIP is used to improve the solution found by the construction heuristic. Subsection 2.3.2 presents the improvement heuristic.

2.3.1 The compact MIP formulation of the problem

We define the set D of days and the set T' as the set containing the first 7 timeslots of all days. All other parameters have been introduced in Section 2.1. Left to define are the decision variables:

$$z_{ert} := \begin{cases} 1 & \text{if event } e \text{ is assigned to room } r \text{ on timeslot } t \\ 0 & \text{otherwise} \end{cases}$$

$$y_e := \begin{cases} 1 & \text{if event } e \text{ is not assigned} \\ 0 & \text{otherwise} \end{cases}$$

$$u_e := \begin{cases} 1 & \text{if event } e \text{ is assigned to a last timeslot of the day} \\ 0 & \text{otherwise} \end{cases}$$

$$v_{st} := \begin{cases} 1 & \text{if student } s \text{ is assigned to an event on timeslots } t, t+1 \text{ and } t+2. \\ 0 & \text{otherwise} \end{cases}$$

$$w_{sd} := \begin{cases} 1 & \text{if student } s \text{ is assigned to exactly one event on day } d \\ 0 & \text{otherwise} \end{cases}$$

$$w'_{sd} := \begin{cases} 1 & \text{if student } s \text{ is assigned to at least one event on day } d \\ 0 & \text{otherwise} \end{cases}$$

Events can not be assigned to all possible timeslots and rooms. Therefore, z_{ert} is

only defined for the triples (e, r, t) if $t \in T_e$ and $r \in R_e$. This already ensures hard constraints two and four.

The objective function consists of two parts. The first part of the objective function takes care that the number of students that have a requested event not assigned is minimized. This part minimizes the distance to feasibility. The second part consists of three terms that account for the number of soft constraint violations. Finding a solution with a distance to feasibility of zero is the most important goal. Therefore, we introduced the weight factors W_h and W_s and applied the values $W_h = 1000$ and $W_s = 1$.

Now we are able to present the MIP:

$$\min \quad W_h \sum_{e \in E} N_e y_e + W_s \Big(\sum_{e \in E} N_e u_e + \sum_{s \in S} \sum_{t \in T'} v_{st} + \sum_{s \in S} \sum_{d} w_{sd} \Big)$$

$$\sum_{e \in E_s \cap E_t} \sum_{r \in R_e} z_{ert} \leq 1 \qquad \forall s \in S, \forall t \in T \tag{2.13}$$

$$\sum_{e \in E_r \cap E_t} z_{ert} \leq 1 \qquad \forall r \in R, \forall t \in T \tag{2.14}$$

$$\sum_{r \in R_e} \sum_{t \in T_e} z_{ert} + y_e = 1 \qquad \forall e \in E \tag{2.15}$$

$$\sum_{t \in T_e | t < t'} \sum_{r \in R_e} z_{ert} \geq \sum_{t \in T_f | t \leq t'} \sum_{r \in R_f} z_{frt} \quad \forall e, f \in E | p_{ef} = 1, \forall t' \in [1, 44] \tag{2.16}$$

$$\sum_{t \in \{8,17,26,35,44\}} \sum_{r \in R_e} z_{ert} - u_e \leq 0 \qquad \forall e \in E \tag{2.17}$$

$$\sum_{t=t'}^{t'+2} \sum_{e \in E_s \cap E_t} \sum_{r \in R_e} z_{ert} - v_{st'} \leq 2 \qquad \forall s \in S, \forall t' \in T' \tag{2.18}$$

$$\sum_{t=9*d}^{9*d+8} \sum_{e \in E_s \cap E_t} \sum_{r \in R_e} z_{ert} - M w'_{sd} \leq 0 \qquad \forall s \in S, \forall d \in D \tag{2.19}$$

$$\sum_{t=9*d}^{9*d+8} \sum_{e \in E_s \cap E_t} \sum_{r \in R_e} z_{ert} + w_{sd} \geq 2 w'_{sd} \qquad \forall s \in S, \forall d \in D \tag{2.20}$$

$$z_{ert} \in \{0, 1\} \qquad \forall e \in E, \forall r \in R_e, \forall t \in T_e \tag{2.21}$$

$$y_e \geq 0 \qquad \forall e \in E \tag{2.22}$$

$$u_e \geq 0 \qquad \forall e \in E \tag{2.23}$$

$$v_{st'} \geq 0 \qquad \forall s \in S, \forall t' \in T' \tag{2.24}$$

$$w_{sd} \geq 0 \qquad \forall s \in S, \forall d \in D \tag{2.25}$$

$$w'_{sd} \in \{0, 1\} \qquad \forall s \in S, \forall d \in D \tag{2.26}$$

Constraint (2.13) ensures that no student has to attend more than one event at a time. Only one event can be put into each room on any timeslot. This is enforced by

constraints (2.14). Constraint (2.15) imposes that every event is assigned to at most one timeslot and one room. If there exists a precedence relation between two events, they should be scheduled in the correct order during the week. This is fulfilled by constraint (2.16). Constraint (2.17) accounts for the soft constraint that no student has to attend an event at the last timeslot of a day. Students do not have to attend more than two events consecutively. This is taken into account by constraint (2.18). Constraints (2.19) and (2.20) together account for that a student is not assigned to exactly one event on a certain day. We take $M = 9$, this value is large enough to always fulfill constraint (2.19). Note that except for the $z-$variables and the w'-variables, all decision variables are elements of $\mathbb{R}^+$. They are automatically put on zero or one, because of the binary $z-$variables.

2.3.2 Description of the improvement heuristic

The compact MIP is too big to solve in a reasonable amount of time, but it can be used to complete a partial solution. A partial solution is represented by a set E^p of events that have been assigned to a timeslot. The timeslot to which an event $e \in E^p$ has been assigned is denoted by τ_e. The events in E^p have not been assigned to a room, but it is known that a feasible room assignment exists for those events. Given a partial solution, the number of decision variables in the MIP is reduced a lot by substituting: $\forall e \in E^p, \forall t \in T \setminus \tau_e : z_{ert} = 0$. If the set E^p is sufficiently large, then CPLEX is able to solve the MIP quickly. Hence, each event $e \in E^p$ is reassigned to a room and each event $f \in E \setminus E^p$ to a room and timeslot within a small computation time.

The construction heuristic constructs a valid solution that we try to improve by solving MIP with input a partial solution derived from the known valid solution. We denote MIP with input a partial solution represented by E^p with $MIP(E^p)$.

The improvement heuristic constructs a partial solution represented by the set E^p. First, it initializes the set $E^p := E$. The partial solution is constructed by deleting events from E^p. Events that are not scheduled in the current valid solution are immediately removed. It seems obvious to remove events that cause a lot of soft constraint violations. Therefore, we introduce $SCV(e)$ as the number of soft constraint violations of an event e. This number is determined for all events assigned to a timeslot and room in the current valid solution. The value of $SCV(e)$ is calculated for each $e \in E^p$ as follows:

1. Initialize $SCV(e) := 0$.

2. If τ_e is a last timeslot of the day, then $SCV(e)$ is increased by N_e.

3. For each student for whom e is the only event on a day, $SCV(e)$ is increased by one.

4. For each student who has three events in a row, one of which is e, $SCV(e)$ is increased by one.

If only events with a large value of $SCV(e)$ are not in E^p, then it is almost impossible to fit those events into the partial solution on different timeslots than in the old solution. Therefore, the improvement heuristic chooses one event e^* from E^p and removes it from E^p. The other events removed from E^p are events that increase the probability that a new solution is found with e^* assigned to a timeslot where it has fewer soft constraint violations.

The heuristic starts by choosing an event e^* that is not assigned in the given valid solution. If the current solution is feasible, then event e with maximum value of $SCV(e)$ is chosen. To prohibit that always the same event is chosen, a list L of length 20 that contains the events that were most recently used as event e^* is kept up.

The parameter $c(t, e^*)$ is the number of events $e \in E^p$ with $\tau_e = t$ and $e \in C_{e^*}$. The total number of students attending an event e with $\tau_e = t$ and $e \in C_{e^*}$ is denoted with $cs(t, e^*)$. In other words, it is the total number of students attending an event on timeslot t that conflicts with e^* in the current valid solution. With T^c we model the set of timeslots for which all assigned events in the current valid solution are removed from E^p. The timeslots in T^c are feasible timeslots for e^* with the largest probability that e^* can be assigned to one of those timeslots in a better solution. These are the timeslots t with minimum value of $cs(t, e^*)$. If this is equal for two timeslots, then the timeslot with the maximum number of soft constraint violations is chosen.

The improvement heuristic goes as follows:

1. Initialize $L = \emptyset$ and $NOT = 5$.
2. $E^p = E$.
3. Select $e^* \in E \setminus L : e^*$ is not assigned; If all $e \in E \setminus L$ have been assigned, then take $e^* = \max_{f \in E \setminus L} SCV(f)$.
4. $T^c = T_{e^*}$.
5. Reduce T^c to NOT timeslots with $\min_{t \in T^c} cs(t, e^*)$; if this is equal, then take $\max_{t \in T^c} \sum_{f \in E | t = \tau_f} SCV(f)$.
6. $\forall e \in E^p$: If $\tau_e \in T^c$, then remove e from E^p.
7. $\forall f \in E^p$: If $c(\tau_f, e^*) <= 3$ and f conflicts with e^*, then remove f from E^p.
8. Solve $MIP(E^p)$, maximum allowed computation time is 5 seconds.
9. Add e^* to L and delete the last element of L.
10. If no improvement found for 15 iterations, then $NOT = NOT + 1$.
11. If no feasible solution found without soft constraint violations and there is computation time left, then go to step 2.

In step 7 events that conflict with e^* are removed from E^p to enlarge the probability that an event e^* is assigned to a different timeslot. In the current solution these events have been assigned to a timeslot τ that has three or less events conflicting with e^*. By

removing the conflicting events from E^p they can be assigned to a different timeslot and event e^* can be assigned to timeslot τ. Rarely the solution time of $MIP(E^p)$ becomes larger than a few seconds. If this occurs, then CPLEX is terminated after 5 seconds and the best found solution is only accepted if it has an objective value at least as good as the old one.

2.4 Computational results

The competition organizers constructed 24 instances. The first 8 instances were released at the start of the competition. Two weeks before the deadline another 8 instances were released and the last 8 instances were kept secret. Those were released after the final ranking of the competitors had been made. Remark that we did not participate in the competition, because it was not allowed to make use of CPLEX.

For a fair comparison of the created solutions the organizers provided a benchmark program to determine the maximum available computation time on a single processor machine. All our computations have been done on an Intel core T8300, 2.4 GHz processor with 2.0 GB internal memory. The available computation time on this machine was 364 seconds.

Table 2.1 presents the computational results of the construction heuristic described in Section 2.2 for all 24 instances. To solve RMP, CPLEX 11.2 has been used. The column with head 'c.t.(s)' gives the computation time of the construction heuristic. Column 'dtf' contains the distance to feasibility. The number of times a student is assigned to three consecutive events is represented by the next column. Column '1 a day' gives the number of times a student has only one event on a day and the number of students having an event on the last timeslot of the day is given in the column 'eod'. The right most column presents the total number of soft constraint violations.

The construction heuristic is able to assign all events to a timeslot and a room for all 24 instances. In other words, it is able to find a feasible solution for each instance. The number of soft constraint violations is quite large, but we took them into account only in a minor way. We can conclude that the construction heuristic succeeds in fulfilling our main goal, which was providing a solution with distance to feasibility of zero.

The computation time of the construction heuristic depends on the number of events. Instances with more events need significantly more computation time. Table 2.1 also shows that for most instances a lot of computation time is left to improve the solution found.

The improvement heuristic is run until the allowed computation time is fully used or until there are no violated constraints left. Table 2.2 presents the computational results if the improvement heuristic is used with the solution found by the construction heuristic as input. The last column shows the percentage of the violated soft constraints after the construction heuristic that have been solved by the improvement heuristic.

The number of soft constraint violations decreased significantly. Especially shortly after the start of the improvement heuristic a lot of large improvements have been

Table 2.1: Computational results of the construction heuristic

I	$\|E\|$	$\|R\|$	$\|S\|$	c.t.(s)	dtf	3 cons	1 a day	eod	s.c.
1	400	10	500	45	0	1080	12	584	1676
2	400	10	500	37	0	1080	13	601	1694
3	200	20	1000	3	0	453	674	0	1127
4	200	20	1000	3	0	376	596	157	1129
5	400	20	300	15	0	732	14	175	921
6	400	20	300	21	0	735	14	203	952
7	200	20	500	3	0	217	244	119	580
8	200	20	500	2	0	164	229	166	559
9	400	10	500	39	0	1314	23	713	2050
10	400	10	500	35	0	1092	11	599	1702
11	200	10	1000	3	0	339	571	157	1067
12	200	10	1000	3	0	218	544	472	1234
13	400	20	300	21	0	779	17	244	1040
14	400	20	300	20	0	759	17	234	1010
15	200	10	500	3	0	206	331	153	690
16	200	10	500	2	0	121	305	108	534
17	100	10	500	3	0	96	2	0	98
18	200	10	500	2	0	673	22	125	820
19	300	10	1000	17	0	530	572	708	1810
20	400	10	1000	40	0	608	622	115	1345
21	500	20	300	36	0	825	13	176	1014
22	600	20	500	253	0	1399	36	809	2244
23	400	20	1000	29	0	1841	20	753	2614
24	400	20	1000	22	0	673	690	429	1792

found. For one of the 24 instances a solution has been found without any soft constraint violation. The computation time necessary to find this solution was 144 seconds.

To get an impression of how good the total algorithm performs, we compare our results with the results of the five finalists of the competition. For these five finalists the organizers did 10 runs with different random seed for each instance. The results of those 50 runs were ranked from 1 to 50 and the rankings of all 240 runs of a finalist were summed up. The finalist with the lowest value of this sum was the winner of the competition. Making a fair comparison with the results of the finalists is very difficult, because our heuristic is deterministic. We applied the same ranking method for 6 finalists and counted our deterministic result ten times. This ranking method would have ranked us number 3.

Another comparison with the results of the five finalists is made in Table 2.3. The columns 2 to 6 show the number of runs of an instance for which a finalist did not get a distance to feasibility of zero. The finalists have been ranked according to

Table 2.2: Computational results after the improvement heuristic

I	$\|E\|$	$\|R\|$	$\|S\|$	dtf	3 cons	1 a day	eod	s.c.	Improvement
1	400	10	500	0	637	16	462	1115	33.47 %
2	400	10	500	0	710	9	438	1157	31.70 %
3	200	20	1000	0	122	230	0	352	68.77 %
4	200	20	1000	0	171	308	0	479	57.57 %
5	400	20	300	0	234	1	38	273	70.36 %
6	400	20	300	0	441	2	148	591	37.92 %
7	200	20	500	0	33	77	0	110	81.03 %
8	200	20	500	0	23	39	0	62	88.91 %
9	400	10	500	0	925	5	623	1553	24.24 %
10	400	10	500	0	577	5	473	1055	38.01 %
11	200	10	1000	0	143	173	0	316	70.38 %
12	200	10	1000	0	303	284	0	587	52.43 %
13	400	20	300	0	338	4	92	434	58.27 %
14	400	20	300	0	444	14	61	519	48.61 %
15	200	10	500	0	137	187	0	324	53.04 %
16	200	10	500	0	82	116	0	198	62.92 %
17	100	10	500	0	0	0	0	0	100 %
18	200	10	500	0	101	0	0	101	87.68 %
19	300	10	1000	0	426	449	519	1394	22.98 %
20	400	10	1000	0	409	406	107	922	31.45 %
21	500	20	300	0	420	5	111	536	47.14 %
22	600	20	500	0	1145	24	741	1910	14.88 %
23	400	20	1000	0	1137	3	650	1790	31.52 %
24	400	20	1000	0	370	420	176	966	46.09 %

the final ranking of the competition, so number 1 in Table 2.3 is the winner of the competition. Our heuristic provides a solution with distance to feasibility of zero for all 24 instances. There is no finalist that also succeeds in always finding a feasible solution. Only finalist number 3 finds a feasible solution for almost all runs. It can be concluded that our algorithm performs excellent in generating solutions with distance to feasibility zero. Note that the construction heuristic finds the feasible solutions in a small computation time.

The columns sc1 to sc5 of Table 2.3 present for each finalist the average number of violated soft constraints of solutions found with distance to feasibility zero. The last column contains the number of violated soft constraints of our heuristic. Unfortunately it is not known which soft constraints are violated by the finalists. Therefore, we can not say anything about which algorithm can deal the best with a certain type of constraint.

For instances for which it is hard to find a feasible solution (like 2,9,10 and 19) we outperform other approaches. Our algorithm performs worse on most of the easier instances and the instances with a lot of precedence constraints. This can be explained by the fact that our improvement heuristic is not capable of coping well with the soft

Table 2.3: Comparison with solutions of the five finalists

I	*f1	f2	f3	f4	f5	TUe	sc1	sc2	sc3	sc4	sc5	scTUe
1	0	5	0	7	7	0	883.4	560.4	1730.5	151.3	1936.0	1124
2	2	8	0	6	10	0	1565.9	660.5	1913.6	6.5	-	1156
3	0	0	0	0	0	0	237.3	529.9	389.7	732.6	333.9	447
4	0	0	0	0	0	0	370.0	683.2	480.2	727.5	559.7	477
5	0	0	0	0	0	0	6.8	21.0	679.9	128.3	20.9	323
6	0	0	0	2	0	0	4.2	64.5	977.4	319.8	266.6	553
7	0	0	0	0	0	0	7.5	97.7	354.1	3.8	183.6	140
8	0	0	0	0	0	0	0.0	24.1	1.3	80.6	24.5	0
9	0	7	0	7	10	0	1868.6	0.3	2100.4	0.0	-	1687
10	7	2	4	0	10	0	1850.0	0.0	2065.3	0.1	-	1136
11	0	0	0	0	1	0	288.3	716.0	352.6	898.0	699.2	343
12	0	0	0	7	1	0	352.7	1046.8	616.4	495.3	1176.3	678
13	0	0	0	2	0	0	128.3	102.6	911.1	399.9	475.6	503
14	0	0	0	0	0	0	4.1	0.4	983.5	97.0	407.9	538
15	0	0	0	0	0	0	93.1	460.6	310.6	142.7	268.2	306
16	0	0	0	0	0	0	17.1	251.8	5.8	131.2	178.4	166
17	0	0	0	0	0	0	4.9	19.0	9.8	116.4	106.2	0
18	0	1	0	0	0	0	14.1	42.9	339.9	264.8	314.3	72
19	8	10	0	1	10	0	2027.0	-	2080.8	43.8	-	1263
20	0	6	0	10	0	0	505.0	1312.0	640.5	-	919.3	887
21	0	0	0	2	0	0	27.1	3.6	876.3	211.0	336.8	550
22	1	0	0	0	4	0	612.0	0.0	1839.0	82.7	1424.7	1946
23	0	1	0	10	0	0	330.5	564.8	1043.0	-	701.3	1797
24	0	3	0	0	0	0	124.2	889.3	963.4	129.2	518.0	1011

* In the cases finalist 1 did not find a feasible solution, it even did not find a valid solution. So dtf $= \infty$ for those cases.

constraints of three events in a row and one event a day. A promising approach seems to be to find a feasible solution with our construction heuristic and then apply (parts of) the algorithm of the winner of the competition to minimize the violated number of soft constraints.

2.5 Concluding remarks

In this chapter we presented an original heuristic that constructs a feasible solution for the post enrolment course timetabling problem of the second international timetabling competition. The heuristic fixes a set of events for a timeslot given a feasible solution of an LP-relaxation which we found by using column generation.

The construction heuristic is able to assign all the events to a timeslot and a room without violating any of the hard constraints in all 24 instances of the competition. In comparison with the five finalists of the competition the construction heuristic performs very well in generating a solution with distance to feasibility of zero. Most

finalists have one or more instances where their algorithm does not find a feasible solution.

An improvement heuristic to decrease the number of soft constraint violations has been presented. Every iteration a set of events is reassigned by solving an integer program. The number of soft constraint violations resulting after the improvement heuristic is still quite large and leaves room for improvement. As future research it would be interesting to combine our construction heuristic and the local search algorithm of one of the finalists of the competition to reduce the number of soft constraint violations.

The organizers of the competition choose to design instances that have a solution with distance to feasibility zero and without soft constraint violations. In reality it is not known whether such a solution exists and then lower bounds become interesting. If the column generator would always generate the best column, for example by solving the IP presented in Subsection 2.2.3, then the optimal solution of the MP is always found. The corresponding solution value would be a lower bound for the distance to feasibility. Solving the IP for generating the best columns is too time consuming, therefore development of a fast exact algorithm to determine the best column to add is interesting future research.

Chapter 3

A course timetabling problem at the TU Eindhoven

In February 2005, outraged students of the Department of Industrial Design were protesting at the TU Eindhoven (The Netherlands). Why were they doing this? The academic year of these roughly 350 students of Industrial Design is split into a number of periods. In every period, every student must follow a number of small *courses*. There is a pool of roughly 55 courses to choose from, and every student submits a list with his/her 10 preferred courses to the department. Based on all these preference lists (and on many other constraints), a scheduler of the department then assigns roughly 4 courses to every student.

The scheduler was a human decision-maker who essentially applied a hand-woven greedy assignment procedure, starting filling the most popular courses with students with the lowest id-numbers. In 2005, however, the number of students increased substantially, and as a consequence the greedy approach did not yield acceptable results anymore. The students were very dissatisfied with the work of the human scheduler. More than 150 out of the 350 students were assigned to fewer courses than requested. Therefore, they needed to do courses that were not on their preference list.

The people from the Department of Industrial Design realized that they had a problem and they did not know how to settle this problem themselves. The number of students had increased substantially, and the timetabling problem had become much larger, much harder, and much more complex. Hence, they turned to us, mathematicians. They explained their problem (in problem formulation No. 1), and we happily told them that we were able to solve it: the problem (in formulation No. 1) can be modeled as a network flow problem, and hence is solvable in polynomial time. Unfortunately, it turned out that formulation No. 1 was not a complete formulation of the problem. They had forgotten to inform us about a number of additional restrictions that led to a new, more difficult assignment problem (in formulation No. 2), which eventually turned out to be NP-hard.

This chapter is a report on the course assignment problem of the Department of Industrial Design. We will describe the assignment problem in its (incomplete) formulation No. 1 and in its (complete) formulation No. 2. We show that formulation No. 1 yields a tractable problem, whereas formulation No. 2 does not. Our main

contribution is a careful case study of the complete problem formulation. We apply lexicographical optimization and design four elegant integer linear programming models for it, with roughly 6000 variables and roughly 7000 constraints. Putting these ILP models into CPLEX yields excellent results within moderate computation times. We describe the ILP models in detail, and we report on our computational experiments with the real-world data of the Department of Industrial Design.

The rest of this chapter is structured in the following way. In Section 3.1 we present references to the literature of a closely related course timetabling problem. Section 3.2 contains a detailed description of the problem we solved for the Department of Industrial Design. We prove that the timetabling problem in formulation No. 2 is NP-hard, by proving some independent subproblems to be NP-hard. These proofs are given in Section 3.3. The four ILP models used in the lexicographical optimization procedure will be described in Section 3.4. Section 3.5 contains the computational results and some concluding remarks are given in Section 3.6.

3.1 A closely related timetabling problem

A general introduction into the field of course timetabling is provided in Section 1.4. The problem of the Department of Industrial Design can also be categorized as a post enrolment course timetabling problem. One of the main differences with the post enrolment course timetabling problem of ITC2 is that courses can be split in multiple *sections*. If the expected number of students for a course is too large for one room, then courses are split into multiple sections that can be taught by different teachers.

Closely related to the timetabling problem of the Department of Industrial Design is the *student scheduling problem (SSP)*, in which the weekly timetable is fixed and a section may take more than one timeslot. Students are required to take a certain number of courses, which they have to select themselves after the timetable is made available. The problem consists of assigning a student to a specific section of a course for a given fixed timetable such that students are satisfied, there are no time conflicts, section sizes are balanced and room capacities are respected. The main difference between the SSP and our timetabling problem is that for the SSP *all* courses on the preference list of the students have to be assigned to students. This results in most practical cases into an empty feasible solution set. Other differences are that we consider an ordered preference list and in the SSP the maximum number of students for a course is a soft constraint.

Cheng et al. (2003) discuss problem SSP as it generally applies to high schools in North America. They define the problem as the assignment of students to sections of courses offered at various times during the week. The objective is to fulfil student requests, providing a conflict-free schedule. They show that the problem is NP-hard and discuss various multi-commodity flow formulations with fractional capacities and integral gains.

Laporte and Desrochers (1986) give a mathematical formulation of the student scheduling problem. They formulate the problem as an optimization problem splitting the requirements into hard and soft ones. The only hard constraint in their model

is that student course selections must be respected. Time conflicts for students are soft constraints. When time conflicts occur students are advised to make a different course selection. The problem is then solved in three phases: In the first one the algorithm searches for an admissible solution, in the second section enrollments are balanced and in the third the room capacities have to be respected.

Tripathy (1992) formulated the student scheduling problem as an integer linear programming problem and used Lagrangian Relaxation to solve it. Sabin and Winter (1986) use a greedy approach that is moderated by an intelligent ordering of the students. Miyaji et al. (1988) apply goal programming. The student scheduling problem is also considered in Busam (1967) and Feldman and Golumbic (1989).

3.2 Problem description

At our first meeting, the planner from the Department of Industrial Design explained the problem to us in a certain problem formulation No. 1; see Subsection 3.2.1. This problem can be modeled as a network flow problem, and hence is solvable in polynomial time, Ahuja et al. (1993). This network flow model will also be given in Subsection 3.2.1.

Unfortunately, we learnt after some time that formulation No. 1 was *not* a complete formulation of the problem. The planner actually had forgotten to tell us about a number of additional restrictions that lead us to a new, more difficult assignment problem, formulation No. 2. Subsection 3.2.2 describes formulation No. 2.

3.2.1 Problem formulation No. 1

At the first meeting with the planner of the Department of Industrial Design, we were told that every student hands in a preference list of at most 10 courses and requests a certain number of courses. The only constraints are that a student can not do two courses at the same time and there is a maximum number of students that can be assigned to a course. This subsection contains a more detailed description of problem formulation No. 1 and its network flow model.

All courses are provided in the set C of courses. For each course c an *upper bound* C_c^{max} on the number of students is given. This maximum number of students depends on the preference of the teacher and the room capacity in which the course is given. Each course has one weekly meeting time, which is already fixed. This weekly meeting time always consists of two consecutive hours. Two such consecutive hours are defined as one *timeslot*. The weekly meeting time of a course is chosen from a set T of disjoint timeslots. $T(c)$ is the timeslot that is the weekly meeting time of course c. Hence, one of the constraints in the model is that courses c_i and c_j can not be assigned to one student if $T(c_i) = T(c_j)$.

We define S as the set of students. For each student s the requested number r_s of courses is given, which on average is about four. P_s is defined as the set of positions on the preference list for which student s filled in a course. Table 3.1 gives a few examples of preference lists. Column i gives the course on position i of the preference

list of the corresponding student. The parameter c_{sp} is introduced as the course on position p of the preference list of student s.

Table 3.1: Example of preference lists

Student	r_s	1	2	3	...	10
s040202	4	DAC03	DA247	DA125	...	DA405
s040203	4	DA619	DA125	DA201	...	DA616
s040204	1	DA418	DA242			

In summary: the input of problem formulation No. 1 consists of:

- A set T of timeslots.
- A set C of courses; for every course $c \in C$ a timeslot $T(c)$ and a maximum number C_c^{max} of participating students is given.
- A set S of students; for every student $s \in S$ a set P_s of filled positions of the preference list, a course c_{sp} for each position $p \in P_s$ and a requested number r_s of courses is given.

The goal is to assign as many courses to students as possible, while:

- The number of courses assigned to student s does not exceed r_s.
- Courses assigned to a student are on his/her preference list.
- Courses assigned to a student do not conflict in time.
- No course exceeds its maximum number of assigned students.

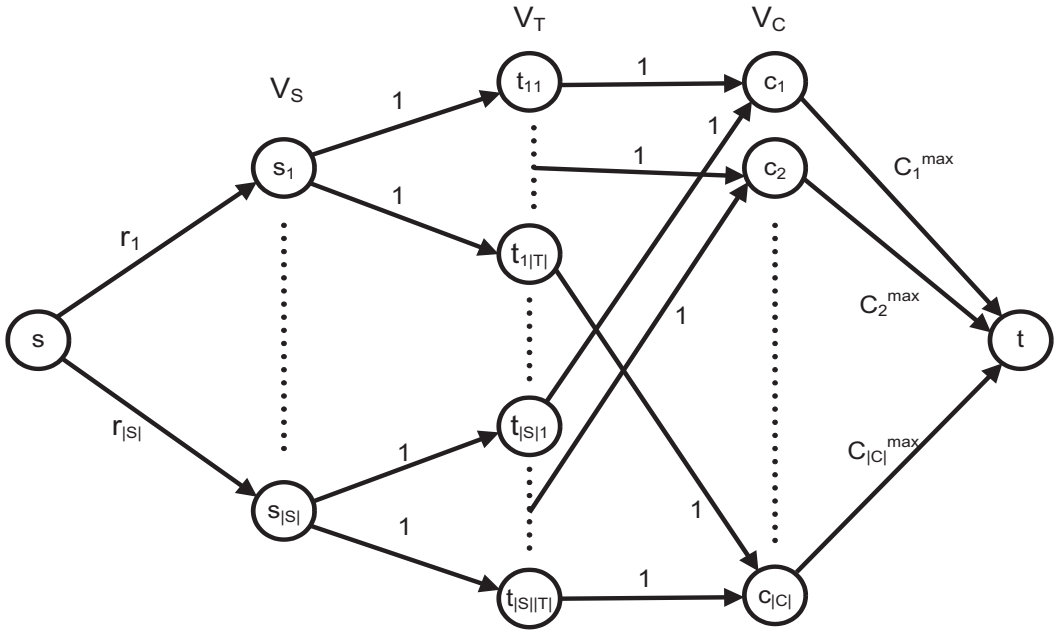

Figure 3.1: The network flow model of problem formulation No. 1

As stated before, problem formulation No. 1 can be modeled as a network flow problem. An illustration of the network with its arc capacities can be found in Figure

3.1. The layered network has a source node connected to the first layer of nodes of which each node corresponds to a student. The set of student nodes is denoted by V_S. The capacity of the arc between a student node and the source node is equal to the requested number of courses of the student.

For each student and each timeslot, a node is defined in the second layer, which is denoted as the set of nodes V_T. Therefore, this set V_T contains $|S||T|$ nodes. Each student node has $|T|$ outgoing arcs with capacity one to a node in V_T that corresponds to a timeslot.

The third layer of the graph contains a node for every course. This set of course nodes is denoted by V_C. Each course is given in exactly one timeslot. Therefore, if a course is on the preference list of the student, this course node is connected with the node in V_T that corresponds to its timeslot and this student. For example, in Figure 3.1 course number 1 is given in timeslot 1. Because course 1 is chosen by student 1 and student $|S|$, there are arcs from nodes t_{11} and $t_{|S|1}$ to course node c_1.

Each course node is connected to the sink with its maximum number of students as its capacity. Maximizing the flow through this network maximizes the number of assigned courses.

3.2.2 Problem formulation No. 2

As we received the first data set from the Department of Industrial Design, we were very surprised: there suddenly were also *lower bounds* C_c^{min} on the number of students participating in course c. This yields the new option of a course being canceled if there are not enough students attending. An exception to this rule is that a course cannot be canceled if it is compulsory for some students. This new option can not be modeled within the flow model. In fact, the problem becomes NP-hard; see Section 3.3. After looking at the data more carefully and after conversations with the planner of the Department of Industrial Design we noticed there were additional restrictions. The remainder of this subsection explains these additional restrictions and describes the problem into more detail.

An academic year is divided into a certain number of teaching periods. For instance, the academic year 2005-2006 is divided into six teaching periods. We define such a period as a *block*. The Department of Industrial Design wants us to schedule two consecutive blocks simultaneously. Therefore, set B is introduced as the set of blocks that have to be scheduled simultaneously.

In problem formulation No. 1 we assumed the *workload* of all courses was equal. However, there are courses with a workload of 40 hours and courses with a workload of 80 hours. In the remainder of this chapter a workload of 1 corresponds to a workload of 40 hours. In Section 3.3 we prove that having courses with a workload 1 and courses with a workload 2 makes the problem already NP-hard.

In problem formulation No. 2 the definition of r_s is adjusted into the requested total workload of student s for all blocks in B. For every student s, also a maximum requested workload r_{sb} for each block $b \in B$ separately is given. This is necessary because the requested workload of a student is not always equally divided over all blocks $b \in B$. One example is when $B = \{b_1, b_2\}$ and student s has to do a practical

training in block b_2 and wants to do two courses in block b_1. In this case he sets: $r_s = 2$, $r_{sb_1} = 2$ and $r_{sb_2} = 0$.

It was assumed in problem formulation No. 1 that a course has one meeting every week, hence it has a single timeslot. However, there are also courses which have more than one weekly meeting, hence have more than one timeslot. If such a course is assigned to a student, the student has to be available at all timeslots.

The set C of courses contains courses with multiple sections, meaning that the course is repeated during the week. Table 3.2 contains course DA242 as an example. The timeslots in the table are encoded. For example, the code B1TM2 stands for the second part of Tuesday morning in block 1. Course DA242 has five sections which all have two timeslots.

Table 3.2: Examples of courses

Course	Section	Timeslots	wlb1	wlb2	Min	Max
DA242	DAG242-1	B1TM2, B1TA1	1	0	0	30
	DAG242-2	B1TM2, B1TA2	1	0	0	30
	DAG242-3	B1TM2, B1WA1	1	0	0	30
	DAG242-4	B1TM2, B1WA1	1	0	0	30
	DAG242-5	B1TM2, B1WA2	1	0	0	30
DA247	DAG247-1	B1WA2, B2WA2	1	1	5	15
	DAG247-2	B1WA2, B2WA2	1	1	5	15

We define I as the set of sections offered to the students. For each course there is at least one section in the set I and for every section $i \in I$ its course $c(i) \in C$ is given. In problem formulation No. 2 there are no maximum and minimum number of students for a course like in problem formulation No. 1, but a minimum number C_i^{min} and a maximum number C_i^{max} of students for each section $i \in I$. The meeting times for each section $i \in I$ are given as the set $T(i) \subseteq T$ of timeslots, where T denotes the set of meeting times of the blocks in B. There are a few courses, for example literature studies, which are not assigned to a timeslot and thus $T(i) = \emptyset$ for a section i of such a course.

For each section $i \in I$ and block $b \in B$ the parameter $w(i, b)$ is defined as the workload of section i in block b. Hence for a section i with a workload of 80 hours in block b we have $w(i, b) = 2$. It is necessary to define the workload for each section and each block, because there are sections with workload 2 distributed over different blocks. See for example course DA247 in Table 3.2. The workloads of a section in block 1 and 2 are denoted with $wlb1$ and $wlb2$ in Table 3.2. Min and Max denote the minimum and maximum number of students for a section.

It is possible that courses are given in two blocks that are not scheduled simultaneously. If a student has been assigned to a section of such a course in the first block, then he needs to be assigned to the same section of this course in the second block. This implies there are students already preassigned to sections at the moment the schedule of the second block is being constructed. Therefore, we introduce the set F

of *fixations* which contains combinations (s, i) for which section i is already assigned to student s.

If students have specific needs, for instance when they almost finish their studies and only have one course left to pass, a course on the preference list of the student can be set to *urgent*. Also first year students have to do some compulsory courses, these courses can also be set to urgent. Courses marked as urgent should be assigned as much as possible, even if it would lead to course attendance below the minimum. We define U as the set of combinations (s, p) for which course c_{sp} is considered urgent for student s.

In summary: the input of problem formulation No. 2 consists of:

- A set B of blocks that have to be scheduled simultaneously.
- A set T of timeslots.
- A set C of courses; for every course c its set I_c of sections.
- A set S of students; for every student s a total requested workload r_s, a maximum requested workload r_{sb} for each block separately, a set P_s of filled positions on the preference list and for each position $p \in P_s$ a course c_{sp} is given.
- A set I of sections; for every section i its course $c(i)$, a minimum C_i^{min} and maximum C_i^{max} number of students, a set $T(i) \subseteq T$ of timeslots and its workload $w(i, b)$ for each block $b \in B$ is given.
- A set U of combinations (s, p) for which course c_{sp} is urgent for student s.
- A set F of combinations (s, i) for which section i is already preassigned to student s.

Our main goal is to assign workload to students as much as possible, obeying the following hard constraints:

- Maintaining the number of students in a section below the maximum size prescribed.
- The workload assigned to student s is less than or equal to r_s.
- The workload assigned to student s in block b is less than or equal to r_{sb}.
- Sections assigned to a student do not conflict in time.
- Students are only assigned to a section of a course on their preference list.
- Students are only assigned to one section of a course.
- Student s is assigned to section i if $(s, i) \in F$.

Two soft constraints are that all urgent courses are assigned and all section size lower bounds are respected.

3.3 Some complexity results

Cheng et al. (2003) prove that the Student Scheduling Problem is NP-hard. The Student Scheduling Problem is a special case of problem formulation No.2, with as main difference that the total workload of the courses on the preference list is always equal to the requested workload. Therefore, all courses on the list of a student have to be assigned in the student scheduling problem. This proves already that the timetabling problem defined as problem formulation No. 2 is NP-hard. This doesn't give any

guidance on the prospects of solving any particular subproblem. In the rest of this section we prove again that problem formulation No. 2 is NP-hard by identifying some independent NP-hard subproblems. These subproblems result from adding one constraint to problem formulation No. 1.

In this section we sometimes assume that there are no timeslots, which is the same as giving each course a unique timeslot. In the network flow model of problem formulation No. 1 we could then connect all nodes V_S in the first layer directly to the nodes V_C of the third layer if this course is on the preference list of the corresponding student. These edges would have a capacity of one.

In the first subproblem, the additional constraint is a lower bound on the number of students in a course. There are no timeslots, there is only one section for each course c with a minimum and a maximum number of participating students. The workload of all courses is one, and only one block has to be scheduled. Formally, problem P_{min} is defined as follows:

> **Instance:** A set C of courses; for every course $c \in C$ a minimum number C_c^{min} and a maximum number C_c^{max} of participating students. A set S of students; for every student $s \in S$ a preference list of some courses in C, and a number r_s of requested courses.
>
> **Question:** Does there exist an assignment such that (i) every student s gets exactly r_s courses from its preference list, and such that (ii) for every course c the number of assigned students is either zero (if the course does not take place) or falls between the bounds C_c^{min} and C_c^{max}?

Theorem 3.3.1. *Problem P_{min} is NP-complete.*

Proof. The proof is done by reduction from the *exact cover by 3-sets* problem: Given a ground set $X = \{x_1, \ldots, x_n\}$ and a set $T = \{t_1, \ldots, t_m\}$ of 3-element subsets of X, can one select $T' \subseteq T$ such that every element of X occurs in exactly one member of T'? Exact cover by 3-sets is NP-complete, Garey and Johnson (1979).

From an instance of the exact cover by 3-sets problem, we construct a corresponding instance of problem P_{min} with n students $x_1, \ldots, x_n$ and with m courses $t_1, \ldots, t_m$. Every student s has a demand of one course ($r_s = 1$), and every course c has minimum and maximum capacity three ($C_c^{min} = C_c^{max} = 3$).

Assume X possesses an exact cover T'. Assign student x_s to course t_c if and only if $x_s \in t_c$ and $t_c \in T'$. Since T' is an exact cover of X, every student x_s will be assigned to exactly one course t_c. The course t_c is assigned to three students if it is in T', and to zero students if it is not in T'. This shows that the constructed instance of P_{min} is a yes-instance. The converse statement can be seen in a similar way. □

In the second subproblem, we take problem formulation No. 1 and additionally allow courses with a workload of 2. We consider a situation with only one section for each course c, only a single block without any timeslots and there is no minimum capacity of courses. Problem P_{wl} is defined as follows:

> **Instance:** A set C of courses; for every course $c \in C$ a workload $wl_c \in \{1, 2\}$ and a maximum capacity C_c^{max} of participating students. A set S

of students; for every student $s \in S$ a preference list of some courses in C, and a desired workload r_s.

Question: Does there exist an assignment such that (i) every student s gets courses with a total workload r_s from its preference list, and such that (ii) for every course c the number of assigned students is at most C_c^{max}?

Theorem 3.3.2. *Problem P_{wl} is NP-complete.*

Proof. The proof is done by reduction from the 3-SAT variant where every variable occurs exactly twice in negated and exactly twice in unnegated form. Consider an arbitrary instance of this 3-SAT variant.

- For every variable x_k, we introduce two corresponding students $st(x_k)$ and $st(\overline{x_k})$ who both request a workload of two.
- For every variable x_k, we also introduce a corresponding variable-course $VC(x_k)$ which has a workload of two and a capacity of one. $VC(x_k)$ is in the preference list of $st(x_k)$ and $st(\overline{x_k})$.
- For every clause c_j, we introduce a clause-course $CC(c_j)$ with a workload of one and a capacity of two. $CC(c_j)$ is in the preference list of a student $st(x_k)$ (respectively $st(\overline{x_k})$) if and only if x_k (respectively $\overline{x_k}$) occurs as a literal in clause c_j.

Note that in any feasible assignment, student $st(x_k)$ (respectively student $st(\overline{x_k})$) will either do course $VC(x_k)$ or the two courses $CC(c_{j_1})$ and $CC(c_{j_2})$ for which literal x_k (respectively literal $\overline{x_k}$) occurs in clauses c_{j_1} and c_{j_2}.

Assume that the 3-SAT instance is a yes-instance, and consider a corresponding satisfying truth-assignment. If x_k is set to TRUE, then we assign student $st(x_k)$ to $VC(x_k)$, and student $st(\overline{x}_k)$ to the two clause-courses that correspond to the clauses containing $\overline{x}_k$. If x_k is set to FALSE, we assign $st(x_k)$ to the clause-courses that correspond to the clauses containing x_k, and student $st(\overline{x_k})$ to $VC(x_k)$. Then each student receives his requested workload, and every course $VC(x_k)$ gets only a single student. Since every clause has at most two FALSE literals, the corresponding clause-course will get at most two students. So every yes-instance of the 3-SAT problem leads to a yes-instance of the timetabling problem.

Now assume that the constructed instance of problem P_{wl} is a yes-instance. Then every student $st(x_k)$ receives a workload of 2, which implies that the student must either be assigned to a variable-course $VC(x_k)$, or to two clause-courses $CC(c_{j_1})$ and $CC(c_{j_2})$. If student $st(x_k)$ is assigned to $VC(x_k)$, we set x_k to TRUE. If student x_k is assigned to two clause-courses, then we set x_k to FALSE. Since each clause-course $CC(c_j)$ is assigned to at most two students, every clause contains at most two FALSE literals. Hence, every yes-instance of P_{wl} corresponds to a yes-instance of 3-SAT. □

In the third subproblem the possibility that a course has multiple timeslots is added to problem formulation No. 1. We consider the timetabling problem where all courses have a workload of one, there is no minimum capacity for courses, one section for each course c and one block has to be scheduled. Problem P_{mt} is defined as follows:

Instance: A set T of timeslots; A set C of courses; for every course $c \in C$ a set $T(c) \subseteq T$ of timeslots and a maximum capacity C_c^{max} of participating students. A set S of students; for every student $s \in S$ a preference list of some courses in C, and a number of requested courses r_s.

Question: Does there exist an assignment such that (i) every student s gets exactly r_s courses from its preference list, and (ii) for every course c the number of assigned students is at most C_c^{max}, and such that (iii) the assigned courses to a student do not have a timeslot in common?

Theorem 3.3.3. *Problem P_{mt} is NP-complete.*

Proof. The proof is done by reduction from the *independent set* problem: Given a graph $G = (V, E)$ and an integer k, does there exist a subset $V' \subseteq V$ with $|V'| \geq k$ and $|V| = n$ such that no two vertices in V' are connected?

From an arbitrary instance of the independent set problem we construct a corresponding instance of problem P_{mt} with one student that requests k courses. For every edge $e \in E$ we introduce a timeslot t_e and for every node $v \in V$ we introduce a course c_v with capacity one and timeslots $t_e \in T(c_v)$ if v is a vertex of edge e. Note that the number of timeslots of a course is equal to the degree of the corresponding vertex in G. The preference list of the student contains all n courses.

Assume that the independent set instance is a yes-instance with independent set V' and $|V'| = k$. If $v \in V'$, then c_v is assigned to the student. The student receives its requested k courses and the assigned courses have no timeslot in common. Trivially, all courses have no more students assigned than their capacity.

Assume that the constructed instance of problem P_{mt} is a yes-instance. Then the student is assigned to k courses which all have different timeslots. This implies that the nodes corresponding with an assigned course are not connected and that the selected nodes form an independent set. Hence, every yes-instance of P_{mt} corresponds to a yes-instance of independent set. □

The independent set problem is polynomially solvable in the case all nodes have degree less than or equal to two. Therefore, the special case of problem P_{mt} where every course has one or two timeslots is not proven to be NP-complete. The complexity result of this problem is given in the following theorem.

Theorem 3.3.4. *P_{mt} is NP-complete, even if restricted to the class of instances in which $|T(c)| \leq 2, \forall c \in C$.*

Proof. The proof is done by reduction from the *chromatic index* problem: Given a simple graph $G = (V, E)$ and a positive integer K, can E be partitioned into disjoint sets $E_1, \ldots, E_k$, with $k \leq K$, such that, for $1 \leq i \leq k$, no two edges in E_i share a common endpoint in G.

From an instance of the chromatic index problem, we construct a corresponding instance of problem P_{mt} where each course has one or two timeslots. For every node $v_i \in V$ we introduce a timeslot $t(v_i)$ and for every edge $e_j \in E$ we introduce a course $C(e_j)$ with a capacity of 1 student. If $e_j = \{v_{i_1}, v_{i_2}\}$, then course $C(e_j)$ has timeslots

$t(v_{i_1})$ and $t(v_{i_2})$. We introduce K students, $s_1, \ldots, s_K$, which all have a preference list containing all courses. We define $p \equiv |E| \pmod K$. Students s_1 to s_p request $\lceil |E|/K \rceil$ courses and students s_{p+1} to s_K request $\lfloor |E|/K \rfloor$ courses.

Assume G contains a chromatic index K. Consider K sets E_i in the partition, of which some may be empty. Let E_1 and E_2 be two disjoint sets with $|E_1| > |E_2| + 1$, then there exist two disjoint sets E_1' and E_2' such that $|E_1' \cup E_2'| = |E_1 \cup E_2|$ and $|E_1'| = |E_1| - 1$ and $|E_2'| = |E_2| + 1$ and both E_1' and E_2' form a matching. Therefore we can assume: $||E_i| - |E_j|| \leq 1 \forall i, j$. This implies that there exists a solution to the chromatic index problem with p sets containing $\lceil |E|/K \rceil$ edges and the other sets containing $\lfloor |E|/K \rfloor$ edges. Now assign student s_i to course $C(e_j)$ if $e_j \in E_i$. The first p students are assigned to the courses corresponding to the p bigger sets E_i, so each student gets its requested number of courses. The courses assigned to a student all have different timeslots, because the edges in this set form a matching. Each edge is assigned to at most one set E_i, therefore each course is also assigned to at most one student. So every yes-instance of the chromatic index problem leads to a yes-instance of the timetabling problem.

Now assume that the constructed instance of problem P_{mt} is a yes-instance. If student s_i is assigned to course $C(e_j)$ then add edge e_j to set E_i. All assigned courses have different timeslots, therefore the edges in the same set are not connected and form a matching. There are K students which result in K disjoint sets E_i and all $|E|$ courses have been assigned, hence every constructed yes-instance of P_{mt} corresponds to a yes-instance of the chromatic index problem. □

The fourth subproblem considers problem formulation No. 1, but courses with multiple sections are added. We define this problem as P_{ms}. In this problem every course has a workload of one, no minimum number of students, only one block has to be scheduled and each section has at most one timeslot. Problem P_{ms} is defined as follows:

> **Instance:** A set T of timeslots; A set C of courses; A set I of sections; for every section $i \in I$ its course $c(i)$, a timeslot $t_i \in T$ and a maximum capacity C_i^{max} of participating students. A set S of students; for every student $s \in S$ a preference list of some courses in C, and a number r_s of requested courses.
>
> **Question:** Does there exist an assignment such that (i) every student s is assigned to exactly r_s sections of courses from its preference list, and (ii) for every section i the number of assigned students is at most C_i^{max} and (iii) a student is assigned to only one section of a course and (iv) the assigned sections to a student do not have a timeslot in common?

Theorem 3.3.5. *Problem P_{ms} is NP-complete.*

Proof. The proof is done by reduction from the 3-SAT problem. Consider an arbitrary instance of the 3-SAT problem.

- For each variable x in clause c_j we introduce a variable-course $VC(c_j, x)$. These courses have two literal-sections, $LS(c_j, x)$ and $LS(c_j, \overline{x})$, with capacity one. The

timeslots are chosen such that $T(LS(c_j, \overline{x})) = T(LS(c_{j+1}, x))$ and $T(LS(c_n, \overline{x})) = T(LS(c_1, x))$ where n is the number of clauses in which variable x occurs.

- For every clause c we introduce a clause-course $CC(c)$ with one clause section $CS(c)$ with capacity 2 and timeslot $T(CS(c))$.
- For every literal l in clause c we introduce a dummy-course $DC(c, l)$. For each dummy-course $DC(c, l)$ we introduce one dummy-section $DS(c, l)$, with capacity 1. If l is the literal corresponding with variable x, then timeslot of this dummy-section is equal to $T(LS(c, l))$.
- For every variable x, we introduce one student $st(x)$ that requests a number of courses equal to the number of times the variable x occurs in a clause. Its preference list contains the variable-courses $VC(c, x)$ if clause c contains variable x.
- For every literal l in clause c we introduce a student $st(c, l)$ that requests two courses from its preference list that contains three courses. Assume l is a literal of the variable x, then student $st(c, l)$ requests variable-course $VC(c, x)$, clause-course $CC(c)$ and the dummy-course $DC(c, l)$.

Assume that the 3-SAT instance is a yes-instance, and consider a corresponding truth-assignment. If variable x is TRUE, then $st(x)$ is assigned to the literal-sections $LS(c, x)$. If literal l of clause c is TRUE, then assign student $st(c, l)$ to literal section $LS(c, \bar{l})$ and $DS(c, l)$. For each FALSE literal l, student $st(c, l)$ is assigned to $LS(c, l)$ and to the clause-section $CS(c)$. All the sections assigned to $st(c, l)$ have different timeslots and the requested number of courses for student $st(c, l)$ is assigned. Because clause c is TRUE, at most two students are assigned to $CS(c)$ and all other sections are only assigned to at most one student. So every yes-instance of the 3-SAT problem leads to a yes-instance of the timetabling problem.

Now assume that the constructed instance of the timetabling problem is a yes-instance. At most two students are assigned to $CC(c)$. Therefore, at least one of the literals of the clause c is not assigned and evaluates to TRUE. If student $st(x)$ is assigned to the literal-sections $LS(c, x)$, then the variable x is evaluated to TRUE and if this student is assigned to the $LS(c, \overline{x})$ it is FALSE. If student $st(c, l)$ is assigned to literal section $LS(C, \bar{l})$, then the literal l is evaluated to TRUE. So every yes-instance of the timetabling problem leads to a yes-instance of the 3-SAT problem. □

Now we consider the special case of problem P_{ms}, where there are no timeslots. The complexity result of this problem is given in the following theorem.

Theorem 3.3.6. *Problem P_{ms} restricted to the class of instances without timeslots is polynomially solvable.*

Proof. The problem can be modeled as a network flow model. An illustration of the network with its arc capacities can be found in Figure 3.2.
The layered network has a source node connected to the first layer of nodes of which each node corresponds to a student. The set of student nodes is denoted with V_S. The capacity of the arc between a student node and the source node is equal to the requested number of courses of the student and takes care that a student doesn't get more courses assigned than requested.

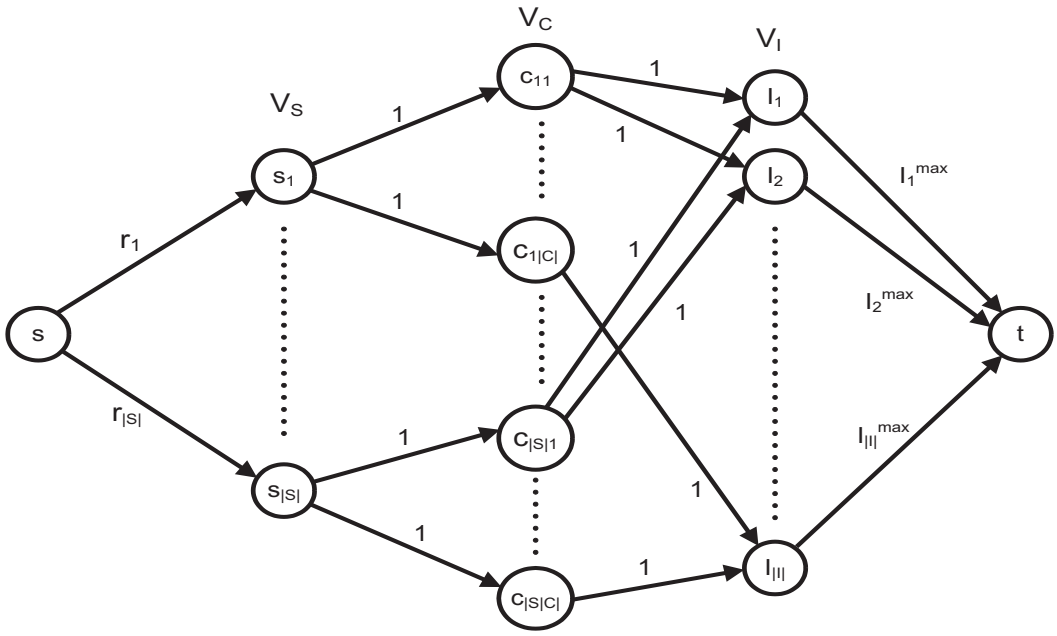

Figure 3.2: The network flow model for problem P_{ms} without timeslots

For each combination of a student and a course a node is defined in the second layer. This layer is denoted as the set V_C of course-nodes which contains $|S||C|$ nodes. There is an outgoing arc with capacity one from a student node to a node in V_C if the course is on the preference list of the corresponding student. Each student node has therefore a number of outgoing arcs equal to the length of the preference list. In Figure 3.2 both student 1 and student $|S|$ have course 1 and course $|C|$ on their preference lists.

The third layer of the graph contains a node for every section. This set of section-nodes is defined as V_I. Between section-nodes and course-nodes there is an arc with capacity 1 if this section belongs to this course. For example, in Figure 3.2 course number 1 has two sections (I_1 and I_2) and course $|C|$ has one section ($I_{|I|}$).

Each section-node is connected to the sink with its maximum number of students as its capacity. Maximizing the flow through this network maximizes the number of assigned courses. □

3.4 The lexicographic optimization algorithm

To build a schedule that best fits the needs for the students, the problem is formulated and solved as a *lexicographic optimization* problem. Lexicographic optimization is a form of multi-criteria optimization. Assume we have objectives $f_i, i = 1, \ldots, m$. If solution x_1 optimizes f_1, then we say solution x_2 optimizes f_2 under the condition $f_1(x_2) = f_1(x_1)$. Recursively x_i optimizes f_i under the constraining conditions that $f_1(x_i) = f_1(x_1), \ldots, f_{i-1}(x_i) = f_{i-1}(x_{i-1})$.

The timetabling problem is split into four subproblems which are formulated as

an integer linear programming problem. The goals of the four subproblems are:

1. Maximize the number of assigned courses with an urgency.
2. Minimize the shortage of students to reach the minimum number of students of a section. Because of urgencies, some sections must be taught, but may not have enough students with the corresponding course on their preference list. As many students as possible are assigned to those sections.
3. Maximize the total assigned workload. We try to assign a workload r_s to every student s.
4. Create a better timetable by optimizing the following goals:
   - To assign as many courses as possible on top of the preference lists,
   - To spread students as equally as possible over the sections of a course,
   - To discourage that one student gets a lot of courses which are on the bottom of his preference list,
   - To minimize the number of sections that have to be taught.

All parameters have already been introduced in Section 3.2. What is left to define are the decision variables. These are defined as follows:

$$y_i := \begin{cases} 1 & \text{if section } i \text{ is assigned to one or more students} \\ 0 & \text{otherwise} \end{cases}$$

$$z_{si} := \begin{cases} 1 & \text{if section } i \text{ is assigned to student } s \\ 0 & \text{otherwise} \end{cases}$$

The following constraints have to be fulfilled in all four subproblems:

$$\sum_{i \in I_{csp}} z_{si} \leq 1 \qquad \forall s \in S, \forall p \in P_s \tag{3.1}$$

$$\sum_{p \in P_s} \sum_{i \in I_{csp}} w(i,b) z_{si} \leq r_{sb} \qquad \forall s \in S, \forall b \in B \tag{3.2}$$

$$\sum_{p \in P_s} \sum_{i \in I_{csp}} \sum_{b \in B} w(i,b) z_{si} \leq r_s \qquad \forall s \in S \tag{3.3}$$

$$\sum_{s \in S} \sum_{p \in P_s | i \in I_{csp}} z_{si} \leq C_i^{max} y_i \qquad \forall i \in I \tag{3.4}$$

$$\sum_{p \in P_s} \sum_{i \in I_{csp}, t \in T(i)} z_{si} \leq 1 \qquad \forall s \in S, \forall t \in T \tag{3.5}$$

$$z_{si} = 1, y_i = 1 \qquad \forall s \in S, \forall i \in I | (s,i) \in F \tag{3.6}$$

$$y_i \in \{0,1\} \qquad \forall i \in I \tag{3.7}$$

$$z_{si} \in \{0,1\} \qquad \forall s \in S, \forall i \in I \tag{3.8}$$

Constraint (3.1) takes care that at most one section of a course is assigned to a student. The workload assigned to a student has to be less than or equal to the

requested workload each block separately and all blocks together. This is fulfilled by constraints (3.2) and (3.3). Constraint (3.4) enforces that the maximum number of students for a section is not exceeded and constraint (3.5) imposes that at each timeslot only one section is assigned to each student. If $(s, i) \in F$ then section i has to be assigned to student s, which is fulfilled by constraint (3.6).

As explained above, the problem is split into four subproblems which are solved sequentially. The goal of the first subproblem is to maximize the number of assigned courses with an urgency. The constraint that a section needs to have more than a minimum number of students is not a restriction in this subproblem, because at least one section of a course must be given if there is a student with an urgency for this course. This first subproblem can be solved with the following ILP formulation:

$$\max \sum_{(s,p)\in U} \sum_{i\in I_{csp}} z_{si} = U^{max}$$

subject to:

$$(y, z) \text{ satisfy } (3.1) - (3.8)$$

The second subproblem minimizes the total shortage S_{min} of students to reach the minimum number of students of a section, keeping the maximum number of assigned courses with an urgency equal to U^{max}. Sections that have to be given because they are assigned to students with an urgency for the corresponding course, are assigned to other students such that the minimum number of students for those sections is reached. The decision variable s_i is defined as the shortage of students for section i and gets a value larger than zero if it is not possible to assign at least its minimum number C_i^{min} of students. This gives the following ILP formulation:

$$\min \sum_{i\in I} s_i = S^{min}$$

subject to:

$$\sum_{(s,p)\in U} \sum_{i\in I_{csp}} z_{si} = U^{max}$$

$$\sum_{s\in S} \sum_{p\in P_s | i\in I_{csp}} z_{si} + s_i \geq C_i^{min} y_i \qquad \forall i \in I$$

$$s_i \in \mathbb{Z}_+ \qquad \forall i \in I$$

$$(y, z) \text{ satisfy } (3.1)\text{-}(3.8)$$

The third subproblem maximizes the total workload assigned to students with the restrictions that U^{max} and S^{min} keep their optimal values. This maximum workload is denoted by W^{max} and is determined by the following model:

$$\max \sum_{s\in S}\sum_{p\in P_s}\sum_{i\in I_{c_{sp}}}\sum_{b\in B} w(i,b)z_{si} = W^{max}$$

subject to:

$$\sum_{i\in I} s_i = S^{min}$$

$$\sum_{(s,p)\in U}\sum_{i\in I_{c_{sp}}} z_{si} = U^{max}$$

$$\sum_{s\in S}\sum_{p\in P_s | i\in I_{c_{sp}}} z_{si} + s_i \geq C_i^{min} y_i \qquad \forall i \in I$$

$$s_i \in \mathbb{Z}_+, \qquad \forall i \in I$$

$$(y,z) \text{ satisfy (3.1)-(3.8)}$$

To 'optimize' the timetable a fourth subproblem is solved in which we try to:

- Minimize the average position of the assigned courses,
- Minimize the number of students that are assigned to more than one course on the positions 7 up to 10 on their preference list,
- Spread the students over the sections of a course,
- Minimize the number of sections that must be taught.

The objective function is separated into four terms and has to be minimized under the restrictions that U^{max}, S^{min} and W^{max} keep their optimal values.

The first term in the objective function of the fourth subproblem is: $W_p \sum_{s\in S}\sum_{p\in P_s}\sum_{i\in I_{c_{sp}}}\sum_{b\in B} w(i,b)pz_{si}$. Note that it contains a weight factor W_p. This objective function assigns courses as high as possible on the preference lists. Assigning a course on top of a preference list, $p = 1$, adds only once the workload to the objective function. If a course on the bottom of the list, $p = 10$, is assigned, this adds 10 times the workload to the objective function.

We discourage that one student gets a lot of courses of his 7th up to 10th position of his preference list. If a student gets more than one course from these positions, then a penalty W_e is paid for each 'extra' course from these positions. Therefore, the decision variable E_s is introduced for every student s. This variable is equal to the 'extra' number of courses assigned to student s which are on the 7th up to 10th position of his preference list.

If a course has multiple sections, students have to be spread as equally as possible over the sections. Therefore, the decision variable I_c^{max} is introduced as the number of students assigned to the section of course c with the most students assigned. Also the spread S_c of course c is introduced and is equal to the sum over all sections of course c of the difference between I_c^{max} and the assigned number of students in each section. S_c is added to the objective function with a weight factor W_s.

To save time for teachers, the fourth and last goal is introduced. It minimizes the number of sections that have to be taught. The weight factor W_i has been introduced and a fourth term has been added to the objective.

This results into the final ILP formulation:

$$\min W_p \sum_{s\in S}\sum_{p\in P_s}\sum_{i\in I_{c_{sp}}}\sum_{b\in B} w(i,b)pz_{si} + W_e\sum_{s\in S}E_s + W_s\sum_{c\in C}S_c + W_i\sum_{i\in I}y_i$$

subject to:

$$\sum_{s\in S}\sum_{p\in P_s|i\in I_{c_{sp}}} z_{si} \le I^{max}_{c(i)} \qquad \forall i\in I$$

$$\sum_{i\in I_c}(I^{max}_c - \sum_{s\in S}\sum_{p\in P_s|i\in I_{c_{sp}}} z_{si}) = S_c \qquad \forall c\in C$$

$$\sum_{p=7}^{10}\sum_{i\in I_{c_{sp}}} z_{si} \le 1 + E_s \qquad \forall s\in S$$

$$\sum_{s\in S}\sum_{p\in P_s}\sum_{i\in I_{c_{sp}}}\sum_{b\in B} w(i,b)z_{si} = W^{max}$$

$$\sum_{i\in I} s_i = S^{min}$$

$$\sum_{(s,p)\in U}\sum_{i\in I_{c_{sp}}} z_{si} = U^{max}$$

$$\sum_{s\in S}\sum_{p\in P_s|i\in I_{c_{sp}}} z_{si} + s_i \ge C^{min}_i y_i, \qquad \forall i\in I$$

$$E_s \in \mathbb{Z}_+ \qquad \forall s\in S$$

$$I^{max}_c, S_c \in \mathbb{Z}_+ \qquad \forall c\in C$$

$$s_i \in \mathbb{Z}_+ \qquad \forall i\in I$$

(y,z) satisfy (3.1)-(3.8)

The first constraint enforces that I^{max}_c will contain the maximum number of students assigned to one of the sections of course c. The spread of the students over the sections of course c is determined by the second constraint. If more than one course on the positions 7 to 10 of the preference list of student s is assigned, then the value of E_s becomes positive. This is ensured by the third constraint. All other constraints have been explained before.

3.5 The computational results

The computational results for the academic year 2005-2006 are given in this section. This academic year was divided into six blocks. Blocks 1 & 2, blocks 3 & 4 and blocks 5 & 6 are scheduled simultaneously. In all blocks the meetings were on Tuesday

morning, Tuesday afternoon, Wednesday morning and Wednesday afternoon. Every morning and afternoon is split into two parts. So all blocks contain eight timeslots. More details about the input are given in Table 3.3. The abbreviation wl stands for workload.

Table 3.3: Data for academic year 2005-2006

Blocks	$\|T\|$	$\|S\|$	$\|C\|$	$\|I\|$	$\|U\|$	available wl	requested wl
1 & 2	16	356	51	79	590	1504	1416
3 & 4	16	328	64	88	279	1545	1288
5 & 6	16	302	58	89	151	1544	1333

The number of students requesting workload decreased during the academic year, because of students who were doing a practical training and students who were finishing their studies. In blocks 5 & 6 the average requested workload per student is larger than in blocks 3 & 4, which is caused by students who still hope to reach the required workload for the academic year by doing some 'extra' courses.

The large number of urgencies in blocks 1 & 2 can be explained by the fact that first-year students have been preassigned to courses, because they are not able to make a choice themselves. All first-year students have six compulsory courses, which they have to do in the first-year. These six courses are set as urgent and only two out of these six courses can be done in each block. The urgencies in blocks 5 & 6 can be explained by students who are finishing their education and only have some specific courses left to do.

The models introduced in Section 3.4 are solved by the standard IP solver CPLEX 11.2. The computations are done on an Intel core T8300, 2.4 GHz processor with 2.0 GB internal memory. The values of the weight factors were $W_p = 10, W_s = 1$, $W_e = 100$ and $W_i = 1$. Assigning courses on top of the preference list is much more important than trying to spread the students over sections and minimizing the number of sections. Therefore, W_s and W_i have been chosen much smaller. To take care that not one specific student gets only courses on the bottom of his list, we choose a value of W_e that is significantly bigger than W_p. The results for the academic year 2005-2006 are given in Table 3.4.

The computation time of CPLEX given in Table 3.4 is the time it needs to solve the fourth subproblem. The computation time of the first three subproblems is even shorter. What can be concluded is that the computation time of CPLEX is very small.

In blocks 1 & 2 a requested workload of 1416-1369 = 47 could not be assigned, in blocks 3 & 4 a requested workload of 27 and in blocks 5 & 6 a requested workload of 33 could not be assigned. Especially in blocks 1 & 2 this is partly caused by the small difference between the requested and available workload. However, the main causes are preference lists for which it was impossible to assign the requested workload. Examples of such wrongly chosen preference lists are:

- An empty preference list, because students did not hand it in on time,
- A preference list with less than 10 courses,

Table 3.4: Results for the academic year 2005-2006

	block 1 & 2	block 3 & 4	block 5 & 6
Runtime CPLEX (s)	1.99	1.28	4.01
U^{max}	439	273	134
S^{min}	0	0	0
W^{max}	1369	1261	1300
average position	3.386	3.763	3.114
bad positions	1	2	7
spread	22	32	46
booked sections	77	84	87

- A preference list with not enough different timeslots in one of the two blocks,
- A preference list with the same course in more positions. There was even a student with ten times the same course on his preference list.

If all students would hand in a preference list with 10 different courses and enough different timeslots, then in blocks 1 & 2 only five students would not be assigned to their requested number of courses and in blocks 3 & 4 and blocks 5 & 6 only three students.

Table 3.4 also shows that in blocks 1 & 2 only 439 out of 590 urgency requests could be assigned. This can be explained by the fact that in these blocks all six courses on the preference list of first-year students are set as urgent, of which at most 4 are assigned. This gives at least two not assigned courses with an urgency for each first-year student.

The average position denotes the average of the positions of the courses assigned to a student. On average students request a workload of 4, which usually corresponds to four courses. For example, if courses on the positions 1,3,5 and 7 are assigned, then the average position for this student is 4. Hence, it can be concluded that students are assigned a lot of courses which are at the top of their preference list.

If a student is assigned to i courses on 7th up to 10th position on his preference list, then he has $i - 1$ bad positions. Also from the number of bad positions it can be concluded that students are assigned to courses at the top of their preference lists. The number of bad positions increases slightly during the year.

Table 3.4 also shows the number of booked sections. If you compare those with the number of offered sections you see that in blocks 1 & 2 and blocks 5 & 6 two sections have not been assigned and in blocks 3 & 4 four sections have not been assigned.

The spread of the booked sections is also presented. The spread is equal to the value of $\sum_{c \in C} S_c$. Take as example block 1 & 2. The number of assigned sections is 77 and the number of courses is 51, which are 26 "extra" sections. A spread of 22 shows that the students have been spread equally over the sections of a course.

Each of the four parts of the objective function can be optimized separately by setting the other three weight factors to zero. This has been done and the results are shown in Table 3.5. The table shows that the best average position that is possible for the first two blocks is 3.382. This shows that we hardly lose good positions by

minimizing the number of sections and spreading the students over the sections of a course. The same can be said about the number of bad positions. This shows that excellent values of the weight factors have been chosen. The spread of the students over the sections of a course and the number of assigned sections is a bit further from the optimal values.

Table 3.5: Results for the academic year 2005-2006 if each of the four objectives is optimized separately

	block 1 & 2	block 3 & 4	block 5 & 6
average position	3.382	3.735	3.108
bad positions	1	2	7
spread	0	3	0
booked sections	64	68	71

3.6 Conclusions

We have formulated, analyzed and solved a real-world timetabling problem that showed up at the Department of Industrial Design of the TU Eindhoven. Our successful approach was a lexicographic optimization algorithm that contains four subproblems that we formulated as an ILP formulation. The running time that CPLEX needs for solving the ILP problems is negligible. An advantage of an ILP approach is its flexibility. Our experience is that the constraints of the timetabling problem change every academic year and even during the academic year.

The administration and the students of the Department of Industrial Design were highly satisfied with the timetables generated by our program. Most students now receive courses that are at the top of their preference lists. There still are a few students who are not satisfied, but in most cases this turns out to be solely their own fault: they failed to specify correct preferences in the correct format.

Chapter 4

Planning shunting movements at a railway station

In the Netherlands the capacity of the railway stations has become one of the main bottlenecks in the logistic planning proces. Next to all the timetabled passenger and cargo trains, many shunting movements between platform tracks and shunting areas are carried out in and around the large stations in the railway network. Shunting movements have to provide passenger trains with the right composition of rolling stock and facilitate the inside and outside cleaning and the short term maintenance of the rolling stock.

Constructing an executable plan for all train movements at a railway station has become an impractical job. The shunt planning is highly dependent on the rolling stock circulation and the timetable for the passenger trains and it is performed shortly before execution which results in a high time pressure. Advanced decision support systems would help shunt planners a lot in creating good shunt plans and speeding up this shunt planning.

As explained in Section 1.5, a shunt planning problem can be split into three subproblems, Freling et al. (2005):

1. A matching problem that matches arriving train units to departing train units.
2. A parking problem for storing train units at the shunting tracks.
3. A routing problem for routing train units to the shunting tracks.

In Section 1.5 the three problems are described in more detail. The routing problem is the central topic of this chapter.

Kroon et al. (2008) present a model that is capable of solving the matching and parking subproblem in an integrated manner. One of the main goals in their parking subproblem is to select positions and compositions of the trains at the shunt yard, such that the operations in the next morning can start up as smoothly as possible. Their model incorporates that trains are composed of several train units and tracks can be approached from two sides. Their model minimizes the number of train units from the same train service that have to be split and the number of tracks with multiple subtypes of train units parked at it.

Also DiStefano and Koci (2004) did research related to the parking problem. They looked at how to order trains on the available shunting tracks in order to minimize the number of required shunting movements on the next morning. They assume that each track is long enough to host the trains assigned to it. Their main objective is minimizing the number of required shunting tracks. They consider several variants of their shunting problems, distinguished from each other by the ends of the shunting tracks that can be used for entering or leaving these tracks. For several variants of their problem they provide computational complexity results.

Tomii et al. (1999) and Tomii and Zhou (2000) describe a genetic algorithm that handles storage of train units and several related processes, such as cleaning and maintenance. However, their shunting problem is of a less complex nature than the general shunting problem, since in their context at most one train unit can be parked on each shunting track at the same time.

Papers on positioning trams and buses in their storage depots have been written by Winter and Zimmermann (2000), Blasum et al. (1999), DiMiele and Gallo (2001) and Hamdouni et al. (2006). Winter and Zimmermann (2000) focus on storage areas in which trams are stored one behind the other in dead-end sidings. They look at the variant with the so-called midnight constraint. The midnight constraint means that the earliest departure takes place after the last arrival. Their model assigns trams to depot positions, thereby minimizing the number of necessary shunting movements. They also describe real-time dispatch strategies. Blasum et al. (1999) study similar problems focusing on a smooth start-up process of the tram system in the early morning. DiMiele and Gallo (2001) describe a model for parking buses in a storage area based on Minimal Non-Crossing Matching and Generalized Assignment. This model also takes into account the fact that the vehicles have different lengths. Moreover, they present an approach for dealing with the case without midnight constraint, so the earliest departure takes place before the last arrival. Another application of bus dispatching is described by Hamdouni et al. (2006). They generate robust solutions by having as little different bus types as possible in each lane of the depot, and by grouping in each lane the buses of the same type as much as possible together.

The problem of assigning passenger trains to platforms has some similarities with the routing problem. For this platform assignment problem the arrival and departure times of a set of trains at a certain railway station are given. The question is whether the trains can be routed through the railway station in such a way that trains do not conflict with each other, Zwaneveld (1997), Zwaneveld et al. (1996) and Zwaneveld et al. (2001). Trains are assigned to platforms such that there are no route conflicts between timetable trains and the number of necessary shunting movements is minimized. The problem is proved to be NP-hard and modeled as a weighted node packing problem on a conflict graph. A similar problem is studied by Billionnet (2003).

Cornelsen and DiStefano (2007) also look at assigning trains to platforms given a timetable. They model the platform assignment problem as a graph coloring problem on a conflict graph. The vertices of the graph represent the trains and two vertices are adjacent if the corresponding trains cannot be assigned to the same platform due to their arrival and departure times. The main difference with the model of Zwaneveld et al. (1996) is that Cornelsen and DiStefano (2007) don't take into account any

shunting movement nor the capacity of the switch zone. They distinguish between variants with and without the midnight constraint and present several complexity results and approximation methods.

Fuchsberger (2007) formulates a multi-commodity flow type model for the train platforming problem. The model has been tested successfully on real-life instances of the Swiss Railways. Carey and Carville (2003) describe a heuristic that adjusts a draft timetable of a station to obtain a conflict-free routing.

The goal of the routing problem is to determine a time instant and route for the necessary shunting movements. A prototype model has been developed in earlier research by Van den Broek (2002). This model assumes that the routes for all the shunting movements are fixed beforehand and verifies that each shunting movement can be scheduled at a time instant that each element of the infrastructure is occupied by at most one movement at a time. The model is a mixed integer program that is solved by CPLEX.

This chapter is based on Van den Broek and Kroon (2007). The rest of the chapter is structured in the following way. Section 4.1 contains a detailed description of the problem. Two MIP models are described in Section 4.2. Both minimize the number of shunting movements that can not be planned. In the first model the routes of the trains are fixed beforehand and in the second model the routes are determined by the model. Section 4.3 presents computational results for some larger railway stations in the Netherlands for both models. Some concluding remarks are given in Section 4.4.

4.1 Problem description

At the larger railway stations, timetable related shunting movements between platform tracks and shunting areas are necessary in the following cases:

1. *Extending a train.* Rolling stock is added to a passenger train in order to increase the train's capacity. A shunting movement is necessary to bring the train unit from a shunting area to a platform track.

2. *Shortening a train.* Rolling stock is uncoupled from an arriving passenger train. A shunting movement is necessary to bring the uncoupled train unit from a platform track to a shunting area.

3. *Starting trains.* These are the first trains in the morning that have to depart from a station. To provide these trains with rolling stock, a shunting movement from a shunting area to a platform track is necessary.

4. *Ending trains.* These are trains that arrive at a station and after arrival, the rolling stock of these trains is not used anymore on the same day. The rolling stock has to be brought from a platform track to a shunting area.

Besides the timetable related shunting movements, also many shunting movements related to the cleaning and maintenance of the rolling stock have to be carried out.

However, these shunting movements usually take place at the shunting areas themselves, without too much interaction with the timetabled trains. We only consider shunting movements that use infrastructure outside the shunting areas.

For each shunting movement, an appropriate route over the railway infrastructure and an appropriate time instant have to be determined. This is particularly relevant for the shunting movements between a shunting area and a platform area of a station. The capacity of the infrastructure between those areas has to be shared by passenger trains, cargo trains and shunting movements. The goal is to schedule and route the shunting movements between the passenger and cargo trains. This section gives a formal description of this planning problem.

Each train movement has a unique time instant which corresponds with an event on the platform tracks. This unique time instant is called the *plan time* of the movement. The plan time of an arriving passenger train corresponds with the arrival time and the plan time of a departing passenger train is equal to the departure time of that train. The time instant at which a cargo train passes a platform track is its plan time. For a shunting movement the plan time corresponds with the arrival or departure on a platform track.

Each shunting movement has a given departure and arrival track as input. These tracks can be a shunting area, a shunting track or a platform track. During day time a large part of the rolling stock serves as a passenger train and will not be parked at a shunting area. This means that the capacity of the shunting areas will be sufficient during day time and only has to be verified during night. It is assumed that the capacity and the detailed layout of a shunting area are not relevant. The focus is on the capacity of the infrastructure *between* the platform area of a station and the shunting area, not on the shunting area itself. Because the rolling stock circulation is known, it has been verified already whether the capacity of the shunting area is sufficient. As a consequence the shunting area can be seen as a set of tracks with sufficient capacity.

Depending on the infrastructure of a railway station, a shunting movement can be routed along several routes to get from its departure track to its arrival track. These routes differ in the used tracks, switches and crosses. The possible routes between a pair of tracks consist of one *priority route* and a maximum of nine possible *alternative routes*.

Moreover, each shunting movement has a feasible *time window* which contains the allowed plan times of the shunting movement. This time window is given by an earliest and a latest possible plan time which are based on the availability of railway tracks, platform tracks, and shunting tracks. For example, a shunting movement bringing empty rolling stock from a platform track to a shunting area cannot start before the passengers got out after the arrival of the rolling stock on the platform track and has to be completed before the next train arrives at the same platform track.

Passenger and cargo trains are planned far before the actual operations. *Planned* means that the arrival- and departure tracks, the route and the plan time of a movement are fixed. Shunting movements are train movements with a relatively low priority. They are preferably planned only briefly before operations. Otherwise there is the risk that they have to be replanned several times, e.g. due to a modified rolling

stock circulation or due to the fact that additional passenger or cargo trains have to be facilitated on the infrastructure.

Planners build some robustness into the plans by taking into account a certain *headway time* between each pair of train movements. This means that a certain minimum number of minutes has to be scheduled between the plan times of two consecutive movements that use a common element of the infrastructure. This minimum amount of time is given by the planning norms. These norms depend for example on whether trains are cargo trains or passenger trains, or whether trains are arriving or departing trains.

A *saw movement* is a movement that arises when the arrival track can not be reached from the departure track by one forward movement, see Figure 4.1. Most of these saw movements arise when the arrival and departure track are parallel to each other or when at least one of the two tracks is a track which can only be approached from one side. Rolling stock that executes a saw movement has to change direction on a track in between. In Figure 4.1 such a track can be found at ②. Such a track is called a *saw track*. At a saw track the driver has to walk from one side of the train to the other side. This means that the plan times of the two parts of a saw movement have to be separated in time and results in the occupation of the saw track for a number of minutes. The minimum amount of time between the plan times of the two parts of a saw movement depends on the length of the train and is given. Usually, there are several saw tracks to choose from. Which saw track is chosen depends on the other train movements that use the possible saw tracks. However, we assume that for every saw movement, the saw track is given.

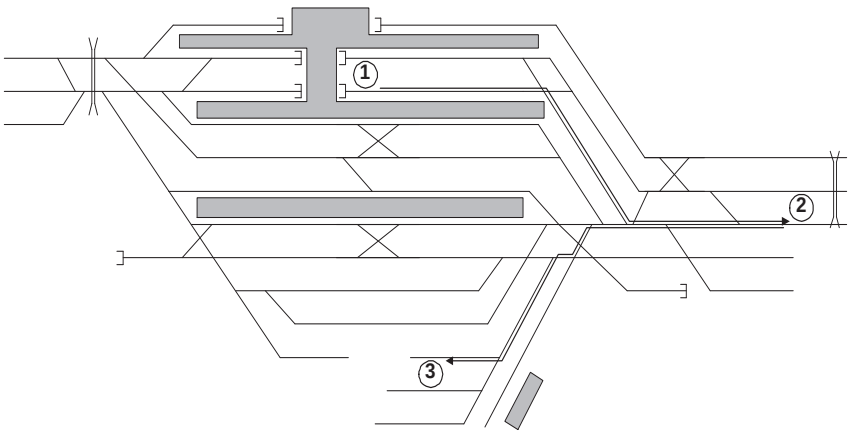

Figure 4.1: Example of a saw movement

A tool has been developed that determines for each shunting movement an appropriate route from its departure track to its arrival track within its time window. This route does not conflict with the train movements which have already been planned nor with other shunting movements. It is not allowed to change the arrival tracks, the departure tracks and the plan times of the passenger and cargo trains.

4.2 The mathematical programming model

Two MIP models have been developed, both will be described in this section. The models check whether it is possible to plan all the necessary shunting movements between the planned movements. In the first model, the route of each shunting movement is given and the plan time has to be selected. The second model is an extension of the first one. In the second model it is also allowed to select the route of each train movement from a pre-specified set.

4.2.1 Parameters used in the models

In this section we introduce all the parameters that we use in the MIP models. We start with introducing what a train movement exactly is. The safety headway times between two trains depend on whether the trains are departing or arriving. But is a cargo train that does not stop an arriving or a departing train? In our definition a train movement has to arrive at or depart from a platform track or a track parallel to the platform tracks. A cargo train that passes a station is split into two movements. The first one arrives at or parallel to a platform track and the second movement departs from this track. The set of all movements is denoted by S.

Some movements have been planned and some have not been planned. Therefore, a set S_p of planned movements and a set S_n of not planned movements is introduced. The set S_p contains all timetable trains, cargo trains and the already planned shunting movements. The set S_n contains the shunting movements that have not been planned yet.

The plan time of a movement is the time that the movement arrives at or departs from a platform track. In the model this is the number of minutes after the start time of the planning horizon. For every movement j, the release date r_j is the first possible time instant that is allowed to be the plan time of movement j. The due date d_j of movement j is the last possible time instant that is allowed to be the plan time of movement j. For a movement $j \in S_p$, the release date and the due date are equal to the already determined plan time. Each movement $j \in S_n$ has a time window $[r_j, d_j]$, which contains all possible time instants that are allowed to be the plan time of movement j.

Trains can arrive and depart from more than one track. A long passenger train can arrive at platform tracks 7A, 7B and 7C. Also, several movements use more tracks than just their arrival and departure tracks. These other tracks are part of the route and have to be empty when the movement is carried out. The set T_j is defined as the set of tracks that are used by movement j along its given route. This set contains at least the arrival and departure tracks of movement j.

To check that tracks on the route of a movement are not occupied, the concept of a *successor* of a movement is introduced: A successor of movement j with respect to track t is a movement that is allowed to use track t after movement j has arrived at track t or after it has left track t and there is still rolling stock left on this track. Every other movement that uses track t and is not a successor of movement j has to be carried out before movement j or after all successors of movement j have been carried

out. For example: if an ending train arrives at a platform, the arriving movement has as its successor the shunting movement that brings the rolling stock to a shunting area. As long as the shunting movement has not been carried out, no other movement is allowed to use the platform track. The set σ_j of successors of movement j and the set σ_j^t of successors of movement j with respect to track t can be deduced from the input data. If a track is a shunting area or empty after movement j, then movement j has no successor with respect to this track.

Trains that only pass a station are separated in an arriving movement and a departing movement. The departing movement is the successor of the arriving movement with respect to an appropriate track.

In the model, saw movements are split into two or even more movements. The first movement is the movement from the departure track to the saw track, and the second one departs from the saw track and arrives at the arrival track of the original movement. The second part is the successor of the first part.

Two movements are defined to be *route dependent* if they have an element of the infrastructure in common in their routes. Such an infra-element can be a track, a switch or a crossing. If two movements have no infra-element in common, then they do not directly influence each other's plan time. The plan times of two route dependent movements have to be separated by the headway time of which the value is given by the planning norms. The parameter l_{jk} is the required minimum amount of time between the plan times of the route dependent movements j and k if movement j passes the common infra-element before movement k. Note that the value of the headway times are independent of the routes of the movements. The transfer time of a driver and reversing the direction of the rolling stock are also covered by this parameter if movements j and k together form a saw movement.

Obviously, a movement which has a time window around 6:00 am does not influence a shunting movement which has a time window around 9:00 pm. If two movements have time windows that are in time far from each other, then the order of operation of the movements is known a priori. On the other hand, two movements are *time dependent* if their time windows differ less than the norm between those two movements. This means that movements j and k are time dependent if and only if $r_j < d_k + l_{kj}$ and $d_j > r_k - l_{jk}$. If two movements are route dependent and time dependent, then they are *dependent*. Summarizing, the definition of dependency of two movements is as follows:

Two movements are *dependent* if they have an element of the infrastructure in common in their routes and the order of the movements on the common infra-elements is unknown a priori. Two movements are independent if one of the two conditions is not satisfied. Note that the order of two movements is the same on all common infra-elements.

If two train units have to be combined, then they arrive at the track where they are combined from the same direction or from different directions. If they arrive from the same direction, then the order in which they enter the track is known and the train unit that arrives last is the successor of the one that arrives first. If the train units arrive from different directions and at least one of them has not been planned yet, then they may arrive in either order. Because of technical and safety reasons

they can not arrive at the same time, hence the movements are dependent. By giving each movement the combined train as its successor, the order of operation with the combined train is known and hence both movements are independent of that train. Now each movement can arrive at the track after the other one has arrived.

If a train arrives on a track and is split into two parts, then we have a similar situation. If both parts leave the track into the same direction, then the order in which they leave the track is known and the train unit that leaves last is the successor of the other. If they leave the track in different directions and at least one of them has not been planned yet, then they may leave the track in either order. Again, because of technical and safety reasons, they can not leave the track at the same time, so the departing movements are dependent. By giving the arriving train both departing movements as its successor with respect to the arrival track, the departing movements are independent of the arriving train. Now the departing movements can leave the track in either order.

Assume that movement k is a successor of movement j and that movement m uses their common track. To avoid that movement m is scheduled over the common track between movements j and k, movement m is dependent of the movements j and k if $d_m + l_{mj} > r_j$ and $r_m < d_k + l_{km}$. If movements j, k and m are not fulfilling these constraints, then it is known a priori whether m is carried out before j or after k. Hence, the order of operation of the three movements is known a priori and the movements are independent.

In the model the following parameter is used to indicate whether two movements j and k are dependent:

$$a_{jk} := \begin{cases} 1 & \text{if movements } j \text{ and } k \text{ are dependent} \\ 0 & \text{if movements } j \text{ and } k \text{ are independent} \end{cases}$$

4.2.2 Model with fixed routes (MFR)

The goal of this model is to verify whether it is possible to plan all the movements that have not been planned yet within their time windows. Therefore, the model minimizes the number of not planned movements that can not be planned within their time window. If the objective value is zero, then it can be concluded that the capacity of the infrastructure is sufficient. If the objective value is strictly positive, then not all the shunting movements in S_n can be planned within their time window.

The decision variables used in the model with fixed routes are the following:

- y_j = the plan time of movement j in minutes
- $U_j = \begin{cases} 1 & \text{if movement } j \text{ can not be planned within its time window} \\ 0 & \text{if movement } j \text{ can be planned within its time window} \end{cases}$
- $x_{jk} = \begin{cases} 1 & \text{if movement } j \text{ is operated before movement } k \text{ on the common infra} \\ 0 & \text{otherwise} \end{cases}$

The decision variable y_j is a real variable which gives the plan time of movement j as the number of minutes after the start time of the planning horizon. The decision

variable U_j can be derived from the decision variable y_j. If the determined plan time y_j falls in the time window $[r_j, d_j]$, then U_j is set to zero. Otherwise the variable will be set to one. Both types of decision variables are only defined for movements $j \in S_n$. For planned movements $j \in S_p$, the plan time y_j is given and U_j is always zero. These values are already substituted in the model. The decision variable x_{jk} is only defined if movements j and k are dependent and gives the order in which the movements j and k have to be operated on the common infrastructure elements.

The problem with fixed routes is formulated as the following MIP model:

$$\min \quad \sum_{j \in S_n} U_j$$

subject to:

$$y_j \geq r_j \qquad \forall j \in S_n \tag{4.1}$$

$$y_j \leq d_j + U_j M \qquad \forall j \in S_n \tag{4.2}$$

$$x_{jk} + x_{kj} = 1 \qquad \forall j,k \in S \text{ with } a_{jk} = 1 \tag{4.3}$$

$$y_j + l_{jk} \leq y_k + (1 + U_j - x_{jk})M \qquad \forall j,k \in S \text{with} a_{jk} = 1 \tag{4.4}$$

$$y_j + l_{jk} \leq y_k \qquad \forall j,k \in S_n \text{with} k \in \sigma_j \tag{4.5}$$

$$x_{jm} = x_{km} \qquad \forall j,k,m \in S, \text{ with } a_{jm} = a_{km} = 1, \exists t \in T_m : k \in \sigma_j^t \tag{4.6}$$

$$x_{jk} \in \{0,1\} \qquad \forall j,k \in S \text{ with } a_{jk} = 1 \tag{4.7}$$

$$y_j \in \mathbb{R}^+ \qquad \forall j \in S \tag{4.8}$$

$$U_j \in \{0,1\} \qquad \forall j \in S \tag{4.9}$$

Constraints (4.1) enforce that a not planned movement gets a plan time that is larger than its release date and constraints (4.2) handle that the plan time of a not planned movement preferably does not exceed its due date. Constraints (4.3) and (4.4) take care that there is enough time between the plan times of two dependent movements. Constraints (4.2) and (4.4) contain a big-M, a large constant integer value. Remark that constraint (4.4) is binding only if $U_j = 0$ and $x_{jk} = 1$. If movement k is a successor of movement j, then its plan time has to be larger than the plan time of movement j plus the required headway time between movements j and k. Constraints (4.6) ensure that movement m, which uses track t and is dependent of movements j and k, is operated before movement j or after movement k.

4.2.3 Model with variable routes (MVR)

In order to increase the flexibility of the model (MFR), an extension (MVR) has been developed that also selects the routes for all train movements. In order to facilitate the additional flexibility, a few parameters and decision variables have been added. The added parameters are the following:

- R_j = the set of possible routes of movement j.

- T_{jr} = is the set of tracks used by movement j if route $r \in R_j$ is chosen.
- $a_{jrks} = \begin{cases} 1 & \text{if movements } j \text{ by route } r \text{ and } k \text{ by route } s \text{ are dependent} \\ 0 & \text{if movements } j \text{ by route } r \text{ and } k \text{ by route } s \text{ are independent} \end{cases}$

The definition of dependency is exactly the same as in model (MFR). The model also has to determine the route of each movement. As a consequence, a new decision variable has been introduced, namely:

$$z_{jr} = \begin{cases} 1 & \text{if movement } j \text{ has to be routed along route } r \\ 0 & \text{otherwise} \end{cases}$$

The model minimizes the weighted number of not planned movements that can not be planned within their time window. The parameter w_n represents the penalty if a not planned movement can not be planned.

A second term has been added to the objective function. This term has to prevent that a lot of alternative routes are chosen, especially for the already planned movements. Alternative routes are less comfortable for the passengers, since they usually use more switches. The parameter w_{jr} represents the penalty if alternative route $r \in R_j$ is chosen for movement j. These penalties have to be much smaller than the penalty w_n, because the most important objective is to find a solution such that all movements can be planned. Now the MIP model (MVR) can be described as follows:

$$\min \quad w_n \textstyle\sum_{j \in S_n} U_j + \sum_{j \in S} \sum_{r \in R_j} w_{jr} z_{jr}$$

subject to:

$$y_j \geq r_j \qquad \forall j \in S_n \qquad (4.10)$$

$$y_j \leq d_j + U_j M \qquad \forall j \in S_n \qquad (4.11)$$

$$y_j + l_{jk} \leq y_k \qquad \forall j,k \in S_n \text{ with } k \in \sigma_j \qquad (4.12)$$

$$\textstyle\sum_{r \in R_j} z_{jr} = 1 \qquad \forall j \in S \qquad (4.13)$$

$$x_{jk} + x_{kj} = 1 \qquad \forall j,k \in S : \exists r \in R_j, \exists s \in R_k \text{ with } a_{jrks} = 1 \qquad (4.14)$$

$$y_j + l_{jk} \leq y_k + M(3 + U_j - x_{jk} - z_{jr} - z_{ks}) \qquad \forall j,k \in S, \forall r \in R_j, \forall s \in R_k \text{ with } a_{jrks} = 1 \qquad (4.15)$$

$$z_{mq} - 1 \leq x_{jm} - x_{km} \leq 1 - z_{mq}, \qquad \forall j,k,m \in S, \text{ with } d_m + l_{mj} > r_j, r_m < d_k + l_{km}, \forall q \in R_m \exists t \in T_{mq} : k \in \sigma_j^t \qquad (4.16)$$

$$x_{jk} \in \{0,1\} \qquad \forall j,k \in S \text{ with } \exists r \in R_j, \exists s \in R_k : a_{jrks} = 1 \qquad (4.17)$$

$$z_{jr} \in \{0,1\} \qquad \forall j \in S, \forall r \in R_j \qquad (4.18)$$

$$y_j \in \mathbb{R}^+ \qquad \forall j \in S \qquad (4.19)$$

$$U_j \in \{0,1\} \qquad \forall j \in S \qquad (4.20)$$

Several constraints of (MVR) are the same as those in (MFR). Therefore, we only describe the differences. For each movement exactly one route has to be chosen. Constraint (4.13) enforces this. If there are routes r and s for movements j and k such that these movements along these routes are dependent, then there has to be sufficient time between the plan times of these movements if routes r and s are selected. Constraints (4.14) and (4.15) take care for this. Obviously, constraint (4.15) has an effect on the plan times of movements j and k only if $U_j = 0$ and $x_{jk} = z_{jr} = z_{ks} = 1$. Note again that the value of the headway time l_{jk} is independent of the chosen routes r and s. If route q of movement m uses track t and this track is occupied by successive movements j and k during part of the time window of movement m, then m is not allowed to use track t as long as this track is occupied. If route q is chosen for movement m, then m has to be carried out before movement j or after movement k. Constraints (4.16) handle this.

4.3 Application to railway stations in the Netherlands

To see whether the models can be solved quickly and effectively, they have been tested for three railway stations in the Netherlands. These stations are Groningen, Utrecht and Zwolle. A description of the stations is given in Section 4.3.1. Thereafter the computational results of MFR are presented in Section 4.3.2 and of MVR in Section 4.3.3.

4.3.1 Introduction of railway stations

Railway station Groningen is a relatively small station in the northern part of the Netherlands with one shunting area (see Figure 4.1). Because the shunting area is located parallel to the platform tracks, many saw movements are required. There are even a few saw movements that have to be split into three parts, because the shunting area can only be reached from one side. A typical 24-hour weekday for station Groningen has about 650 train movements where the saw movements have been split already into separate movements. Approximately 200 of these 650 movements are shunting movements.

A second railway station that is used to test the model is station Utrecht. This is the largest railway station in the Netherlands. Trains from this station depart to all directions of the Netherlands. Utrecht has three shunting areas, a large one at the southern part of the station (OZ), a small (Landstraat) and large (Cartesius weg) one at the northern part of the station. The shunting area Landstraat can only be reached from a few platform tracks and can only be entered from one direction. The shunting area OZ is a large shunting area which is always entered or left via the same track. To get to this shunting area, a shunting movement needs to use the same infrastructure as a lot of trains that arrive from or depart to the southern and eastern part of the Netherlands. Shunting area Cartesius weg is a relatively new shunting area that can be entered by only a few tracks. A huge number of trains going to and coming from

the western and northern part of the Netherlands are using the same infrastructure as a shunting movement to or from this shunting area.

Not many trains start or end at railway station Utrecht, so there is a relatively small number of shunting movements. A typical weekday for station Utrecht has approximately 2100 movements including almost 200 shunting movements. Saw movements hardly ever occur in Utrecht, because there are shunting areas on both sides of the station.

The third railway station that is used for testing the model is station Zwolle, which lies in the north-eastern part of the Netherlands. This station is chosen because it is known as one of the hardest railway stations of the Netherlands with respect to shunting. This is caused by the fact that it has several small shunting areas and many shunting movements are related to the internal and external cleaning of rolling stock. A typical weekday at station Zwolle has approximately 900 movements including 175 shunting movements.

4.3.2 Computational results of MFR

Data sets from real shunt plans have been derived to test whether the MIP problems can be solved in an acceptable computation time. The plan times of the already planned passenger and cargo trains were kept the same as in the real plans. For the shunting movements a time window was derived. In the first data set of each station, it was assumed that all movements use their priority route and in the second data set their routes were taken as in the real plan.

The standard MIP solver CPLEX 11.2 is used to solve the model introduced in Section 4.2.2. The computations were carried out on an Intel core T8300, 2.4 GHz processor with 2.0 GB internal memory. The model is tested on the stations introduced in the previous section. A planning horizon has been chosen from Tuesday 5:00 am to Wednesday 1:00 am during a normal week. Table 4.1 shows the computation times CPLEX needs to solve the model for the different stations. For all three railway stations the model is solved quickly.

Table 4.1: Computation times of CPLEX for solving MFR

	All preferred routes		Real plan	
	Objective	Time (sec)	Objective	Time (sec)
Groningen	8	2.48	5	5.46
Utrecht	5	0.25	2	0.125
Zwolle	6	1.05	6	3.59

The data sets were derived from a real plan. Assuming that a real plan is conflict-free, it should be possible to plan all shunting movements. But the model found shunting movements that could not be planned for each tested railway station. This can be explained by the fact that planners have the opportunity to violate norms.

At first glance it seems a bit strange that Utrecht has such a small computation time in comparison with Groningen and Zwolle. This can be explained by the fact that Utrecht has fewer shunting movements relative to the number of passenger and cargo trains. A second reason is that there are hardly any saw movements. Especially saw movements are responsible for many dependent movements and in this way require a lot of variables and constraints. A third cause is that the capacity of the infrastructure is so scarce that there is not much to decide for the model.

4.3.3 Computational results of MVR

Model MVR is also solved with CPLEX 11.2 on the same computer. The model has been tested with the same data sets as used for testing model MFR. The values of the weight factors are taken 1000 for w_n and r for w_{jr}. The latter implies that lower numbered routes are preferred, which represents the current practice. Table 4.2 presents the computation times that CPLEX needs to solve the model for the three stations and for varying numbers of allowed alternative routes.

Table 4.2: Computation times of CPLEX for solving MVR

# alternative routes	Groningen		Utrecht		Zwolle	
	Obj	Time (sec)	Obj	Time (sec)	Obj	Time (sec)
0	8000	2.61	5000	0.54	6000	1.84
1	5003	59.52	3068	3.54	5001	14.87
2	5003	97.19	2117	6.48	5001	133.33
3	5003	444.56	2117	8.13	5001	174.63
4	5003	448.20	2121	12.21	5001	2360.37
5	5003	1038.76	2121	13.20	5001	4348.46
6	5003	1039.72	2127	26.15	5001	4844.04
7	5003	1040.15	2210	29.15	5001	5307.83
8	5003	1048.25	2210	35.28	5001	6072.05
9	5003	1046.87	2210	45.35	5001	7081.69

If for all movements only the priority route is allowed, then the number of shunting movements that can not be planned is obviously the same as in the previous section. In this case station Utrecht has five shunting movements that can not be planned. The objective value of 3068 in the case of one possible alternative route, shows that two more shunting movements could be planned and that 68 train movements have been scheduled on their first alternative route. The value of 68 denotes that a lot of conflicts between planned movements have been solved by using the first alternative route. If no alternative routes are allowed and two planned movements conflict, then we detect this conflict before running CPLEX. This can not be seen in the objective value. This shows that the model can also be used to find non-conflicting routes for planned movements.

For station Groningen already three of the eight infeasible shunting movements can be planned if one alternative route is allowed. For all other movements at station Groningen no alternative route is necessary. The computation times for station Groningen increase a lot if alternative routes are possible.

The computation time for solving the model for station Utrecht is small even if nine alternative routes are allowed. Knowing that Utrecht is a much bigger railway station than Groningen, this seems a bit strange. But in Groningen the number of shunting movements in comparison with the planned movements is much larger. Also the large number of saw movements cause the larger computation time for Groningen.

For station Zwolle there are six shunting movements that can not be planned if all movements have to take their priority route. If one alternative route is allowed, then still five shunting movements can not be planned. Even if nine alternative routes are permitted for a movement, then still five movements could not be planned. The computation times of CPLEX for station Zwolle increase significantly if more routes for a movement are allowed.

4.4 Concluding remarks

Two MIP models have been introduced for scheduling and routing shunting movements between shunting areas and platform areas of railway stations. Both models try to plan all the shunting movements that have not been planned yet in between the already planned train movements. In the first model the routes of all movements are given and can not be changed. This model can be solved very quickly by the MIP solver CPLEX 11.2. The second model allows selecting the route of a movement, which results in much more feasible solutions. Nevertheless, for station Utrecht the model can be solved very quickly.

For stations Zwolle and Groningen, the solution process takes too much computation time. Therefore, we want to develop a branch & bound algorithm that is able to prove optimality for the second model much faster than CPLEX. To get a better understanding of the problem, we start to look at the most basic model of the routing problem: a job shop scheduling problem with blocking and no-wait precedence constraints between two operations of the same job. This is the central topic of the next chapter, in which we present a branch & bound algorithm for this problem.

Table 4.2 shows that for railway stations Groningen and Zwolle no conflicts between planned movements have been solved by model MVR and only a very small number of shunting movements could be planned in model MVR while they could not be planned by model MFR. Next to extending model MFR with more allowed routes, other extensions are possible. The current model MFR picks randomly a plan time for a shunting movement out of the set of allowed plan times for this movement. Recently, an extended model of MFR has been developed that determines plan times for shunting movements such that the sum of the remaining number of possible plan times for each shunting movement is maximized. Another possible extension of model MFR is to let the model determine the saw track of a saw movement.

Chapter 5

No-wait and blocking job shops

5.1 Introduction

In earlier chapters the routing problem has been introduced as part of the shunt planning process at larger railway stations. Two mixed integer programming models for solving this routing problem have been presented. The computation time of CPLEX to solve the model that determines the route and plan time of a shunting movement was too large for practical use for stations Groningen and Zwolle. Therefore, we focus first on a simplified model of the routing problem, namely the job shop scheduling problem with blocking or no-wait precedence constraints. In the three-field notation of Graham et al. (1979), these problems are written as $J|no-wait|C_{max}$ and $J|blocking|C_{max}$.

The traditional job shop scheduling problem is to create a schedule for a set of jobs on a set of machines, subject to the constraint that each machine can handle at most one job at a time and preemption is not allowed. Each job consists of a specified sequence of processing steps. The processing of a job on a machine is called an operation. Each operation occupies one dedicated machine for a given duration. The objective is to schedule the jobs such that the maximum of their completion times is minimized.

The traditional job shop is a notoriously intractable NP-hard combinatorial optimization problem. It is not only NP-hard, Garey and Johnson (1979), but it belongs to the worst in practice. In spite of a great deal of research, there are instances of modest size for which it is beyond our current understanding to solve them to optimality. A specific instance with 10 machines and 10 jobs proposed by Muth and Thompson (1963) remained unsolved for over 20 years. This instance was finally solved by Carlier and Pinson (1989).

Several review papers on the traditional job shop have appeared. Jain and Meeran (1999) present a detailed review and computational comparison between the main solution techniques that have been applied. They clearly highlight the strengths and weaknesses of the algorithms. A second example of an overview paper is Blazewicz et al. (1996).

Most applied algorithms for the traditional job shop have been branch & bound algorithms based on the disjunctive graph model that has been introduced by Roy and

Sussman (1964). The decisions in this model indicate the sequence in which operations are processed on each machine. Once these sequences have been determined, the start time of an operation is the longest path to the node of this operation. Branch & bound algorithms of this type have been proposed by Carlier and Pinson (1989), Applegate and Cook (1991) and Brucker et al. (1994). Martin and Shmoys (1996) propose several lower bounding procedures and show how to incorporate them into a branch & bound algorithm. Unlike all other research done, their branch & bound algorithms are not based on the disjunctive graph formulation, but have a time oriented branching scheme. Their approach obtained the best known lower bounds on a lot of instances. We present and apply their lower bounding procedures later in this chapter.

Also many heuristics have been applied to the traditional job shop. A simulated annealing approach has been presented by Van Laarhoven et al. (1992). Taillard (1994) and Nowicki and Smutnicki (1996) apply tabu search and a genetic algorithm has been presented by Yamada and Nakano (1992). As far as we know, no heuristics have been developed with a performance guarantee.

The traditional job shop problem assumes that there are buffers with infinite capacity available. If an operation has finished processing on one machine and the next machine is not available it can be put in a buffer. However, in many production processes those intermediate buffers are not available. Also in the routing problem of Chapter 4 those buffers are not available. An example: if an arriving train has to be shunted to a shunt yard, then this train can be modeled as a job with three operations. The first operation is the arrival of the train, the second operation is the occupation of a platform track and the third operation is the shunting movement. The train always occupies infra-elements and therefore the problem can not be modeled with a traditional job shop.

The class of machine scheduling problems without intermediate buffers can be characterized by *no-wait* and/or *blocking* precedence constraints. In a no-wait environment an operation has to start immediately after his predecessor operation has been completed. So in a no-wait job shop a job must be processed from start to completion without any interruption either on or between machines. Blocking occurs when a job, having completed processing on a machine, has to stay on the machine until the successor machine becomes available and the job can be transferred. This implies that the actual time an operation keeps its machine occupied is not known in advance. For illustration we take again the example of the arriving train that has to be shunted to a shunt yard; there is a no-wait precedence constraint between the arriving train and the operation that models the platform occupation and a blocking precedence constraint between the platform occupation and the shunting movement.

Scheduling problems with no-wait and blocking precedence constraints have a lot more applications. Modern manufacturing environments can frequently be modeled as a no-wait or blocking scheduling problem. Nowadays, companies tend to design production lines without buffers in order to limit the accumulation of inventory, which can be expensive. As a consequence of the lack of buffer space, a machine cannot release a completed operation. The machine must hold it until the next machine on the routing becomes available. Another application are robotic cells. In robotic cells, jobs must be transported from machine to machine and these transports are

done by robots. It is necessary to assign the transport operations to robots and to schedule the machine and robot operations. A last example comes from the railways. D'Ariano et al. (2007) present a branch & bound algorithm that they apply to a train scheduling problem faced by railway infrastructure managers during real-time traffic control. During disruptions or delays the infrastructure manager has to prevent that delays are propagated to other trains in the network and has to ensure the feasibility of the new plan. The authors model train trips as jobs that have to be scheduled on track segments between two block signals that can handle at most one train at a time. If trains having reached the end of the track segment cannot enter the next segment because it is occupied by another train, then a blocking occurs.

In traditional job shops the presence of humans as an additional problem solver is implicitly assumed. Scheduling automated, unmanned manufacturing systems is therefore significantly different. Blocking constraints in combination with the assumption that human intervention is not possible can lead to a *deadlock* if the machines are not adequately planned. Deadlock is a situation in which operations in a set remain indefinitely blocked, because each of them requests access to a machine held by some other operation in the same set. A traditional approach to this problem is to solve the scheduling problem without considering storage space or transportation machines. While realizing the schedule some real-time control policy has to handle storage space and the transportation machine and has to avoid a deadlock situation. This separate consideration of scheduling and deadlock avoidance may lead to unsatisfactory performance. Mati et al. (2001) illustrate the superiority of an integrated approach through a job shop like example.

The goal of the job shop scheduling problem with no-wait or blocking precedence constraints is to assign start times to operations such that no deadlocks are created while the makespan is minimized. Note that the completion time is the start time plus process time, but for the job shop with blocking precedence constraints this is not necessarily equal to the time that the machine is released.

In view of all the applications, it is very surprising that not much research has been done on the no-wait job shop problem. The problem usually appears when the process technology enforces that operations within a job are processed one after another without waiting and usually has nothing to do with lack of storage capacity. There are production processes where for example the temperature requires that one operation immediately follows its predecessor. A common example for this problem is the production line in a steel company, where molden steel undergoes a series of operations. Another example of a production process where no-wait constraints occur is the food processing industry, where the canning operation immediately has to follow the cooking to ensure freshness.

The no-wait job shop is known to be NP-hard in the strong sense, even when it is restricted to two machines, Sahni and Cho (1979). Timkovsky (1985) proves that the no-wait job shop with two machines in which all operations have unit process time, is strongly NP-hard. Woeginger (2004) proved that the no-wait job shop problem does not possess a polynomial time approximation scheme, unless $P = NP$, (i) for the case of two machines with at most five operations per job and (ii) for the case of three machines with at most three operations per job. Bansal et al. (2005) show

that if each job has at most two operations, the problem has a PTAS if the number of machines is not part of the input.

Papadimitriou and Kanellakis (1980) prove that $F_3|blocking|C_{max}$ is NP-hard in the strong sense. The job shop problem is a generalization of the flow shop problem, hence $J_3|blocking|C_{max}$ is also NP-hard in the strong sense. An extensive survey on complexity results and applications on machine scheduling problems with blocking and/or no-wait precedence constraints is given in Hall and Sriskandarajah (1996)

In the next section we give a more detailed problem description of the job shop scheduling problem with blocking and/or no-wait precedence constraints and we present a MIP formulation. Mascis and Pacciarelli (2002) formulate the job shop scheduling problem with blocking and/or no-wait constraints as an alternative graph, which is a generalization of the disjunctive graph of Roy and Sussman (1964). This alternative graph is explained in Section 5.3. In Section 5.4 we present a job insertion heuristic. The lower bounds that we have applied in the branch & bound algorithms are introduced in Section 5.5. In Section 5.6 new branch & bound algorithms are presented and the computation times of those branch & bound algorithms are compared with the computation times of solving a MIP formulation with CPLEX and with the computation times of Mascis and Pacciarelli (2002). Because the no-wait job shop problem for small instances can be solved in a very small computation time, we provide a more in depth discussion on this problem in Section 5.7 and provide computation times for larger instances. Some final remarks and conclusions are given in Section 5.8.

5.2 Detailed problem description

We model a job shop problem with a set $\{O_1, \ldots, O_n\}$ of operations with only precedence constraints between operations that belong to the same job. The number of operations of a job does not have to be the same for each job. Each operation j has a process time p_j and requires one dedicated machine from the set $M = \{m_1, \ldots, m_m\}$ of machines on which it has to be processed without preemption. A machine can only process one operation at a time. Dummy operations O_0 and O_{n+1} are added for the start and the finish, this gives a set $O = \{O_0, \ldots, O_{n+1}\}$ of operations. The dummy operations O_0 and O_{n+1} have process time zero and do not require a machine.

Each job consists of a chain of operations with a precedence constraint between two successive operations in the chain. The successor of operation i is modeled with σ_i. This means that the start time of operation σ_i must be greater or equal to the completion time of operation i. Note that operation O_0 precedes all other operations and operation O_{n+1} is a successor of all other operations.

Any feasible solution is called a *schedule* and is an assignment of start times to the operations such that:

- Each operation is processed without interruption.
- No operation starts before its predecessor has been completed.
- Each machine processes at most one operation at a time.

The objective is to minimize the start time of operation O_{n+1}.

A deadlock situation occurs if and only if an operation O_1 holds machine m_1, while waiting for machine m_2 for processing. Operation O_2 holds m_2 while waiting for machine m_3, and so on, to operation k holding m_k while waiting for m_1. Whenever there is a deadlock of two or more operations, human intervention or a *swap* is needed. See Figure 5.1 for an example of a swap. Like Mascis and Pacciarelli (2002), we consider the blocking job shop problems with swapping allowed and with swapping not allowed. For the job shop problem where swapping is not allowed, deadlock prevention is a main issue.

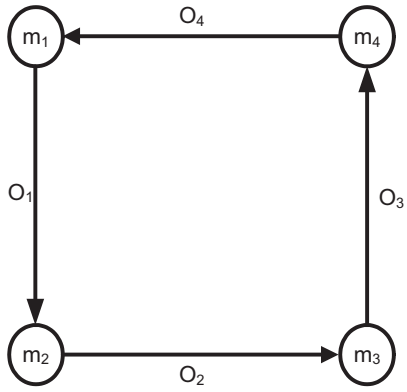

Figure 5.1: An example of a swap of four operations.

Mascis and Pacciarelli (2002) distinguish four types of job shop scheduling problems. First, the ideal (I) type that has infinite capacity buffers available. Hence, all operations immediately leave a machine after completion. This is exactly the well-studied traditional job shop problem. The second type is the no-wait job shop problem (NW) in which all precedence constraints between two operations within a job are of the no-wait type. In the third and fourth type, all precedence constraints between two operations of a job are of the blocking type. The distinction is made between blocking with swapping (BWS) allowed and the type where swapping is not allowed (BNS).

We define the set P of ordered pairs of operations with a precedence constraint in between. To distinguish the type of precedence constraint we also introduce $P^{nw} \subseteq P$ as the set of ordered pairs of operations with a no-wait precedence constraint and $P^b \subseteq P$ as the set of ordered pairs of operations with a blocking precedence constraint. If two operations have to be processed on the same machine, then they are defined as *machine-dependent*. The set D is defined as the set of pairs of machine-dependent operations.

All four job shop problem classes can be modeled with a MIP formulation. In the MIP formulation two different types of decision variables are used. Every operation j has its own start time s_j and $\forall\{i,j\} \in D$:

$$x_{ij} := \begin{cases} 1 & \text{if } i \text{ is scheduled before } j \text{ on the joint machine} \\ 0 & \text{otherwise} \end{cases}$$

Most of the constraints are the same for all four types of job shop problems. That's why we present all four types in one MIP formulation:

$$\min \quad s_{n+1}$$

subject to:

$$\begin{aligned}
s_{\sigma_i} &= s_i + p_i && \forall (i,\sigma_i) \in P^{nw} && (5.1)\\
s_{\sigma_i} &\geq s_i + p_i && \forall (i,\sigma_i) \in P && (5.2)\\
s_j &\geq s_{\sigma_i} - M(1-x_{ij}) + \epsilon && \forall (i,\sigma_i) \in P^b, \forall \{i,j\} \in D && (5.3)\\
s_j &\geq s_i + p_i - M(1-x_{ij}) && \forall \{i,j\} \in D && (5.4)\\
x_{ij} + x_{ji} &= 1 && \forall \{i,j\} \in D && (5.5)\\
x_{ij} &\in \{0,1\} && \forall \{i,j\} \in D && (5.6)\\
s_j &\in \mathbb{R}^+ && \forall j \in O && (5.7)
\end{aligned}$$

The goal is to minimize the start time of the dummy operation O_{n+1}. This is equivalent to minimizing the makespan. If there is a no-wait precedence constraint between operations i and σ_i, then operation σ_i has to start immediately after i is completed. Constraint (5.1) enforces this. The ordinary precedence constraints of the ideal job shop are given by constraints (5.2). Note that those are also necessary for blocking precedence constraints. Constraint (5.3) prevents that deadlocks occur, by taking care that an operation of another job does not start on the machine before the machine is released. Constraints (5.4) and (5.5) are the ordinary disjunctive constraints of the ideal job shop. We give the big-M a value equal to the sum of all process times ($M = \sum_{j \in O} p_j$).

Taking $\epsilon = 0$ gives a MIP formulation of the blocking job shop problem where swapping is allowed. If ϵ is a small positive number, then a solution of the MIP does not contain a swap. Ramaswamy and Joshi (1996) prove that adding constraint (5.3) with $\epsilon > 0$ provides an optimal deadlock-free schedule, so without a swap. They also prove that the sequence obtained is optimal, but the start times may have a small computational error of ϵ. This gives a maximum error of 2ϵ between the start times of every pair of operations. Therefore, taking $\epsilon < \frac{1}{2n}$ will always give the optimal sequence of the operations. We took $\epsilon = \frac{1}{2n+1}$.

5.3 The alternative graph

Traditional job shop problem instances are usually represented by the *disjunctive graph* introduced by Roy and Sussman (1964). A disjunctive graph $G = (N, P, A)$ consists of a node set N, conjunctive arc set P and disjunctive arc set A. Figure 5.2 illustrates the disjunctive graph of an ideal job shop with three jobs and three machines. Each job consists of three operations that are processed on a different machine. The nodes correspond to operations. A solid directed arc is a conjunctive arc and represents a precedence relation within a job. A directed arc from node i to node σ_i has a length p_i. Dashed arcs are drawn between two operations that have to

be operated on the same machine. Each dashed arc in the figure represents a pair of arcs with contradicting direction of which one has to be chosen. The chosen arc represents the order of the two operations on the common machine. These dashed arcs get a length equal to the process time of the operation on the tail of the arc. In Figure 5.2 the length of the arcs are not presented.

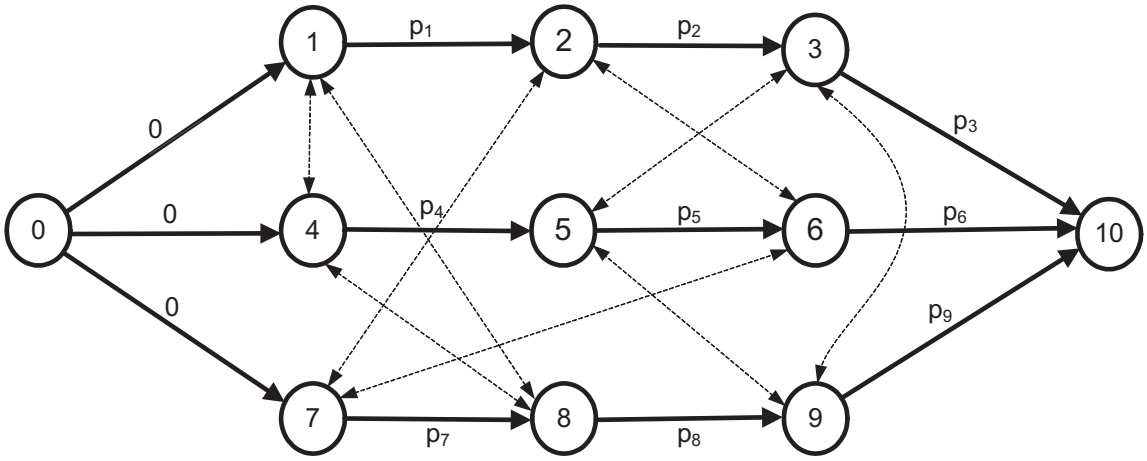

Figure 5.2: A disjunctive graph with three jobs and three machines.

From the node of operation O_0 there is a directed arc with length zero to all first operations of a job. Each node of a last operation of a job has a directed arc to the node of operation O_{n+1} with length equal to its process time. The ideal job shop problem is equivalent to the problem of directing all undirected edges in the disjunctive graph in such a way that the length of the longest path from node O_0 to node O_{n+1} is minimized.

Mascis and Pacciarelli (2002) model the no-wait and blocking job shop problems by means of a generalization of the disjunctive graph that they call *an alternative graph*. This graph is also represented with $G = (N, P, A)$, with a set N of nodes, a set P of fixed precedence constraints and a set A of pairs of directed arcs. Such a pair of directed arcs is called an *alternative* pair of arcs. Each pair of alternative arcs represents a constraint that a machine can only process one operation at a time. Every alternative arc (i, j) has a certain length l_{ij}.

In the disjunctive graph all pairs of disjunctive arcs form the set of alternative arcs in the alternative graph. All the pairs are of the form $((i, j), (j, i))$ with i and j operations that have to be processed on the same machine. So for the ideal job shop problem the alternative graph is exactly the disjunctive graph.

Let us assume there is a no-wait precedence constraint between the operations i and σ_i. In the alternative graph this is modeled with a pair of fixed arcs (i, σ_i) and (σ_i, i) having length $l_{i\sigma_i} = p_i$ and $l_{\sigma_i i} = -p_i$. In Fig 5.3 the alternative graph for a no-wait job shop with three jobs and two machines is shown. The fixed arcs have been drawn with solid or dotted arcs and the pairs of alternative arcs are dashed. The constraint that only one operation can be processed on one machine at the same time is represented by those dashed arcs, exactly like in the disjunctive graph.

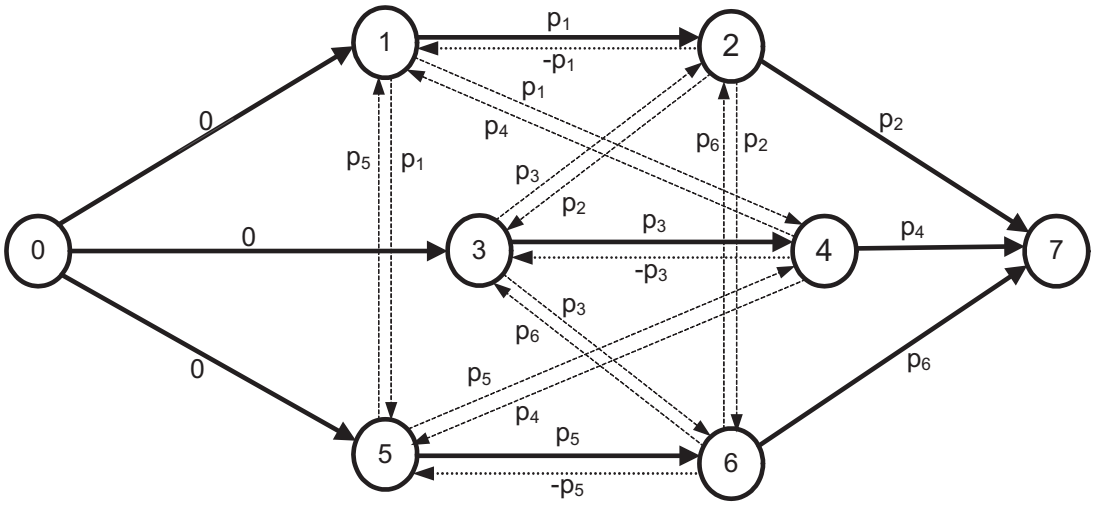

Figure 5.3: The alternative graph of a no-wait job shop with three jobs and two machines

For the no-wait job shop a preprocessing step is possible that significantly increases the length of the alternative arcs. Take operations i and j that have to be processed on the same machine and denote their jobs with J_i and J_j. Initially the length of the arc (i, j) is $l_{ij} = p_i$. Assume that operation i starts at time zero and operation j starts immediately when operation i completes, hence at time $l_{ij} = p_i$. Due to the no-wait constraints also the start times of all other operations of jobs J_i and J_j have been determined. If operations k and l belonging to job J_i, respectively J_j, have to be operated on the same machine at the same time, then it can be concluded that the start time between operations i and j has to differ more than the current value of l_{ij} and the value of l_{ij} can be raised by one. This step can be repeated until a value for l_{ij} is found where no operations of jobs J_i and J_j conflict. So $\forall\{i, j\} \in D$ the values of l_{ij} are initialized to the minimum time that has to be in between the start times of operations i and j if operation i is processed before operation j on their common machine.

In the blocking job shop we distinguish blocking operations and non-blocking operations. Non-blocking operations are the last operations of a job. They leave their machine immediately after completion. For two operations i and j that have to be operated on the same machine this leads to three possible cases. The first case arises if operations i and j are both non-blocking. Then the alternative pair of arcs is a pair of disjunctive arcs known from the ideal job shop.

In the second case operations i and j are both blocking. If operation i is processed before operation j, then operation σ_i must start before operation j can start. Otherwise the common machine has not been released. This gives two mutually exclusive inequalities: $s_{\sigma_i} \leq s_j$ or $s_{\sigma_j} \leq s_i$. These relations can be modeled by the pair of alternative arcs (σ_i, j) and (σ_j, i), both with length zero. In Figure 5.4 this pair of alternative arcs is illustrated in the left picture. The pair of alternative arcs is depicted with the dotted lines.

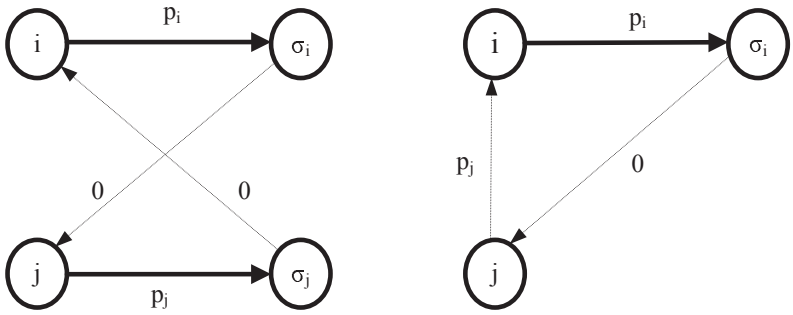

Figure 5.4: The alternative arcs for operations i and j. Both operations require the same machine.

The third case appears if one operation is blocking and one is non-blocking. Assume operation i is blocking and operation j is non-blocking. If operation i precedes operation j, then operation j can start processing as soon as operation σ_i has started. If operation j precedes operation i, then operation i can start immediately when operation j is completed. This gives the alternative inequalities: $s_{\sigma_i} \leq s_j$ or $s_j + p_j \leq s_i$. Those relations are modeled with the pair of alternative arcs (j, i) with length p_j and (σ_i, j) with length zero as shown with dotted lines in the right picture of Figure 5.4.

From each pair of alternative arcs one arc has to be chosen. The choice of this arc represents the order of the corresponding two operations on the joint machine. A *selection* S in the alternative graph is defined as a set of arcs containing at most one arc from each pair of alternative arcs. For each pair $((i, j), (k, l))$ of alternative arcs there are three possibilities:

1. If $(i, j) \in S$, then (i, j) is *selected* in S.

2. If $(k, l) \in S$, then (i, j) is *forbidden* in S.

3. If $(i, j) \notin S$ and $(k, l) \notin S$, then the pair is *unselected*.

A selection is called *complete* if of each pair of alternative arcs exactly one arc is chosen. If the graph $G(S) = (N, P \cup S)$ has no strictly positive length cycle then a selection S is called *consistent*. Each feasible schedule can be associated with a complete consistent selection on the corresponding alternative graph and the makespan of the schedule is the length of a longest path from node O_0 to node O_{n+1}. The length of the longest path from node i to node j is denoted by $lp(i, j)$. Hence the makespan of a complete selection is the length of a critical path and corresponds with $lp(0, n + 1)$. If there is no path between the operations i and j, then we set $lp(i, j) = -\infty$.

A distinction is made between blocking precedence relations where swapping is allowed and where swapping is not allowed. If we have a selection where a swap is necessary, then a cycle of alternative arcs with length zero is created and this is consistent. Therefore, the alternative arcs corresponding to no-swap blocking precedence relations get a small positive length ϵ to take care that a cycle of blocking operations becomes inconsistent.

5.4 Job insertion heuristic

One of the best heuristics for the ideal job shop problem is the shifting bottleneck procedure of Adams et al. (1988). For each machine a one-machine scheduling problem is solved to optimality. This one-machine scheduling problem is a relaxation of the job shop problem by relaxing the constraint that only one operation can be operated at a time for all other machines. The machine with the highest minimum maximum lateness for its one-machine problem is sequenced. Every time after a machine is sequenced, the one-machine problems of all previously sequenced machines are solved again keeping the sequences on the other previously sequenced machines fixed. So the heuristic sequences the machines one by one, taking each time the machine identified as a bottleneck among the machines not yet sequenced. The shifting bottleneck procedure applied to blocking or no-wait job shop problems results very easily in infeasible selections and is therefore not a useful heuristic for those types of job shops.

Many of the proposed heuristics for the ideal job shop are greedy algorithms based on list schedules with priority dispatching rules, see for example Brucker et al. (1994). Mascis and Pacciarelli (2002) observe that a dispatching rule with high probability fails in finding a feasible solution for the no-wait or blocking job shop. They proved that deciding whether it is possible to extend a partial feasible schedule to a feasible solution is NP-complete. This also means that finding a feasible extension of a selection in the alternative graph or proving that it does not exist is an NP-complete problem. As a consequence, any heuristic that incrementally constructs a solution does not necessarily end up with a feasible solution.

Mascis and Pacciarelli (2002) present four fast greedy algorithms and apply them to all four types of job shop problems. These greedy algorithms repeatedly enlarge a consistent selection in the alternative graph and conclude either with a complete consistent selection or with a non-consistent selection (no feasible solution). All four heuristics select one arc at a time. If selecting an alternative arc leads to a positive length cycle in the alternative graph, its alternative is selected. If both arcs in an unselected pair cause a positive length cycle, then the procedure fails in finding a feasible solution. The four heuristics differ only in the selection criteria of the alternative arc that is selected.

Gröflin and Klinkert (2009) have developed a new neighborhood for the blocking job shop. This neighborhood always provides feasible neighbors. Unlike the traditional job shop, generating feasible neighbor solutions is not trivial for the blocking job shop. Neighborhoods based on a simple swap of adjacent operations typically do not yield a feasible solution. Based on the new neighborhood they developed a tabu search algorithm for the blocking job shop that allows swapping. Brizuela et al. (2001) propose a genetic algorithm for the no-wait job shop and the blocking job shop with swapping allowed.

Meloni et al. (2004) present a rollout metaheuristic (see Bertsekas et al. (1997)), which is applicable to all four types of job shops. The heuristic works like a constructive procedure that iteratively extends a partial selection in the alternative graph to a complete selection. Each step they evaluate each candidate arc according to some scoring function and add the arc that has the best score to the partial selection. This

scoring function is basically a look-ahead strategy, guided by sub-heuristics. Also this rollout metaheuristic does not always end up with a feasible solution for the blocking or no-wait job shops.

We have developed a job insertion heuristic that always finds a feasible solution. The job insertion heuristic is described in the next subsection. In the last subsection we compare for benchmark instances the solution generated by the job insertion heuristic with the best solution found by Mascis and Pacciarelli (2002), Meloni et al. (2004) or Gröflin and Klinkert (2009).

5.4.1 Description of the job insertion heuristic

Due to the no-wait and blocking precedence constraints a greedy heuristic usually does not end up with a feasible solution. We have developed a heuristic that inserts the jobs one by one and that always generates a feasible solution. In the worst case, a job that has to be inserted starts after completion of the jobs that were already in the system. The job insertion heuristic adds every job consecutively to the system only preserving the already fixed sequences of operations on the machines. The job insertion problem is defined as follows: Given a feasible schedule and a new job that has not been scheduled yet. The problem is to find a feasible insertion of the new job into the schedule such that the makespan is minimized.

In this chapter the symbol j is used as an index representing an operation. To keep the notation clear, we use the symbol c to denote a job, since a job is a chain of operations. We define the set C of jobs and introduce the set O_c of operations of job c. The set F is the set of operations that have been inserted. For each operation in F, the order with respect to other operations in F that require the same machine has been fixed. The problem $JIP(c, F)$ is the job insertion problem in which job c is the job that has to be inserted and the set F represents the current selection. In other words: problem $JIP(c, F)$ sequences all pairs of machine-dependent operations one of which is in job c and one of which is in F such that the makespan is minimized. After solving $JIP(c, F)$, the sequence of all pairs of machine-dependent operations that have been inserted is fixed, so not the completion times are fixed. Problem $JIP(c, F)$ is formulated as a MIP, like it has been introduced in Section 5.2 and is solved to optimality by CPLEX. Because there are almost no sequences of machine-dependent operations to determine, there are also hardly any x-variables in the MIP model. This results in a small computation time of CPLEX.

In our description of the heuristic we also need the set C^* of jobs that have been inserted. All necessary definitions have been given, hence we are now able to present the job insertion heuristic:

1. Initialize $F := \emptyset, C^* := \emptyset$.
2. Identify $c_b = \max_{c \in C \setminus C^*} \sum_{j \in O_c} p_j$.
3. Solve $JIP(c_b, F)$.
4. $F := F \cup O_{c_b}$ and $C^* := C^* \cup c_b$.

5. If $C^* \neq C$, then go to step 2.

6. Reoptimize

Step 2 selects the job that has to be inserted. The job with the largest sum of process times is selected. Entering step 6 of the heuristic means that a feasible solution has been found. In the reoptimization step we take one job c out of the system and solve the job insertion problem $JIP(c, O \setminus O_c)$. The sequence of the machine-dependent pairs of operations with one operation in c is substituted by the corresponding sequence in the solution of $JIP(c, O \setminus O_c)$. A reoptimization cycle starts by taking out the job with largest sum of process times and proceeds until no improvement is found for each job.

5.4.2 Computational results of the job insertion heuristic

To be able to compare the solutions generated by the job insertion heuristic with the solutions of other approaches, the same data set has been taken as Mascis and Pacciarelli (2002) and Gröflin and Klinkert (2009). The computations have been done on instances derived from the well known benchmark data sets for the ideal job shop. The instances Abz5-9 were introduced by Adams et al. (1988), instances Orb1-10 by Applegate and Cook (1991), instances La1-40 by Lawrence (1984) and instances Ft6, Ft10 and Ft20 are the instances proposed by Muth and Thompson (1963). The job insertion problems $JIP(c, F)$ have been solved with the standard IP solver CPLEX 11.2. The computations have been done on an Intel core T8300, 2.4 GHz processor with 2.0 GB internal memory.

In Table 5.1 the makespan of the generated schedules of a subset of the introduced instances are presented for all four types of job shop problems. The makespan is given for the cases with and without the reoptimization step in columns 'wr' and 'no'. The column 'ct(s)' presents the computation time in seconds for the job insertion heuristic with the reoptimization step. Note that all instances in the table have ten jobs and ten machines. The table shows that the reoptimization step improves the makespan significantly for most of the instances. We conclude that the reoptimization step is a useful step in the job insertion heuristic, especially for the blocking job shop.

Table 5.2 compares the solutions found by the job insertion heuristic for the no-wait and blocking job shop with heuristic solutions known from literature. For the ideal type, blocking without swapping and the no-wait type, the best results have been found by one of the four greedy heuristics introduced by Mascis and Pacciarelli (2002). These results are presented in the columns 'MP'. For some instances with 10 jobs and 10 machines the best solution has been found by Meloni et al. (2004). In Table 5.2 this is denoted with a $*$ left above the value of the makespan. Meloni et al. (2004) tested their rollout metaheuristic only for instances with 10 jobs and 10 machines. If a cell contains a bar this indicates that none of the heuristics found a feasible solution. The columns 'JIH' contain the makespan of the solutions generated by the job insertion heuristic and the columns 'OPT' contain the optimal makespan value of an instance if this value is known. Note that some of the optimal solution

Table 5.1: Computational results of the job insertion heuristic with and without reoptimization.

	I			BWS			BNS			NW		
Inst	no	wr	ct(s)	no	wr	ct(s)	no	wr	ct(s)	no	wr	ct(s)
Abz5	1350	1258	4.34	1782	1782	2.98	1995	1818	5.90	2656	2656	1.00
Abz6	969	965	1.61	1432	1290	4.42	1719	1590	4.59	1838	1838	0.81
Ft10	1005	988	2.23	1283	1247	4.04	1304	1158	7.71	1769	1769	0.97
Orb01	1191	1105	4.01	1355	1322	5.96	1458	1407	4.71	1799	1799	1.05
Orb02	899	892	3.31	1328	1243	5.82	1414	1269	4.63	1731	1615	1.68
Orb03	1059	1050	2.15	1274	1256	3.57	1426	1426	2.79	1686	1637	1.33
Orb04	1080	1043	2.90	1459	1256	5.63	1502	1405	4.31	1983	1941	1.31
Orb05	928	928	1.00	1159	1088	5.55	1449	1449	2.64	1499	1468	1.16
Orb06	1112	1085	3.04	1273	1218	7.44	1549	1449	4.37	1870	1870	0.78
Orb07	416	411	1.69	639	570	3.89	683	633	8.18	823	810	1.23
Orb08	965	929	3.43	1233	1113	7.10	1345	1324	4.56	1319	1319	0.70
Orb09	970	970	1.12	1241	1192	5.45	1234	1221	3.78	1721	1721	0.75
Orb10	1033	1009	1.70	1246	1237	3.74	1637	1567	4.78	1614	1614	0.69
La16	1021	1000	2.08	1259	1201	6.18	1346	1345	4.79	1819	1786	1.45
La17	814	809	1.61	1286	1098	8.38	1333	1061	6.09	1690	1579	1.72
La18	895	885	1.73	1298	1103	7.65	1393	1263	6.30	1606	1578	1.65
La19	893	848	3.53	1285	1163	6.29	1360	1268	9.39	1689	1689	0.72
La20	922	915	1.73	1446	1239	4.84	1449	1238	4.28	1714	1714	0.73

values have been determined with a branch & bound algorithm that is presented in Section 5.6.

For the blocking with swapping job shop the results of Gröflin and Klinkert (2009) are presented. Therefore, 'GK' is the head of the first column for the type BWS. They performed for each instance five independent runs. The column contains the best solution found. Also in this case a $*$ above the value of the makespan means that the solution value has been found by Meloni et al. (2004).

The job insertion heuristic outperforms the best results found by the greedy heuristics of Mascis and Pacciarelli (2002). For the blocking without swapping and the no-wait type the results for the instances with 10 jobs and 10 machines have a comparable quality as the solutions found by Meloni et al. (2004). An advantage of the job insertion heuristic is that it guarantees to find a feasible solution while the rollout metaheuristic does not necessarily find one. For the blocking with swapping job shop the job insertion heuristic also provides solutions of comparable quality of Gröflin and Klinkert (2009) and Meloni et al. (2004).

For the ideal and no-wait job shop the computation times of the job insertion heuristic are small. The computation times for the blocking job shop instances are larger, especially for instances larger than 20 jobs and 10 machines. But even for the hardest instances the computation times are acceptable and mainly caused by the reoptimization step.

It can be conclude that the job insertion heuristic gives good solutions in a small computation time. Main advantage of the job insertion heuristic compared to greedy heuristics and the rollout metaheuristic is that it always finds a feasible solution.

Table 5.2: Computational results of the job insertion heuristic

		I			BWS			BNS			NW		
I	size	MP	JIH	OPT	GK	JIH	OPT	MP	JIH	OPT	MP	JIH	OPT
Abz5	10x10	*1250	1258	1234	*1595	1782	1468	*1838	1818	1641	*2726	2656	2150
Abz6	10x10	*947	965	943	*1222	1290	1145	*1419	1590	1249	*2178	1838	1718
Abz7	20x15	753	717	-	1224	1103	-	2675	1312	-	2538	2096	-
Abz8	20x15	783	710	-	1226	1198	-	-	1223	-	3145	2091	-
Abz9	20x15	777	735	-	1166	1139	-	1976	1204	-	2350	1944	-
Ft06	6x6	55	55	55	63	63	63	74	73	69	73	73	73
Ft10	10x10	*943	988	930	1103	1247	1068	*1447	1158	1158	*1882	1769	1607
Ft20	20x5	1338	1195	1165	1495	1566	-	1796	1628	-	1820	1735	1532
Orb01	10x10	*1081	1105	1059	1268	1322	1175	*1335	1407	1256	1675	1799	1615
Orb02	10x10	*889	892	888	1092	1243	1041	*1370	1269	1144	*1814	1615	1485
Orb03	10x10	*1031	1050	1005	*1188	1256	1160	1407	1426	1311	1647	1637	1599
Orb04	10x10	*1006	1043	1005	1203	1256	1146	*1460	1405	1246	*1836	1941	1653
Orb05	10x10	*890	928	887	1083	1088	995	*1414	1449	1203	*1609	1468	1365
Orb06	10x10	*1026	1085	1010	1265	1218	1199	*1448	1449	1266	1788	1870	1555
Orb07	10x10	*403	411	397	*517	570	483	*565	633	527	* 880	810	689
Orb08	10x10	*919	929	899	1028	1113	995	*1223	1324	1139	*1542	1319	1319
Orb09	10x10	*943	970	934	*1046	1192	1039	*1329	1221	1130	1542	1721	1445
Orb10	10x10	*944	1009	944	*1153	1237	1146	1612	1567	1367	*1773	1614	1557
La01	10x5	666	666	666	832	899	793	1066	1050	881	1106	1084	971
La02	10x5	694	685	655	793	878	793	1077	969	900	1296	1059	937
La03	10x5	735	620	597	747	797	715	884	884	808	955	956	820
La04	10x5	679	598	590	769	814	743	881	965	859	1039	1007	887
La05	10x5	593	593	593	698	745	664	995	891	732	1094	920	777
La06	15x5	926	926	926	1180	1275	1060	1568	1317	1194	1758	1454	1248
La07	15x5	984	890	890	1091	1122	1016	1553	1327	1127	1609	1351	1172
La08	15x5	873	892	863	1125	1342	1040	1386	1335	1173	1580	1336	1244
La09	15x5	986	951	951	1223	1387	1141	1579	1410	1305	2430	1521	1358
La10	15x5	1009	958	958	1203	1284	1096	1629	1371	1218	1506	1451	1287
La11	20x5	1239	1222	1222	1584	1622	-	2017	1780	-	2226	1889	1619
La12	20x5	1039	1039	1039	1391	1405	-	1800	1603	-	1935	1577	1414
La13	20x5	1161	1150	1150	1548	1801	-	2024	1693	-	1867	1748	1580
La14	20x5	1301	1292	1292	1620	1665	-	2228	1894	-	2381	1833	1578
La15	20x5	1369	1238	1207	1650	1601	-	2226	1789	-	2134	1840	1671
La16	10x10	*956	1000	945	*1109	1201	1060	*1231	1345	1148	*1659	1786	1575
La17	10x10	*788	809	784	*982	1098	929	*1146	1061	968	*1665	1579	1371
La18	10x10	*848	885	848	1078	1103	1025	*1334	1263	1077	*1950	1578	1417
La19	10x10	*842	848	842	1093	1163	1043	*1314	1268	1102	*1690	1689	1482
La20	10x10	*907	915	902	*1118	1239	1060	*1357	1238	1118	1971	1714	1526
La21	15x10	1241	1092	1046	1545	1712	-	2475	1737	-	3033	2336	2030
La22	15x10	1032	983	927	1458	1393	-	2430	1589	-	2668	2391	1852
La23	15x10	1131	1032	1032	1611	1637	-	-	1940	-	3296	2428	2021
La24	15x10	999	1010	935	1571	1695	-	2489	1752	-	2473	2653	1972
La25	15x10	1071	1009	977	1499	1560	-	2001	1845	-	2750	2451	1906
La26	20x10	1378	1258	1218	2162	2421	-	3147	2270	-	3687	2890	2467
La27	20x10	1353	1278	1235	2175	2182	-	3505	2497	-	3488	3073	2611
La28	20x10	1322	1260	1216	2071	2201	-	4419	2293	-	4806	3166	2546
La29	20x10	1392	1290	-	2124	1882	-	3298	2273	-	3086	2897	2300
La30	20x10	1476	1355	1355	2171	2140	-	3208	2317	-	3392	2837	2452
La31	30x10	1871	1784	1784	3167	2932	-	-	3157	-	4838	4295	-
La32	30x10	1942	1850	1850	3418	3406	-	5710	3701	-	5469	4666	-
La33	30x10	1897	1719	1719	3131	3023	-	-	3071	-	-	4162	-
La34	30x10	1934	1747	1721	3205	2985	-	-	3185	-	5016	4346	-
La35	30x10	2017	1888	1888	3311	3195	-	5955	3548	-	5360	4540	-
La36	15x15	1347	1317	1268	1932	1957	-	3209	2088	-	3651	3355	2685
La37	15x15	1547	1447	1397	2053	1991	-	3811	2255	-	4326	3542	2831
La38	15x15	1342	1296	1196	1875	1889	-	-	1982	-	3872	3197	2525
La39	15x15	1361	1278	1233	1950	2038	-	2765	1969	-	-	3054	2660
La40	15x15	1340	1283	1222	1936	1971	-	-	2099	-	4422	3277	2564

5.5 Lowerbounds

A lot of research has been done to find strong lower bounds for the ideal job shop problem in a small computation time, but they are still lacking. There are very strong lower bounds, but they take too much computation time to use them in a branch & bound algorithm. In this section we describe some of the lower bounds applied to the ideal job shop and investigate whether they are applicable for the blocking and no-wait job shop.

A trivial lower bound is the *job bound*. This is the maximum over all jobs of the sum of process times of the operations in a single job. Equivalent, the maximum over all machines, of the sum of process times on a single machine provides the *machine bound*.

Applegate and Cook (1991) describe a cutting-plane method for obtaining stronger lower bounds for the ideal job shop. They make use of eight types of valid inequalities given by others and themselves. This polyhedral approach seems to be inappropriate for solving the job shop problem.

By relaxing the constraint that a machine can handle one operation at a time for all machines except one, the problem becomes a one-machine scheduling problem with a head, body and tail for each operation. Each operation j has a head r_j which is equal to the sum of the process times of the operations that precede j in its job. The sum of process times of the operations that succeed j in its job is called the tail q_j. Each operation j also has a body p_j. By setting an upper bound this problem can be transformed in the following one-machine scheduling problem: $1|r_j|L_{max}$. Unfortunately this one-machine scheduling problem is also NP-hard, but it is possible to solve the problem in acceptable computation time. Bratley et al. (1973) introduce the *one-machine bound* which is the maximum over all machines of the makespan for the one-machine subproblems. This bound can be found by performing a binary search on the upper bound that is set.

The one-machine scheduling problem $1|r_j|L_{max}$ can be relaxed by dropping the non-preemption constraint. An $O(n \log n)$ algorithm for solving this preemptive one-machine scheduling problem to optimality is presented by Jackson (1956). Jackson's rule creates a preemptive earliest due date schedule. The maximum of the makespans of the preemptive one-machine problems is a lower bound. This preemptive one-machine bound is not much worse than the bound resulting from the non-preemptive one-machine problems, but less computation time is needed.

Carlier and Pinson (1990) provide an algorithm that strengthens this preemptive one-machine bound and maintain polynomial computability. They present an $O(n^2)$ algorithm based on Jackson's rule to solve $1|r_j, pmtn|L_{max}$ and that updates all heads and tails of the operations of the one-machine problem. In Carlier and Pinson (1994) they present an $O(n \log n)$ implementation of the same algorithm. Such an update of heads and tails is called a *Carlier-Pinson update*. Martin and Shmoys (1996) show that those updates are the same as adjusting each head (or tail) of an operation to the earliest time that the operation can start such that it processes continuously and such that there is a preemptive schedule for the remaining operations on that machine.

Martin and Shmoys (1996) propose several procedures that determine lower bounds

of the ideal job shop. For each procedure they show the resulting bound and the necessary time to determine the bound. Two of the lower bounds that they introduced are the *iterated Carlier-Pinson* and the *Carlier-Pinson shave* lower bounds. We choose to apply those two lower bounds and they are the central topic of the rest of this section. In the first subsection we explain the two lower bounds in more detail and in the second subsection we apply them to the blocking and no-wait job shop and present computational results.

5.5.1 Iterated C-P and C-P shave lower bounds

A job shop feasibility problem is: given a time T, does there exist a feasible schedule that completes before or at time T. Martin and Shmoys (1996) consider the feasibility problem and use a bisection search on the time T to reduce the optimization problem to the feasibility problem. For each operation they define the *processing window* as the interval in which the operation is allowed to start processing. For instance: if operation j has head r_j and tail q_j, then $[r_j, T - q_j - p_j]$ is the processing window of operation j.

In the same paper Martin and Shmoys present several approaches to determine restricted processing windows $[u_j, v_j]$, in which u_j is the earliest possible start time of operation j and v_j is the last possible start time of operation j. In this section we describe two approaches that determine restricted processing windows, namely iterated Carlier-Pinson and Carlier-Pinson shave. The explanation of each procedure is only focused on increasing the start of the processing window, because decreasing the end of the processing window can be done analogously.

We start with considering the one-machine relaxation for machine m to compute tighter processing windows. Each operation k on this machine has a restricted processing window $[u_k, v_k]$. We tighten the processing windows of these operations by adjusting heads r_k and tails q_k as introduced by Carlier and Pinson (1989). Initially operation k has head $r_k = u_k$, body p_k and tail $q_k = T - p_k - v_k$.

For any subset S of the set of operations that have to be operated on machine m we define:

$$p(S) = \sum_{l \in S} p_l, \quad r(S) = \min_{l \in S} r_l, \quad q(S) = \min_{l \in S} q_l$$

Note that the length of a feasible schedule is at least $r(S) + p(S) + q(S)$. Given an operation k that has to be operated on machine m, $k \notin S$, (k,S) is called an *ascendent set* if:

$$r(S) + p(S) + p_k + q(S) > T, \tag{5.8}$$

$$r_k + P(S) + p_k + q(S) > T. \tag{5.9}$$

Inequality (5.8) implies that in a feasible schedule for operations $S \cup \{k\}$, operation k is processed first or last and inequality (5.9) implies that k cannot be processed first. Hence, ascendent set (k, S) implies that if there exists a feasible schedule for the one-machine problem, k must be processed last among $S \cup \{k\}$. Given such an ascendent

set (k, S), we can update $u_k := \max\{u_k, c(S)\}$, with $c(S) = \max\{r(S') + p(S')|S' \subseteq S\}$ a lower bound on the first possible time that all operations in S can complete processing.

Heads and tails of the operations of a one-machine problem have to be updated alternately until no new updates are found. Such an update of heads and tails for a one-machine problem is a Carlier-Pinson update. In Carlier and Pinson (1994) an $O(n \log n)$ algorithm that updates all heads and tails has been presented. Martin and Shmoys (1996) introduced an $O(n^2)$ algorithm for performing Carlier-Pinson updates for the one-machine problem which they show is faster when a sequence of closely related problems is solved. The general idea of their algorithm is to perform all updates by repeatedly applying an $O(n)$ algorithm that performs updates for all ascendent sets (k, S) with a fixed value of $q(S)$. For the exact details we refer to their paper.

Martin and Shmoys apply the algorithm of adjusting heads and tails on one machine in an iterated fashion. If the processing window of a certain operation j is updated, then also the processing window of other operations of the same job can be updated. Let us assume that u_j is updated to u_j^*, then u_{σ_j} can be updated to $u_j^* + p_j$. So updating processing windows on one machine, can lead to further updates on other machines. The Carlier-Pinson update procedure has to be performed in an iterated fashion until no updates are found on any machine or until one of the processing windows is empty. If one of the processing windows is empty, then the feasibility problem has been solved. Hence, no feasible schedule exists of length T or less, which provides a lower bound of value T. If there is no possible update left and no processing window is empty, then the procedure is not able to prove that no feasible solution exists for the considered feasibility problem. Martin and Shmoys applied a binary search on the value of T between known upper and lower bounds to determine the *iterated C-P* lower bound. Applegate and Cook (1991), Brucker et al. (1994) and Carlier and Pinson (1990) also applied this procedure, but this only becomes clear when looking at their computational results.

Another lower bound introduced in Martin and Shmoys (1996) is the *C-P shave* lower bound. For a certain operation j with processing window $[u_j, v_j]$, it is assumed that j starts at the latest at w_j. Hence, v_j is replaced by w_j and all other processing windows are unchanged. Then it is tried to prove that there does not exist a feasible schedule, for example by using the iterated C-P algorithm. If it is proven that no feasible schedule exists, then operation j can not start before w_j+1 and the processing window of operation j can be replaced by $[w_j + 1, v_j]$. In other words: the portion $[u_j, w_j]$ is shaved off the processing window of j. The maximum value of w_j has been determined by first trying to shave off one unit. If this provides an update of u_j, then the shave trial length is doubled. This proceeds until u_j is not updated. Then a binary search is performed to find the maximum value of w_j. If the iterated C-P algorithm is used to prove infeasibility, then Martin and Shmoys call it C-P shaving. Updates of the processing window of one operation can lead to updates of processing windows of other operations, so also C-P shaving is repeated until no further updates are found.

A last lower bound that has been introduced in Martin and Shmoys (1996) is

double shave, in which they use C-P shave to prove that a feasible schedule does not exist. This gives an excellent lower bound for the ideal job shop, but the computation time is huge.

We applied the iterated C-P and C-P shave lower bound for the blocking and no-wait job shop problems. The double shave lower bound has not been used, because of the large computation time. Note that the $O(n^2)$ algorithm of Martin and Shmoys (1996) has been implemented to perform all C-P updates for the one-machine scheduling problems.

The extra restrictions on the precedence constraints for the blocking and no-wait job shops can be used to strengthen the lower bound. For the two blocking types the only difference with the ideal type is the moment at which a machine is released. The time that a machine is occupied by an operation is not necessarily equal to the process time of the operation. Therefore, the process time p_j of operation j can sometimes be replaced by the minimal occupation time p'_j which is initially equal to p_j. The minimal occupation time p'_j of operations j can be updated if $v_j + p'_j < u_{\sigma_{j+1}}$. Then p'_j can be set to $u_{\sigma_{j+1}} - v_j$. Note that the lower bounds for both blocking types are equal to each other.

For the no-wait job shop we can explicitly use that a successor operation immediately has to start when its predecessor completes. Let us assume that there is a no-wait precedence constraint between the operations j and σ_j. If the earliest possible start time u_j of operation j is increased to u^*_j, then $u_{\sigma_j} := u^*_j + p_j$. For the no-wait type updating u_j to u_j* means that the u-values of all operations in the same job can be increased with $u^*_j - u_j$.

5.5.2 Computational results

In contrast with Martin and Shmoys (1996) we did not apply binary search on T between a known upper and lower bound. We start with trying to improve the best known lower bound with one. If the lower bound is improved we try to improve the new lower bound with 2, 4, 8, 16 and so on until no improvement of the lower bound is found. Then we apply binary search between the last two tested values of T. For the ideal job shop the computation times are a bit larger than with binary search, but for the blocking and no-wait job shop the computation times are significantly smaller. As initial lower bound for the computational experiments we always took the job bound.

Table 5.3 shows the iterated C-P and C-P shave lower bounds for 18 benchmark instances with 10 jobs and 10 machines that we will use as test set during the rest of this chapter and that have also been used by Mascis and Pacciarelli (2002). The optimal solution values are presented in the third column of each type.

The computation times for finding the C-P shave lower bounds for these instances were always less than two seconds and mostly less than one second. To determine the iterated C-P lower bounds takes a computation time in the order of magnitude of a few hundreds of seconds.

Surprisingly, for all 18 instances the iterated C-P lower bound of the blocking job shop is exactly the same as the lower bound of the ideal job shop. Also for the no-wait job shop, the iterated C-P lower bound is not much larger than the iterated C-P lower

Table 5.3: Lower bounds found by iterated C-P and C-P shave

	I			BWS/BNS			NW		
I	it C-P	shave	OPT	it C-P	shave	OPT	it C-P	shave	OPT
Abz5	1126	1202	1234	1126	1208	1468	1127	1422	2150
Abz6	889	943	943	889	959	1145	909	1263	1718
Ft10	855	919	930	855	924	1068	863	1079	1607
Orb01	975	1020	1059	975	1027	1175	975	1173	1615
Orb02	812	874	888	812	879	1041	815	1072	1485
Orb03	906	975	1005	906	978	1160	912	1109	1599
Orb04	898	979	1005	898	982	1146	898	1203	1653
Orb05	810	869	887	810	871	995	864	988	1365
Orb06	946	995	1010	946	1005	1199	971	1205	1555
Orb07	363	393	397	363	395	483	372	480	689
Orb08	894	899	899	894	905	995	905	1028	1319
Orb09	901	931	934	901	931	1039	909	1140	1445
Orb10	923	944	944	923	958	1146	943	1153	1557
La16	901	945	945	901	958	1060	901	1177	1575
La17	777	784	784	777	805	929	801	1060	1371
La18	803	848	848	803	866	1025	814	1104	1417
La19	755	830	842	755	837	1043	766	1050	1482
La20	836	902	902	836	915	1060	861	1143	1526

bound for the ideal job shop.

Also the C-P shave lower bounds for the blocking job shops are hardly better than the C-P shave lower bounds for the ideal job shop. The C-P shave lower bound for the no-wait job shop is significantly larger than the lower bounds for the blocking and ideal job shop. But a disadvantage of the C-P shave lower bound is the much larger computation time. If we compare the relative gap between the lower bounds and the optimal solution values, this shows that this relative gap is very large for the no-wait job shop in comparison with the ideal job shop.

5.6 Branch & bound algorithms

Several branch & bound algorithms for the ideal job shop have been developed. Some of them are briefly described in the first subsection. Thereafter, we introduce five branch & bound algorithms that we have implemented. In Subsection 5.6.2 the building blocks that all five branch & bound algorithms have in common are presented. The main question that is answered in Subsection 5.6.3 will be whether spending time on better lower bounds like iterated C-P and C-P shave is profitable in a branch & bound algorithm for a blocking or no-wait job shop. The iterated C-P and C-P shave algorithms can also be used to reduce the length of the processing windows. Both window reduction techniques have been applied in a branch & bound algorithm. Those two branch & bound algorithms and their computational results are presented

in Subsection 5.6.4. In the last subsection the computation times of the best branch & bound algorithm that we have developed are compared with the computation times of the branch & bound algorithm of Mascis and Pacciarelli (2002) and the computation times of CPLEX for solving the MIP introduced in Section 5.2.

5.6.1 Branch & bound algorithms for the ideal job shop

Carlier and Pinson (1989) were the first to solve the instance with 10 machines and 10 jobs proposed by Muth and Thompson (1963). In their branch & bound algorithm Carlier and Pinson solve for each machine the preemptive one-machine subproblem and use the maximum as the lower bound for each node. Only for a few pairs of operations they test for ascendent sets. In a second paper Carlier and Pinson (1990) introduce immediate selections of disjunctive constraints which lead to adjustments of heads and tails. They present a polynomial algorithm for optimally adjusting heads and tails and fixing disjunctive constraints which provides an efficiently pruning of the search tree. In Carlier and Pinson (1994) they propose an $O(n \log n)$ algorithm to determine all possible immediate selections. This provides exactly the same result as one Carlier-Pinson update.

The branch & bound algorithm that Applegate and Cook (1991) present uses the *edge-finding* algorithm. The main difference between edge-finder and the algorithm described by Carlier and Pinson (1989) is that edge-finder tests for ascendent sets for all subsets of jobs on a machine while Carlier and Pinson only test for all pairs of jobs on the machine and one heuristically determined subset. As mentioned before, also Carlier and Pinson improved their first branch & bound algorithm by looking for ascendent sets for all subsets of operations on a machine. Applegate and Cook always pick the subproblem with the minimum preemptive lower bound. Then they fix the order of two operations i and j that gives the biggest direct increase in the preemptive lower bound. So they maximize the minimum of the preemptive lowerbounds of the two created nodes.

Brucker et al. (1994) present a branching scheme based on a block approach. Their algorithm assumes that in every node a feasible solution is found corresponding with a complete selection in the disjunctive graph. They define a *block* as a sequence of successive operations on the critical path of the disjunctive graph if the sequence contains at least two nodes and has a maximal number of operations which have to be processed on the same machine. The successor nodes in the search tree have an operation from the block that has to be scheduled before (after) all other operations in the block or is not the first (last) operation of the block.

Martin and Shmoys (1996) show how to incorporate their proposed lower bounding procedures into a branch & bound algorithm. They present two branch & bound algorithms with a time-oriented lower bound and they use the processing windows in their branching scheme. Their first branch & bound algorithm starts at time zero and assigns an available operation or delays this operation. This means a node is created where this operation starts at time zero and a node where at least another operation uses the machine first. So they explicitly use the fact that an optimal schedule will always be an active schedule. For blocking or no-wait job shops an optimal schedule is

not necessarily an active schedule, therefore we do not apply this branching scheme. The idea behind their second branching scheme is to determine when there is not much idle time on a machine and then branch on the operation that goes first in this time period. They define a tight set S, which is a set of operations on the same machine which cause the machine to operate without interruption from $r(S)$ until $T - q(S)$. Then they start branching on which operation starts on $r(S)$. Even for ideal job shop instances tight sets are hard to find. For blocking or no-wait job shops it is even more difficult to determine tight (or nearly tight) sets, because there is a large gap between the one-machine bound and the optimal makespan. Therefore, we did also not apply this branching scheme.

5.6.2 Common building blocks of our branch & bound algorithms

All the branch & bound algorithms that are presented in the following subsections have been based on the alternative graph. Each node in the search tree represents a selection in the alternative graph. Always two new nodes are created in which one of the alternative arcs of a certain selected pair of alternative arcs is selected. The rest of this section describes properties that all five developed branch & bound algorithms have in common.

Essential in the branch & bound algorithms for the ideal job shop are the immediate selection rules introduced by Carlier and Pinson (1990). Mascis and Pacciarelli (2002) present similar immediate selection rules for the blocking and no-wait job shop and also for those problems they are essential. The immediate selection rules significantly reduce the computation time of a branch & bound algorithm, because the number of necessary branches is reduced a lot. We have also applied those selection rules in each branch & bound algorithm.

Let us consider a selection S and an unselected pair $((i,j),(k,l))$ of alternative arcs. Arc (i,j) can be added to S if we are able to prove that no optimal solution exists in which arc (k,l) is selected. The immediate selection rules that have been applied are:

1. If $lp(l,k) + l_{kl} > 0$, then selecting arc (k,l) creates a positive length cycle in the alternative graph. Arc (i,j) can be selected.

2. If $u_k + \lfloor l_{kl} \rfloor > v_l$, then selecting arc (k,l) gives infeasibility. Arc (i,j) can be selected.

3. If $lp(i,j) \geq l_{ij}$, then arc (i,j) is redundant and can be added to S.

Important for a branch & bound algorithm is that a good initial feasible solution is provided. A good initial solution takes care that many branches of the search tree are cut off as early as possible. We apply the job insertion heuristic to get a good feasible solution for the root node. A heuristic that is used in each node of the search tree should be very fast, therefore we do not use the job insertion heuristic for this. We even do not apply any heuristic in other nodes than the root node. New feasible

solutions are only found when a complete consistent selection in the alternative graph has been reached. The new feasible solution has makespan $lp(O_0, O_{n+1})$.

In all branch & bound algorithms that we will introduce, the search strategy is depth-first. This search strategy is mainly chosen because it usually needs a less amount of memory. A second reason is that it leads quickly to new feasible solutions. Disadvantage of this search strategy is that whenever early choices made in the search tree are unfortunate choices, we can get stuck in this part of the search tree resulting in a large computation time.

To summarize:

The branch & bound algorithms that are introduced in the next subsections have the following in common:

- Depth-first search strategy,
- Initial solution is provided by the job insertion heuristic,
- Immediate selection rules are applied,
- No heuristic is applied, except in the root node.

5.6.3 Comparison of the lower bounds

In this section three branch & bound algorithms that we have developed are compared. They only differ in the used lower bound. The first branch & bound algorithm, which we call A1, does not use any sophisticated lower bound. A tree is pruned if $lp(O_0, O_{n+1})$ is larger or equal than the value of the currently best known upper bound. A second branch & bound scheme, A2, determines the iterated C-P lower bound in each node of the search tree and the third branching scheme, A3, applies the C-P shave lower bound in each node. The search tree is pruned if the lower bound of a node is larger or equal than the value of the currently best known solution value.

To determine the pair of alternative arcs to branch on, we define for each alternative arc (i, j) the minimal length of the critical path after selection of arc (i, j), namely $mlcp(i, j) = lp(O_0, i) + l_{ij} + lp(j, O_{n+1})$. As a branching strategy in all three branching schemes we choose the pair of alternative arcs $\{(i, j), (k, l)\}$ with maximum value of $mlcp(i, j) + mlcp(k, l)$ over all pairs of alternative arcs. After a pair of alternative arcs has been chosen, we create two new nodes and in each node one of the alternative arcs is fixed. Then the immediate selection rules are applied on both nodes. The node with the smallest length of the critical path in the alternative graph is chosen to proceed the algorithm.

Table 5.4 shows the computational results of the branch & bound algorithms A1, A2 and A3 for all four types of job shop problems. For the blocking job shop with swapping allowed we did not do all the computations for algorithm A3. The computation times were too large and don't provide additional information. We can immediately draw the conclusion that applying C-P shave as a lower bound in a branch & bound procedure for the blocking and no-wait job shop is too time consuming. For the ideal job shop the C-P shave lower bound proves its strength. This confirms the

Table 5.4: Computational results for algorithms A1, A2 and A3

	I				BWS				BNS				NW			
I	Obj	A1	A2	A3	Obj	A1	A2	A3	Obj	A1	A2	A3	Obj	A1	A2	A3
Abz5	1234	128	93	77	1468	1198	2713	-	1641	195	471	62927	2150	4	7	1041
Abz6	943	60	3	20	1145	312	838	-	1249	108	264	21607	1718	1	1	121
Ft10	930	4717	4850	396	1068	204	608	-	1158	49	114	22376	1607	2	4	461
Orb01	1059	564502	277410	898	1175	1804	4130	-	1256	181	455	79484	1615	1	2	253
Orb02	888	12	6	36	1041	379	1093	-	1144	64	190	21381	1485	2	2	217
Orb03	1005	136070	148805	4656	1160	224	553	-	1311	8	18	1863	1599	2	4	459
Orb04	1005	358	377	103	1146	161	400	-	1246	95	277	37519	1653	2	3	351
Orb05	887	104	160	124	995	56	140	16505	1203	20	48	5153	1365	3	5	430
Orb06	1010	13513	5131	312	1199	2312	5353	-	1266	254	630	102520	1555	1	1	71
Orb07	397	20	11	28	483	720	1732	-	527	303	776	84221	689	3	5	557
Orb08	899	17321	1637	138	995	130	324	-	1139	22	54	8410	1319	1	2	217
Orb09	934	55	46	45	1039	63	139	17684	1130	17	43	6633	1445	1	2	136
Orb10	944	37	19	84	1146	77	192	27660	1367	12	25	2275	1557	3	4	501
La16	945	20892	365	41	1060	202	582	-	1148	72	218	26570	1575	2	3	210
La17	784	1155	2	30	929	239	575	-	968	32	72	6551	1371	2	3	276
La18	848	92	3	31	1025	203	648	-	1077	100	315	30428	1417	2	3	399
La19	842	6	7	38	1043	923	2191	-	1102	94	220	22880	1482	2	3	439
La20	902	6	8	37	1060	64	164	16749	1118	39	91	8355	1526	1	1	135
TOT		759048	438933	7094		9270	22375	-		1665	4281	551153		35	55	5274

remark of Martin and Shmoys (1996) that C-P shave especially works well when the gap between the one-machine bound and the optimal makespan is small.

Algorithm A1 has computation times for the blocking and no-wait job shop that are significantly smaller than those of algorithm A2. Therefore, we can conclude that in those cases applying the iterated C-P lower bound in a branch & bound algorithm gives larger computation times than using the length of the critical path as lower bound. Like with the C-P shave lower bound we conclude that applying the iterated C-P lower bound for the ideal job shop is profitable compared to only using the length of the critical path.

5.6.4 Comparison of the windows reduction techniques

In the last section we showed that determining the iterated C-P or C-P shave lower bound in each node of the search tree takes too much computation time, but this does not mean we can not use the iterated C-P or C-P shave algorithms. Both algorithms are able to reduce the processing windows of operations significantly. Let us assume we have a currently best known solution value UB. The iterated C-P or C-P shave algorithm can be applied to a feasibility instance with $T = UB - 1$ that can be created in a newly generated node. Note that we are not interested any more in a feasible solution with makespan UB, therefore we take $T = UB - 1$ for a feasibility instance. Once during the execution of an algorithm an empty processing window of an operation arises, then no better feasible solution exists in this part of the tree and the node can be pruned. If no empty processing window is found, then probably some of the processing windows have been reduced in length.

The first possible start time of each operation i is now given by u_i which is larger or equal than $lp(0, i)$. The last possible start time of each operation j is v_j, which can be smaller than $UB - 1 - lp(j, n+1)$. We use the reduced processing windows in the branching strategy. For each alternative arc (i, j) an estimation of the resulting makespan $em(i, j)$ is determined:

$$em(i, j) = u_i + l_{ij} + (UB - 1 - v_j).$$

The pair of alternative arcs $\{(i, j), (k, l)\}$ is chosen with maximum $em(i, j) + em(k, l)$. Of the two created nodes we select the node of the fixed alternative arc that has the smallest value of em. After a node has been selected the immediate selection rules are applied.

The two processing window reduction techniques provide us two new branch & bound algorithms. The first new branch & bound algorithm, A4, uses iterated C-P for processing windows reduction and the second new branch & bound algorithm A5 applies C-P shave for processing windows reduction. Algorithms A4 and A5 are compared with algorithm A1 that has been introduced in the last subsection. Algorithm A1 does not apply windows reduction and therefore it also does not use the newly introduced branching scheme.

Table 5.5 presents the computational results for the branch & bound algorithms A1, A4 and A5 for all four types. For the ideal job shop windows reduction with C-P shave is clearly the best approach. But we are mainly interested in the blocking

Table 5.5: Computational results for algorithms A1, A4 and A5

	I				BWS				BNS				NW			
I	Obj	A1	A4	A5	Obj	A1	A4	A5	Obj	A1	A4	A5	Obj	A1	A4	A5
Abz5	1234	128	101	14	1468	1198	1376	30874	1641	195	277	5061	2150	4	4	87
Abz6	943	60	2	4	1145	312	345	7012	1249	108	109	1530	1718	1	1	11
Ft10	930	4717	2438	34	1068	204	321	7604	1158	49	69	1846	1607	2	2	39
Orb01	1059	564502	144484	73	1175	1804	2353	47772	1256	181	311	5941	1615	1	2	21
Orb02	888	12	4	5	1041	379	481	7083	1144	64	91	1447	1485	2	2	21
Orb03	1005	136070	72209	253	1160	224	413	9187	1311	8	12	137	1599	2	2	44
Orb04	1005	358	194	13	1146	161	202	3686	1246	95	97	1808	1653	2	2	31
Orb05	887	104	42	11	995	56	79	1195	1203	20	28	465	1365	3	4	55
Orb06	1010	13513	1681	26	1199	2312	3144	70890	1266	254	532	11704	1555	1	1	6
Orb07	397	20	6	6	483	720	956	25736	527	303	357	6986	689	3	3	59
Orb08	899	17321	3678	11	995	130	184	3809	1139	22	30	541	1319	1	1	23
Orb09	934	55	26	7	1039	63	84	1088	1130	17	18	268	1445	1	1	10
Orb10	944	37	8	9	1146	77	93	1977	1367	12	16	146	1557	3	3	54
La16	945	20892	158	7	1060	202	279	4727	1148	72	107	1854	1575	2	2	18
La17	784	1155	2	5	929	239	352	6753	968	32	78	1340	1371	2	2	25
La18	848	92	2	5	1025	203	340	5171	1077	100	138	2168	1417	2	2	41
La19	842	6	6	7	1043	923	1221	23800	1102	94	125	1951	1482	2	2	38
La20	902	6	4	5	1060	64	94	1871	1118	39	48	809	1526	1	1	13
TOT		759048	225045	495		9270	12317	290235		1665	2443	45802		35	37	596

and no-wait job shop problems and both processing windows reduction algorithms are ineffective for those types. The degree of occupation of the machines is too low in those cases, hence there are not many C-P updates and thus the processing windows are hardly reduced. This leads us to the conclusion that spending time on processing windows reduction by using iterated C-P or C-P shave is also not profitable for the blocking and no-wait job shops.

5.6.5 Comparison with other algorithms

In the last two subsections we have presented five branch & bound algorithms and gave computational results on 18 instances with 10 jobs and 10 machines. Algorithm A1 clearly outperforms the other four branch & bound algorithms. Now the remaining question is: "How well does algorithm A1 perform in comparison with other solution approaches?". This question will be answered in this subsection.

In Section 5.2 a MIP formulation of the job shop problem has been presented for all four types. The job shop problems have also been solved by putting this MIP formulation in the general purpose solver CPLEX. The only setting of CPLEX that had been changed was the branching strategy, we set it on depth-first. This was necessary to prevent an out-of-memory error.

To our knowledge Mascis and Pacciarelli (2002) are the only ones that have implemented an exact algorithm for blocking or no-wait job shops. They have also developed a branch & bound algorithm. The most important differences between their algorithm and algorithm A1 are:

1. The branching scheme. They branch on the pair of alternative arcs $\{(i,j),(k,l)\}$ with maximum $lp(0,i)+l_{ij}+lp(j,n+1)$ and then select arc (k,l);
2. They use the minimum makespan of the one-machine problems $1|r_j,pmtn|L_{max}$ as a lower bound and adjust heads and tails according to the approach of Carlier and Pinson (1994);
3. The initial solution. They take the best solution value found by the four greedy heuristics that they have introduced.

Table 5.6 compares the computation times of algorithm A1 with the computation times of CPLEX 11.2 and of the branch & bound algorithm of Mascis and Pacciarelli (2002). Note that the computation times of their branch & bound algorithm have been found on a Pentium II 350 MHz PC while Algorithm A1 and CPLEX have been run on an Intel core T8300, 2.4 GHz processor with 2.0 GB internal memory.

A few cells in Table 5.6 contain the abbreviation n.s.f. that stands for 'no solution found'. For two instances of the blocking job shop CPLEX was not able to find a feasible solution and terminates after a long computation time with this remark.

For the blocking and no-wait job shop problems algorithm A1 clearly outperforms CPLEX. The branch & bound algorithm of Mascis and Pacciarelli (2002) has computation times that are more than forty times larger than the computation times of algorithm A1. Even if we take into account the difference in processor, it seems reasonable to conclude that algorithm A1 is a faster algorithm for the blocking and no-wait job shop. The difference in computation times can mainly be explained by the above mentioned differences one and two.

Table 5.6: Computational results of algorithm A1, solving MIP formulation with CPLEX (C) and the branch & bound algorithm of Mascis & Pacciarelli (MP)

	I				BWS				BNS				NW			
I	Obj	C	MP	A1	Obj	C	MP	A1	Obj	C	MP	A1	Obj	C	MP	A1
Abz5	1234	3211	3079	128	1468	96085	42441	1198	1641	27069	4670	195	2150	170	770	4
Abz6	943	8	90	60	1145	62863	15196	312	1249	47577	3071	108	1718	69	93	1
Ft10	930	62200	3313	4717	1068	35675	25228	204	1158	39623	6216	49	1607	42	195	2
Orb01	1059	> 200000	38072	564502	1175	124304	43669	1804	1256	63716	8311	181	1615	24	61	1
Orb02	888	32	453	12	1041	5909	13687	379	1144	20057	498	64	1485	95	169	2
Orb03	1005	33939	16376	136070	1160	n.s.f	15630	224	1311	10537	570	8	1599	36	157	2
Orb04	1005	23828	1570	358	1146	6456	11107	161	1246	8798	3255	95	1653	47	239	2
Orb05	887	182	2288	104	995	4861	9009	56	1203	2986	796	20	1365	185	1026	3
Orb06	1010	3039	10259	13513	1199	102192	115148	2312	1266	69783	12184	254	1555	13	30	1
Orb07	397	542	1035	20	483	14490	35226	720	527	34339	13255	303	689	122	435	3
Orb08	899	316	1264	17321	995	36242	8593	130	1139	3798	1034	22	1319	49	198	1
Orb09	934	573	1788	55	1039	6152	7513	63	1130	n.s.f.	879	17	1445	59	62	1
Orb10	944	246	409	37	1146	7789	10890	77	1367	4514	353	12	1557	249	843	3
La16	945	326	726	20892	1060	52621	11627	202	1148	32089	5531	72	1575	89	124	2
La17	784	32	35	1155	929	9085	10416	239	968	12124	2087	32	1371	103	139	2
La18	848	33	109	92	1025	6710	13838	203	1077	34611	3748	100	1417	145	196	2
La19	842	59	213	6	1043	19844	23390	923	1102	17455	2367	94	1482	168	371	2
La20	902	22	42	6	1060	9834	5660	64	1118	10448	1471	39	1526	23	178	1
TOT		-	81120	759048		601112	418266	9270		443524	70297	1665		1688	5286	35

5.7 More on the no-wait job shop

In the last section it has been shown that branch & bound algorithm A1 solves the no-wait job shop benchmark instances with 10 jobs and 10 machines within a few seconds. To the best of our knowledge, the only other exact algorithm for the no-wait job shop problem is the branch & bound algorithm of Mascis and Pacciarelli (2002). They only presented the optimal solutions of those instances with 10 jobs and 10 machines. Therefore, we run branch & bound algorithm A1 also for medium size benchmark instances for the no-wait job shop. In Table 5.7 we present the computation times and the optimal solution values of those instances. Also the optimal solution values of the instances swv1-5 have been added. These instances have been introduced in Storer et al. (1992).

Table 5.7: Computation times of algorithm A1 for medium size no-wait job shops

I	size	Opt	c.t.(s)
La06	(15,5)	1248	40
La07	(15,5)	1172	22
La08	(15,5)	1244	16
La09	(15,5)	1358	38
La10	(15,5)	1287	21
La11	(20,5)	1619	19438
La12	(20,5)	1414	3506
La13	(20,5)	1580	13768
La14	(20,5)	1578	3162
La15	(20,5)	1671	20128
La21	(15,10)	2030	226
La22	(15,10)	1852	285
La23	(15,10)	2021	490
La24	(15,10)	1972	356
La25	(15,10)	1906	133

I	size	Opt	c.t.(s)
La26	(20,10)	2467	107076
La27	(20,10)	2611	193806
La28	(20,10)	2546	106120
La29	(20,10)	2300	72144
La30	(20,10)	2452	17398
La36	(15,15)	2685	695
La37	(15,15)	2831	1963
La38	(15,15)	2525	1106
La39	(15,15)	2660	2515
La40	(15,15)	2564	1505
Ft20	(20,5)	1532	11830
swv01	(20,10)	2318	16694
swv02	(20,10)	2417	31316
swv03	(20,10)	2381	87750
swv04	(20,10)	2462	298923
swv05	(20,10)	2333	10593

To get a proof of optimality takes most of the computation time for algorithm A1. A very good feasible solution is found after a short computation time, which means that the branch & bound algorithm can also be used as a heuristic. We test the performance of algorithm A1 applied as heuristic. This was inspired by the following remark in Bansal et al. (2005): "In contrast with the ideal job shop problem, which has a huge amount of literature devoted to heuristics and exact algorithms, there is a relatively small number of papers devoted to experimental analysis of practical algorithms devoted to the no-wait job shop". Heuristics for the no-wait job shop problem are the central topic for the second part of this section. First, an overview of the literature on heuristics are presented. After that, the best of these heuristics are compared with branch & bound algorithm A1 with a bounded computation time.

Raaymakers and Hoogeveen (2000) present a simulated annealing approach to

solve no-wait job shop problems with parallel machines. A fast deterministic variable neighborhood search algorithm and a non-deterministic hybrid algorithm (GASA) using elements of genetic algorithms (GA) and simulated annealing (SA) have been introduced by Schuster and Framinan (2003). Recently, a hybrid genetic algorithm has been proposed by Pan and Huang (2009). MacChiaroli et al. (1999) and Schuster (2006) present tabu search algorithms. The tabu search algorithm of Schuster (2006) outperforms the GASA algorithm of Schuster and Framinan (2003) and has a very small computation time.

Framinan and Schuster (2006) tried a metaheuristic, which was recently introduced by Ghosh and Sierksma (2002), named complete local search with memory (CLM). This metaheuristic records all explored solutions to avoid the exploration of already visited solutions. The complete local search heuristic has comparable quality solutions as the tabu search algorithm of Schuster (2006), but the algorithm has been run 30 times to find this solution with an average computation time that is larger than the computation time of the tabu search algorithm of Schuster.

A complete local search with limited memory (CLLM) heuristic has been developed by Zhu et al. (2009). To apply complete local search with memory (CLM) for solving a combinatorial optimization problem, large memory is needed. Also it is very time-consuming to verify whether a new sequence of jobs is already in the memory. Zhu et al. (2009) modify CLM to limit the memory size and introduce a strategy to refresh the contents of the memory. Their heuristic outperforms the algorithms of Schuster (2006) and Framinan and Schuster (2006), but uses more computation time.

Bozejko and Makuchowski (2009) provide a hybrid tabu search algorithm. They compare their approach to the tabu search algorithm of Schuster (2006) by setting the computation time of Schuster as maximum computation time. In spite of the use of a processor of 800 MHz compared to 1400 MHz used by Schuster, their algorithm found better solutions for almost all instances. They also present results without the time bound of Schuster. Unfortunately, they do not give the used computation times.

We try to compare algorithm A1 with the heuristics that found the best solutions, namely the hybrid tabu search algorithm of Bozejko and Makuchowski (2009) and CLLM of Zhu et al. (2009). Table 5.8 shows the performance of branch & bound algorithm A1 with bounded computation time. The last column contains the average computation time of 20 runs of CLLM. The best solution value found for those 20 runs is shown in column 'Zhu'. We run algorithm A1 for an amount of time equal to this average computation time and the objective value found is given in Table 5.8. For the medium size instances the objective values found by algorithm A1 are of a slightly better quality than the solutions found by Zhu et al. (2009). Note that we only did one run within the allowed computation time and they did 20 runs with on average this computation time. They don't specify the used processor, hence stating a fair conclusion is impossible. For the larger instances algorithm CLLM seems to perform better than algorithm A1.

Column 'bks' in Table 5.8 gives the best known solution value of an instance. If this value is put in boldface, then it is the optimal solution value. The objective values found by Bozejko and Makuchowski (2009) are presented in the column 'Boz'. The best solutions have been obtained by their hybrid tabu search algorithm. A

Table 5.8: Comparison of algorithm A1 with heuristics of Bozejko and Makuchowski (2009) and Zhu et al. (2009)

I	size	bks	Boz	Zhu	A1	ct(s)
La11	(20,5)	**1619**	1621	1671	1654	447
La12	(20,5)	**1414**	1434	1452	1451	498
La13	(20,5)	**1580**	1580	1624	1595	640
La14	(20,5)	**1578**	1610	1691	1578	465
La15	(20,5)	**1671**	1686	1694	1686	484
La21	(15,10)	**2030**	2030	2048	2030	306
La22	(15,10)	**1852**	1852	1887	1852	354
La23	(15,10)	**2021**	2021	2032	2021	307
La24	(15,10)	**1972**	1972	2015	1972	422
La25	(15,10)	**1906**	1906	1917	1906	297
La26	(20,10)	**2467**	2506	2553	2598	812
La27	(20,10)	**2611**	2675	2747	2755	834
La28	(20,10)	**2546**	2552	2624	2722	810
La29	(20,10)	**2300**	2300	2489	2427	778
La30	(20,10)	**2452**	2452	2665	2572	822
La31	(30,10)	3498	3498	3745	3708	2588
La32	(30,10)	3882	3882	4028	4337	2698
La33	(30,10)	3454	3454	3749	3976	2587
La34	(30,10)	3659	3659	3824	4161	2754
La35	(30,10)	3552	3552	3760	3945	2615
La36	(15,15)	**2685**	2685	2685	2692	535
La37	(15,15)	**2831**	2831	2962	2977	505
La38	(15,15)	**2525**	2525	2617	2571	497
La39	(15,15)	**2660**	2687	2697	2706	558
La40	(15,15)	**2564**	2580	2594	2709	283
swv01	(20,10)	**2318**	2318	2328	2344	641
swv02	(20,10)	**2417**	2417	2418	2430	777
swv03	(20,10)	**2381**	2381	2415	2517	757
swv04	(20,10)	**2462**	2462	2542	2635	675
swv05	(20,10)	**2333**	2333	2333	2555	712
swv06	(20,15)	**3278**	3290	3376	3449	1136
swv07	(20,15)	3188	3188	3271	3357	1176
swv08	(20,15)	3423	3423	3530	3949	1149
swv09	(20,15)	3270	3270	3307	3355	1053
swv10	(20,15)	3462	3462	3488	3790	1142

large number of times they found an optimal solution. Unfortunately, Bozejko and Makuchowski (2009) did not say anything on the used computation times which makes it impossible to make a fair comparison between the algorithms.

5.8 Concluding remarks

Five branch & bound algorithms have been presented for the no-wait and blocking job shop. The computational results showed that applying iterated C-P or C-P shave as a lower bound in each node of the search tree does not reduce the computation times. Also spending time on processing window reduction by using iterated C-P or C-P shave is too time consuming. The branch & bound approach that uses the length of the longest path in the alternative graph as lower bound performs the best. Hence, a strong lower bound is still lacking and is an interesting issue for future research.

The branch & bound algorithm with the smallest computation times (A1) has been compared to the computation times of Mascis and Pacciarelli (2002) and the computation times that CPLEX needs to solve the MIP formulations of the problems. Algorithm A1 clearly outperforms these two solution approaches for the no-wait and blocking job shop.

We only applied the job insertion heuristic in the root node of the search tree and did not apply a heuristic in all other nodes. It would be interesting to see whether a heuristic in combination with a different search strategy would provide better computational results. A main difficulty that has to be taken care of is the amount of memory that is needed.

The central topic of this chapter has been a simplified model of the routing problem, which was the main topic of Chapter 4. Elements that should be added to the simplified model to come closer to the real-world routing problem are multiprocessor operations and timelags between two operations. Those extensions can easily be modeled with the alternative graph and branch & bound algorithm A1 is still applicable. A third extension that is necessary to model the routing problem with fixed routes is more difficult structures of precedence relations. If an arriving train is split into two parts, both parts should be modeled as a successor of the operation that models the platform occupation. So the simplified model has to be extended to make it possible that an operation has two successors. Introducing those complex precedence relations can be modeled with the alternative graph. Once those three additional elements have been added to the model, we should be able to solve the routing problem with fixed routes. Unfortunately, we did not implement this. Therefore, we have not been able to test algorithm A1 on the real-world applications described in Chapter 4.

The research on no-wait and blocking job shop scheduling problems has been initiated by the large computation times of MVR that has been introduced in Chapter 4. The additional allowed routes for train services have to be added to the job shop model. This can be done by introducing for each operation a set of possible machine sets. From this set, one set of machines has to be chosen on which the operation is processed. This problem is known as the multi-mode job shop problem, Brucker and Neyer (1998). Addition of those sets of machine sets for each operation leads to

a much more difficult problem that can not be modeled with the alternative graph. Therefore, an interesting topic for future research is the multi-mode job shop with no-wait and blocking precedence constraints.

Bibliography

Aarts, E. H. L. and J. K. Lenstra (Eds.) (1997). *Local search in combinatorial optimization*. Discrete Mathematics and Optimization. Wiley, Chichester, UK.

Adams, J., E. Balas, and D. Zawack (1988). The shifting bottleneck procedure for job shop scheduling. *Management Science 34*, 391–401.

Ahuja, R. K., T. L. Magnanti, and J. B. Orlin (1993). *Network flows: theory, algorithms, and applications*. Upper Saddle River, New Jersey, USA: Englewood Cliffs: Prentice-Hall, Inc.

Applegate, D. and W. Cook (1991). A computational study of the job-shop scheduling problem. *ORSA Journal on Computing 3*(2), 149–156.

Atsuta, M., K. Nonobe, and T. Ibaraki (2008). Itc-2007 track2: An approach using general csp solver. Submission to ITC2007 track 2.

Baker, K. R. (1974). *Introduction to sequencing and scheduling*. John Wiley & Sons.

Bansal, N., M. Mahdian, and M. Sviridenko (2005). Minimizing makespan in no-wait job shops. *Mathematics of Operations Research 30*(4), 817–831.

Bertsekas, D., J. Tsitsiklis, and C. Wu (1997). Rollout algorithms for combinatorial optimization. *Journal of Heuristics 3*, 245–262.

Billionnet, A. (2003). Using integer programming to solve the train-platforming problem. *Transportation Science 37*, 213–222.

Blasum, U., M. R. Bussieck, W. Hochstättler, C. Moll, H.-H. Scheel, and T. Winter (1999). Scheduling trams in the morning. *Mathematical Methods of Operations Research 49*(1), 137–148.

Blazewicz, J., W. Domschke, and E. Pesch (1996). The job shop scheduling problem: Conventional and new solution techniques. *European Journal of Operational Research 93*(1), 1–33.

Blazewicz, J., K. Ecker, E. Pesch, G. Schmidt, and J. Weglarz (2007). *Handbook on scheduling: models and methods for advanced planning (international handbooks on information systems)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc.

Borndörfer, R. M., M. Grötschel, and A. Löbel (2003). Duty scheduling in public transit. In W. Jäger and H.-J. Krebs (Eds.), *Mathematics-Key Technology for the Future*, pp. 653–674. Berlin: Springer-Verlag.

Bozejko, W. and M. Makuchowski (2009). A fast tabu search algorihm for the no-wait job shop problem. *Computers & Industrial Engineering 56*, 1502–1509.

Bratley, P., M. Florian, and P. Robillard (1973). On sequencing with earliest starts and due dates with application to computing bounds for the $(n/m/g/f_{max})$ problem. *Naval Research Logistics Quarterly 20*(1), 57–67.

Brizuela, C. A., Y. Zhao, and N. Sannomiya (2001). No-wait and blocking job-shops: challenging problems for ga's. In *Systems, Man, and Cybernetics, 2001 IEEE International Conference on*, Volume 4, pp. 2349–2354.

Brucker, P. (2007). *Scheduling algorithms* (Fifth ed.). Berlin: Springer-Verlag.

Brucker, P., B. Jurisch, and B. Sievers (1994). A branch and bound algorithm for the job-shop scheduling problem. *Discrete Applied Mathematics 49*, 107–127.

Brucker, P. and J. Neyer (1998). Tabu-search for the multi-mode job-shop problem. *OR Spektrum 20*, 21–28.

Burke, E. K. and P. de Causmaecker (Eds.) (2003). *Practice and theory of automated timetabling IV, 4th international conference, selected revised papers*, Volume 2740 of *Lecture Notes in Computer Science*. Springer.

Burke, E. K., K. Jackson, J. H. Kingston, and R. F. Weare (1997). Automated university timetabling: The state of the art. *The Computer Journal 40*(9), 565–571.

Burke, E. K., B. McCollum, J. P. McMullan, and R. Qu (2006). Examination timetabling: A new formulation. In E. K. Burke and H. Rudova (Eds.), *Proceedings of the 6th International Conference on the Practice and Theory of Automated Timetabling*, pp. 373–375.

Burke, E. K. and S. Petrovic (2002). Recent research directions in automated timetabling. *European Journal of Operational Research 140*(2), 266–280.

Burke, E. K. and M. A. Trick (Eds.) (2005). *Practice and theory of automated timetabling V, 5th international conference, revised selected papers*, Volume 3616 of *Lecture Notes in Computer Science*. Springer.

Busam, V. A. (1967). An algorithm for class scheduling with section preference. *Communications of the ACM 10*(9), 567–569.

Cambazard, H., E. Hebrard, B. O'Sullivan, and A. Papadopoulos (2008). Local search and constraint programming for the post-enrolment course timetabling problem. Proceedings of the Conference on the Practice and Theory of Automated Timetabling (PATAT 2008), Montreal, Canada.

Carey, M. and S. Carville (2003). Scheduling and platforming trains at busy complex stations. *Transportation Research A 37*, 195–224.

Carlier, J. and E. Pinson (1989). An algorithm for solving the job-shop problem. *Management Science 35*(2), 164–176.

Carlier, J. and E. Pinson (1990). A practical use of jackson's preemptive schedule for solving the job-shop problem. *Annals of Operations Research 26*, 269–287.

Carlier, J. and E. Pinson (1994). Adjustments of heads and tails for the job-shop problem. *European Journal of Operational Research 78*, 146–161.

Carter, M. W. and G. Laporte (1995). Recent developments in practical examination timetabling. In E. K. Burke and P. Ross (Eds.), *PATAT*, Volume 1153 of *Lecture Notes in Computer Science*, pp. 3–21. Springer.

Carter, M. W. and G. Laporte (1997). Recent developments in practical course timetabling. In E. K. Burke and M. W. Carter (Eds.), *PATAT*, Volume 1408 of *Lecture Notes in Computer Science*, pp. 3–19. Springer.

Chen, B., C. N. Potts, and G. J. Woeginger (1998). A review of machine scheduling: complexity, algorithms and approximability. In *Handbook of Combinatorial Optimization, Vol. 3*, pp. 21–169. Boston, MA: Kluwer Acad. Publ.

Cheng, E., S. Kruk, and M. J. Lipman (2003). Flow formulations for the student scheduling problem. See Burke and de Causmaecker (2003), pp. 299–309.

Chiarandini, M., C. Fawcett, and H. H. Hoos (2008). A modular multiphase heuristic solver for post enrolment course timetabling. Proceedings of the Conference on the Practice and Theory of Automated Timetabling (PATAT 2008), Montreal, Canada.

Conway, R. W., W. L. Maxwella, , and L. W. Miller (1967). *Theory of scheduling*. Addison-Wesley Publishing Co., Reading, Mass.-London-Don Mills, Ont.

Cornelsen, S. and G. DiStefano (2007). Track assignment. *Journal of Discrete Algorithms 5*(2), 250–261.

D'Ariano, A., D. Pacciarelli, and M. Pranzo (2007). A branch and bound algorithm for scheduling trains in a railway network. *European Journal of Operational Research 183*(2), 643–657.

Daskalaki, S. and T. Birbas (2005). Efficient solutions for a university timetabling problem through integer programming. *European Journal of Operational Research 160*(1), 106–120.

de Werra, D. (1985). An introduction to timetabling. *European Journal of Operational Research 19*(2), 151–162.

Desaulniers, G., J. Desrosiers, and M. M. Solomon (2001). Accelerating strategies in column generation methods for vehicle routing and crew scheduling problems. In C. C. Ribiero and P. Hansen (Eds.), *Essays and Surveys in Metaheuristics*, pp. 309–324. Kluwer, Boston, MA.

Desrosiers, J. and M. E. Lübbecke (2005). A primer in column generation. In G. Desaulniers, J. Desrosiers, and M. M. Solomon (Eds.), *Column Generation*, Chapter 1, pp. 1–32. Springer-Verlag, New York.

DiMiele, F. and G. Gallo (2001). Dispatching buses in parking depots. *Transportation Science 35*(3), 322–330.

DiStefano, G. and M. L. Koci (2004). A graph theoretical approach to the shunting problem. In *Proceedings of the 3rd Workshop on Algorithmic Methods and Models for Optimization for Railways (ATMOS 2003)*, Volume 92 of *Electronic Notes in Theoretical Computer Science*.

Even, S., A. Itai, and A. Shamir (1976). On the complexity of timetable and multicommodity flow problems. *SIAM Journal of Computing 5*(4), 691–703.

Feldman, R. and M. Golumbic (1989). Constraint satisfiability algorithms for interactive student scheduling. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence (IJCAI 89)*, pp. 1010–1016.

Framinan, J. M. and C. Schuster (2006). An enhanced timetabling procedure for the no-wait job shop problem: a complete local search approach. *Computers & Operations Research 33*(5), 1200–1213.

Freling, R., R. M. Lentink, L. G. Kroon, and D. Huisman (2005). Shunting of passenger train units in a railway station. *Transportation Science 39*(2), 261–272.

French, S. (1982). *Sequencing and scheduling: An introduction to the mathematics of the job-shop*. Chichester: Ellis Horwood Ltd. Ellis Horwood Series in Mathematics and its Applications.

Fuchsberger, M. (2007). Solving the train scheduling problem in a main station area via a resource constrained space-time integer multicommodity flow. Master's thesis.

Garey, M. R. and D. S. Johnson (1979). *Computers and intractability - a guide to NP-completeness*. San Fransisco: W.H. Freeman and Company.

Gaspero, L. D., B. McCollum, and A. Schaerf (2007). The second international timetabling competition (itc 2007): Curriculum-based course timetabling (track 3). Technical report, Queens's University.

Ghosh, D. and G. Sierksma (2002). Complete local search with memory. *Journal of Heuristics 8*(6), 571–584.

Graham, R. L., E. L. Lawler, J. K. Lenstra, and A. H. G. R. Kan (1979). Optimization and approximation in deterministic sequencing and scheduling: a survey. *Ann. of Discrete Math. 5*, 287–326. Discrete optimization (Proc. Adv. Res. Inst. Discrete Optimization and Systems Appl., Banff, Alta., 1977), II.

Gröflin, H. and A. Klinkert (2009). A new neighborhood and tabu search for the blocking job shop. *Discrete Applied Mathematics In Press, Corrected Proof*, –.

Hall, N. G. and C. Sriskandarajah (1996). A survey of machine scheduling problems with blocking and no-wait in process. *Operations Research 44*(3), 510–525.

Hamdouni, M., G. Desaulniers, O. Marcotte, F. Soumis, and M. van Putten (2006). Dispatching buses in a depot using block patterns. *Transportation Science* (3), 364–377.

Hoekert, W. (2001). Het maken van diensten voor rangeerpersoneel (creating schedules for shunting crew). Master's thesis, Erasmus University, Rotterdam, the Netherlands. In Dutch.

Hutter, F., D. Babić, H. H. Hoos, and A. J. Hu (2007). Boosting verification by automatic tuning of decision procedures. In *Proceedings of Formal Methods in Computer Aided Design (FMCAD'07)*, Washington, DC, USA, pp. 27–34. IEEE Computer Society.

ITC (2002). International timetabling competition 2002: www.idsia.ch/files/ttcomp2002.

ITC (2007). International timetabling competition 2007: www.cs.qub.ac.uk/itc2007.

Jackson, J. R. (1956). An extension of johnson's results on job lot scheduling. *Naval Research Logistics Quarterly 3*(3), 201–203.

Jain, A. S. and S. Meeran (1999). A state-of-the-art review of job-shop scheduling techniques. *European Journal of Operations Research 113*(2), 390–434.

Khachiyan, L. G. (1979). A polynomial algorithm in linear programming. *Soviet Mathematics Doklady 20*, 191–194.

Kostuch, P. (2004). The university course timetabling problem with a three-phase approach. See Burke and Trick (2005), pp. 109–125.

Kroon, L. G., R. M. Lentink, and A. Schrijver (2008). Shunting of passenger train units: An integrated approach. *Transportation Science 42*(4), 436–449.

Laporte, G. and S. Desrochers (1986). The problem of assigning students to course sections in a large engineering school. *Computational Operations Research 13*, 387–394.

Lawrence, S. (1984). Supplement to resource constrained project scheduling: An experimental investigation of heuristic scheduling techniques. Technical report, GSIA, Carnegie Mellon University, Pittsburg, PA.

Lentink, R. M. (2006). *Algorithmic decision support for shunt planning*. Ph. D. thesis, Erasmus University Rotterdam, The Netherlands.

Lewis, R. and B. Paechter (2007). Finding feasible timetables using group-based operators. *IEEE Transactions on Evolutionary Computation 11*(3), 397–413.

Lewis, R., B. Paechter, and B. McCollum (2007). Post enrolment based course timetabling: A description of the problem model used for track two of the second international timetabling competition. Technical report, Cardiff University.

Lübbecke, M. E. and J. Desrosiers (2005). Selected topics in column generation. *Operations Research 53*(6), 1007–1023.

MacChiaroli, R., S. Mole, and S. Riemma (1999). Modelling and optimization of industrial manufacturing processes subject to no-wait constraints. *International Journal of Production Research 37*(11), 2585 – 2607.

Martin, P. and D. B. Shmoys (1996). A new approach to computing optimal schedules for the job-shop scheduling problem. In *IPCO*, pp. 389–403.

Mascis, A. and D. Pacciarelli (2002). Job-shop scheduling with blocking and no-wait constraints. *European Journal of Operational Research 143*(3), 498–517.

Mati, Y., N. Rezg, and X. Xie (2001). Geometric approach and taboo search for scheduling flexible manufacturing systems. *IEEE Transactions on Robotics and Automation 17*, 805–818.

Mayer, A., C. Nothegger, A. Chwatal, and G. Raidl (2008). Solving the post enrolment course timetabling problem by ant colony optimization. Proceedings of the Conference on the Practice and Theory of Automated Timetabling (PATAT 2008), Montreal, Canada.

McCollum, B. (2006). A perspective on bridging the gap in university timetabling. In E. K. Burke and H. Rudova (Eds.), *Proceedings of the 6th International Conference on the Practice and Theory of Automated Timetabling*, Volume 3867 of *Lecture Notes in Computer Science*, Brno, pp. 3–24. Springer.

McCollum, B., P. McMullan, E. K. Burke, A. J. Parkes, and R. Qu (2007). The second international timetabling competition: Examination timetabling track. Technical report, Queen's University.

McCollum, B., A. Schaerf, B. Paechter, P. McMullan, R. Lewis, A. Parkes, L. D. Gaspero, R. Qu, and E. K. Burke (2009). Setting the research agenda in automated timetabling: The second international timetabling competition. *INFORMS Journal on Computing*. to appear.

Meloni, C., D. Pacciarelli, and M. Pranzo (2004). A rollout metaheuristic for job shop scheduling problems. *Annals of Operations Research 131*, 215–235.

Miyaji, I., K. Ohno, and H. Mine (1988). Solution method for partitioning students into groups. *European Journal of Operational Research 33*(1), 82–90.

Muller, T. (2008). Itc2007 solver description: A hybrid approach. Proceedings of the Conference on the Practice and Theory of Automated Timetabling (PATAT 2008), Montreal, Canada.

Muth, J. F. and G. L. Thompson (1963). *Industrial scheduling*. Prentice-Hall.

Nowicki, E. and C. Smutnicki (1996). A fast taboo search algorithm for the job shop problem. *Management Science 42*(6), 797–813.

Pan, J. C. and H. Huang (2009). A hybrid genetic algorithm for no-wait job shop scheduling problems. *Expert Systems with Applications 36*(3), 5800–5806.

Papadimitriou, C. H. and P. C. Kanellakis (1980). Flowshop scheduling with limited temporary storage. *Journal of the ACM 27*(3), 533–549.

Petrovic, S. and E. K. Burke (2004). University timetabling. In *Handbook of scheduling: algorithms, models, and performance analysis*, Chapter 45. CRC Press.

Pinedo, M. L. (2005). *Planning and scheduling in manufacturing and services (Springer series in operations research and financial engineering)*. Springer.

Qualizza, A. and P. Serafini (2004). A column generation scheme for faculty timetabling. See Burke and Trick (2005), pp. 161–173.

Raaymakers, W. H. M. and J. A. Hoogeveen (2000). Scheduling multipurpose batch process industries with no-wait restrictions by simulated annealing. *European Journal of Operational Research 126*(1), 131–151.

Ramaswamy, S. E. and S. B. Joshi (1996). Deadlock-free schedules for automated manufacturing workstations. *IEEE Transactions on Robotics and Automation 12*(3), 391–400.

Rossi-Doria, O., M. Sampels, M. Birattari, M. Chiarandini, M. Dorigo, L. M. Gambardella, J. Knowles, M. Manfrin, M. Mastrolilli, B. Paechter, L. Paquete, and T. Stützle (2002). A comparison of the performance of different metaheuristics on the timetabling problem. See Burke and de Causmaecker (2003), pp. 329–354.

Roy, B. and B. Sussman (1964). Les problèm d'ordonnancement avec constraintes disjonctives. *Note DS No.9 bis, SEMA*.

Sabin, G. C. W. and G. K. Winter (1986). The impact of automated timetabling on universities - a case study. *Journal of Operations Research Society 37*, 689–693.

Sahni, S. and Y. Cho (1979). Complexity of scheduling shops with no-wait in process. *Mathematics of Operations Research 4*(4), 448–457.

Schaerf, A. (1999). A survey of automated timetabling. *Artificial Intelligence Review 13*(2), 87–127.

Schmidt, G. and T. Ströhlein (1980). Timetable construction - an annotated bibliography. *The Computer Journal 23*(4), 307–316.

Schuster, C. J. (2006). No-wait job shop scheduling: tabu search and complexity of subproblems. *Mathematical Methods of Operations Research 63*(3), 473–491.

Schuster, C. J. and J. M. Framinan (2003). Approximative procedures for no-wait job shop scheduling. *Operations Research Letters 31*(3), 308–318.

Storer, R., S. Wu, and R. Vaccari (1992). New search spaces for sequencing instances with application to job shop scheduling. *Management Science 38*, 1495–1509.

Taillard, E. (1994). Parallel taboo search technique for the job shop scheduling problem. *ORSA Journal on Computing 6*(2), 108–117.

Talbi, E.-G. (2009). *Metaheuristics : from design to implementation.* John Wiley & Sons.

Timkovsky, V. G. (1985). On the complexity of scheduling an arbitrary system. *Soviet Journal of Computer ad System Sciences 5*, 46–52.

Tomii, N. and L. J. Zhou (2000). Depot shunting scheduling using combined genetic algorithm and pert. In J. Allan, R. Hill, C. Brebbia, G. Sciutto, and S. Sone (Eds.), *Computer in Railways VII*, pp. 437 – 446. WIT Press, Southampton.

Tomii, N., L. J. Zhou, and N. Fukumara (1999). An algorithm for station shunting scheduling problems combining probabilistic local search and PERT. In I. Imam, Y. Kodratoff, A. El-Dessouki, and M. Ali (Eds.), *Multiple approaches to intelligent systems: 12th international conference on industrial and engineering applications of artificial intelligence and expert systems*, Volume 1611 of *Lecture Noten in Computer Science*, pp. 788 – 797. Springer Berlin / Heidelberg.

Tripathy, A. (1992). Computerised decision aid for timetabling - a case analysis. *Discrete Applied Mathematics 35*(3), 313–323.

van den Broek, J. J. J. (2002). Toets op inplanbaarheid van rangeerbewegingen. Master's thesis, Eindhoven University of Technology.

van den Broek, J. J. J. and L. G. Kroon (2007). A capacity test for shunting movements. In F. Geraets, L. Kroon, A. Schöbel, D. Wagner, and C. Zaroliagis (Eds.), *Algorithmic Methods for Railway Optimization*, Volume 4359 of *Lecture Notes in Computer Science*, pp. 108–125. Springer.

van Laarhoven, P., E. Aarts, and J. K. lenstra (1992). Job shop scheduling by simulated annealing. *Operations Research 40*(1), 113–125.

Winter, T. and U. T. Zimmermann (2000). Real-time dispatch of trams in storage yards. *Annals of Operations Research 96*, 287 – 315.

Woeginger, G. J. (2004). Inapproximability results for no-wait job shop scheduling. *Operations Research Letters 32*(4), 320–325.

Yamada, T. and R. Nakano (1992). A genetic algorithm applicable to large-scale job-shop problems. In *Parallel Problem Solving from Nature 2*, pp. 283–292.

Zampieri, A. and A. Schaerf (2006). Modelling and solving the italian examination timetabling problem using tabu search. In E. Burke and H. Rudova (Eds.), *Proceedings of the 6th International conference on the Practice and Theory of Automated Timetabling*, Brno, pp. 487 – 491.

Zhu, J., X. Li, and Q. Wang (2009). Complete local search with limited memory algorithm for no-wait job shops to minimize makespan. *European Journal of Operational Research 198*(2), 378–386.

Zwaneveld, P. J. (1997). *Railway planning, routing of trains and allocation of passenger lines*. Ph. D. thesis, Erasmus University Rotterdam.

Zwaneveld, P. J., S. Dauzère-Pérès, C. P. M. van Hoesel, L. G. Kroon, H. E. Romeijn, M. Salomon, and H. W. Ambergen (1996). Routing trains through railway stations: Model formulation and algorithms. *Transportation Science 30*, 181–194.

Zwaneveld, P. J., L. G. Kroon, and S. P. M. van Hoesel (2001). Routing trains through a railway station based on a node packing model. *European Journal of Operational Research 128*(1), 14–33.

Samenvatting

In het dagelijks leven kom je overal om je heen plannen en roosters tegen. Neem bijvoorbeeld het Nederlandse spoorboekje, het speelschema van de Nederlandse eredivisie en de dienstroosters van zusters in ziekenhuizen of van de thuiszorg. Zowel de technieken voor het oplossen van dergelijke plannings- en roosterproblemen, als de rekenkracht van computers zijn enorm verbeterd. Tegenwoordig zijn we in staat om uitstekende plannen en roosters te construeren. Het doel in dit proefschrift is om dergelijke plannings- en roosterproblemen uit de praktijk op te lossen met geavanceerde technieken die gebaseerd zijn op gemengde geheeltallige programmering. In het vervolg wordt dit afgekort tot MIP, dat staat voor mixed integer programming.

Zowel vanuit theoretisch als praktisch oogpunt hebben we onderzocht of oplosmethoden gebaseerd op MIP geschikt zijn voor een drietal planningsproblemen uit de praktijk. Voor elk probleem is een exact algoritme of een heuristiek ontwikkeld die een goede of zelfs optimale oplossing oplevert in een rekentijd die klein genoeg is voor de praktische toepassing. Elk van de algoritmen is gebaseerd op een MIP formulering van het betreffende probleem. Als het oplossen met behulp van de solver CPLEX te veel rekentijd vergt hebben we een alternatieve aanpak ontwikkeld.

Twee praktische problemen die we bestudeerd hebben zijn roosterproblemen die universiteiten ondervinden. In dergelijke roosterproblemen wordt het wekelijkse lesrooster voor bijeenkomsten van vakken gemaakt. De vakken moeten worden toegewezen aan een lesuur en aan een collegezaal of lokaal. Het doel daarbij is het minimaliseren van de overlap van bijeenkomsten die door dezelfde studenten worden bijgewoond. In een lesrooster-gebaseerd roosterprobleem ontstaan er conflicten volgens het lesrooster dat gepubliceerd wordt door de universiteit en niet als gevolg van inschrijving van studenten. In de twee roosterproblemen die wij bestudeerd hebben, selecteren de studenten de vakken die ze willen bijwonen voordat het rooster wordt opgesteld. In paragraaf 1.4 wordt beschreven waar de twee roosterproblemen gepositioneerd worden in het onderzoeksgebied van onderwijs gerelateerde roosterproblemen.

De derde toepassing die we bestudeerd hebben is het plannen van rangeerbewegingen op treinstations. Naast alle geplande reizigerstreinen en goederentreinen moeten er rondom grote stations in een spoorwegnetwerk rangeerbewegingen uitgevoerd worden tussen de perronsporen en de opstelterreinen. Rangeerbewegingen voorzien reizigerstreinen van de juiste materieelsamenstelling. Treinen die hun dienst starten moeten soms van materieel voorzien worden dat afkomstig is van een opstelterrein en materieel van eindigende treinen moet soms naar een opstelterrein gebracht worden. In de spitsuren rijden treinen meestal met zo'n groot mogelijke capaciteit. Echter, buiten de spits heeft een vervoerder een overschot aan materieel. Dit overschot wordt

dan op een opstelterrein geparkeerd. Paragraaf 1.5 beschrijft het rangeer planproces bij de Nederlandse Spoorwegen en beschrijft het probleem dat wij in dit proefschrift beschouwen.

Het eerste roosterprobleem dat we hebben bestudeerd is geïntroduceerd tijdens de tweede International Timetabling Competition (ITC2), welke georganiseerd is in 2007. Studenten hebben zich reeds ingeschreven voor een aantal vakken. Deze vakken moeten worden toegewezen aan een lokaal en een lesuur op zodanige wijze dat alle studenten hun gewenste vakken kunnen bijwonen. Dit onder de voorwaarden dat er niet twee vakken die gevraagd worden door een zelfde student, toegewezen worden aan hetzelfde lesuur en dat er niet twee vakken aan hetzelfde lokaal worden toegewezen in hetzelfde lesuur. Er zijn tevens enkele gewenste voorwaarden die het rooster aantrekkelijker maken. Een voorbeeld hiervan is dat zo min mogelijk vakken op het laatste lesuur van de dag geroosterd moeten worden. Voor een gedetailleerde beschrijving van dit roosterprobleem verwijzen we naar hoofdstuk 2.

Het betreffende roosterprobleem hebben we geformuleerd als een MIP, maar oplossen met CPLEX kostte veel te veel rekentijd. Daarom hebben we een heuristiek ontworpen die vakken toewijst aan lesuren gebaseerd op een LP-oplossing die geconstrueerd is met kolomgeneratie. Een kolom correspondeert met een toegestane toewijzing van vakken aan een lesuur. We krijgen een geheeltallige oplossing door kolommen één voor één te fixeren. De gegenereerde oplossing wordt verbeterd door een verzameling van vakken te selecteren die opnieuw worden toegewezen door middel van het oplossen van een MIP. Uit de vergelijkingen van de resultaten van onze heuristiek met de resultaten van de vijf finalisten van de ITC2 blijkt dat onze aanpak concurrerend is en zelfs zeer succesvol in het genereren van toegelaten oplossingen voor de moeilijkere instanties.

Hoofdstuk 3 beschrijft een zeer succesvolle aanpak van een roosterprobleem van de Industrial Design faculteit aan de TU Eindhoven. Studenten van deze faculteit kiezen een gedeelte van hun vakken zelf door deze te selecteren uit een enorme verzameling vakken. Ze doen dit door een geordende lijst met hun favoriete vakken voor de komende periode in te leveren. Gebaseerd op deze inschrijvingen en een heleboel andere voorwaarden, wijst de faculteit vakken toe aan studenten. In hoofdstuk 3 lichten we alle voorwaarden van de situatie in Eindhoven toe. Het probleem is opgelost met behulp van lexicografische optimalisering met vier deelproblemen. Voor alle vier de deelproblemen wordt een MIP formulering gegeven die ook alle vier eenvoudig door CPLEX opgelost kunnen worden.

In hoofdstuk 4 worden twee MIP formuleringen gepresenteerd voor het routeringsprobleem dat een onderdeel is van het rangeer planningsproces. Deze MIP modellen proberen de noodzakelijke rangeerbewegingen in te plannen tussen reizigerstreinen en goederentreinen. Beide modellen minimaliseren het aantal rangeerbewegingen dat niet ingepland kan worden als gevolg van een gebrek aan capaciteit van de infrastructuur. In het eerste model liggen alle routes van treinen op voorhand al vast. Het MIP kan dan in korte tijd opgelost worden door CPLEX. In het tweede model worden de routes van alle treinen die gebruik maken van de infrastructuur behorende bij het station ook bepaald door het model. De rekentijden voor het oplossen van dit laatste MIP model met CPLEX waren voor een aantal stations te groot om het model in de

praktijk te kunnen gebruiken.

Vanwege de grote rekentijden hebben we geprobeerd een beter begrip van het routeringsprobleem te krijgen door een vereenvoudigd model van het probleem te bestuderen. Dit vereenvoudigde model is het job shop scheduling probleem met blokkerings en no-wait precedentie voorwaarden. Het traditionele job shop scheduling problem neemt aan dat er buffers beschikbaar zijn met een oneindige capaciteit om jobs in op te slaan nadat de jobs klaar zijn op de ene machine en nog niet kunnen starten op de volgende machine. Echter, in veel productieprocessen zijn deze tussenliggende buffers niet beschikbaar. De klasse van machine scheduling problemen zonder deze tussenliggende buffers kan gekarakteriseerd worden door blokkerings en no-wait precedentie voorwaarden. In een no-wait omgeving moet een operatie onmiddellijk starten nadat zijn voorgaande operatie gecompleteerd is. Ofwel, een job moet van start tot completering geproduceerd worden zonder onderbreking op of tussen machines. Blokkeringsvoorwaarden treden op als een job gecompleteerd is op een machine en hij op deze machine blijft totdat zijn volgende machine beschikbaar komt. Dit impliceert dat de werkelijke tijd dat een operatie de machine bezet niet van te voren bekend is.

We hebben een aantal branch & bound algoritmen ontwikkeld die no-wait en blokkering job shop problemen efficiënt oplossen. We tonen aan dat het spenderen van tijd aan een sterkere ondergrens voor deze problemen niet leidt tot een verbetering van de totale rekentijd. In hoofdstuk 5 introduceren we een job-invoeging heuristiek die gebruik maakt van een MIP formulering van het probleem en we beschrijven de ontwikkelde branch & bound algoritmen in detail. De rekentijden van de algoritmen worden gepresenteerd voor referentie instanties bekend van de traditionele job shop. Wij zijn de eersten die in staat zijn om referentie instanties van gemiddelde grootte van de job shop met no-wait voorwaarden optimaal op te lossen.

MIP-based Approaches for Complex Planning Problems

Summary

Plans and timetables can be found everywhere in our daily lives. Examples are the Dutch railway timetable, the schedule for the Dutch soccer league and the duty-roster of nurses in hospitals or home care. Together with the increase in computing power, solution techniques for solving such real-world optimization problems improved. Currently, we are able to create excellent real-world timetables and plans. Our goal in this thesis is to solve real-world optimization problems with sophisticated solution approaches based on mixed integer programming (MIP).

For three practical planning problems we investigated whether MIP-based approaches are suitable, looking from a theoretical and practical point of view. For each of the problems an exact or heuristic algorithm has been developed that provides a good or even optimal solution in a computation time that is short enough for practical use. Each of the algorithms is based on a MIP formulation of the problem. If solving it with the general purpose solver CPLEX becomes too time consuming, alternative approaches have been developed.

Two practical problems that we studied are *post enrolment course timetabling* problems that arise at universities. Course timetabling is the weekly scheduling of a set of lectures for a set of university courses within a set of rooms and timeslots, minimizing the overlaps of lectures having common students. In a curriculum-based course timetabling problem conflicts arise according to the curricula published by the university and not on the basis of enrolment data. In a post enrolment course timetabling problem students select the courses that they wish to attend. Section 1.3 discusses where to position these two course timetabling problems in the research environment of educational timetabling.

The third application that we studied is the problem of planning shunting movements at railway stations. Next to all the timetabled passenger and cargo trains, many shunting movements between platform tracks and shunting areas are carried out in and around the large stations in a railway network. Shunting movements have to provide passenger trains with the right composition of rolling stock. The rolling stock for trains that start their duty sometimes has to be picked from the shunt yard and the rolling stock of ending trains sometimes has to be brought to a shunt yard. During rush hours, passenger trains are usually operated at full capacity. However,

outside rush hours an operator of passenger trains usually has a surplus of rolling stock. This surplus has to be parked at a shunting area in order to be able to fully exploit the main railway infrastructure. Section 1.4 describes the shunt planning process at Netherlands Railways and embarks the shunting problem that we consider in this thesis.

A post enrolment course timetabling problem was proposed in the second International Timetabling Competition (ITC2) that was organized in 2007. A set of courses has to be assigned to a timeslot and to a room such that all students are able to attend their requested events while no two courses containing the same student have been assigned to the same timeslot and no two courses have been assigned to the same room. There are also soft constraints that make the timetable "nicer". For a detailed description of this post enrolment course timetabling problem we refer to Chapter 2.

We formulated this post enrolment course timetabling problem as a MIP, but solving it with CPLEX took too much computation time. Therefore, we constructed a deterministic heuristic that assigns events to timeslots based on an LP-solution constructed with column generation. A column corresponds with a feasible assignment of events to timeslots. We get an integer solution by fixing columns one at a time. The generated solution is improved by selecting a set of events that are reassigned by solving a MIP. The comparison of the results of our heuristic with the results of the five finalists of the ITC2, shows that our approach is competitive and even very successful in generating a feasible solution for the harder instances.

A successful solution approach for a post enrolment course timetabling problem of the Department of Industrial Design at the TU Eindhoven is described in Chapter 3. Students of this department are allowed to design part of their curriculum by selecting courses from a huge course pool. They do this by handing in ordered preference lists with their favorite courses for the forthcoming time period. Based on this information and on many other constraints, the department then assigns courses to students. In Chapter 3 we explain all the constraints resulting from the situation in Eindhoven. The problem has been solved using lexicographical optimization with four subproblems. For all four subproblems, an elegant MIP formulation is given that easily can be solved with CPLEX.

Chapter 4 presents two MIP formulations for the routing problem that is part of the shunt planning process. Those MIP formulations try to schedule the necessary shunting movements in between the passenger and cargo trains. Both models minimize the number of shunting movements that can not be planned because of lack of capacity of the infrastructure. In the first model the routes of all trains are fixed beforehand and the constructed MIP is easily solved by CPLEX. In the second model the routes of all trains at a railway station are also determined by the model. The computation times for solving this last MIP model with CPLEX were too large for certain railway stations to be used in practice.

Due to those large computation times we tried to get a better understanding of the routing problem by looking at a simplified model of this problem. This simplified model is the job shop scheduling problem with blocking and no-wait precedence constraints. The traditional job shop scheduling problem assumes that there are buffers with infinite capacity available to hold jobs between the finishing on one machine and

the start of processing on the next machine. However, in many production processes these intermediate buffers are not available. The class of machine scheduling problems without intermediate buffers can be characterized by no-wait and/or blocking precedence constraints. In a no-wait environment an operation has to start immediately after its predecessor operation has been completed. So a job must be processed from start to completion without any interruption either on or between machines. Blocking occurs when a job, whose processing has been completed on a machine, remains on the machine until the successor machine becomes available for processing. This implies that the actual time an operation keeps its machine occupied is not known in advance.

Several branch & bound algorithms have been developed that efficiently solve no-wait or blocking job shop problems. We show that spending time on a strong lower bound does not necessarily result in an improvement of the overall computation time. In Chapter 5 we introduce a job insertion heuristic that uses a MIP formulation of the problem and describe the branch & bound algorithms in detail. The computational results on benchmark instances of the traditional job shop are also presented. We are the first that are able to solve medium sized benchmark instances for the no-wait job shop problem to optimality.

Curriculum vitae

John van den Broek was born in Berghem, the Netherlands on March 27, 1977. In 1996 he completed his secondary school education at the *Titus Brandsma Lyceum* in Oss. At the Technical University of Eindhoven he started his study electrical engineering. After finishing the propaedeutics, he changed to the study of technical mathematics at the same university. For a practical training he worked for three months at SINTEF Industrial Management in Trondheim, Norway, where he worked on a facility location problem under economies of scale. In 2002, he graduated with a Master's thesis on a capacity test for shunting movements. This thesis was written at NS Reizigers, the largest railway operator of the Netherlands.

From March 2003 until August 2004, John has been working at NS Reizigers on the further development of the capacity test for shunting movements that was the central topic of his master's thesis. September 2004 to August 2009 he remained working for one day a week at NS Reizigers continuing his earlier work and taking care of a close cooperation between the logistics department of NS Reizigers and his own research.

September 2004 John started for four days a week as a PhD student at the Combinatorial Optimization group at the Technical University of Eindhoven under the supervision of dr.ir. Cor Hurkens. The project was funded by the Dutch BSIK/BRICKS project, theme Intelligent Systems 3: Decision Support Systems for Logistic Networks and Supply Chain Optimization and by the European Union's research project "Algorithms for Robust and Online Railway Optimization: Improving the Validity and Reliability of Large-Scale Systems" (ARRIVAL). As a Ph.D. student he was a member of BETA (Research School for Operations Management and Logistics) and the LNMB (Dutch Network of the Mathematics of Operations Research). For both he successfully completed all the required Ph.D. level courses. In 2005 he was the Ph.D. representative in the board of the LNMB.