DISSERTATION

Intelligent Search Methods for Workforce Scheduling: New Ideas and Practical Applications

ausgeführt zum Zwecke der Erlangung des akademischen Grades eines Doktors der technischen Wissenschaften/der Naturwissenschaften unter der Leitung von

Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Wolfgang Slany
E184, Institut für Informationssysteme
Technische Universität Wien

eingereicht an der Technischen Universität Wien, Fakultät für Technische Naturwissenschaften und Informatik

von

Dipl.-Ing. Nysret Musliu
Matrikelnummer 9627141
Lorenz Müller Gasse 1a/18
A-1200 Wien

Wien, im September 2001 _____

Në përkujtim të babait dhe gjyshit tim, të cilët u vranë nga forcat policore serbe gjatë luftës në Kosovë më 30.03.1999 para derës së shtëpisë së gjyshit tim, vetëm pse ishin shqiptarë,

dhe

per nënen dhe vëllaun tim të vogël, të cilet pas këtyre ngjarjeve tragjike u detyruan që të largohen nga trojet e tyre ne Kosovë, dhe ishin për disa muaj refugjatë

To the memory of my father and grandfather, who were killed from Serb police forces during the war in Kosova on 30.03.1999 in front of the home of my grandfather, only because they were Albanians,

and

to my mother and youngest brother, who after these tragic events were forced to leave their home in Kosova, and were for several months refugees

Abstract

Intelligent search methods, based on heuristic and intelligent pruning of the search space, have been intensively used for solving difficult real-life problems. Examples of hard problems are problems regarding workforce scheduling. These problems appear in many areas of life, like in industrial plants, hospitals, call centers, public transport etc. and have high practical relevance. Results from ergonomic studies indicate that workforce schedules have a profound impact on the health and satisfaction of employees, as well as on their performance at work. Moreover, workforce schedules have to satisfy numerous legal requirements and meet the specific objectives of the individual workplace.

In this thesis, we use intelligent search methods to investigate the computerized generation of solutions for two specific problems in workforce scheduling: rotating workforce scheduling and shift design problem.

We propose a new framework for generating rotating workforce schedules. One of the basic design decisions is to aim at high-quality schedules for realistically sized problems that can be obtained rather quickly, while, at the same time, maintaining human control. Therefore, interaction between the decision-maker and the algorithm consists of four steps: (1) choosing a set of lengths for the work blocks (a work block is a sequence of consecutive days of work), (2) choosing a particular sequence of blocks of work and days-off blocks amongst those that have optimal weekend characteristics, (3) enumerating possible shift sequences for the chosen work blocks subject to shift change constraints and bounds on sequences of shifts, and (4) assignment of shift sequences to work blocks while fulfilling the staffing requirements. The combination of constraint satisfaction with problem-oriented intelligent backtracking algorithms in each of the four steps can produce good solutions for real-world problems in acceptable time. Apart from the fact that the generated schedules meet all of the hard constraints, the new framework also allows for the incorporation of the preferences of the decision-maker with regard to soft constraints that are otherwise more difficult to assess and model. Computational results from benchmark examples found both in the literature

and in real-world problems confirm the viability of our approach.

Regarding the shift design problem, we consider iterative solution methods. We proposed repair steps (moves) to explore the neighborhood of solutions for this problem. In order to generate the neighborhood and accept the solution for the next iteration, basic principles of the so called tabu search technique are used. However, while tabu search finds acceptable solutions for this problem, the complete exploration of the neighborhood (using all defined moves) during each iteration is very time-consuming. We proposed a new approach in combination with tabu search in order to make the search more effective. The basic idea is to exploit knowledge about the shift design problem during the search. Based on the temporal workforce of the current solution and given workforce requirements (distance of the current solution to the optimal solution with respect to most important criteria), selecting repair steps during each iteration is guided so that the repair steps that have a greater chance to improve the solution are used. As a result of this knowledge about the problem, the neighborhood is selectively explored during each iteration, which makes the search more effective. Furthermore, we propose an algorithm for generating a good initial solution, which further improves the effectiveness of the search.

Finally, we included algorithms in the commercial products that allows to take into account user preferences in the decision-making process. This new framework for rotating workforce schedules has been included in a commercial product called First Class Schedule, which is already being used by several companies in Europe since 2000. This product is highly appreciated internationally and currently exists in German, English, Finnish and Dutch. Guided local search was included in the commercial product Operating Hours Assistant, which has also been used successfully in several organizations since the beginning of the year 2001.

Kurzfassung

Intelligente Suchmethoden, die auf Heuristiken und intelligenter Beschneidung des Suchraumes basiert sind, sind intensiv verwendet worden, um schwierige praktische Probleme zu lösen. Beispiele von solchen Problemen sind Schichtplanungsprobleme. Das Finden von Schichtpläne ist überall dort eine kritische Aufgabe, wo eine bestimmte Mitarbeiteranzahl (die sogenannte Besetzungsstärke) garantiert werden muss, z.B. in Hochöfen, Spitälern oder bei Fluggesellschaften. Ergebnisse aus der Arbeitswissenschaft zeigen, dass die Qualität der Schichtpläne einen großen Einfluss sowohl auf die Gesundheit und Zufriedenheit der Mitarbeiter als auch auf ihre Arbeitsleistung hat. Schichtpläne müssen außerdem die Arbeitszeitgesetze erfüllen und mit den wirtschaftlichen Anforderungen des Betriebes in Einklang stehen.

Thema dieser Dissertation ist die Entwicklung von Algorithmen zur automatischen Erzeugung zyklischer Schichtpläne und Generierung von Schichten basierend auf intelligenten Suchverfahren. Zu den zwei wichtigsten Zielen zählen dabei die Qualität der Lösungen - sie sollen zumindest so gut sein, wie von professionellen Planern erstellte - sowie die Zeitersparnis bei der Erstellung. Weiters sollte der Benutzer die Möglichkeit haben, Anforderungen einfließen zu lassen, die sich nur schwer explizit modellieren lassen oder die von individuellen Präferenzen abhängen.

In dieser Dissertation wurde ein neuer Lösungsansatz zur Erzeugung von zyklischen Schichtplänen entwickelt. Der Lösungsansatz besteht aus den folgenden Schritten: (1) Auswahl von Mengen von Arbeitsblöcken mit verschiedenen Längen, (2) Entscheidung über Anzahl der freien Wochenenden und die Reihenfolge der Arbeitsblöcke, (3) Erzeugung von Schichtfolgen, (4) Erzeugung von Schichtplänen (die erzeugten Schichtfolgen werden den Arbeitsblöcken zugewiesen). Für jeden Schritt von 1-4 implementierten wir problemorientierte, intelligente backtracking Algorithmen, die sehr effizient sind, da sie den Suchraum durch die beschriebene Zerlegung der Einschränkungen weitestgehend reduzieren können. Ausgiebige Tests und Benchmarkergebnisse bestätigen

die Zweckmäßigkeit unseres Ansatzes. Die Erzeugung hochwertiger zyklischer Schichtpläne mittels dieses Lösungsansatzes erfolgt für die meisten Problemstellungen innerhalb weniger Sekunden. Ein weiterer Vorteil von unsere Ansatz ist die Möglichkeit, Pläne in Kooperation mit dem Benutzer zu erzeugen. Selbstverständlich erfüllen die erzeugten Pläne alle angegebenen Einschränkungen. Darüber hinaus erlaubt die Interaktion mit dem Planer aber auch die Berücksichtigung anwendungsspezifischer Einschränkungen und Präferenzen, die sonst nur schwierig zu erfassen und zu modellieren wären.

Die Erzeugung von Schichten wird basiert auf iterativen Suchmethoden. Es wurden Reparatur basierte Schritte ('moves') vorgeschlagen, um die Umgebung der Lösung zu erkunden. Um diese Umgebubg zu generieren und die Lösung für die nächste iteration zu selektieren, wurden grundlegende Prinzipien der Tabu Suche verwendet. Während jedoch die Tabu Suche akzeptable Lösungen für dieses Problem anbietet, ist die vollständige Erkundung der Umgebung (unter Verwendung alle definierten 'moves') während jeder Iteration sehr zeitraubend. Um die Suche effektiver zu machen wurde ein neuer Lösungsansatz in Verbindung mit Tabu Suche vorgeschlagen. Die grundlegende Idee ist es, Wissen über das Problem während der Suche zu verwenden. Basierend auf der Belegschaft der aktuelle Lösung und der gegebenen Besetzungsstärke (Distanz der aktuellen Lösung von der optimalen Lösung hinsichtlich der meisten wichtigen Kriterien), werden während jeder Iteration die 'moves' selektiert, die eine größere Chance haben, die Lösung zu verbessern. Infolge dieses Wissens über das Problem wird die Umgebung der Lösung während jeder Iteration nicht vollständig generiert, was die Suche effektiver macht. Außerdem wurde ein Algorithmus vorgeschlagen, der eine gute anfängliche Lösung erzeugt, welche die Effizienz der Suche weiter verbessert.

Der Lösungsansatz zur Erzeugung von zyklische Schichtplänen wurde in dem Softwarepaket First Class Scheduler (FCS) implementiert. FCS ist eine Erweiterung des Programms Schichtplanassistent der XIMES GmbH, welches in früheren Versionen bereits eine effiziente Manipulation und Überprüfung von manuell erstellten Schichtplänen erlaubte. Mit FCS können die wichtigsten Einschränkungen im europäischen Raum modelliert werden. Das Softwarepaket ist in Schichtplanungskreisen international hoch angesehen und liegt derzeit in den Sprachen Deutsch, Englisch, Finnisch und Niederländisch vor. Die Lösungsansätze für Generierung von Schichten wurde in dem Softwarepaket Operating Hours Assistant implementiert. Dieses Softwarepaket wird seit Anfang 2001 von einigen Organisationen erfolgreich verwendet.

Acknowledgments

First and foremost I would like to thank my family for their continuous support all my life long. I got infinite love from my parents. My father sacrificed everything, despite the very hard situation in Kosova, to make it possible for me to continue my studies. He was my teacher of physics and chemistry in primary school and gave me a lot of insight into problem solving. His support was always present for me and he will always remain my role model in my life. I lost him and my grandfather during the war in Kosova, when I was in the middle of work on this thesis, and that was very hard to face. My mother was always here for me to give me support in the hardest situations. I am amazed by her force to deal with the most hard situations of life. Without her love and support this thesis would never have been finished. My younger brothers Xhevdet and Valdet were my inspiration and one more reason to be strong and continue the work on this thesis after I lost my father and grandfather. I would like to thank also my grandparents for their support and for beautiful moments of my life I had with them.

Special thanks go to my supervisor Wolfgang Slany. Discussions with him, his useful suggestions, and his readiness to help me in any situation were very important for successfully finishing this thesis. It was a pleasure to work with him in project Rota and writing papers together. I am also grateful to him and to his wife Kyoko for the moral support they gave me and my family in times when tragic events happened.

My thanks also go to Prof. Georg Gottlob for giving me the opportunity to work in the DBAI group and for his moral support throughout this thesis. It was very nice to work in a good working atmosphere with all colleagues in the DBAI group and to have for a long time Robert Baumgartner as my room-follow. Thanks to Jochen Renz and Christoph Koch for proof reading parts of my thesis.

I would like to thank Johannes Gärtner for being my second advisor and for his ideas and discussions with him about the generation of rotating schedules and throughout the work in the project. I thank Sabine Wahl for the perfect joint work on the commercial software products in which the algorithms described in

this thesis were included. It was very nice to work part of the time at Ximes Corp. I would like to thank all colleagues at Ximes who gave support to me and my family in a time when we needed it most.

Special thanks go to my friend Astrit Ademaj, who was for two years my room-follow in the dormitory and gave me great support in the hardest moments of my life. I would like also to thank all my friends and especially Driton Statovci, Idriz Smaili, Assumpció López Polo and Melchor Moro Oliveres for the support they gave to me in that time. I thank Yll Haxhimusa for reading parts of this thesis.

I am grateful to my relatives who gave me in the first phase of this thesis financial and any kind of other support throughout this thesis: Ismail Musliu, Hamdi Musliu, Ali Beka, Mustafe Musliu, Xhafer Musliu, Liman Beka, Sylejman Beka, Osman Beka, Ruzhdi Musliu, Muhamet Musliu, Selvete Musliu, Sadije Gerbeshi, Zarife Musliu, Nazmije Musliu, and Bahrije Beka.

I wish to thank also Elisabeth Hager, Mimoza Vladi and Douglas Linton for proofreading parts of this thesis.

My research was partially supported by FFF project No. 801160/5979, the Austrian Science Fund Project N Z29-INF, and the Austrian government. I am grateful for that.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

In this thesis, we consider practical approaches for computerized workforce scheduling. In general, workforce scheduling includes many problems that deal with planning and scheduling of a workforce in an organization. These problems appear in different forms in a broad range of workplaces, such as hospitals, industrial plants, call centers, public transportation, and airline companies. Results from ergonomics [11] indicate that workforce schedules affect both the health and the social life of the employees, and can also increase the risk of work-related accidents. Moreover, workforce schedules also impact workforce related costs for the organization. Therefore, it is of a high practical relevance to find workforce schedules that, on the one hand, fulfill the ergonomic criteria for the employees, and, on the other, reduce costs for the organization. However, most workforce scheduling problems are difficult to solve. Most relevant problems in workforce scheduling belong to the class of problems called NP-hard problems. No efficient algorithms are known for these problems and an exact solution, in general cases, is generally not believed to be possible. Despite these hurdles, researchers have developed over the years a number of alternative methods that make it possible to find an acceptable solution to these problems in a reasonable amount of time. One group of such methods is loosely classified as intelligent search methods. This group includes methods that determine the problem's exact solution (for relevant large instances of problem in practice) by intelligently pruning the regions of the search space that cannot contain good solutions. Furthermore, in this group are also included heuristic search methods.

In this thesis, in order to solve the workforce scheduling problems, we rely on such intelligent search methods. A brief review of intelligent search methods

is given in Chapter 2.

1.2 Workforce scheduling problems

The typical process for planning and scheduling of a workforce in an organization consists of several stages. Usually the first stage is to determine the temporal requirements. To do this, the required number of employees with certain qualifications is determined for every time slot of the planning period (e.g., every Monday between 6:00-10:00 there should be 4 employees at work). After this stage, one can proceed to determine the total number of the employees needed, while at the same time designing the shifts, and then assigning these shifts or days-off to the employees. The literature provides different approaches for finding solutions at this stage. One approach is to coordinate the design of the shifts with the assigning of these shifts to employees. This solves both problems as if they were a single one [29]. Other approaches consider days-off scheduling and shift scheduling only after the shifts are designed [8, 46]. One of the disadvantages of the second approach is that considering the design of the shifts separately does not guarantee that a feasible solution for the assignment of the shifts can be found. In contrast, solving the workforce-scheduling problem in several separate stages makes the problem of general workforce scheduling easier to tackle, one of the reasons why this approach is used frequently in practice.

In this thesis, we consider separately the problem of the designing the shifts, and the problem of assigning employees to particular shifts or days-off for rotating schedules. A detailed description of both problems is given in Chapter 3.

For the problem of shift design, we are given the workforce requirements for a certain period of time, constraints about the possible start and the length of shifts, and an upper limit for the average number of duties per week per employee. The aim is to generate solutions that contain shifts (and the number of employees per shift) that fulfill all hard constraints about the shifts, as well as minimize the number of shifts, over- and understaffing, and differences in the average number of duties per week.

After the shifts are generated, the assignment of employees to shifts or days-off for a given period of time can be made. There are two main variants of workforce schedules: rotating (or cyclic) workforce schedules and non-cyclic workforce schedules. In a rotating workforce schedule — at least during the planning stage — all employees have the same basic schedule but start with different offsets. Therefore, while the individual preferences of the employees cannot be taken into account, the aim is to find a schedule that on average is optimal for all employees. In non-cyclic workforce schedules the individual preferences of the employees can

be taken into consideration and the aim is to achieve schedules that fulfill the preferences of most employees. In both variations of workforce schedules other constraints such as the minimum number of employees required for each shift have to be met. In this thesis we will consider the problem of rotating workforce schedules.

1.3 Computerized workforce scheduling

Both problems of shift design and assignment of employees to shifts and days-off are NP-complete [41, 46]. For this reason, they are generally difficult to solve, which corresponds to the extremly large search space and conflicting constraints that are usually encountered. All this reinforces the importance of finding a way to generate practicable schedules in a reasonable amount of time. Regarding the problems we consider in this thesis, experienced professional planners can construct an acceptable solution for many of them by hand. However, the time required to do this can sometimes be very long. For example, rotating workforce schedules can take anywhere from one hour to several days. In addition, because of the large number of possible solutions, the human planners can never be sure whether their solution is the best one. Because of the complexity of the problem and the relatively high number of constraints that must be satisfied and, in case of soft-constraints, optimized, generating shifts and a schedule without the help of the computer in a short period of time is almost impossible — even for relatively minor instances of this problem. Therefore, computerized workforce scheduling has been the focus of interest for researchers for over 30 years.

Different approaches were used to solve problems of workforce scheduling. Examples for the use of exhaustive enumeration are [33] and [15]. Glover and McMillan [29] rely on the integration of techniques from management sciences and artificial intelligence to solve general shift scheduling problems. Balakrishnan and Wong [8] solve a problem of rotating workforce scheduling by modeling it as a network flow problem. Smith and Bennett [63] combine constraint satisfaction and local improvement algorithms to develop schedules for anesthetists. Schaerf and Meisels [61] proposed generalized local search for employee timetable problems. A wide review of previous work on workforce scheduling is given in the Chapter 3.

The critical features of workforce scheduling algorithms are their computational behavior and flexibility for solving a wide range of problems that appear in practice. Recently Laporte [43] assessed rotating workforce scheduling algorithms proposed in the literature saying that "[...] these are often too constraining and not sufficiently flexible for this type of problem".

1.4 Research questions of this thesis

This thesis results from the solving of problems that emerged directly from practice. The aim was to develop algorithms that will automate the process of generating rotating workforce schedules and designing shifts for the Ximes Corp, which specializes in developing software and in consulting on work-hour arrangements. The efficiency of the algorithms and the quality of their solutions are the two most important criteria. In the consulting process, it is essential that a solution be produced in a very short time, so that it can be discussed further. The algorithms should be included in systems that can be employed directly in the consulting process. They should also be able to stand alone, so that they can be easily used in different organizations and by individuals who are not specialized in the process of generating the workforce schedules. This implies that the packages should be flexible, thus making possible a relaxing of the constraints. It should also allow for the inclusion of the user in the decision-making process, especially with regard to the constraints, which in different situations may have different levels of importance.

Before proceeding further, we will summarize the main objectives of this thesis:

- Develop efficient algorithms for generating high quality rotating workforce schedules. The algorithms should solve most real cases in a reasonable amount of time (from a few seconds to a few minutes).
- Develop efficient algorithms for the shift design problem. The algorithms should be able to find good solutions in most real cases in a reasonable amount of the time.
- Include the algorithms in the commercial products First Class Scheduler and Operating Hours Assistant. The systems should include the appropriate interaction in relaxing the constraints, as well as allowing the user to choose between a number of solutions, which fulfill different criteria that might vary in importance depending on the situation.

1.5 Main results

Here we summarize the main results of this thesis:

- We propose and implement a new framework for computerized rotating workforce scheduling, including intelligent backtracking algorithms for each stage of the framework.

The new framework consist of the following steps: (1) choosing a set of lengths for the work blocks (a work block is a sequence of consecutive days of work), (2) choosing a particular sequence of blocks of work and days-off blocks amongst those that have optimal weekend characteristics, (3) enumerating possible shift sequences for the chosen work blocks subject to shift change constraints and bounds on sequences of shifts, and (4) assignment of shift sequences to work blocks while fulfilling the staffing requirements. The combination of constraint satisfaction with problem-oriented intelligent backtracking algorithms in each of the four steps can produce good solutions for real-world problems in acceptable time. Apart from the fact that the generated schedules meet all of the hard constraints, the new framework also allows for the incorporation of the preferences of the decision-maker with regard to soft constraints that are otherwise more difficult to assess and model. Computational results from benchmark examples found both in the literature and in real-world problems confirm the viability of our approach.

- We propose and implement a local search technique based on the tabu search principles for the shift design problem.

  The appropriate neighborhood structure for local search is one of the most important features to reach 'good' solution. In this thesis we propose moves (repair steps) that are used to explore the neighborhood. We experimented with different neighborhoods over practical and randomly generated examples. In order to accept the solution for next iteration, basic principles of the tabu search technique are used, where we experimented with two variants of making solutions tabu. Furthermore, we experimented with different initial solution and proposed an algorithm for generation of a good initial solution, which improves the effectiveness of the search. The local search based technique gives good results in real life examples and randomly generated examples, which were generated with a random generator we proposed for the shift design problem.

- We propose and implement a method, which by exploiting knowledge about the shift design problem guides the selection of the neighborhood during each iteration in the local search.

  The complete exploration of the neighborhood using all defined

moves during each iteration is very time consuming for the shift design problem. We proposed a new approach in combination with tabu search to make the search more effective. Based on the temporal workforce of the current solution and given workforce requirements (distance of the current solution from the optimal solution with respect to the most important criteria), selecting the repair steps during each iteration is guided so that the repair steps that will most likely improve the solution are used. As a result of this knowledge about the problem, the neighborhood is selectively explored during each iteration, which makes the search more effective. Computational results in real life examples and randomly generated problems show the advantages that this ingredient gives to the tabu search technique.

- We include algorithms and other parts that allow taking into account the user in the decision-making process in commercial products.

  This new framework for rotating workforce schedules has been included in a commercial product called First Class Scheduler, which is already being used by several companies in Europe since 2000. This product is highly appreciated internationally and currently exists in German, English, Finnish and Dutch. The guided local search was included in the commercial product Operating Hours Assistant, which has also been used successfully in several organizations since the beginning of year 2001.

Some of the results of this thesis have already been accepted for publication in the journals "Discrete Applied Mathematics" and "Artificial Intelligence Communications" and have appeared in the proceeding of three international conferences.

1.6 Organization of this thesis

This thesis is divided into six chapters. Following this chapter, in Chapter 2, we give a brief review of intelligent search methods, with emphasis on the methods used in this thesis. Chapter 3 introduces workforce scheduling in general and the problems we consider here in detail. This chapter also includes a review of previous efforts made in solving problems related to workforce scheduling. In Chapter 4 a new framework for rotating workforce scheduling and computational results for real life examples is presented. Chapter 5 proposes a solution for the

shift design problem. It also presents a neighborhood search based on tabu search, a guided neighborhood search, and the computational results for both real-life and randomly generated examples. Chapter 6 briefly introduces commercial products in which our algorithms have been included. Chapter 7 concludes and describes work that remains to be done.

Chapter 2

A Brief Review of Intelligent Search Methods

Rapid development of computer sciences has made possible modeling and solving very complex real world problems. Real world problems are often very hard to modulate, heavily constrained and with a large search space, that is a large number of possible solutions. To solve these problems in a reasonable amount of time, researchers have developed many problem-solving techniques. One way to solve problems is by searching. In this Chapter we give a brief overview of intelligent search techniques which have been intensively used for solving hard real world problems.

2.1 Search and problem solving

Search is one of the typical strategies in computer sciences to solve problems that have a so-called combinatorial nature. Some problems with high practical relevance solved by search are given below:

- Route finding: Vehicle routing concerns with finding the routing of vehicles to pick up or supply the costumer with goods in different locations, subject to different constraints like: time restrictions, costs of the routes etc. Routing is also very important in computer networks, bus routing, airline travel etc.
- VLSI design: Designing chips with a million gates is a very complex task that includes many complicated problems like placement of the cells and

routing decision. Search is used to minimize the area of the chip, length of wiring etc.

- Scheduling and sequencing: Such problems appear for example in manufacturing systems where a determined number of jobs should be completed on a set of machines. The task is to determine the order of jobs on machines, such that the completion time of the last job is minimized. Another important application is automatic assembly of complex objects by a robot. Other examples include workforce scheduling, course scheduling, university timetabling, sport events timetabling, scheduling for space missions etc.

- Packing and loading problems: Such problems include for example cutting of rectangular pieces in wood industry from standardized stocks, cutting from the rolls in the textile or paper industry, optimal loading of the items in the specified area etc.

- Robot navigation.

- Applications in medicine: Such problems appear in systems for medical diagnosis, cancer treatment etc.

- Applications in telecommunications: Example of such problems are the optimization of communication network planning, code design which make possible fast and reliable transmitting of information, etc.

- Applications in molecular biology like mapping of DNA sequences.

Further we introduce basic concepts of search by one example.
Problem 1: There is given a table with dimensions $n \times m$ (assuming in he following example that $n = 4$ and $m = 7$). Each of the elements of the table should be filled with one of the symbols M,A,N,-, such that the following constraints are satisfied:

1. In each column there should appear each symbol exactly one time (in general case this constraint can specify a different frequency of appearance for each symbol)

2. Two adjacent elements of the table are not allowed to be assigned in the following sequences: "N M","A M", "N M". Adjacent elements in a row are considered to be consecutive elements. Additionally, the last element of the row is considered adjacent to the first element of the next row , and the next row for the last row is the first row of the table

Table 2.1: A possible feasible solution for example 1

M	M	N	N	-	-	-
A	A	A	A	N	N	N
-	-	M	M	A	A	A
N	N	-	-	M	M	M

3. Blocks of adjacent elements assigned with one of the symbols M,A,N are not allowed to be shorter than 4 and longer than 7.

4. Blocks of elements assigned only with symbol "-" should have length of from 2 to 4

This is a simplified version of a practical problem that will be considered in this thesis.

How shall we find the solution for this problem? First let us make an observation about the number of possible assignments to this table. The number of possible solutions (feasible and infeasible ones) for the search problem is called *search space* of the problem. Let us see how large is the search space for this problem. The table has $4 \times 7 = 28$ elements. As for each of the elements there exist 4 possibilities (four symbols), the space of the possible solutions is: $4^{28} = 7,2 \times 10^{16}$. In this problem we are interested on finding the first solution which fulfills all constraints. For example in Table 2.1 is shown one of the solutions that fulfills all constraints. An algorithm used to find the solution among all possible solutions is called a search algorithm. Sometimes is not only required to find the first solution that fulfills some criteria but also to find the best solution subject to one ore more criteria which can be fulfilled to a specific degree. In this case the problem is called an *optimization problem* and the aim is to find the *optimal* solution.

A very simple algorithm, so that a solution for Problem 1 is found, would be to enumerate all possible solutions and to test each of them whether it fulfills constraints or not until we find the first one. However, this method is not effective. Even for small instances of problems it is impossible to enumerate all possible solutions in a reasonable amount of time. In order that those problems that have a very large search space are solved in a reasonable amount of time, alternative methods have been developed. Those methods do not always guarantee finding an optimal solution, but produce, most of the time, acceptable solutions in practice. In this thesis, we rely in techniques which are loosely classified as intelligent search techniques [58] and part of them as modern heuristic techniques

[51]. We will not consider here some of typical operational research techniques like linear programming, because we focus in this thesis in intelligent search techniques. Before we describe AI search techniques let us say some words about the complexity of the problems.

2.2 Complexity of problems

Complexity theory is a field of computer science which studies the complexity of problems based on the characteristics of theoretically best algorithms that solve them. A problem which can be solved with algorithms, which have similar time or space requirements belongs to class of problems. Two complexity classes are the class P and NP. These two complexity classes include many relevant practical problems.

One problem belongs to class P if it can be solved with an efficient (polynomial) algorithm. The algorithm is polynomial if its execution time grows (rate of growth $O()$) according to a polynomial function compared to the size of the description of the problem's instance. One of the algorithms for sorting called selection sort, for example, is a polynomial algorithm as this algorithm has the rate of growth $O(n^2)$, where n represents the number of the items that should be sorted.

Another complexity class called NP (non-deterministic polynomial) includes many problems that have high practical relevance. The class P is a subset of NP, but the class NP include problems which are called hard or intractable because for these problems no efficient algorithms are known. One of the open questions in computer science is the whether the class P is equivalent to NP.

Problems in NP can be described as decision problems which have one answer "yes" or "no". Suppose that we can guess a solution, which has the answer "yes" to the problem. If we can check in polynomial time that the solution is right then the problem is NP. If every problem in NP can be transformed to a particular problem in polynomial time the problem is called NP-hard. If an NP-hard problem belongs itself to the class NP that problem is called NP complete. Stephen Cook [18] was first (1971) to prove that the problem called 'satisfiability' is NP complete. Karp (1972) [38] presented other problems that are NP complete and during the last 30 years a lot of work has been done in NP completeness theory. A list of many NP complete problems can be found in Garey's and Johnson's book [21] which provides a detailed introduction to the theory of NP completeness.

Many practical problems, including the problems from workforce scheduling

Table 2.2: Comparison of polynomial growth with exponential growth

rate of growth	n=3	n=10	n=20	n=30	n=40
n^2	9	100	400	900	1600
4^n	64	1048576	1099511627776	$1,15 * 10^{18}$	$1,2 * 10^{24}$

that we will consider in this thesis, are NP complete problems. For these problems no such efficient algorithms are known, but nobody has also proved that they do not exist for such problems. Solving one of these problems efficiently would also mean solving all NP complete problems efficiently, as every NP complete problem could then be transformed in polynomial time to a problem for which efficient algorithms would exist.

The general case of Problem 1 is an NP-Complete problem and no efficient algorithm for it is known. Enumeration of all solutions for this problem (exhaustive search) requires an exponential algorithm with a rate of growth $O(4^{n \times m})$. Suppose that the m remains fix and only n changes in value. In Table 2.2 the difference of the rate of growth for this algorithms and selection sort which has growth rate $O(n^2)$ is given for different values of n.

2.3 Search techniques in AI

Although for NP complete problems in general no efficient algorithms exist, researchers have developed over the years a number of alternative methods that make it possible to find an acceptable solution to these problems in a reasonable amount of time. We give a brief review of some of the intelligent search methods in the following.

2.3.1 Exhaustive search

Exhaustive search is a procedure that guarantees finding the best solution, but it is a very ineffective procedure. With this method, all the possible solutions in the search space are systematically enumerated until the best solution is found. In case the evaluation of the best solution is known, the search procedures stop as soon as the best solution is found, otherwise all possible solutions must be examined and the best one is identified at the very end. As the search space of most practical problems is very large, use of exhaustive search is mostly not practical. However, analyzing the procedures for enumerating all solutions is

important, because other more effective methods are based on these procedures. Further we give a brief description of two such procedures called breadth-first search and depth-first search. These two strategies are called blind strategies because they do not have the information about the past cost to the goal solution. In order to describe these procedures, the search process is illustrated with a search tree.

Breadth-first search

In the breadth-first search procedure, first, all nodes of the first level in the tree are visited, then all nodes of level 2 and so on, until the last level when the best (goal) solution is found if one exists. In Figure 2.1 the breadth-first search procedure for Problem 1 is illustrated. First level of the tree represents all possibilities for the element (1,1) of the table. Like described in the problem, these possibilities are the symbols M,A,N,- . For each of the nodes in level one there exist four other possibilities in the element (1,2) of the table. In this manner the tree is expanded until the whole table is filled. As the table has 28 elements the tree will have 28 levels. The solution that fulfills all constraints can be found only in the last level, when all elements of the table are filled.

The breadth-first strategy guarantees that the best solution will be found (in our case the solution that fulfills all constraints). However, space requirements of this strategy are very large, as all nodes should be stored in the memory. In our case, each node has 4 branches (branching factor is 4) and the tree has 28 levels (depth of the tree is 28). Thus the minimum number of nodes that must be stored in memory is 4^{27}. In the general case, if there is a tree with branching factor b and depth d the minimum number of node that must be memorized is b^{d-1} which is bounded by $O(b^d)$, an exponential growth rate in d. Even for very small problems, the amount of memory required by this procedure exceeds practical possibilities.

Another variant of breadth-first search is bidirectional search. The basic idea there is to search from the initial state and the goal state until two searches meet [60].

Depth-first search

In the depth-first search procedure, the left most node in each level of the tree is visited until the point in which it is concluded that the found path in the tree is not the best solution. At this point the procedure backtracks for one level to visit other nodes. In Figure 2.2 the depth-first search procedure for the Problem 1 is

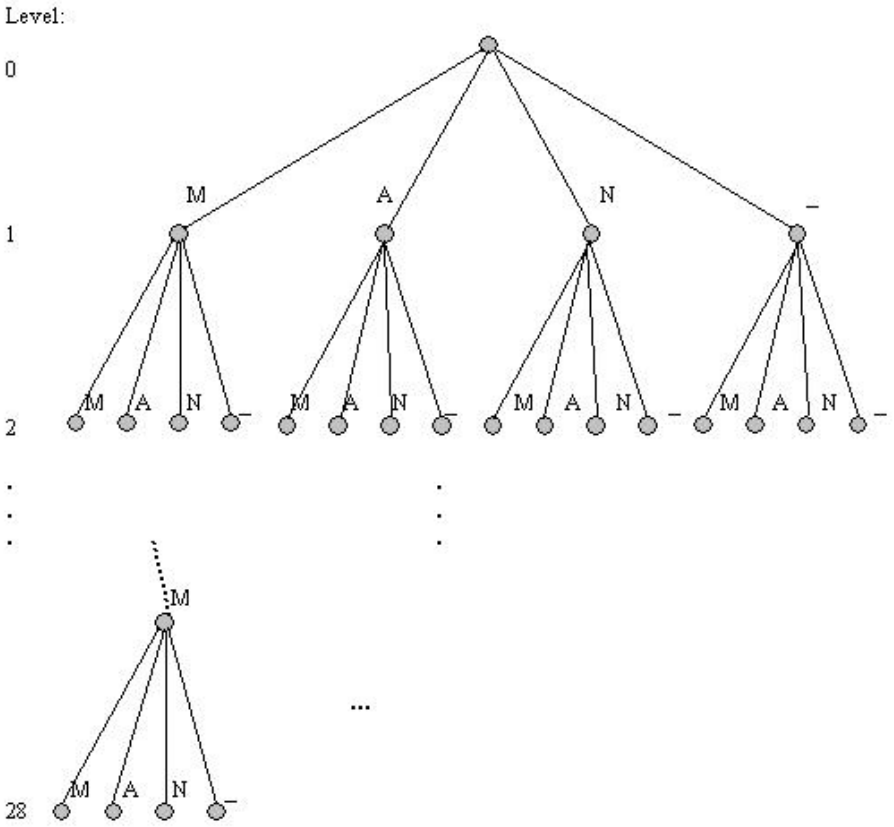


Figure 2.1: Breadth-first search tree for Problem 1

illustrated. First, the first node that consist in the first possibility for element $(1, 1)$ of the matrix (in our case M) is visited, after that the second element of the table is filled and so on, until the first path is reached, which unfortunately does not represent the desired solution. As the path does not represent the solution that fulfills all constraints, backtracking done, and the next node to the right is visited (A). When the 4 nodes are visited in the bottom but the solution is not found, the depth-first search procedure backtracks two levels higher to visit other nodes, and the visiting of nodes is done recursively.

The depth-first strategy guarantees that the best solution will be found (in our case the solution that fulfills all constraints) if the tree is not infinite. Space requirements of this strategy are not large, only d (depth of tree) nodes have to be memorized. However, the time complexity of the depth-first search is also

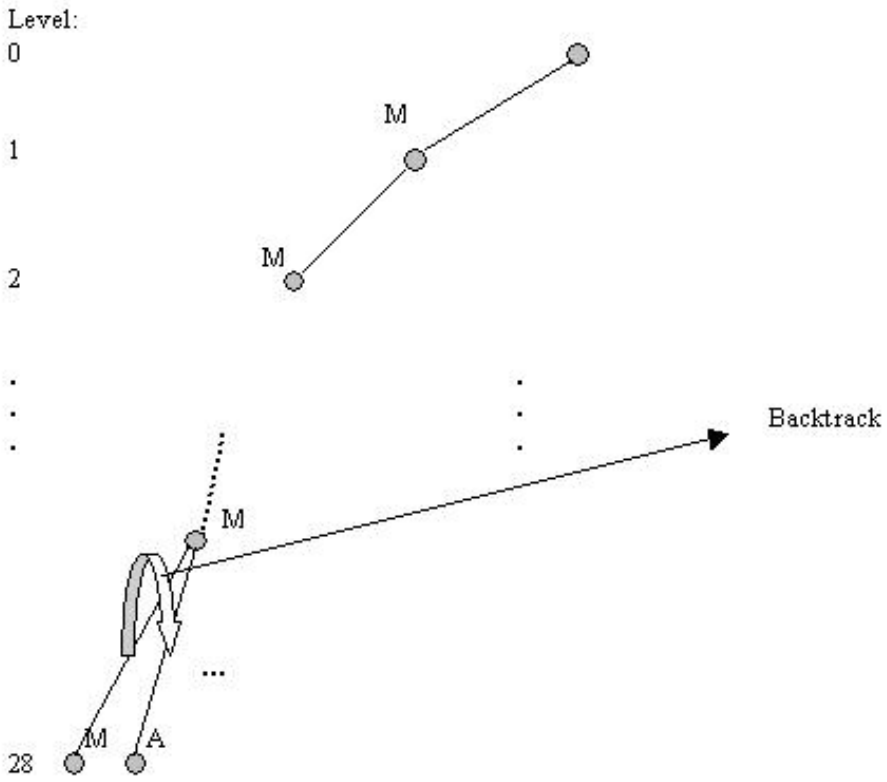


Figure 2.2: Depth-first search tree for Problem 1

$O(b^d)$. The disadvantage of the depth-first search is that it can get stuck in the tree and take a wrong path (direction).

2.3.2 Heuristic based search

Breadth-first search and depth-first search visit systematically the nodes of the tree, and are very impractical even for small instances of problems. In artificial intelligence (AI) methods based on so-called 'heuristic' search have been developed. The word heuristic is derived from the Greek verb 'heuriskein' which means 'to find' or 'to discover'. Newell, Shaw, and Simon (1963) describe heuristics as 'a process that may solve a given problem, but offers no guarantees of doing so'.

Another definition given by Reeves [59] defines heuristics as 'a technique, which seeks good (i.e., near optimal) solutions at a reasonable computational cost without being able to guarantee either feasibility or optimality, or even in many cases to state how close to optimality a particular feasible solution is.'

Search heuristics have been used intensively in AI. The basic idea is to expand not all but only a subset of nodes in the search selected by heuristic. One of the simplest variants of the so called best-first search is based on the heuristic function h, where $h(n)$ represents estimated cost of the cheapest path from the state at node n to the goal state. We use notation from [60] and this approach is called Greedy search.

An extension of best-first search is the search technique called A^* where the evaluation function $f(n)$ is a combination of the heuristic function $h(n)$ and the function $g(n)$ which represents the cost of the path found so far: $f(n) = g(n) + h(n)$. The function h is selected such that it guarantees the so called 'admissibility'. An algorithm is admissible if it always terminates at the optimum solution [51]. For detailed description of the $A*$ search strategy and its extensions we refer the reader to [60], [51], [13].

Heuristic search has been also used extensively in AI for game playing Shannon [62].

2.3.3 Search for constraint satisfaction problems

Many practical problems like routing, scheduling, timetabling, etc. can be defined as constraint satisfaction problems (CSP). Formally, a constraint satisfaction problem is described with a set of n variables $x_1, x_2, \ldots, x_n$, sets of domain values which the variables can take $D(x_1), D(x_2), \ldots, D(x_n)$, and set a of constraints $c_1, c_2, \ldots, c_n$ over the variables $x_1, x_2, \ldots, x_n$. For example, Problem 1 is a constraint satisfaction problem where the variables are elements of the table and domain of each variable are values M,A,N,-. Constraint 2 for example is a binary constraint as it is defined between any of two adjacent elements of the table. This constraint is satisfied if the adjacent elements take only values for which the constraint is specified. The domain of the variables in CSP can be finite or infinite. In Problem 1 the variables have finite domains, as they can take only one of the four values. Constraints are also often divided in to hard and soft constraints. In case of hard constraints, the constraints must be satisfied, and in the case of soft constraints the aim is to satisfy the constraint as good as possible. They are so called optimization problems. For detailed description of constraint satisfaction problems we refer the reader to [68] and [48].

Backtracking search

In the case of CSPs, search space can be significantly reduced by pruning of part of the search space where some of the constraints cannot be satisfied. The basic idea is to test before expanding every node in the search tree if any of constraints are violated. In that case, the part of the tree will not be searched, as it is sure that a feasible solution cannot be found. The search technique that prunes the search space in this manner and is based on depth-first search, is called backtracking search. This strategy has been used for many practical problems and is a way of enumerating all solutions in a much more effective way than basic depth-first search. One way to make backtracking more effective is forward checking [60].

Further we illustrate the backtracking strategy for the Problem 1 in Table 2.3. Column 1 of the table is represented by variable x_1, column 2 with x_2 and so one until the last column with variable x_{28}. In Table 2.3 first 12 steps of backtracking algorithms are presented. The letter U is used to denote that the variable is unsigned. In the first step, backtracking search begins with giving first possible value for the first variable. Every time the variable is assigned, a test for violation of constraints is done. We show in braces which constraints are violated when new variable is assigned with the value. If this is the case, the search backtracks, which means that the variable should be assigned other value or backtracking to the previous step should be done.

Heuristics for constraint satisfaction problems

One way to solve larger instance of CSPs is by using heuristics, which can be used to determine the order of variables to be assigned and selecting values for the variables. Most-constrained variable heuristics [12] are used with forward checking. With this technique, the order of assignment of variables is determined and the variable with the fewest possible value is selected to be assigned. Other heuristics used are the least-constraining-value heuristic [32] where the value of variable is selected such that it rules out the smallest number of values in the variables which are connected to the current variable by constraints.

Other methods used to solve CSPs is the iterative improvement technique. With this method, initially all variables are assigned with a value and the solution is repaired iteratively from the inconsistencies (constraint violations), by changing iteratively the values of variables. Different type of heuristics can be used in choosing the new value of variables. One of the heuristics that has proved to be very effective in solving CSPs is the min-conflict heuristic [52]. With this heuristic, in each iteration, the value of the variable that minimizes the number

Table 2.3: First 12 steps of backtracking algorithm for problem 1

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	...	x_{28}
Step 1:	M	U	U	U	U	U	U	U	U	U	...	U
Step 2:	M	M	U	U	U	U	U	U	U	U	...	U
Step 3:	M	M	M	U	U	U	U	U	U	U	...	U
Step 4:	M	M	M	M	U	U	U	U	U	U	...	U
Step 5:	M	M	M	M	M	U	U	U	U	U	...	U
Step 6:	M	M	M	M	M	M	U	U	U	U	...	U
Step 7:	M	M	M	M	M	M	M	U	U	U	...	U
Step 8:	M	M	M	M	M	M	M	M	U	U	...	U
backtrack: (c1,c3)												
Step 9:	M	M	M	M	M	M	M	A	U	U	...	U
backtrack: (c3)												
Step 10:	M	M	M	M	M	M	M	N	U	U	...	U
backtrack: (c3)												
Step 11:	M	M	M	M	M	M	M	-	U	U	...	U
Step 12:	M	M	M	M	M	M	M	-	M	U	...	U
backtrack: (c4)												
...												

of constraint violations is changed. This technique has been successfully used to schedule observation for the Hubble space telescope.

2.3.4 Local search techniques

We will describe here, in detail, hill climbing, simulated annealing and tabu search technique that are based on local search. Local search techniques, named also ([59], [2]) neighborhood search, are based on the neighborhood of the current solution. Many local search techniques have been developed for different problems. Search, in these techniques, begins with constructing the initial solution, which is after that changed iteratively with so called neighborhood relations (moves) until an acceptable or optimal solution is reached. Neighborhood of solution s is denoted with $N(s)$ and contains all solutions which can be obtained from the solution s transforming it by moves. One solution is said to be global optimum or optimal solution if it has the best evaluation among all solutions in the search space. One solution is said to be a local optimum if it is the best solution in its neighborhood. Imagine the search space like a landscape with many peaks.

Table 2.4: A possible initial solution for Problem 1

A	M	M	-	-	-	-
M	-	A	A	N	N	N
-	A	N	M	A	A	A
N	N	-	N	M	M	M

Table 2.5: One of the neighborhood solutions for solution in Table 2.4

M	M	M	-	-	-	-
A	-	A	A	N	N	N
-	A	N	M	A	A	A
N	N	-	N	M	M	M

The highest peak is a global optimum and other peaks in the landscape are local optima.

In order to explain some concepts of local search, suppose for example that we constructed for Problem 1 the initial solution in Table 2.4. This solution does not fulfill all constraints and in order to find a solution that fulfills all constraints, local search techniques could be applied. First we have to define the moves. Suppose, for example, we define for this problem only one move, which swaps the contents of two cells in any column of the table. We call this move $swap(j, i, k)$, where j represents the column in which the change of elements should be done, and i, k represent elements in column j that should swap their contents. The neighborhood of the solution in Table 2.4 represents now all solution that can be obtained by applying the move $swap(j, i, k)$, where i,j,k take all possible integer values (in this case $0 < j < 8, 0 < i, k < 5$). One possible neighborhood solution for the solution in Table 2.4 is presented in Table 2.5. This solution is obtained by applying the move $swap(1, 1, 2)$.

More moves, of course, can be defined, and they are most of the time specific to the problem in question that has to be solved. Larger neighborhood offers principally better possibilities to reach an optimal solution. However, exploring a larger neighborhood requires also more time. During each iteration, from the neighborhood of the solution based on some criteria (i.e., the solution is better then current solution) one solution is becoming a current solution. Let as mention that it is also possible that the current solution does not change, as the criteria for changing it are not fulfilled. In the next iteration, the new neighborhood for

the current solution is generated and again the descendant of the current solution is selected. This procedure is repeated for a determined number of iterations.

An initial solution can be constructed randomly, or a procedure to find a good initial solution can be applied.

Local search techniques differ mainly in the way they explore neighborhood and in their decision about accepting the descendant of the current solution. Some of the techniques generate complete neighborhood (with all defined moves) of the solution during each iteration. Some of them generate only part of neighborhood, i.e., one neighborhood solution or more. Regarding accepting the descendant solution, some techniques accept only better solutions than the current one, while others accept worse solution too, under some conditions. We further describe these techniques in more detail.

Hill climbing

Hill climbing is an iterative improvement technique. The simplest variant of a hill climbing technique begins with constructing the initial solution which we will call current solution. Iteratively, during the next iterations, the current solution is replaced with the best solution in the neighborhood if that one has a better evaluation then the current solution. If no improves can be made, the procedure stops. Hill climbing can easily get stuck in local optimum if there exist many local optima, which is the case with most practical problems.

One variant of hill climbing is random restart hill climbing. In this procedure, the simple hill climbing is called with different starting random initial solutions and to the end, the best solution found so far from all calls of the procedure is picked. In this case more chances exist to find the global optimum, and that depends on the number of starting points, which are used for running the simple hill climbing. The procedure is stopped after a determined number of iterations. Another criteria for stopping procedures are also used. For example, the procedure can be stopped if after a determined number of iteration no improves to the solution have been made. In Figure 2.3 the version of random restart hill climbing is given (based on [51]). Exit from the main loop is done if the number of maximal iterations is reached, and exit from the hill climbing loop is done if no improves can be done.

Another variant of hill climbing is a method called stochastic hill-climber. This procedure differs from the previous one in that, only one solution from the neighborhood is generated during each iteration. This generated neighborhood solution will be accepted with a probability which depends on its evaluation. In this manner, even a solution that is worse than the current solution, has

```
Initialize s_best
NumberOfIterations=0
Do While NumberOfIterations < MaximalNumberOfIterations

    Generate randomly Initial Solution s_c
    Evaluate s_c
    SolutionImproved=TRUE

    Do While SolutionImproved=TRUE
        Generate neighborhood solutions of solution s_c
        Select best solution s_x from the neighborhood

        If s_x is better than s_c then
            s_c = s_x
        else SolutionImproved=FALSE
    Loop

    NumberOfIterations=NumberOfIterations+1

    if s_c is better then s_best then
    s_best = s_c

Loop
```

Figure 2.3: Random restart hill climber

chances to become the current solution in the next iteration. The pseudo code of stochastic hill-climber is presented in Figure 2.4, where we assume that we have maximization problem.

The parameter T given in the code remains constant during the search. The probability for accepting the solution decreases if the solution s_x is much worse than the solution s_c. If the parameter T is greater, the difference of the current solution from that in the neighborhood has less impact on the probability to accept the generated neighborhood solution for the next iteration.

Simulated annealing

The first basic ideas for simulated annealing (SA) were published by Metropolis et al. [50] as early as in 1953. In the 1980s, Kirkpatrick et al. [39] suggested that

NumberOfIterations=0
Generate randomly Initial Solution s_c
Evaluate s_c
Do While NumberOfIterations < MaximalNumberOfIterations

Generate neighborhood solution s_x of the solution s_c
Select s_x with probability $p = \frac{1}{1+e^{\frac{eval(s_c)-eval(s_x)}{T}}}$

NumberOfIterations=NumberOfIterations+1

Loop

Figure 2.4: Procedure of a stochastic hill climber

simulated annealing could be used for optimization problems. This algorithm is based on the analogy of annealing of solids. Simulated annealing under special circumstances allows accepting of a worse solution from the neighborhood in order to escape from a local optimum. This technique is similar to a stochastic hill climbing, but here the parameter T called temperature changes during the search. Another difference is that the solution that is better than current one is always accepted in the next iteration. The pseudo code of simulated annealing is presented in Figure 2.5. Here we assume we have maximization problem.

In standard SA, the neighborhood solution s_x is selected most of the times randomly, but there exist also implementations in which the selecting of the neighborhood solution is cyclic, such that all neighbors are tried once before any is considered two times. Critical points in SA are the decision about the initial temperature and the function g(T, NumberOfIterations) which is called also cooling ratio and which defines how the temperature changes during the search. Terminate-Condition1 and Terminate-Condition2 are also important parameters that have to be specified. They may be dependent for example on the number of iterations, number of iterations without improvement etc.

For a more detailed description of simulated annealing, the reader is referred to Laarhoven and Aarts [42], Aarts and Korst [1], and Dowsland [20].

Tabu search

First ideas for the tabu search (TS) techniques came from Glover [25] and Hansen [31]. This technique is one of most powerful modern heuristic techniques, which has been used successfully for many practical problems. In this thesis, we will

NumberOfIterations=0
Initialize temperature T
Generate randomly Initial Solution s_c
Evaluate s_c
Do While terminate-condition1

Do While terminate-condition2

Generate neighborhood solution s_x of the solution s_c
if s_x is better than s_c then
$s_c = s_x$
else generate random x in the range (0,1)
if $x < e^{\frac{eval(s_x)-eval(s_c)}{T}}$
then $s_c = s_x$

Loop

T= g(T, NumberOFIterations)
NumberOfIterations=NumberOfIterations+1

Loop

Figure 2.5: Simulated annealing procedure

show how this technique can be used for the shift design problem. Basic idea of this technique is to avoid the cycles (visiting the same solution) during the search using the tabu list. In the tabu list specified information for search history for a fixed specific number of past iteration are stored. The acceptance of the solution for the next iterations in this technique depends not only from its quality, but also from the tabu list. In the tabu list one stores for instance the moves or inverse moves that have been used during specific number of past iterations. The stored moves are made tabu for several iterations. A solution is classified as a tabu solution if it is generated from a move that is present in the tabu list. In this technique, during each iteration, the complete neighborhood (with defined moves) of the current solution is generated. In the basic variant of TS the best solution (not tabu) from the neighborhood is accepted for the next generation. However, it is also possible to accept the tabu solution if it fulfills some conditions, which are determined with, the so called aspiration criteria (i.e., the solution is tabu but has the best evaluation so far). In Figure 2.6 the basic procedure of the is presented.

NumberOfIterations=0
Initialize tabu list
Generate randomly Initial Solution s_c
Evaluate s_c

Do While terminate-condition

Generate all neighborhood solutions of the solution s_c
Find best solution s_x in the neighborhood
if s_x is not tabu solution then $s_c = s_x$

elseif 'aspiration criteria' is fulfilled then $s_c = s_x$

else

find best not tabu solution in the neighborhood s_{nt}
$s_c = s_{nt}$

Update tabu list
NumberOfIterations=NumberOfIterations+1

Loop

Figure 2.6: Basic tabu search procedure

The termination-condition can be fulfilled for example if the maximum number of iterations is exceeded, or no improves have been found for several numbers of iterations, etc.

In this technique, is it possible to accept a worse solution. In this way, tabu search escapes from local optima. However, as it is heuristic technique, it does not always guarantee to find the global optimum.

The tabu list is updated after each iteration. In order to give an idea of the tabu list suppose that the initial solution of the Problem 1 is given in Table 2.6. Further, let us suppose that the best solution in the neighborhood of the solution that can be obtained with move $swap(column, element1, element2)$ is the solution obtained by exchanging elements 2 and 3 in the column 1 (see Table 2.7). In order to obtain this solution, we applied move *swap* with the parameters $swap(1, 2, 3)$. To avoid the cycle, we store the move *swap* with these parameters $swap(1, 2, 3)$ in the tabu list. In this way, the exchange of elements

Table 2.6: Initial solution for TS for example 1

M	-	M	-	-	-	-
N	M	A	A	N	N	N
-	A	N	M	M	A	A
A	N	-	N	A	M	M

Table 2.7: New solution obtained from solution in Table 2.6 by exchanging elements 2 and 3 in column 1

M	-	M	-	-	-	-
-	M	A	A	N	N	N
N	A	N	M	M	A	A
A	N	-	N	A	M	M

2 and 3 in column 1 is made tabu for a specified number of iterations (this depends on the length of the tabu list). Solutions in the neighborhood that will be obtained in the next iterations with $swap(1, 2, 3)$ are considered tabu solutions.

One of the critical points in the tabu search is the length of the tabu list. This parameter most of the time depends on specific problem and usually is determined experimentally. There exists also a variant of the basic TS in which this parameter changes dynamically during the search called reactive tabu search [9]).

The memory in which the visited solutions or moves are stored is called recently-based memory. Another memory used in the tabu search technique is called frequency-based memory (or long term memory). In this memory, there could be stored, for example, the information about the frequency of use of each move during the prespecified number of iterations. The frequency-based memory can be used in different ways. A typical use would be, for example, to diversify the search in the unexplored regions of the search space, for example by applying moves that have not been applied for several iterations or to intensify the search in promising regions of the search space.

For detailed description of tabu search technique the reader is refereed to Glover and Laguna [28] and Glover [26], [27].

Other heuristic techniques

There exist also other heuristic techniques that have been used successfully in problem solving. Here we mentioned them briefly. Genetic algorithms ([34], [30]) were developed from Holland, his colleagues, and his students 1960s and 1970s at the University of Michigan. Currently, GA are applied almost in all areas. Neural networks and recently ant colony optimization heuristics have been also used for solving of combinatorial problems.

The techniques we described in this chapter can also be used in combination with each other. For example, combining genetic algorithms with local search algorithms has given good results for many problems.

Chapter 3

Workforce Scheduling

Workforce scheduling, in general, includes sub-problems which appear in many spheres of life like in industrial plants, hospitals, public transport, airlines companies, universities etc. According to the area, this problem is denoted with different names in the literature. Usual used terms are: workforce scheduling, manpower scheduling, staff scheduling, shift scheduling, employee timetabling, employee scheduling, rostering, personnel scheduling, crew scheduling, labor shift scheduling problem etc. In this chapter, we first give a rather intuitive and semi-formal definition of workforce scheduling problems. Further, we give a detailed description and formal definition for two problems we consider in this thesis (rotating workforce scheduling and shift design). Subsequently, we give a review on work in computerized workforce scheduling which has been the subject of study of researchers from the operational research community and the AI community since more than 30 years.

3.1 Basic concepts

A workforce schedule represents the assignments of the employees to the defined shifts for a period of time. In Table 3.1 one typical representation of workforce schedules is presented. There exist different representations of workforce schedules in different areas, but we can explain the basic definition of workforce scheduling without loss of generality based on this representation.

Subsequently, main features of this schedule are listed:

Employees: This schedule is made for 9 employees (first column of the table). In this table, the employees are represented with numbers 1-9, but other

Table 3.1: One typical week schedule for 9 employees

Employee/day	Mon	Tue	Wed	Thu	Fri	Sat	Sun
1	D	D	D	D	D	-	-
2	-	D	D	D	D	D	D
3	D	-	-	N	N	N	N
4	-	-	-	-	A	A	A
5	A	A	A	A	-	-	-
6	N	N	N	N	N	-	-
7	-	-	A	A	A	A	A
8	A	A	-	-	-	N	N
9	N	N	N	-	-	D	D

representations are also used (i.e., name of the employees). The number of employees is variable and depends on the specific problem. In general workforce scheduling problems, the employees can have different qualifications. Additionally, general problems can include employees with different working times.

Planning period: Determines the length of the schedule. This table represents a one week schedule of 9 employees. The second column of the table corresponds to Monday, the third to Tuesday and so on until Sunday. In practice, the length of planning period varies. Most typical are week and month schedules, although schedules with other lengths also appear in practice.

Shifts: The letters D, N, A in table represent the shifts. A shift is a period of time during which the employees are in duty. This schedule contains three shifts, but in general the number of shifts can change. To be on duty does not mean automatically that the employee is at the workplace. The employee can also be in a so-called stand-by duty (being prepared to work if needed). The sign '-' represents days-off (the employee has time free). Basic features of one shift are the start and the end time. For example, the shift D can be defined as a shift which begins at 6:00 and ends at 14:00, A: 14:00-22:00 and N: 22:00-6:00. Usually, according to the begin time, each shift becomes its name. In this example we call shifts D, A, and N respectively day shift, afternoon shift, and night shift. The structure of shifts in practice can be much more complicated. Shifts can for example include breaks of different lengths, which can be fixed or within time windows (the break can appear anywhere during the specified time period).

Thus, what does this schedule describe? In this schedule is described explicitly when 9 employees will work during one week. The employee 1 works

Table 3.2: Second week for the cyclic schedule from Table 3.1

Employee/day	Mon	Tue	Wed	Thu	Fri	Sat	Sun
1	N	N	N	-	-	D	D
2	D	D	D	D	D	-	-
3	-	D	D	D	D	D	D
4	D	-	-	N	N	N	N
5	-	-	-	-	A	A	A
6	A	A	A	A	-	-	-
7	N	N	N	N	N	-	-
8	-	-	A	A	A	A	A
9	A	A	-	-	-	N	N

from Monday until Friday in a day shift (D) and during Saturday and Sunday has days-off. The second employee has a day-off on Monday and works in a day shift during the rest of the week. Further, the last employee works from Monday until Wednesday in night shifts (N), on Thursday and Friday has days-off, and on Saturday and Sunday works in the day shift. Each row of the table represents the week schedule of one employee.

3.1.1 Cyclic and non-cyclic schedules

One wide division of workforce schedules is in cyclic (rotating) and non-cyclic schedules. If the schedule represented in Table 3.1 is defined as non-cyclic, then, after a week, a new schedule for the employees should be constructed or the present one remains. If the schedule is cyclic, after the first week the first employee will take the schedule of the second employee, the second employee the schedule of third employee and so on, then the last employee will take the schedule of the first employee (see Table 3.2). The cycle length is equal to the number of employees times the planning length, and thus in this case it is 9 weeks. Since each employee takes the schedule of each other employee during the cycle, all employees pass through the same schedule during the cycle. In case of cyclic schedules, all employees have the same working hours on average and are considered to have the same qualifications.

Table 3.3: Temporal requirements for the schedule in Table 3.1

Shift	Mon	Tue	Wed	Thu	Fri	Sat	Sun
D	2	2	2	2	2	2	2
A	2	2	2	2	2	2	2
N	2	2	2	2	2	2	2

3.1.2 Constraints

Workforce schedules most of the times should fulfill several constraints. The workforce scheduling constraints are influenced by the results from ergonomics [11], cost for the organization and individual preferences of employees. It is well known that the schedule can have an impact in the health of the employees and their performance at work. The type of constraints depend on the specific problems. However, we give here a brief description of some constraints which appear almost in every problem.

Temporal requirements

One of the typical constraints for each problem is about workforce requirements. Workforce requirements determines the needed number of employees for each day and time interval during the planning period. For example, in Table 3.1 two employees are present every day in each shift. It follows that this schedule satisfies the requirements of two employees for each day and each shift (see Table 3.3). Shortly, we represent these requirements by a so-called requirement matrix where each row of the matrix represents one shift and each column one day, whereas the elements of the matrix represent needed number of employees for particular shifts and days. In a general workforce problem it is not always necessary that the requirements are fulfilled exactly since this sometimes may not be possible. If that is the case, the aim is to minimize the difference of numbers of employees who are presented in the schedule to the requirements.

$$R_{3,7} = \begin{pmatrix} 2 & 2 & 2 & 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 & 2 & 2 & 2 \end{pmatrix}$$

Number of working hours per week

Average number of working hours is defined for each employee per week. In the general case, employees can have different numbers of working hours. The aim in the schedule is to reach a specified number of working hours for each employee as exactly as possible.

Sequences of shifts

Some sequence of shifts are not allowed or not preferable to be assigned to the employees because of ergonomic criteria. For example, working in the morning shift after a night shift is not allowed because there should exist a minimum period of rest between two duties. Another example includes the limits about the length of sequences of work in the same shift. For example, working in the night shift for more than 4 to 5 successive days is depreciated.

Work and rest periods

Usually the employees should have rest days after a certain number of working days. For example, in the schedule from Table 3.1, no employee will work more than 7 days in a row. Usually the rest period should not be too short or too long, thus, for this reason limits are set. In the schedule of Table 3.1, rest periods have length between 2 to 4 days.

Individual preferences

In the case the schedule is not cyclic, individual preferences of employees can be taken into consideration. Individual preferences of employees for example concern the days the employees want to have off (availability of employees). Another example includes preferences about the work shifts. Some employees for example may prefer to work more during the weekend or in night shifts.

Hard and soft constraints

Constraints in workforce scheduling problem can be hard or soft. In the case they are hard, any valid schedule must fulfill them. In case of soft constraints, violating them is allowed, but the goal is to fulfill them as much as possible. Soft constraints can be for example individual preferences. In this case, it is sometimes impossible to construct the schedule that fulfills constraints of all employees, but the aim is to fulfill as many of the preferences of the employees as possible.

Table 3.4: Possible temporal requirements for one week

Time interval/day	Mon	Tue	Wed	Thu	Fri	Sat	Sun
07:00-11:00	5	5	5	5	5	1	1
11:00-14:30	10	10	10	10	10	4	4
14:30-20:30	12	12	12	12	12	4	4
20:30-06:00	14	14	14	14	14	4	4

3.1.3 Design of shifts

Usually, after the temporal requirements are defined, the first phase in workforce scheduling is to construct the shifts. In this phase the number of employees for each shift is determined. Further, the total number of employees is calculated based on the average number of working hours for a certain period of time, usually one week.

In Table 3.4 an example of temporal requirements is given. In this example the temporal requirements are given for one week. Based on these requirements and some constraints about the shifts, the aim in the shift design problem is to generate legal shifts that fulfill in the best way the temporal requirements. In the next section we will give a detailed description of the problem of shift design we consider in this thesis.

3.2 Problems we consider in this thesis

In this thesis we consider the problem of generation of shifts and the problem of assignment of shifts to the employees for the case of rotating workforce schedules (rotating workforce scheduling). Further, we give a detailed description of these problems.

3.2.1 Shift design

Instance:

- n consecutive time intervals $[a_1, a_2)$, $[a_2, a_3)$, ... $[a_n, a_{n+1})$, all with the same length *slotlength* in minutes. Each interval $[a_i, a_{i+1})$ has an adjoined numbers w_i indicating the optimal number of employees that should be present during that interval.

Time point a_1 represents the begin of the planning period and time point a_n represents the end of the planning period. The format of time points is: *day:hour:minute*. For simplicity the temporal requirements are usually represented using longer time intervals. See one possible representation of temporal requirements for one week in Table 3.4.

- y shift types $v_1, \ldots, v_y$. Each shift type v_j has the following adjoined parameters: v_j.min-start, v_j.max-start which represent the earliest and latest start of the shift and v_j.min-length, v_j.max-length which represent the minimum and maximum lengths of the shift. In Table 3.5 an example of shift types is given.
- An upper limit for the average number of working shifts per week per employee (definition is given later).

Table 3.5: Possible constraints for shift types in the shift design problem

Shift type	Earliest begin	Latest begin	Shortest length	Longest length
M	05:00	08:00	07:00	09:00
D	09:00	11:00	07:00	09:00
A	13:00	15:00	07:00	09:00
N	21:00	23:00	07:00	09:00

Problem:

Generate a set of k shifts $s_1, \ldots, s_k$. Each shift s_l has adjoined parameters s_l.start and s_l.length and must belong to one of the shift types (see formal definition below). Additionally, each real shift s_p has adjoined parameters $s_p.w_i, \forall i \in \{1, \ldots, C\}$ (C represents number of days in the planning period) indicating the number of employees in shift s_p during the day i. The aim is to minimize the four components given below:

- Sum of the excesses of workers in each time interval during the planning period (see formal definition below).
- Sum of the shortages of workers in each time interval during the planning period (see formal definition below).
- Number of shifts k.

- Distance of the average number of duties per week in case it is above a certain threshold.

This is a multi criteria optimization problem. The criteria have different importance depending on the situation. The objective function is a scalar function which combines the four weighted criteria. Let we note that we consider designing of shifts for a week (less days are also possible) and assume that the schedule is cyclic (consecutive element of last time slot of planning period is the first time slot of planning period).

Formal definitions:
The generated shift belongs at least to one of the shift types if:

$$\forall l \in \{1,\ldots,k\}\ \exists j \in \{1,\ldots,y\}\ /$$

$$v_j.\text{min-start} \leq s_l.\text{start} \leq v_j.\text{max-start}$$

$$v_j.\text{min-length} \leq s_l.\text{length} \leq v_j.\text{max-length}$$

Sum of the shortages and excesses (in minutes) of workers in each time interval during the planning period is defined as

$$ShortagesSum = \sum_{d=1}^{n}(Indicator(w_d - \sum_{p=1}^{k} x_{p,d}) * slotlength)$$

$$ExcessesSum = \sum_{d=1}^{n}((Indicator - 1)(w_d - \sum_{p=1}^{k} x_{p,d}) * slotlength)$$

where,

$$Indicator = \begin{cases} 1 & \text{if } w_d - \sum_{p=1}^{k} x_{p,d} \text{ is positive} \\ 0 & \text{otherwise.} \end{cases}$$

$$x_{p,d} := \begin{cases} s_p.w_i & \text{if } d \text{ time slot belongs to the interval of shift } s_p \text{ in the day i} \\ 0 & \text{otherwise.} \end{cases}$$

The average number of working shifts per week per employee (AvD) is defined below:

$$AvD = \frac{(\sum_{i=1}^{k}\sum_{j=1}^{C} s_i.w_j) * AverageNumberOfHoursPerWeek}{\sum_{i=1}^{k}\sum_{j=1}^{C} s_i.w_j * s_i.length}$$

3.2.2 Rotating workforce scheduling

In this section, we describe the problem of assignment of shifts and days-off to employees in case of rotating workforce schedules. This is a specific problem of a general workforce-scheduling problem. The definition is given below:

Instance:

- Number of employees: n.
- Set A of m shifts (activities) : $a_1, a_2, \ldots, a_m$, where a_m represents the special day-off "shift".
- w: length of schedule. The total length of a planning period is $n \times w$ because of the cyclic characteristics of the schedules.
- A cyclic schedule is represented by an $n \times w$ matrix $S \in A^{nw}$. Each element $s_{i,j}$ of matrix S corresponds to one shift. Element $s_{i,j}$ shows which shift employee i works during day j or whether the employee has time off. In a cyclic schedule, the schedule for one employee consists of a sequence of all rows of the matrix S. The last element of a row is adjacent to the first element of the next row, and the last element of the matrix is adjacent to its first element.
- Temporal requirements: $(m-1) \times w$ matrix R, where each element $r_{i,j}$ of matrix R shows the required number of employees for shift i during day j.
- Constraints:
  - Sequences of shifts permitted to be assigned to employees (the complement of inadmissible sequences): Shift change $m \times m \times m$ matrix $C \in A^{(m^3)}$. If element $c_{i,j,k}$ of matrix C is 1, the sequence of shifts (a_i, a_j, a_k) is permitted, otherwise it is not.
  - Maximum and minimum length of periods of consecutive shifts: Vectors $MAXS_m$, $MINS_m$, where each element shows the maximum respectively minimum permitted length of periods of consecutive shifts.
  - Maximum and minimum length of blocks of workdays: $MAXW$, $MINW$.

Problem: Find as many non-isomorphic cyclic schedules (assignments of shifts to employees) as possible that satisfy the requirement matrix, all constraints, and are optimal in terms of weekends without scheduled work shifts (weekends off).

All constraints defined are hard constraints, except one for the weekends-off. Note also that generation of many possible schedules that satisfy all constraints is required. The idea behind this requirement is to give the user the possibility to include other possible soft constraint in the group of schedules which fulfill all hard constraints and maximize the number of weekends off.

In the following, we formally express the constraints:

The requirement matrix R is satisfied if

$$\forall j \in \{1, 2, \ldots, w\}\, \forall k \in \{1, 2, \ldots, m-1\}$$

$$|\{i \in \{1, 2, \ldots, n\} / \; s_{i,j} = a_k\}| = r_{k,j}$$

The other constraints are satisfied if:

For the shift change matrix C:

$$\forall i \in \{1, \ldots, n\}\, \forall j \in \{1, \ldots, w\}\, \exists\, e, f, g \in \{1, \ldots, m\}$$

$$s_{i,j} = a_e \wedge next(s_{i,j}) = a_f \wedge next_2(s_{i,j}) = a_g \;\Rightarrow\; c_{e,f,g} = 1$$

where

$$next(s_{i,j}) = \begin{cases} s_{i,j+1} & \text{if } j < w \\ s_{i+1,1} & \text{if } j = w \text{ and } i < n \\ s_{1,1} & \text{otherwise} \end{cases}$$

and

$$next_k(s_{i,j}) = \underbrace{next(next(\ldots next(}_{k} s_{i,j}) \ldots)),$$

Maximum length of periods of successive shifts:

$$\forall k \in \{1, \ldots, m\}\, \forall i \in \{1, \ldots, n\}\, \forall j \in \{1, \ldots, w\}$$

$$(s_{i,j}, next(s_{i,j}), \ldots, next_{MAXS(k)}(s_{i,j})) \neq \underbrace{(a_k, \ldots, a_k)}_{MAXS(k)+1}$$

Minimum length of periods of successive shifts:

$$\forall k \in \{1, \ldots, m\}\, \forall i \in \{1, \ldots, n\}\, \forall j \in \{1, \ldots, w\}$$

$$\neg(s_{i,j} \neq a_k \wedge next(s_{i,j}) = a_k \wedge (\bigvee_{b \in \{2,\ldots,MINS(k)\}} next_b(s_{i,j}) \neq a_k))$$

Maximum length of work blocks:

$$\forall i \in \{1,\ldots,n\}\, \forall j \in \{1,\ldots,w\}$$

$$(s_{i,j} = a_m \vee next(s_{i,j}) = a_m \vee \ldots \vee next_{MAXW}(s_{i,j}) = a_m)$$

Minimum length of work blocks:

$$\forall i \in \{1,\ldots,n\}\, \forall j \in \{1,\ldots,w\}$$

$$\neg(s_{i,j} = a_m \wedge next(s_{i,j}) \neq a_m \wedge (\bigvee_{b \in \{2,\ldots,MINW\}} next_b(s_{i,j}) = a_m))$$

3.2.3 Complexity of the problems we consider in this thesis

The staff scheduling problem is listed in the book by Gary and Johnson [21]) as problem [SS20] in the list of NP complete problems. The NP completeness of the problem is proved by a transformation from X3C (Exact cover by 3-sets). Garey and Johnson [21] define the staff scheduling problem as follows:

Instance: Positive integers m and k, a collection C of m-tuples, each having k 1's and $m-k$ 0's (representing possible work schedules), a 'requirement' m-tuple $\overline{R}$ of non-negative integers, and a number n of employees.

Question: Is there a schedule $f : C \rightarrow Z_0^+$ such that $\sum_{\overline{c} \epsilon C} f(\overline{c}) \leq n$ and such that $\sum_{\overline{c} \epsilon C} f(\overline{c})\overline{c} \leq n$?

Lau [46] considers the problem of the assignment of shifts to employees subject to demands and shift change constraints (the shift change matrix is two-dimensional), both for cyclic and non-cyclic schedules. The author gave the proof that the problem is NP complete by a polynomial reduction from the 3-SAT problem. In the same paper, the author presents a polynomial algorithm for the case when the shift changes are monotonic (if the shifts are arranged in order according to their start time, the shift numbers assigned to each employee should be monotonically non-decreasing).

Kortsarz and Slany [41] show that it is possible to model the shift design problem as a network flow problem, namely as the cyclic multi-commodity capacitated fixed-charge min-cost max-flow problem. Exploiting this modeling, if one drops the objective to minimize the number of shifts, the problem can be

solved very efficiently with a polynomial min-cost max-flow algorithm. Using a reduction to the Minimum Edge Cost Flow problem (no. [ND32] in Garey and Johnson [21]), Kortsarz and Slany show also that, on the contrary, the general problem is NP complete. The problem in itself is thus hard. It is also hard to approximate: Kortsarz and Slany [41] also show that there is a constant $c < 1$ such that approximating the shift design problem with polynomial bounds on numbers appearing in the requirements within $c \ln n$ is NP hard. Additionally, they convincingly argue that it is very unlikely that the shift design problem admits an efficient algorithm with an approximation guarantee that is substantially better than an $O(\sqrt{n \log M})$ ratio (where M is the largest number in the requirements), which in practice would anyway be unusable. This strongly suggests that attacking the problem using exact methods is not feasible for anything beyond toy problems, and that heuristic methods should thus be employed for realistically sized problems. Similar results exist for the Minimum Edge Cost Flow problem and variants [41]. This problem is one of the more fundamental flow variants with many applications. A sample of these applications include optimization of synchronous networks, source-location, transportation, scheduling (for example, trucks or manpower), routing, and designing networks (for example, communication networks with fixed cost per link used, e.g., leased communication lines), see [41] for references.

3.3 Previous work

Computerized workforce scheduling has interested researchers for over 30 years. In this section, we give brief review of the previous work in workforce scheduling with the emphasis on the problems which show similarities with the problems we consider in this thesis. First we present a review of work in the rotating workforce scheduling and shift scheduling. Further, we give a review of work in the related problems. Let us not that the problem of generating the shifts and assignment of the shifts and days-off to the employees are sometimes solved simultaneously. Survey of algorithms used for different workforce scheduling problems is given by Tien and Kamiyama [66]. Operational research (OR) approaches for employee scheduling are described in book Nanda and Browne [56].

3.3.1 Rotating workforce scheduling

For generation of rotating workforce schedules different approaches were used. Examples for the use of exhaustive enumeration are [33] and [15]. Balakrishnan and Wong [8] solved a problem of rotating workforce scheduling by modeling it as

a network flow problem. We will compare our results in Chapter 4 with the results of Balakrishnan and Wong. Gärtner and Wahl [23] discuss the interaction of vizualization and diverse planning strategies for constructing rotating workforce schedules. They proposed also few heuristics for the manual construction of schedules. Several other algorithms for rotating workforce schedules have been proposed in the literature [35, 36, 40, 45]. Recently Laporte [43] considered developing rotating workforce schedules by hand and showed how the constraints can be relaxed to get acceptable schedules.

3.3.2 Shift scheduling

The shift design problem we consider in this thesis is similar with problem which has been addressed in literature as shift scheduling problem. Typically for this problem is required to generate shifts and number of employees for each shift for a single day. The aim is to obtain the solutions without under-staffing and to minimize number of employees. Let we note that problem of shift design we consider in thesis differs in several aspects from the problem of shift scheduling. First, we consider generation of shifts for a week. We consider also minimizing of number of shifts and distance of the average number of duties per week in case it is above a certain threshold. Moreover, in our problem under-staffing is allowed.

Shift scheduling problem has been solved mainly by using Integer Programming (IP). Dantzig [19] developed the original set-covering formulation for shift scheduling problem:

$$\text{mimimize} \sum_{j=1}^{n} c_j x_j$$

subject to

$$\sum_{j=1}^{n} a_{ij} x_j \geq r_i \text{ for } i = 1, 2, \ldots, m$$

$$x_j \geq 0 \text{ and integer for } n \in N$$

where

n = index for shifts,
x_j = number of employees assigned to shift j,
r_i = number of employees required to work in the ith time period,
c_j = cost of having an employee work shift j,
m =number of time periods to be scheduled over a single day,

$$a_{ij} = \begin{cases} 1 & \text{if the time period } i \text{ is a work period of shift } j \\ 0 & \text{otherwise.} \end{cases}$$

In this formulation for each feasible shift exist one variable. Feasible shifts are enumerated based on possible start, length, breaks and time windows for breaks. When the number of shifts increases rapidly this formulation is not practical.

Bechtold and Jacobs [10] proposed a new integer programming formulation for shift scheduling. In this formulation the modeling of break placements is done implicitly. Authors reported superior results with their model compared to the set covering model. Their approach is limited in the organization which operates less than 24 hours per day.

Thompson [64] introduced fully implicit formulation of shif scheduling problem. The model combines work of Moondra [53] and Bechtold and Jacobs [10] in which respectively the length of shifts and the breaks are modeled implicitly. Authors reported better results compare to model of Bechtold and Jacobs [10].

Another integer programming model for shift scheduling with multiple rest and lunch breaks and break windows was proposed by Aykin [6]. This formulation is applicable to the shift scheduling problem without the limitation in [10]. Compare of different modeling approaches is given in by Aykin [7].

3.3.3 Related workforce scheduling problems

Schaerf and Meisels [61] considered problems of workforce scheduling where the shifts are composed of tasks and the employees can have different qualification. The problems includes different constraints like requirements, ability, availability etc. The authors present a local search framework consisting of one move and a very simple objective function, and proposed generalized local search with three moves and another objective function. For two cases techniques based on the hill climbing were applied. They differ in the amount of the neighborhood they explore during each iteration: The first technique has only one neighborhood solution, while the latter constructs all neighborhood solutions for the predefined moves. The objective function for generalized local search is a combination of two types of constraints and a so called look-ahead factor. All these components in the objective function have different weights, which are changed during the search (shifting penalty mechanism). The authors experimented with problems from nurse scheduling and scheduling in the production line. The results for some instances of the second problem were given. The instances consist of 21 weekly shifts (three per day), 50 employees and a total of 280 tasks. The authors

reported obtaining best results by using the generalized local search technique based on min-conflict hill climbing [52]. According to the authors, tabu search techniques as basis for generalized local search gives comparable results with the above mentioned best technique. The authors claim: "For the generalized local search, the combining effect of the shifting penalty mechanism (which changes continuously the quality of moves) and the presence of different move types made the use of the prohibition mechanism not effective in our experiments". Several other algorithms based on the local search techniques have been proposed in the literature for different workforce scheduling problems [67, 17, 65, 14, 57].

Smith and Bennett [63] combine constraint satisfaction and local improvement algorithms to develop schedules for anaesthetists. Weil and Heus [69] formulate the nurse scheduling problem as a constraint satisfaction problem and proposed an approach to reduce the search space by eliminating interchangeable values. The shifts with similar characteristics, which may be interchangeable, are considered like one shift and thus the domain of each variable is reduced. The approach is used for scheduling two nurses in a two weeks schedule. In this paper no results for practical problem are given. Several other algorithms based on the constraint programming have been proposed in the literature for different workforce scheduling problems [47, 3, 49, 5].

The set-cover formulation presented in the previous section can be used also for the problem of assigning of shifts to employees for a week [54]. This problem is addressed in OR literature as tour scheduling problem. Example of solving tour scheduling problem by using integer programming is the work of Cezik et al. [16]. proposed integer programming model for the weekly tour in the call center. This model is obtained by combining seven daily shift scheduling models in a network flow problem. The model they proposed can accommodate different days-off requirements. Furthermore, their formulation allows finding of solutions which fulfill the the requirements about the difference between starting times of any two consecutive shifts. This model work under assumption that shifts are fully contained in one day without spill-over to the next day.

Different approaches were used to solve general workforce scheduling in which problems of the design of shifts and the assignment of shifts to employees are solved simultaneously. Jackson and Dollard [37] consider a problem where the employees have different skills and availabilities and the shifts with skill requirements are given. The authors proposed a simple greedy algorithm for this problem. Glover and McMillan [29] consider the problem that includes employees with different skills and different working hours, and many constraints about demands for each period, maximum and minimum number of working hours for each employee, constraints about feasible shifts, linking constraints between time periods and so one. The problem of shift design and assignment of shifts to em-

ployees are solved simultaneously. The authors proposed an approach that relies on integration of techniques from management sciences and artificial intelligence. This approach combines tabu search, where for the evaluation of the moves the multiple evaluation criteria is used, with an approach that creates additional criteria which should lead to a configurations which seems promising. Additionally, a problem decomposition is used during the search. In this paper results of 10 tests from real world applications are presented. Each problem involves around 100 employees and a weekly schedule is made. Execution time for each example is around 20 minutes and the optimality of the obtained solutions ranges between 98 to 99 percent. The number of constraints is 3400-9000, and from one million to 4 million variables are involved.

Chapter 4

Efficient Generation of Rotating Workforce Schedules

Generating high-quality schedules for a rotating workforce is a critical task in all situations where a certain staffing level must be guaranteed, such as in industrial plants or police departments. Results from ergonomics [11] indicate that rotating workforce schedules have a profound impact on the health and satisfaction of employees as well as on their performance at work. Moreover, rotating workforce schedules must satisfy legal requirements and should also meet the objectives of the employing organization. In this paper our description of a solution to this problem is being stated. One of the basic design decisions was to aim at high-quality schedules for realistically sized problems obtained rather quickly, while maintaining human control. The interaction between the decision-maker and the algorithm therefore consists of four steps: (1) choosing a set of lengths of work blocks (a work block is a sequence of consecutive days of work), (2) choosing a particular sequence of blocks of work and days-off blocks amongst these that have optimal weekend characteristics, (3) enumerating possible shift sequences for the chosen work blocks subject to shift change constraints and bounds on sequences of shifts, and (4) assignment of shift sequences to work blocks while fulfilling the staffing requirements. The combination of constraint satisfaction and problem-oriented intelligent backtracking algorithms in each of the four steps allows for finding good solutions for real-world problems in acceptable time. Computational results from a benchmark example found in the literature confirmed the viability of our approach. The algorithms have been implemented in commercial shift scheduling software.

4.1 Introduction

Rotating workforce scheduling appears mostly in organizations which operate around the clock in several shifts. We gave a detailed description of this problem in Section 3.2.2. Let us remind briefly the problem. For this problem we have given: A number of employees n, a set of m shifts, the length of the schedule w, temporal requirements which are given as matrix $R((m-1) \times w)$ and constraints about the forbidden sequences of shifts to be assigned to employees, maximum and minimum lengths of periods of consecutive shifts given as vectors $MAXS_m$, $MINS_m$ and maximum and minimum lengths of blocks of workdays.

The aim is to find as many non isomorphic cyclic schedules (assignments of shifts to employees) as possible that satisfy the requirement matrix, all constraints, and are optimal in terms of weekends without scheduled work shifts (weekends off). An important criteria is to generate the workforce schedules efficiently.

The length of schedule is usually one week. In a rotating workforce schedule — at least during the stage of planning — all employees have the same basic schedule but start with different offsets. Therefore, while individual preferences of the employees cannot be taken into account, the aim is to find a schedule that is optimal for all employees on average. The employees in this problem have all the same qualification. Work is done usually in three shifts, although there are case with more shifts. Note that in this problem we consider simultaneously assignments of the days-off and shifts to employees. In the literature these two phases are considered sometimes separately. The problem of rotating workforce scheduling is NP complete [46] and thus hard to solve in general.

To give an idea of the search space of this problem, suppose there are 9 employees and three shifts and the schedule for one week has to be constructed. The search space of this problem is $4^{63} > 8.5 \times 10^{37}$, which is a very large number. This problem is a rather small instance of the problem which we can solve very easily with the approach we proposed in this thesis.

In Chapter 3 we gave a review on previous work on rotating workforce schedules. In this thesis we proposed and implemented a new framework to solve the problem of rotating workforce scheduling, including efficient backtracking algorithms for each step of the framework. Constraint satisfaction is split up into four steps so that the search space is reduced for each step, which provides the possibility of using backtracking algorithms. Computational results show that our approach is efficient for real-sized problems. The main characteristic of our approach is the possibility to generate high quality schedules in a short time interactively, involving the human decision-maker.

Further in this chapter we describe our new framework and the algorithms that were used and give computational results for real life examples and benchmark examples from the literature.

4.2 New four step framework for rotating workforce scheduling

Tien and Kamiyama [66] proposed a five stage framework for workforce scheduling algorithms. This framework consists of the following stages: determination of temporal manpower requirements, total manpower requirement, recreation blocks, recreation/work schedule and assignment of shifts (shift schedule). The first two stages can be seen as an allocation problem and the last three stages are scheduling of days-off and assignment of shifts. All stages are related to each other and can be solved in sequence, but there also are algorithms which solve two or more stages simultaneously.

For the problem formulation examined here, we assume that the temporal and total requirements are already stated. Temporal requirements are given through the requirement matrix and determine the number of employees needed for each day and each shift. For this problem total requirements are represented by the number of employees n. We propose a new framework for solving the problem of assigning days-off and shifts to employees. This framework consists of the four steps given below:

1. choosing a set of lengths of work blocks (a work block is a sequence of consecutive days of working shifts),
2. choosing a particular sequence of work and blocks of days-off amongst these that have optimal weekend characteristics,
3. enumerating possible shift sequences for the chosen work blocks subject to shift change constraints and bounds on sequences of shifts, and
4. assignment of sequences of shifts to blocks of work while fulfilling the staffing requirements.

As a start we explain our motivation for using this framework. The approach we use is focused on interaction with the decision-maker. Thus, the process of generating schedules is only semi-automatic. When our system generates possible candidate sets of lengths of work blocks in step 1, the decision-maker has to select the one solution that reflects his or her preferences best. In this way we

satisfy two goals: On the one hand, an additional soft constraint regarding the lengths of work blocks can be taken into account by this interaction, on the other hand the search space for step 2 is significantly reduced. Consequently it is possible to solve step 2 much more effectively. In step 2 our main concern is to find the best solution for weekends off. The user's selection made in step 1 can impact features of weekends off versus length of work blocks since these two constraints are the ones that — in practice — are in conflict most frequently. The decision-maker can decide if he or she prefers an optimal length of work blocks or better features for weekends off. With step 3 we satisfy two more goals. First, because of the shift change constraints and the bounds on the number of consecutive shifts per sequence, each work block has only few legal shift sequences (terms). Thus in step 4 backtracking algorithms will very quickly find assignments of terms to the work blocks so that the requirements are fulfilled (if shift change constraints with days-off exist, their satisfaction is checked at this stage). Second, a new soft constraint is introduced. Indeed, as we generate several shift plans, they will contain different terms. The user has then the possibility to eliminate some undesired terms, and therefore to eliminate solutions containing these terms. Terms can have impact on the fatigue and sleepiness of employees and are therefore very important when high-quality plans are desired.

4.2.1 Determination of lengths of work blocks

A work block is a sequence of workdays between two days-off. An employee has a workday scheduled for day j if he or she is assigned a shift different from the days-off shift a_m. In this step the feature of work blocks we are interested in is its length only. Other features of work blocks (e.g., shifts of which the work block is made of, start and end of the block, etc.) are not known at this time. As the schedule is cyclic each employee has the same schedule and consequently the same work blocks for the entire planning period.

Example: The weekly schedule for 9 employees given in Table 4.1 (D, A and N are abbreviations for day shift, afternoon shift and night shift) consists of two work blocks of length 6, four work blocks of length 5 and two of length 4 in the order (4 6 5 4 6 5 5 5). By re-arranging the order of blocks, other schedules can be constructed, for example the schedule with the order of work blocks (5 5 6 5 4 4 5 6). We will represent schedules with the same work blocks, but a different order of work blocks using unique solutions, so-called class solution, where blocks are shown in decreasing order. The class solution for the example given above will therefore be {6 6 5 5 5 5 4 4}.

It is clear that even for small instances of problems many class solutions can

Table 4.1: A possible schedule with work blocks in the order (46546555)

Employee/day	Mon	Tue	Wen	Thu	Fri	Sat	Sun
1	D	D	A	A			
2	A	A	A	N	N	N	
3		D	D	N	N	N	
4		A	A	A	A		
5			D	D	D	D	D
6	D			D	D	D	N
7	N				A	A	N
8	N	N				A	A
9	A	N	N				

be found. Our main concern in this step is to generate all possible class solution, or as many as possible for large instances of the problem.

A class solution is nothing but an integer partition of the sum of all working days scheduled for one employee during the entire planning period. To find all possible class solutions in this step we have to deal with the following two problems:

- Generation of restricted partitions, and
- Elimination of those partitions for which no schedule can be created.

Because the elements of a partition represent lengths of work blocks and because constraints about maximum and minimum length of these work blocks are given, not all partitions must be generated. Still, the maximum and minimum lengths of days-off blocks impact the maximum and minimum permitted number of elements in one partition, since between two work blocks there always is one block of days-off, or a block for recreation. In summary, partitions that fulfill the following criteria have to be generated:

- Maximum and minimum value of elements in a partition. These two parameters are respectively maximum and minimum permitted length of work blocks,
- Minimum number of elements in a partition:

$$MINB = \left\lceil \frac{DaysOffSum}{MAXS(m)} \right\rceil$$

DaysOffSum: Sum of all days-off that one employee has scheduled during the entire planning period.

- Maximum number of elements in a partition:

$$MAXB = \left\lfloor \frac{DaysOffSum}{MINS(m)} \right\rfloor$$

The set of partitions that fulfill the criteria given above is a subset of the set of all possible partitions. It is possible to first generate the full set of all possible partitions and then eliminate those that do not fulfill the constraints given by the criteria. However, this approach is inefficient for large instances of the problem. Our idea was it to use restrictions for pruning while the partitions are generated. We implemented a procedure based on this idea for the generation of restricted partitions. This procedure searches for all legal paths (legal partitions) in a search tree. Levels of the tree represent components of the partition and branches correspond to assignments of possible work blocks to components. To prune the search tree, the procedure uses the information about the maximum and minimum number of elements in a partition, the sum of elements of a partition, and the fact that the values of the components of the partition should be in decreasing order.

Pseudo code of procedure we use is given below. The set P contains elements of a partition of N.

Initialize N, $MAXB$, $MINB$, $MAXW$, $MINW$
'Value of arguments for first procedure call
$Pos = 1$, $MaxValue = MAXW$

'Recursive procedure
RestrictedPartitions($Pos, MaxValue$)

    $i = MINW$
    Do While ($i <= MaxValue \wedge \neg PartitionIsCompleted$)

        Add to the set P element i
        $PSum$ = Sum of elements in a set P

        If ($PSum = N \wedge Pos \geq MINB$) Then
            $PartitionIsCompleted$ = true
            Store partition (set P)
        'Pruning
        ElseIf ($Pos < MAXB \wedge PSum \leq N - MINW$) Then

'Recursive call
RestrictedPartitions($Pos + 1, i$)
EndIf
$i = i + 1$
Remove last element from set P

Loop

End

Not all restricted partitions can produce a legal schedule that meets the requirements of work force per day (in this step we only test whether we have the desired number of employees for a whole day, not for each shift).

Example: Supposing we have to find a one week schedule for 9 employees so that every day 6 employees are present in three shifts. Additionally, suppose that the length of work blocks can range from 4 to 7 and the length of days-off blocks from 2 to 4. Possible class solutions are restricted partitions of the number 42 = 7 days × 6 employees. Some of these restricted partitions are legal, some are not. For example, with restricted partition {6 6 5 5 5 5 5 5} a legal schedule can be generated, while this is not possible for the restricted partition {7 5 5 5 5 5 5 5}. For the latter no days-off distribution that can produce a legal schedule subject to the day requirements (6 employees) exists.

As we only want to have class solutions that get us a legal shift plan, we eliminate all restricted partitions that cannot fulfill the work force per day requirements. A restricted partition will be legal if at least one distribution of days-off exists that fulfills the work force per day requirements. In the worst case all distributions of days-off have to be tested if we want to be sure that a restricted partition has no legal days-off distribution. It is possible to first generate all days-off distributions and then test each permutation of restricted partitions if at least one satisfying days-off distribution can be found. This approach is rather ineffective when all class solution have to be generated, as many of them will not have legal days-off distribution and thus the process of testing takes too long for large instances of typical problems. We implemented a backtracking algorithm for testing the restricted partitions. The procedure is a search for a first legal path (distribution of work and days-off blocks that fulfills the requirements per day) in a tree. The levels of the tree represent blocks (odd levels represent work blocks and even levels represent days-off blocks) and the branches of the tree correspond to the allocation of blocks of work and days-off. To prune the

search tree, the procedure uses its knowledge of the number of employees needed each day and the permitted number of blocks of work and days-off of each type. Additionally, as we want to obtain the first class solutions as soon as possible, we implemented a three stages time restricted test. Consequently no time is lost with restricted partitions which do not have any legal days-off distribution at the beginning of the test.

The pseudo code of algorithm for testing the restricted partitions is given below (without the time restriction).

INPUT: Restricted partition, possible days-off blocks.

Initialize vectors $W(NumberOfUniqueWorkBlocks)$ and $F(NumberOfUniqueDaysOffBlocks)$ with unique work blocks, respectively with unique days-off blocks (for example unique work blocks of class solution {5 5 4 4 3} are blocks 5, 4 and 3).

'i represents one work or days-off block. It takes values from 1 to $numberOfWorkBlocks * 2$ (after each work block comes a days-off block and our aim is to find the first schedule that fulfills the requirements per day). For the first procedure call $i = 1$
'Recursive procedure
PartitionTest(i)

$k = 1$
If i is odd

Do While($k \leq NumberOfUniqueWorkBlocks$)
Assign block i with work block $W(k)$

If $i = LastBlock - 1$ then
Req= for each day the number of employees does not get larger than the requirement and the number of work blocks of type $W(k)$ does not get larger than the number of blocks of type $W(k)$ in the class solution
'Pruning
If Req =true then
PartitionTest($i + 1$)
Endif

Else

'Only partial schedule until block i is tested

Req = requirements for number of employees during each day are not overfilled and number of work blocks of type $W(k)$ does not get larger than the number of blocks of type $W(k)$ in the class solution

If Req =true then

PartitionTest$(i+1)$

Endif

Endif

$k = k + 1$

Loop

Else

Do While$(k \leq NumberOfUniqueDaysOffBlocks)$

Assign block i with days-off block $F(k)$

If $i = lastblock$ then

$SumTest$ =Test if sum of all days-off is as required

If $SumTest$ =true then

Class solution has at least one days-off distribution

Interrupt test

Endif

Else

'Only partial schedule until block i is tested

$FreeTest$ =Test if not more employees than required have free (test is done for each day)

'Pruning

If $FreeTest$ =true then

PartitionTest$(i+1)$

Endif

Endif

$k = k + 1$

Loop

Endif

End

4.2.2 Determination of distribution of blocks of work and days-off that have optimal weekend characteristics

Once the class solution is known, different shift plans can be produced subject to the order of work blocks and to the distribution of days-off. For each order of blocks of the class solution there might be various distributions of days-off. At this point we introduce a new soft constraint. This constraint is relevant for weekends off. It is our intention to find the best solution (or more solutions if they are not dominated) for each order of work blocks for weekends off. We want to maximize the number of weekends off, to maximize the number of long weekends off (the weekend plus Monday or Friday is free) and to find solutions that have a "better" distribution of weekends off. Distribution of weekends off will be evaluated by the following method: every time two weekends off appear directly after each other the distribution gets a negative point. A certain distribution of weekends off is better than other distributions, if it has less negative points. Priority is given to the number of weekends off followed by the distribution of weekends off. Finally, the number of long weekends off is considered only if the others are equal. Possible candidates are all permutations of the work blocks found in a class solution. Each permutation may or may not have days-off distributions. If the permutation has at least one days-off distribution it is then our aim to find the best solutions for weekends off. The best solutions are those that cannot be dominated by another solution. We state that solution $Solut_1$ dominates solution $Solut_2$ in the following cases:

- $Solut_1$ has the same number of weekends off as $Solut_2$, the evaluation of the weekends distribution of $Solut_1$ is equal to the one of $Solut2$, and $Solut_1$ has more long weekends off than $Solut_2$.
- $Solut_1$ has the same number of weekends off as $Solut_2$ and the evaluation of the weekends distribution of $Solut_1$ is better than the one of $Solut2$.
- $Solut_1$ has more weekends off than $Solut_2$.

At this point two remarks have to be made. First, because some of the permutations of the class solutions may not have any days-off distribution, we

use time restrictions to find days-off distributions. In other words, if the first days-off distribution is not found in a predetermined time, the next permutation is tested. Second, for large instances of problems, too many days-off distributions may appear and this may impede the search for the best solution. Interrupting the test can be done manually depending on the size of the problem.

For large instances of the problem it is in practice impossible to generate all permutations of class solutions and the best days-off distributions for each permutation in a reasonable amount of time. In these cases our main concern is to enumerate as many solutions as possible which have the best days-off distribution and can be found in a predetermined time. All solutions found are arranged based on weekend attributes so that the user can easily decide which distribution of days-off and workdays he or she wants to continue with. The user may select one of the solutions solely based on weekends, but sometimes the order of work blocks may also be a deciding factor. For example, one can prefer the order of work blocks (7 6 3 7 6 3 7 6) to the order (7 7 7 6 6 6 3 3).

For finding legal days-off distributions for each permutation of a class solution we use a backtracking procedure similar to the one for testing the restricted partitions in step 1, except for here the distribution of work blocks is already set. After the days-off distributions for a given order of work blocks are found, searching the best solutions based on weekend characteristics is a comparatively trivial task and does not take very long.

Selected solutions in step 2 have a set distribution of work blocks and days-off blocks, and in the final step the assignment of shifts to the work blocks has to be done.

4.2.3 Generating permitted shift sequences for each work block

In step 2, work and days-off blocks have been fixed. We still have to assign shifts to the employees. Again we use a backtracking algorithm, but to make this algorithm more efficient we introduce another intermediate step. The basic idea of this step is the following: For each work block, construct the possible sequences of shifts subject to the shift change constraints and the upper and lower bounds on the length of sequences of consecutive shifts of the same kind. Because of these constraints, the number of these sequences (we will call them terms) is not too large. Therefore, our backtracking algorithm — which only manipulates this limited set of terms — is much more efficient than the classical backtracking approach, where for each position in the work blocks all shift possibilities would have to be tried and the test for shift change constraints would have to be done in a much more time-consuming manner.

Example: Suppose the solution selected by the user in step 2 has the distribution of work blocks (6 4 4 6 5 4 5).

Shifts: Day (D), Afternoon (A) and Night (N).

Inadmissible shift changes: (N D), (N A), (A D).

Minimum and maximum lengths of consecutive shifts: D: 2-6, A: 2-5, N: 2-4.

Our task is it to create legal terms for work blocks of length 6, 5, and 4.

For work block of length 6 the following terms exist:
DDDDDD, DDDDAA, DDDDNN, DDDAAA, DDDNNN, DDAAAA, DDNNNN, DDAANN, AAAANN, AAANNN, AANNNN.

Block of length 5:
DDDDD, DDDAA, DDDNN, DDAAA, DDNNN, AAAAA, AAANN, AANNN.

Block of length 4:
DDDD, DDAA, DDNN, AAAA, AANN, NNNN.

This approach is very appropriate when the number of shifts is not too big. When the number of shifts is big we arrange shifts with similar characteristics in groups of so-called shift types. For example if there is a separate day shift for Saturday that starts later than the normal day shift, these two shifts can be grouped together. This integration of similar shifts into shift types allows us to have a smaller number of terms per work blocks and therefore reduces the overall search space. At the end a transformation from shift types to the substituted shifts has to be done. A similar approach has been applied by Weil and Heus [69]. They group different days-off shifts into one shift type and thus reduce the search space. Different days-off shifts can be grouped into one days-off shift only if they are interchangeable (the substitution has no impact on constraints or evaluation).

The process of constructing the terms usually does not take long given that the length of work blocks in the vast majority of cases is less than 9 and some basic shift change constraints always appear because of legal working time restrictions.

4.2.4 Assignment of shift sequences to work blocks

Once we know the terms we can use a backtracking algorithm to find legal solutions that satisfy the requirements for each shift during each day. The size of the

search space that should be searched with this algorithm is:

$$\Pi_{i=1}^{b} N_t(i)$$

where b is the number of work blocks and $N_t(i)$ is the number of legal terms for block i.

If we would not use terms the search space would be of size:

$$(m-1)^{SumOfAllWorkDays}$$

Of course we would have more constraints, in the latter case for instance the shift change constraints, but the corresponding algorithm would be much slower because the constraints would have to be tested not only once as in our solution.

The procedure we implemented in this step is a search for all legal paths (legal schedules) in a search tree. The levels of the tree represent work blocks and the branches of the tree correspond to the allocation of terms (a sequence of shifts). To prune, the procedure uses the information about the needed number of employees for each shift (for each day). Say the terms test for shift change constraints is done without consideration of shift a_m (days-off). If there are shift change constraints that include days-off, the test of the solution has to be done later for these sequences.

Pseudo code for the backtracking algorithm based on terms is given below.

INPUT: distribution of work and days-off blocks

Generate all legal shift sequences for each work block
'Value of argument for first call of procedure ShiftAssignment
$i = 1$

'Recursive procedure
ShiftAssignment(i)

j = Number of shift sequences of block i
$k = 1$
Do While ($k \leq j$)

Assign block i with sequence number k
If $i = lastblock$ then

Req = Test if requirements are fulfilled and shift change constraints are not violated (in this stage we test for forbidden shift sequences that include days-off)
If *Req* = true then
Store the schedule
Endif
Else
'Only partial schedule until block i is tested
PTest=Test each shift if not more than needed employees are assigned to it
'Pruning ...
If *Ptest* =true then
ShiftAssignment$(i+1)$
Endif
Endif
$k = k + 1$
Loop
End

There are rare cases when, even if there is a work and days-off distribution, no assignment of shifts can be found that meets the temporal requirements for each shift on each day because of shift change constraints. In these cases constraints about minimum and maximum length of periods of consecutive shifts must be relaxed to obtain solutions.

4.3 Computational results

In this section we report on computational results obtained with our approach. We implemented our four step framework in a software package called First Class Scheduler (FCS) which is part of a shift scheduling package called Shift-Plan-Assistant (SPA) of XIMES[1] Corp. All our results in this section have been obtained on an Intel Pentium II 333 MHz based computer. Our first two examples are taken from a real-world sized problems and are typical for the kind

[1] http://www.ximes.com/

Table 4.2: First Class Scheduler solution for problem 1

Employee/day	Mon	Tue	Wed	Thu	Fri	Sat	Sun
1	D	D	D	D	D	D	
2			D	D	D	D	
3	D	D	D	D			
4	D	D	D	D	D	D	
5	D	D			D	D	

of problems for which FCS was designed. Further, we compared our results with results from a paper of Balakrishnan and Wong [8]. They solved problems of rotating workforce scheduling through the modeling in a network flow problem. Their algorithms were implemented in Fortran on an IBM 3081 computer. They applied their technique to several examples taken from previous literature. We compare our results to three benchmark problems.

Problem 1: An organization operates in one 8 hours shift: Day shift (D). From Monday to Saturday 4 employees are needed, whereas on Sunday no employees are needed. These requirements are fulfilled with 5 employees which work on average 38,4 hours per week. A rotating week schedule has to be constructed which fulfills the following constraints:

1. Length of periods of successive shifts should be: D: 2-6
2. Length of work blocks should be between 2 and 6 days and length of days-off blocks should be between 1 and 4
3. Features for weekends off should be as good as possible

We note that days-off blocks of length 1 are not preferred, but otherwise no class solution for this problem would exist.

Using FCS in step 1, all class solutions are generated in 4.5 seconds: {6 4 4 3 3 2 2}, {6 6 3 3 2 2 2}, {6 6 4 3 3 2}, {6 6 4 4 2 2}, {6 6 6 2 2 2}, {6 6 6 3 3}. We select the class solution {6 6 4 4 2 2} to proceed in the next step. In step 2 FCS generates 6 solutions after 0.6 seconds. Each of them has one long weekend off. We select a solution with the distribution of work blocks (6 4 4 6 2 2) to proceed in the next steps. Step 3 and 4 are solved automatically. We obtain the first and only existing schedule after 0.02 seconds. This solution is shown in Table 4.2.

The quality of this schedule stems from the fact that there are at most 8 consecutive work days with only a single day-off in between them. This constraint is very important when single days-off are allowed. This example showed a small instance of a problem with only one shift; nevertheless, even for a such instances it is relatively difficult to find high quality solutions subject to this constraint.

Let us note here that the same schedule can be applied for a multiple of 5 employees (the duties are also multiplied) if the employees are grouped in teams. For example if there are 30 employees they can be grouped in 5 teams. Each of them will have 6 employees.

Problem 2: An organization operates in three 8 hours shifts: Day shift (D), Afternoon shift (A), and Night shift (N). From Monday to Friday three employees are needed during each shift, whereas on Saturday and Sunday two employees suffice. These requirements are fulfilled with 12 employees which work on average 38 hours per week. A rotating week schedule has to be constructed which fulfills the following constraints:

1. Sequences of shifts not allowed to be assigned to employees are:
   (A D), (N D), (N A)
2. Length of periods of successive shifts should be: D: 2-7, A: 2-6, N: 2-5
3. Length of work blocks should be between 4 and 7 days and length of days-off blocks should be between 2 and 4
4. Features for weekends off should be as good as possible

Using FCS in step 1, the first class solution is generated after 0.07 seconds and we interrupt the process of generation of class solutions after 1.6 seconds when already 7 class solutions have been generated out of many others: {6 6 5 5 5 5 5 5 5 5}, {6 6 6 5 5 5 5 5 5 5 4}, {7 6 5 5 5 5 5 5 5 5 4}, {7 6 6 5 5 5 5 5 5 4 4}, {7 7 5 5 5 4 4 4 4 4 4 4}, {7 7 7 6 5 5 5 5 5 5}, and {7 7 7 6 6 5 5 5 5 4}. The first solution has the highest number of most optimal blocks, namely those with length 5, but entails weak features for weekends off. For this reason we select the class solution {7 7 7 6 5 5 5 5 5 5} to proceed in the next step. In step 2 the optimal solution for the distribution of blocks (7 7 7 6 5 5 5 5 5 5), with 6 weekends off from which 3 are long, is found in less than 2 seconds. The first 11 solutions are generated in 11 seconds, where we have one solution with 6 weekends off, 4 of which are long, and a distribution of weekends off that is acceptable. This solution has the order of work blocks as follows: (7 7 6 5 5 5 7 5 5 5). We select this solution to proceed in the next steps. Step 3 and 4

Table 4.3: First Class Scheduler solution for problem 2

Employee/day	Mon	Tue	Wed	Thu	Fri	Sat	Sun
1	D	D	D			A	A
2	A	A	A	A			
3	D	D	D	D	D		
4			A	A	A	N	N
5					N	N	N
6	N	N			A	A	A
7	A	A	N	N			
8	A	A	A	A	A		
9	N	N	N	N	N		
10	N	N	N	N	N		
11		D	D	D	D	D	D
12	D			D	D	D	D

are solved automatically. We obtain a first schedule which is given in Table 4.3 after 0.17 seconds and the first 50 schedules after 4 seconds. The decision maker can eliminate some undesired terms. Besides these solutions there exist also a large amount of other solutions which differ in terms with each other. If a better distribution of weekends off would have been sought, this could have been found through another class solution, for example {7 7 7 7 7 7 5 5 5} found in step 1 after 16 seconds, at the cost of longer work sequences.

Problem 3: The first problem from literature for which we discuss computational results for First Class Scheduler is a problem solved by Butler [15] for the Edmonton police department in Alberta, Canada. Properties of this problem are:
Number of employees: 9
Shifts: 1 (Day), 2 (Evening), 3 (Night)

Temporal requirements:

$$R_{3,7} = \begin{pmatrix} 2 & 2 & 2 & 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 3 & 3 & 3 & 2 \\ 2 & 2 & 2 & 2 & 2 & 2 & 2 \end{pmatrix}$$

Constraints:

- Length of work periods should be between 4 and 7 days

Table 4.4: Solution of Balakrishnan and Wong [8] for the problem from [15]

Employee/day	Mon	Tue	Wed	Thu	Fri	Sat	Sun
1	A	A	A	A	A	A	
2		N	N	N	N	N	
3			A	A	A	A	A
4			D	D	D	D	D
5	D	D			N	N	N
6	N			A	A	A	A
7	A	A			D	D	D
8	D	D	D	D			N
9	N	N	N	N			

- Only shift 1 can precede the rest period preceding a shift 3 work period
- Before and after weekends off, only shift 3 or shift 2 work periods are allowed
- At least two consecutive days must be assigned to the same shift
- No more than two 7-day work periods are allowed and these work periods should not be consecutive

Balakrishnan and Wong [8] solve this problem using a network model and they needed 73.54 seconds to identify an optimal solution of the problem. This solution is given in Table 4.4. We use D for 1, A for shift 2, N for shift 3, and if the element of the matrix is empty the employee has free.

Before we give our computational results some observations should be made. First constraint two and three cannot be represented in our framework. Let us note here that in all three examples given, we cannot model the problem exactly (the same was true for Balakrishnan and Wong's [8] approach to the original problems), which is to a high degree due to the different legal requirements found in the U.S./Canadian versus those found in the European context, but we tried to mimic the constraints as closely as possible or to replace them by similar constraints that appeared more meaningful in the European context. Having said this, let us proceed as follows: The other constraints can be applied in our model and are left like in the original problem. As mentioned, we include additional constraints about maximum length of successive shifts and minimum and maximum length of days-off blocks. In summary, additional constraint used for First Class Scheduler are:

Table 4.5: First Class Scheduler solution for the problem from [15]

Employee/day	Mon	Tue	Wed	Thu	Fri	Sat	Sun
1	D	D	D			D	D
2	D	N	N			A	A
3	A	N	N	N	N		
4	A	A	A	A	A		
5		D	D	N	N	N	
6		A	A	A	A	A	A
7				D	D	N	N
8	N			A	A	A	N
9	N			D	D	D	D

- Not allowed shift changes: (N D), (N A), (A D)
- Length of days-off periods should be between 2 and 4
- Vector $MAXS_3 = (7, 6, 4)$

In our model we first generate class solutions. Class solutions that exist for the given problem and given constraints are:

{6 6 6 6 6 5 5 5}, {7 6 6 6 6 5 5 4}, {7 7 7 7 7 5 5}, {6 6 6 6 6 6 5 4}, {7 6 6 6 5 5 5 5}, {7 6 6 6 6 6 4 4}, {7 7 6 6 5 5 5 4}, {7 7 6 6 6 5 4 4}, {7 7 7 4 4 4 4 4 4}, {7 7 7 5 5 5 5 4}, {7 7 7 6 5 5 4 4}, {7 7 7 6 6 4 4 4}, {7 7 7 6 6 6 6}, {7 7 7 7 5 4 4 4}, {7 7 7 7 6 6 5}, {7 7 7 7 7 6 4}, {7 7 6 5 5 5 5 5}.

The first solution is generated in 0.14 seconds and all solutions in 4.38 seconds. We select in this step the class solution with the highest number of optimal blocks: {7 7 6 5 5 5 5 5}

In step 2 the distributions of work and days-off periods that gives best results for weekends is found. We select the best solution offered in this step from First Class Scheduler that has this order of work blocks (7 5 7 5 5 6 5 5). The computations of the system took 0.73 seconds.

Step 3 and 4 are solved automatically. The first solution given in Table 4.5 is generated after 0.39 seconds and the first 50 solutions in 4.29 seconds.

There exist also many other solutions that differ only in terms that they contain. Undesired solutions can then be eliminated through the elimination of

unwanted terms.

Problem 4 (Laporte et al. [44]): There exist three non overlapping shifts D, A, and N, 9 employees, and requirements are 2 employees in each shift and every day. A week schedule has to be constructed that fulfills these constraints:

1. Rest periods should be at least two days-off
2. Work periods must be between 2 and 7 days long if work is done in shift D or A and between 4 and 7 if work is done in shift N
3. Shift changes can occur only after a day-off
4. Schedules should contain as many weekends as possible
5. Weekends off should be distributed throughout the schedule as evenly as possible
6. Long (short periods) should be followed by long (short) rest periods
7. Work periods of 7 days are preferred in shift N

Balakrishnan and Wong [8] need 310.84 seconds to obtain the first optimal solution. The solution is given in Table 4.6. The authors report also about another solution with three weekends off found with another structure of costs for weekends off.

In FCS constraint 1 is straightforward. Constraint 2 can be approximated if we take the minimum of work blocks to be 4. Constraint 3 can also be modeled if we take the minimum length of successive shifts to be 4. For maximum length of successive shifts we take 7 for each shift. Constraints 4 and 5 are incorporated in step 2, constraint 6 cannot be modeled, and constraint 7 is modeled by selecting appropriate terms in step 3.

With these given parameters for this problem there exist 23 class solutions which are generated in 5 seconds. For each class solution there exist at least one distribution of days-off, but it could be that no assignment of shifts to the work blocks exist because the range of blocks with successive shifts are too narrow in this case. Because in this problem the range of lengths of blocks of successive shifts is from 4 to 7 for many class solution no assignment of shifts can be found. Class solution {7 7 6 5 5 4 4 4} gives solutions with three free weekends, but they are after each other. Class solution {7 7 7 7 5 5 4} gives better distribution

Table 4.6: Solution of Balakrishnan and Wong [8] for the problem from [44]

Employee/day	Mon	Tue	Wed	Thu	Fri	Sat	Sun
1	A	A	A	A			D
2	D	D	D	D			
3		D	D	D	D	D	
4			A	A	A	A	A
5			N	N	N	N	N
6	N	N			A	A	A
7	A	A			D	D	D
8	D			N	N	N	N
9	N	N	N				

Table 4.7: First Class Scheduler solution for the problem from [44]

Employee/day	Mon	Tue	Wed	Thu	Fri	Sat	Sun
1	D	D	D	D	D		
2		D	D	D	D	D	D
3	D			N	N	N	N
4					A	A	A
5	A	A	A	A			
6	N	N	N	N	N		
7			A	A	A	A	A
8	A	A				N	N
9	N	N	N			D	D

of weekends. If we select this class solution in step 1, our system will generate 5 solutions in step 2 in 1.69 seconds. We selected a solution with the order of work blocks to be (7 7 4 7 5 7 5). Step 3 and 4 are solved simultaneously and the first solution was arrived at after 0.08 seconds. 18 nonisomorphic solutions were found after 0.5 seconds. One of the solutions is shown in Table 4.7.

With class solution {7 7 7 7 7 7} the same distribution of weekends off can be found as in [44].

As we see we can arrive at the solutions much faster than Balakrishnan and Wong [8], though in interaction of the human decision maker. Because each step is very fast, the overall process of constructing an optimal solution still does not

take very long.

Problem 5: This problem is a larger problem first reported in [33]. Characteristics of this problem are:

Number of employees is 17 (length of planning period is 17 weeks).

Three nonoverlapping shifts.

Temporal requirements are:

$$R_{3,7} = \begin{pmatrix} 5 & 4 & 4 & 4 & 4 & 4 & 3 \\ 5 & 4 & 4 & 4 & 4 & 4 & 4 \\ 4 & 3 & 3 & 3 & 4 & 4 & 4 \end{pmatrix}$$

Constraints:

- Rest-period lengths must be between 2 and 7 days
- Work-periods lengths must be between 3 and 8 days
- A shift cannot be assigned to more than 4 consecutive weeks in a row
- Shift changes are allowed only after a rest period that includes a Sunday or a Monday or both
- The only allowable shift changes are 1 to 3, 2 to 1, and 3 to 2

Balakrishnan and Wong [8] need 457.98 seconds to arrive at the optimal solution which is given in Table 4.8.

With First Class Scheduler we cannot model constraints 3, 4, and 5 in their original form. We allow changes in the same block and for these reason we have other shift change constraints. In our case the following shift changes are not allowed: 2 to 1, 3 to 1, and 3 to 2. Additionally, we limit the rest period length from 2 to 4 and work periods length from 4 to 7. Maximum and minimum length of blocks of successive shifts are given with vectors $MAXS_3 = (7, 6, 5)$ and $MINS_3 = (2, 2, 2)$.

With these conditions the first class solution {6 6 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5} is found after 0.22 and the first 9 solutions after 14.2 seconds. Of course there exist much more class solutions, but finding all class solutions will take too much time for this large problem. If we choose the first solution with the most optimal blocks we will obtain solutions with 5 weekends off, even though the weekends are one after each other. We arrive at a better solution with the following class solution: {7 6 5 5 5 5 5 5 5 5 5 5 5 5 5 4}. In step 2 we stop the process of

Table 4.8: Solution of Balakrishnan and Wong [8] for the problem from [33]

Employee/day	Mon	Tue	Wed	Thu	Fri	Sat	Sun
1	D	D	D	D	D	D	D
2	D			D	D	D	D
3	D	D	D				
4	D	D	D	D	D	D	
5				N	N	N	N
6	N	N			N	N	N
7	N	N	N				
8	A	A	A	A	A	A	A
9	A			A	A	A	A
10	A	A	A				
11	D	D	D	D	D	D	D
12			N	N	N	N	N
13	N				N	N	N
14	N	N	N	N			
15		A	A	A	A	A	A
16	A			A	A	A	A
17	A	A	A				

generation of distributions of work and days-off blocks after 20 seconds and we get 3 solutions. From these solutions we select the solution with the order of blocks (7 6 5 5 5 5 5 5 5 5 5 5 5 4 5 5). The first solution (steps 3 and 4) is generated after 0.73 seconds and the first 50 solutions after 4.67 seconds. The first solution is given in Table 4.9.

As you can see the solution has a much worse distribution of weekends than the solution from [8] but our solution has no blocks of length 8 and has many optimal blocks (of length 5). Our solutions also have not more than 5 successive night shifts (seven night shifts are considered too much).

Much better distributions of weekends-off can be found with FCS if the maximum length of work days is increased to 8. In this case step 2 of FCS takes longer because it depends directly on the number of blocks.

One disadvantage of FCS is that the user has to try many class solutions to find an optimal solution. However, the time to generate solutions in each step is so short that interactive use is possible. Other advantages of interactively solving these scheduling problems is the possibility to include the user in the decision

Table 4.9: First Class Scheduler solution for the problem from [33]

Employee/day	Mon	Tue	Wed	Thu	Fri	Sat	Sun
1	D	D	D	D	D	D	D
2			D	D	D	D	D
3	D			D	D	D	D
4	D			D	D	D	A
5	A			A	A	A	A
6	A			A	A	A	N
7	N				A	A	A
8	A	A				A	A
9	A	A	A			N	N
10	N	N	N			N	N
11	N	N	N			N	N
12	N	N	N				
13	D	D	D	N	N		
14		A	A	A	N	N	
15		D	D	N	N		
16	D	D	A	A	A		
17	A	A	A	N	N		

process. For example one may prefer longer blocks but better distribution of weekends to shorter work blocks but worse distribution of weekends.

One disadvantage of FCS is that the user has to try many class solutions to find an optimal solution. However, the time to generate solutions in each step is so short that interactive exploration of alternatives is possible. Another advantage of the interactive solving of scheduling problems of this type is the possibility to include the user in the decision-making process. For example one may prefer longer work blocks but better distribution of weekends to shorter work blocks but worse distribution of weekends. Moreover, this interaction facilitates a better understanding of how requirements can shape the solution space. The system thereby helps to relax requirements when the solution space is very tight. For some well-known scheduling problems with only very few good solutions (e.g., metropolitan and continental rota [22]) FCS finds exactly these required solutions. However, further work will be needed to improve early identification of dead ends.

With FCS one can model all the important constraints that arise with shift scheduling problems in the central European context. The different legal require-

ments in the U.S. have yet to be taken into account (see [24]). Nevertheless, the package is already internationally recognized and German, English, Finnish, and soon Dutch versions are available.

Chapter 5

Local search for shift design

Designing shifts is one of the important stages in the general workforce scheduling problem. For the problem of shift design, we are given the workforce requirements for a certain period of time, constraints about the possible start and the length of shifts, and an upper limit for the average number of duties per week per employee. The aim is to generate solutions that contain shifts (and the number of employees per shift) that fulfill all hard constraints about the shifts, as well as minimize the number of shifts, over- and understaffing, and differences in the average number of duties per week. In this chapter we consider solving this problem by using iterative improvement methods. First we propose repair steps (moves) to explore the neighborhood of solutions for this problem. In order to generate the neighborhood and accept the solution for the next iteration, basic principles of the tabu search technique are used. However, while tabu search can find acceptable solutions for this problem, the complete exploration of the neighborhood (using all defined moves) in each iteration is very time consuming. We proposed a new approach in combination with tabu search in order to make the search more effective. The basic idea is to exploit the knowledge about the shift design problem during the search. Based on the temporal workforce of the current solution and given workforce requirements (distance of the current solution to the optimal solution with respect to the most important criteria), selecting repair steps during each iteration is guided so that the repair steps that have a greater chance to improve the solution are used. As a result of this knowledge about the problem, the neighborhood is selectively explored during each iteration, which makes the search more effective. Furthermore, we propose an algorithm for generating a good initial solution, which further improves the effectiveness of the search. Computational results in a real life problems and in randomly generated problems show the advantages of these ingredients. The

algorithms are part of a commercial product and the system has shown to work well in real life problems.

5.1 Introduction

Like we described in the Chapter 3 the typical process of planning and scheduling a workforce in an organization consists of several stages [66]. The fist stage in this process is to determine the temporal requirements. In this stage the number of needed employees of each qualification is found for every time slot of the planning period (e.g., every Monday between 6:00-10:00 there should be 4 employees at work). After this stage one can proceed to determine the total number of employees needed, interrelated with designing the shifts and then assigning these shifts and days-off to employees. There exist different approaches in the literature how to solve these stages. One of the approaches is to coordinate the design of the shifts and the assignment of the shifts to the employees, and to solve it like one problem [29, 37]. Other approaches consider days-off scheduling and shift scheduling only after the shifts are designed [8, 46, 55]. One of the disadvantages of the second approach is that considering the design of shifts separately does not guarantee that a feasible solution for the assignment of these shifts can be found. However, solving the workforce scheduling problem in several separate stages makes the problem of general workforce scheduling easier to tackle.

In this chapter we consider the problem of designing the shifts. We have given a detailed description of the problem we consider in this thesis in Chapter 3. Let us briefly remind here the definition of this problem. For this problem we are given:

- The temporal requirements for each slot for each day during the planning period.
- A collection of *shift types*. A shift type determines the minimum and maximum length and the earliest and latest start of a shift.
- An upper limit for the average number of working shifts per week per employee.

The aim is to generate a set of shifts which fulfill all constraints specified by the shift types and the number of workers for each day such that the following numbers are minimized:

- Sum of the excesses of workers in each time interval during the planning period.

- Sum of the shortages of workers in each time interval during the planning period.
- Number of shifts k.
- Distance of the average number of duties per week in case it is above a certain threshold.

The problem that includes constraints for the minimization of the number of shifts is NP hard and also hard to approximate. This is the reason why we rely on local search techniques for solving this problem.

The appropriate neighborhood structure is one of the most important features to reach 'good' solutions for local search techniques. In this chapter we propose moves (repair steps) that will be used to explore the neighborhood of the current solution. We use basic principles of the well-known tabu search method for the exploration of the neighborhood, where tabu solutions will not be accepted for a number of iterations to make possible escaping from a local optimum, except if the solution has a better cost than the best current one. A procedure for generating a good initial solution for this problem was also developed. In order to make the search more effective, we introduced a new method that guides the search during the exploration of the neighborhood. A basic feature of the guided search procedure is that the search is concentrated only in the days in which a shortage or excess is present. Moreover, some of the moves are applied only to specific shift types in the region in which the shortage or excess appears. In the computational results, we show results of experiments concerning different kinds of tabu mechanisms, lengths of the tabu list, initial solutions etc. Computational results show that the effectiveness of the search and the quality of the solution is improved by including the knowledge about the problem during the search and also the initial solution. These techniques were implemented in a commercial product called Operating Hours Assistant.

5.2 Local search for shift design

The basic idea of local search techniques [2, 26, 27, 28, 39, 59] is to improve initial solution iteratively during the search. A descendant of the current solution is selected from the neighborhood of the solution, which is constructed through changing the current solution using so called 'moves'. Basically, the differences between individual local search techniques are found in the way they explore the neighborhood and the criteria on how to accept the descendant of the current solution. The appropriate neighborhood structure for these techniques is one

of the most important features to reach 'good' solutions. In this section, we first give definitions of the moves used for exploring the neighborhood for the shift design problem. Afterwards, we describe other features of the technique we used concerning the generation of neighborhood, acceptance of the descendant solutions and aspiration criteria.

5.2.1 Neighborhood relations

A solution in the shift design problem consists of shifts and the duties of the shifts for each day during the planning period. The neighborhood of the solution is obtained by introducing altogether new shifts, removing a shift, or changing one of the three main features of a shift, namely, start, length, and duties per day. Basic moves presented below change these features.

***'ChangeDuty'*:**

The duties of the shift for the particular day are decreased or increased for 1 employee. For example, suppose that shift F1 was assigned 5 employees during the first day. By applying this move to shift F1 (on the first day) two new shifts that respectively are assigned 4 and 6 employees in the first day can be obtained.

***'ChangeLength'*:**

The length of the shift is made longer or shorter for one time unit. For example suppose that the length of shift F1 is 8 hours and a time unit of 30 minutes is used. By applying this move to shift F1, two new shifts with lengths 7,5 h and 8,5 h can be obtained.

***'ChangeStart'*:**

The starting time of a shift is moved forward or backward by one time unit (the length is left unchanged). For example, suppose that the shift F1 begins at 8:00 and the time unit is 15 minutes. By applying this move to shift F1 two new shifts can be obtained, starting at 8:15 and 7:45.

***'CreateShift'*:**

A new shift not having all duties empty is created. This move is implemented indirectly, such that for each shift type we always keep a so called 'reserve' shift which has all duties 0 and exists only as a virtual shift in the solution. A new shift can be created for example by applying the move *'ChangeDuty'* to the reserve shift. From the following two other moves that will be defined later, new shifts can be created too: *'ExchangeDuties'* and *'SplitShift'*.

***'RemoveShift'*:**

One shift is removed from the solution. This move is implemented in an indirect way, such that only one reserve shift (for each shift type) is kept in a solution. Double shifts are always joined every second iteration.

The moves we defined before are basic moves. Next, we define the composite moves, which are combinations of basic moves.

***'MoveBorders'*:**

The borders of two shifts is shifted right or left. The starting time of first shift (shift that begins earlier) and the end of the second shift remain fixed. For example, suppose that the solution contains shifts F1: (6:00-14:30) and S1: (14:30-23:00). By shifting the border of these two shifts for 30 minutes backwards, two new shifts will be created: F1: (6:00-14:00) and S1: (14:00-23:00)

***'ExchangeDuties'*:**

In a particular day, one duty is passed from one shift to the other. For example, suppose that shift F1 has these duties (2 2 3 3 3 3 0) and shift F2 these duties (1 1 1 1 1 1 1), where each element of array represents duties of shifts for particular day. Shifts F1 and F2 will have new duties by applying this move on these shifts and day 1: F1: (1 2 3 3 3 3 0); F2: (2 1 1 1 1 1 1).

***'JoinShifts'*:**

Whole duties of one shift (X) are added to the other one and the shift X is eliminated from the solution. For example, suppose that a solution contains shifts F1: (8:00-16:00) (3 3 3 3 3 3 3) and T1: (9:00-17:00) (2 1 1 1 1 1 1). By applying this move the shift T1 is deleted and shift F1 becomes: F1: (8:00- 16:00) (5 4 4 4 4 4 4).

***'SplitShift'*:**

A shift is divided into two different shifts. Newly created shift differs from the original shift by one time unit in the start or length (four possibilities). When possible, one duty per day is transferred from the original shift to the newly created shift. For example, by applying this move to shift F1: (7-15) (2 3 3 3 3 3 3), one of obtained solutions will contain shifts F1: (7-15) (1 2 2 2 2 2 2) and F2: (7-15:30) (1 1 1 1 1 1 1), the time unit being 30 minutes.

***'ChangeStartKeepEndFix'*:**

The start of a shift is moved forwards or backwards, while its end is kept fixed. For example if we use a time unit of 30 minutes, by applying this move to shift F1: (7:00-15:00), a new shift F2: (6:30-15:00) or F3: (7:30-15:00) can be obtained.

Moves *'ChangeDuty'* and *'ExchangeDuties'* may be still enhanced by applying these moves for a specific number larger than one of days.

5.2.2 Generation of neighborhood

We experimented with different variants for the generation of the neighborhood. In a first variant, only basic moves are applied in every iteration. In another variant, all moves (basic and composite ones) are applied to generate a neighborhood of solutions. However, in this case, for each move, not all neighborhood solutions are generated. Indeed, in all moves except the move *'ChangeDuty'* the exploration of neighborhood is interrupted as soon as a new better solution than the previous one is found.

5.2.3 Tabu mechanism

In order to avoid cycles during the search, a tabu list is used. The tabu list prohibits some of the solutions in the neighborhood to be accepted for the next iteration unless they fulfill certain criteria. We have experimented with two kinds of tabu lists. In the first variant, for each of the moves described above, we add the *inverse move* to the list. For example, the inverse of the move that increases by 1 the duty of shift X on day i, is the move that decreases by 1 the duty of shift X on day i. For each applied move in memory an inverse move is stored, including to which shifts the move was applied (one or two shifts) and also the day the move has been applied. The stored move in memory will be tabu for a certain number of iterations. In the second variant, certain characteristics of a solution itself are considered tabu in combination with each other: the number of shifts, the difference of the duties in each time slot, and the average number of duties per week. Solutions with the same features are considered tabu for a specified number of iterations.

5.2.4 Selection criteria

The best solution from the neighborhood, if it is not tabu, becomes the current solution in the next iteration. The tabu solution will be accepted only if it has the best fitness found so far (aspiration criteria).

5.2.5 Fitness function

The fitness function is a scalar function which combines four weighted criteria.

$$\begin{aligned} Fitness \;=\; & W1 \times ExcessInMinutes + W2 \times ShortageInMinutes + \\ & W3 \times NumberOFShifts + W4 \times DifferenceOfDutiesPerWeek \end{aligned}$$

For a more efficient calculation of the excess and the shortage, a history of change of the solution is stored and these parameters are updated based on this history.

5.3 Including the knowledge about the problem during the search

In this section, we describe a new approach that guides the search for the shift design problem, intending to make the search more effective. Furthermore, the algorithm for generating a good initial solution is presented.

The amount of neighborhood that should be explored during every iteration with defined moves can be reduced by using the knowledge about the problem during the search. The basic idea is to take moves which probably will most improve the solution. A similar technique for constraint satisfaction problems was used by Minton et al. [52]. This method is based on analyzing the 'distance' of the current solution to the optimal solution, with respect to the shortage and excess, which are the most important criteria in this problem. Just to give an idea, suppose that on day i between time 8:00-16:00 one employee is missing. A complete neighborhood search, among other moves, will perform also moves that decrease the duties of the shift for each day, which cannot contribute to any improvement in this situation. Furthermore, while this shortage appears at a particular day and particular shift, it is better to take moves that include only this day and shift types which cover this region: If, for example, the night shift ranges from 23:00-7:00, the moves in this shift probably will not bring any improvement in this situation. Our technique, based on the shortage/excess and shift types that could be used, determines the moves and other parameters that should be used for the neighborhood exploration of the current solution. Below we describe our technique in detail.

1. Find the shortage/excess with the longest length.
2. Determine the contribution of each shift type in the shortage or excess: high, middle, small.
3. Begin with the shift type that has the highest contribution to the shortage/excess.
4. Determine the position of the shift types that contributes in the shortage/excess relative to the shortage/excess: left, middle, right.

5. Shortage/excess is classified in short, middle, or long relative to the shift type for which the neighborhood will be generated.

6. Based on the shortage/excess properties, position, and contribution of the shift type relative to the shortage/excess, determine moves, days, and a shift type with which the neighborhood will be explored.

7. If there exist improvements, repeat the steps from the beginning, or else if there exist no improvements, but still shift types that contribute in the shortage/excess then take the next shift type and continue with step 4. In case there exist no improvements and no more shift types that contribute to shortage/excess, then, for the next iteration do a complete exploring of the neighborhood (like in basic tabu search).

The result of this technique is that the neighborhood will be generated using only some of the moves, and that these will be applied only to particular days and for particular shift types. One other important issue is the order of the moves. Since the procedure exits from the iteration for some of the moves as soon as a better solution than the current one is found, the moves that have more chances to improve the solution should be tried in the beginning. However, it is not always clear how to determine the order of the moves. For example, if the shortage/excess is long and the contribution of a certain shift type is high, then, logically, to improve the solution at this specific shortage/excess point, a move that increases/decreases duties of that particular shift type could be used. However, suppose we have an example where the shortage/excess is short. This kind of shortage/excess can be repaired for example by a moves *'ChangeDuty'*, *'ExchangeDuties'*, *'ChangeLength'*, *'MoveBorders'*, etc. and here the order of the moves is not clear. According to the properties of the shortage/excess, the contribution of shift types and their relative position, there exist 27 possible combinations (each of the properties is classified in three classes). Some of these combinations are almost the same regarding the conclusion about the moves that should be taken. In our technique, the order of the moves is the logical order when possible. In cases when is not completely clear which order of the moves should be taken, the order of the moves is taken like in the case of basic tabu search described previously. Note that if no improvement can be found through this procedure, a complete search of neighborhood is done to allow the acceptance of solutions with worse fitness and thus make an escape from a local optimum possible.

Example: In Figure 5.1 the requirements are given (only the first two days can be seen) for the first problem among the 30 randomly generated problems for which we will give computational results later. Applying the above procedure, the

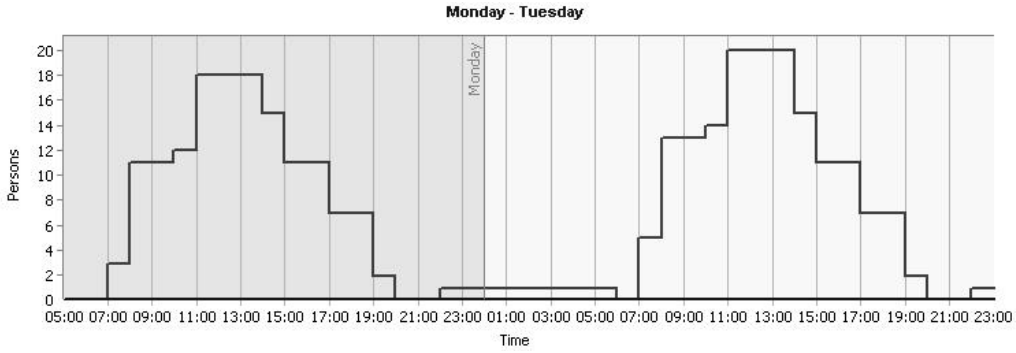


Figure 5.1: Requirements in example 1 for Monday and Tuesday

longest shortage found would be the one that starts in the first day at 22:00, with length 8 hours which has a shortage of one employee. The highest contribution in this shortage is from the night shift. The shortage is long and in the middle of the region of a night shift. The guided search procedure explores first the neighborhood that is constructed by applying the move *'ChangeDuty'* (duties are only increased) only for day one.

5.3.1 Initial Solution

Another ingredient given to the techniques described above is a 'good' initial solution. The basic idea in constructing a 'good' initial solution is that in every change of the requirements there should begin or end a shift. Initial solutions constructed in this manner, of course, do not always contain all shifts needed to construct an optimal solution but contain at least some of those which could have either the correct start or end. Below, the procedure that generates the initial solution is described.

- Detect all time points (day:hour:minute) in the planning period in which a positive difference of requirements (increase of the requirements) exist. The increases that begin at the same time (hour:minute) but on different days are considered like an increase even if they have different values (values of increase for each day are stored).
- For each increase from the previous step, determine all shift types with start regions to which the increase belongs. For each such shift type construct one shift with starting time at the point of time of the increase (time where the increase begins) and optimal length (based on the shift type). Additionally,

mark the starting time of all shifts constructed in this step as constant during the search.

- For each increase detected in step 1, select the first shift among the shifts constructed for that increase in the previous step. Give to the shift the duties which correspond to the values of increase for each day.
- Detect all time points (day:hour:minute) in the planning period in which a negative difference of requirements (decrease of the requirements) exist. The decreases that begin at the same time (hour:minute) but on the different days are considered like a decrease even if they have different values.
- From the decreases from the previous step, eliminate decreases that can be reached with any of the shifts constructed before, based in the increase of duties.
- For each remaining decrease determine all shift types, the end region (which is the time interval in which the shift type can have its end) of which contains the decrease and for each determined shift type construct one shift with end corresponding to the time of decrease as well as having minimal legal length (determined from the constraints for the shift types). Additionally, mark the end of each shift constructed in this step as constant during the search.

5.4 Computational results

In this section, we report computational results for 30 artificial examples generated by the random generator. Additionally we report computational results for one real world example. The randomly generated examples can be found on the site http://www.dbai.tuwien.ac.at/proj/Rota/. We describe first how the random examples are generated. Later, we show results of experiments that were conducted with basic and composite moves and the different types of tabu lists. Further, we investigate the impact of the initial solution and guidance of the search by including the knowledge about the problem. The techniques we described in this chapter are implemented in a software package called Operating Hours Assistant of Ximes[1] Corp. and the system works well on real life examples. All our results in this section have been obtained on an Intel P2 333 Mhz.

[1] http://www.ximes.com/

5.4.1 Randomly generating problem instances

The basic idea for random examples is to generate for each shift type a specific number of time intervals which fulfill constraints about the start and length of the shifts. Each time interval corresponds to one legal shift. Randomly, 1-5 time intervals are generated for each shift type. The start of the time interval is generated randomly from the possible starts in the start region of a shift type and the length of time intervals is one random possible length (determined by the shift type). We use four shift types, and thus the optimal solutions to the problem can have from 1 to 20 shifts. In Table 5.1 constraints for shift types are given. The time interval is taken randomly to be 15, 30 or 60 minutes.

Duties for each time interval are generated such that initially, for each time interval, a random number between 1-5 is generated with equal probability. The time interval will have assigned (as number of employees) this random number for each day during the week with probability 0.9 (probability that the duties change from the standard value is 0.1). If this value should change, the new value is again generated randomly using the same probability distribution. During the weekend the probability that the value changes is 0.6. If the value should change, again a new random number is generated (the probability that either Sunday or Saturday changes from this number is again 0.1 percent).

Weights for shortage and excess are 1, for number of shifts the weigth is equal to the length of the time unit in minutes (15, 30, or 60) and for duties per week it is 1000. The maximum number of duties per week is set to 5. The average number of hours per week is 38,5. As we want to generate the problems for which we can calculate the fitness in advance, if the duties per week exceed 5 (for optimal solution regarding excess, shortage and number of shifts), we put the weights of the duties per week to be 0. This means we consider indeed only minimizing the shortage, excess and number of shifts.

Table 5.1: Constraints about shifts for the random generator

Shift type	Earliest begin	Latest begin	Shortest length	Longest length
M (Morning shift)	05:00	08:00	07:00	09:00
D (Day shift)	09:00	11:00	07:00	09:00
A (Afternoon shift)	13:00	15:00	07:00	09:00
N (Night shift)	21:00	23:00	07:00	09:00

5.4.2 Computational results over random examples

In this section, we give results of four experiments over 30 randomly generated examples. The aim of these experiments was to determine the impact of composite moves, the length of the tabu list, and variants of the tabu mechanism. Techniques are compared regarding the dependence of the fitness from the number of evaluations and quality of the solution found after a certain number of evaluations. All results are given for one run of the algorithm, as the latter is deterministic. For each technique the algorithm is allowed to begin the next iteration only if a maximum number of 100000 evaluations is not exceeded for each example and the search is interrupted each time if after 200 seconds no improvements could be found.

Let us note that during the search the position and length of reserve shifts is changed if no improves can be made to the solution.

Basic moves

In this experiment the basic tabu search technique is used. During every iteration, by using only basic steps, a complete neighborhood of the solution is generated. For each of the tabu variants four different length of tabu list are used (5,10,20,40).

Results of this experiment over 30 examples were not satisfiable. While in two tabu variants the results were improved with increasing the tabu length, the best results were poor. In order to improve the results, we included the composite moves. The results are shown in the next experiment.

Impact of composite moves

In this experiment, tabu search is used and all moves are applied (basic and composite moves) for the generation of the neighborhood. However, we do not now explore the complete neighborhood for each move. Indeed, in all moves except the move *'ChangeDuty'* the exploration of neighborhood is interrupted as soon as a new better solution than the previous one if found. The order of the moves in tabu search is the same like the order they were described in Section 5.2.1, except that move *'ChangeStartKeepEndFix'* is tried after move *'ChangeLength'*. The experiment was conducted for two variants of tabu search and four possible lengths of the tabu list (5, 20, 40, 70).

The best results were obtained by applying the second variant of the tabu list (solution tabu) and with a length of the tabu list of 20. These results are shown in Table 5.2. For each example, information for the best fitness found,

number of evaluations needed, and time for finding this fitness is given. In the second column the fitness of the best-known solution is given (although this is most of the time an optimal solution, we cannot fully guarantee it). The best results obtained by applying the first variant of the tabu list (moves tabu) were slightly weaker. When using all moves, the tabu list has not such a high impact as in the case when only the basic moves are used.

Impact of the initial solution

The aim of this experiment is to investigate the impact of the initial solution in the technique, which was applied in the previous experiment. In the previous experiment the initial solution is a simple one. For each shift type it contains one empty shift template. Now the search begins from the good initial solution which is generated like described in the Section 5.3.1. Length of tabu list is taken to be 20 and the variant of making solution tabu is applied.

In Table 5.4.2 the results of the tabu search with good initial solution are shown. These results are better (compared to the results in Table 5.2 with respect to the fitness of solutions number of evaluations and time needed to generate best solutions.

Including the knowledge about the problem during the search

In this experiment, we show the results obtained by exploiting the knowledge about the problem (see Section 5.3) in the tabu search with good initial solution (TSwIS). We will call this technique tabu search and guided search with initial solution (TSaGSwIS).

In Table 5.4 the results of TSaGSwIS are presented. This techniques show the best results with respect to all criteria: fitness of solutions, number of found optimal solutions, number of evaluations per solution, and the time in which these solution were found. With this technique, 50 % of the optimal solutions are found in only one run.

5.4.3 Computational results on real-world problems

In this section we present the results for one real world problem taken from a call center.

Problem 1: This problem is a real problem from a call center. Temporal requirements for this problem are given in Table 5.5. Constraints about the shift

Table 5.2: Results for 30 examples using TS with the variant of a solution-type tabu list

		Tabu Search with solution tabu		
Ex.	Best known fitness	Fitness	Number of evaluations	Time in sec
1	480	2040	22322	126
2	300	750	62191	390
3	600	600	79326	470
4	480	1590	100388	660
5	480	480	25045	146
6	420	480	19175	102
7	270	1080	41525	245
8	150	195	100597	665
9	150	300	54318	350
10	330	1620	93847	578
11	30	30	2531	13
12	90	90	21550	124
13	105	105	24745	153
14	195	4305	100193	680
15	180	180	2045	10
16	225	540	100779	728
17	540	3600	100019	680
18	720	720	97919	611
19	180	2970	69726	473
20	540	540	53840	327
21	120	195	34349	237
22	75	75	9301	56
23	150	2430	75001	520
24	480	480	28656	165
25	480	2160	94991	635
26	600	720	46458	288
27	480	540	80685	495
28	270	1380	72192	429
29	360	1620	69738	431
30	75	75	6774	41

types which can be used in the solution are given in Table 5.6. Weights of the criteria are: $W1 = W2 = 1, W3 = 30, W4 = 1000$. Average number of working

Table 5.3: Results for 30 examples using TS with good initial solution

		Tabu Search with good initial solution		
Ex.	Best known fitness	Fitness	Number of evaluations	Time in sec
1	480	480	10012	54
2	300	390	36182	219
3	600	1020	80854	453
4	480	1590	101057	617
5	480	480	8613	47
6	420	420	5977	30
7	270	570	15497	100
8	150	615	15631	116
9	150	225	13034	85
10	330	450	80666	502
11	30	30	346	2
12	90	90	11976	70
13	105	105	2197	16
14	195	495	90481	628
15	180	180	148	1
16	225	540	73411	480
17	540	1170	75119	488
18	720	720	18979	129
19	180	195	34173	220
20	540	540	20790	109
21	120	120	2674	19
22	75	90	4158	23
23	150	570	39309	248
24	480	480	10120	60
25	480	1050	46090	309
26	600	660	31033	211
27	480	480	8537	48
28	270	270	7297	46
29	360	390	23856	148
30	75	75	986	6

hours is 38.5 and an upper limit of the average number of duties per week is 5.

In Table 5.7 the solution produced from the TsaGSwIS is shown. The second

Table 5.4: Results for 30 examples using TSaGSwIS

		Tabu search and guided search with good init. solut.		
Ex.	Best known fitness	Fitness	Number of evaluations	Time in sec
1	480	480	691	8
2	300	420	2623	30
3	600	900	85917	552
4	480	1170	67207	508
5	480	480	2299	18
6	420	420	887	6
7	270	630	17468	116
8	150	180	5722	80
9	150	225	10348	109
10	330	510	69811	503
11	30	30	68	1
12	90	90	1664	15
13	105	105	2494	21
14	195	390	46569	449
15	180	180	10	0
16	225	375	41182	289
17	540	1110	51679	471
18	720	720	1759	21
19	180	195	6985	73
20	540	540	27534	186
21	120	120	279	3
22	75	105	2414	19
23	150	540	3026	38
24	480	480	8674	52
25	480	690	29465	286
26	600	600	4151	26
27	480	480	7755	54
28	270	270	694	8
29	360	390	9461	81
30	75	75	379	4

variant of the tabu mechanism (solution tabu) is used and the length of the tabu list is 20.

Table 5.5: Temporal requirements for the call center problem

Time interval/day	Mon	Tue	Wed	Thu	Fri	Sat	Sun
07:00-08:00	5	5	5	5	5	1	1
08:00-08:30	10	10	10	10	10	4	4
08:30-09:30	12	12	12	12	12	4	4
09:30-10:00	14	14	14	14	14	4	4
10:00-12:00	17	17	17	17	17	4	4
12:00-13:00	17	17	17	17	17	9	9
13:00-17:00	20	20	20	20	20	9	9
17:00-18:00	18	18	18	18	18	8	8
18:00-18:30	20	20	20	20	20	5	5
18:30-19:30	18	18	18	18	18	5	5
19:30-20:00	16	16	16	16	16	5	5
20:00-22:00	13	13	13	13	13	5	5

Table 5.6: Constraints about shifts in the call center problem

Shift type	Earliest begin	Latest begin	Shortest length	Longest length
M (Morning shift)	05:00	08:00	07:00	09:00
D (Day shift)	09:00	11:00	07:00	09:00
A (Afternoon shift)	13:00	15:00	07:00	09:00

Table 5.7: Solution for the call center problem with TSaGSwIS

Shift	Time	Mon	Tue	Wed	Thu	Fri	Sat	Sun
M1	07:00-15:00	5	5	5	5	5	1	1
M2	08:00-15:00	5	5	5	5	5	0	0
D1	09:00-17:00	2	2	2	2	2	0	0
D2	10:30-19:30	5	5	5	5	5	0	0
D3	09:00-18:00						3	3
E1	15:00-22:00	10	10	10	10	10		
E2	13:00-22:00	3	3	3	3	3	5	5

The produced solution has no excess and a shortage of 3.99 % . The average number of duties per week is 4.82. The solution is generated after 3771 evaluations and in about 40 seconds.

Chapter 6

Practical Applications

Algorithms described in Chapters 4 and 5 are included in a software packages First Class Scheduler (FCS) and Operating Hours Assistant (OPA). The consultants of Ximes Corp. have already successfully used these systems in several organizations for shift design and the construction of rotating workforce schedules. The systems are also installed in several companies in Austria, Germany, Switzerland, Holland and UK. In this chapter, we give a brief description of these applications with illustrative examples.

6.1 First Class Scheduler

First class scheduler (FCS) is a part of the commercial product SHIFTPLAN-ASSISTANT (SPA 4.0) of Ximes Corp. (This company is specialized in developing software and in consulting work-hours arrangements, especially for shift-models). FCS supports semiautomatic generation of rotating workforce schedules. In this system, the generation of rotating schedules is conducted by interacting with the decision maker based on the new framework and algorithms that we have already presented in Chapter 4. In FCS most important constraints in the central European context can be modeled. This package is already internationally highly appreciated and exists in German, English, Finnish and Dutch. The product is used successfully since the year 2000 in several organizations by the consultants of Ximes Corp. for generating rotating workforce schedules. The system is also installed and works in many organizations such as for example, VOEST ALPINE AG, Österreichische Post AG, NOVARTIS Pharma AG, Opel Austria GmbH, Semperit Reifen Ges.m.b.H., Ciba Spezialitätenchemie AG, Fachklinik Schleswig etc. Next we introduce the this system with an illustrative example.

Example: There are three non overlapping shifts D (day shift), A(afternoon shift), and N(night shift) and 8 groups (each group has 3 employees). For shifts D and A, 6 employees are required each day, whereas for Night shift (N), 3 employees are required each day. The non-permitted sequences of shifts are: "N D","N A","A D","N-N","N-D","N-A". All these data are provided from the SHIFTPLAN-ASSISTANT to the FCS. Further in the FCS, the weekly rotating workforce schedules are generated in interaction with the decision maker as follows.

Definition of hard constraints

The constraints concerning length of work, day off blocks and length of blocks of consecutive shifts should be defined by the decision maker (see Figure 6.1). In this example, the work blocks are not allowed to be longer than 6 days and shorter than 4. The day off blocks should be between 2 and 4 days. Blocks of consecutive shifts should be of length 2-6, 2-5 and 2-4, respectively, for the D, A and N shifts.

Choosing a set of lengths of work blocks

The possible set of work blocks (see Figure 6.2) are generated with the algorithm described in Chapter 4 under given constraints about the requirements and the lengths of the work and day off blocks. In this stage, the decision maker should select a preferable set of work blocks based on his/her preferences, but the order of the work blocks is not yet determined. From the possible set of work blocks (class solution), in the next step, the order of these blocks is determined based on the weekend features (number of weekends off and number of long weekends off). Note that before the set of work blocks is generated it should be tested in case at least one possible distribution of day off exist for these work blocks, such that temporal requirements for work and rest are fulfilled. For this example, in this stage, the class solution {6 6 5 5 5 4 4} is selected.

Choosing a particular sequence of work and day off blocks among those that have optimal weekend characteristics

Different solutions regarding distribution of day off can be found for given work blocks. Based on the algorithm described in Chapter 4, for each possible order of work blocks, the best distribution of day off blocks subject to a number of weekends off, a number of long weekends off and a distribution of weekends during the planning period is found. In Figure 6.3, the best possible solutions for different

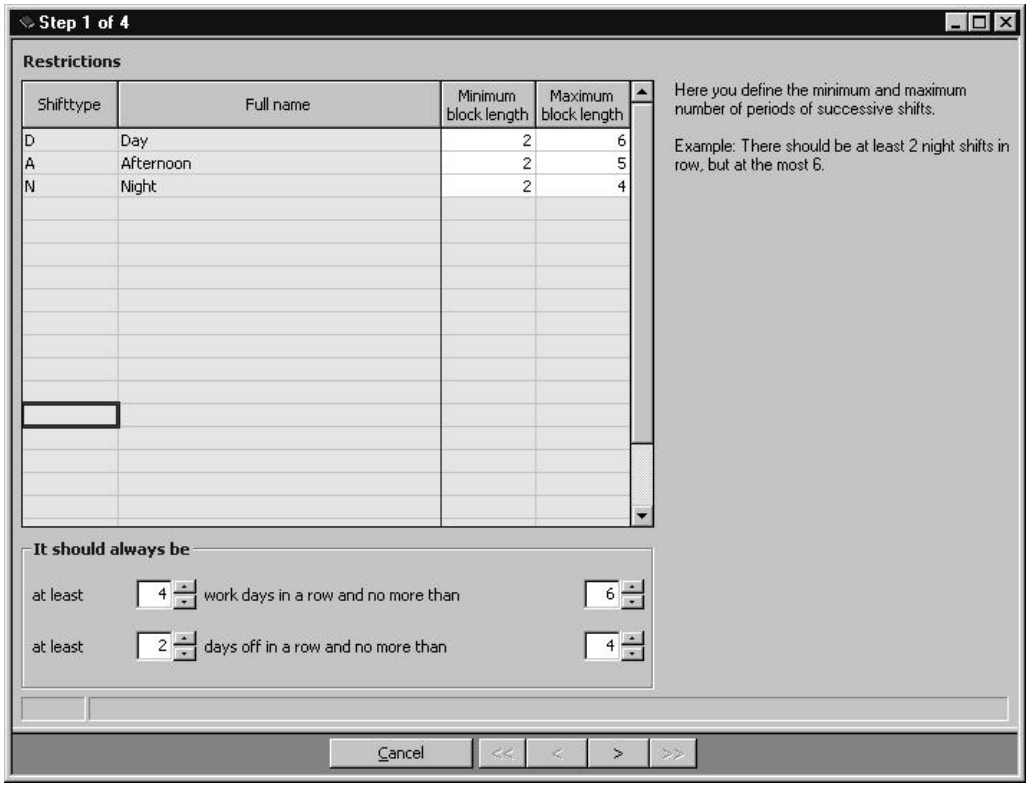


Figure 6.1: Definition of hard constraint in First Class Scheduler

orders of work blocks for the class solution selected in Step 2 are represented. The decision maker is also here included to select one of the solutions based on his preferences.

Generation of schedules

After the distribution of the day off and work blocks is determined, the assignment of shifts to employees can be done. This is the last phase of the generation of rotating workforce schedules. Internally, before the algorithm assigns the shifts to the employees, possible shift sequences for the chosen work blocks subject to shift change constraints and bounds on sequences of shifts are enumerated. In Figure 6.4, a bunch of the generated schedules is shown on the left side. These schedules fulfill all hard constraints and have same characteristics subject to weekends off. However, these schedules are built up from different shift sequences. On the right side the shift sequences of these schedules are shown. The user can

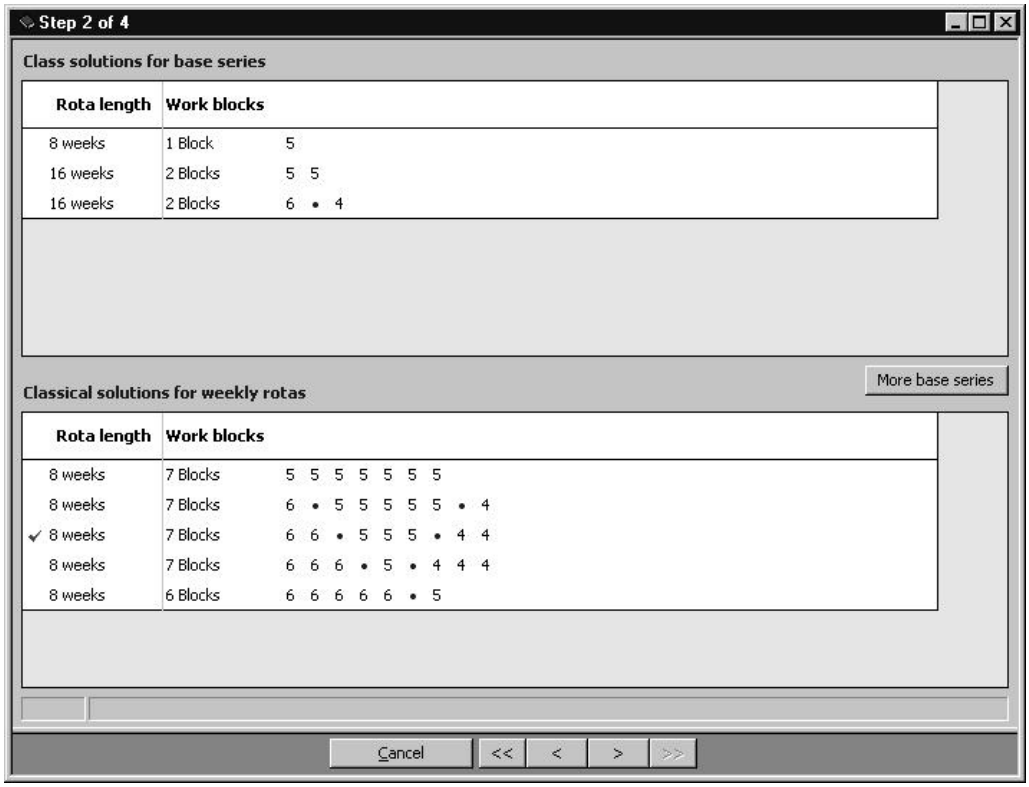


Figure 6.2: Selecting of the possible length of work blocks in First Class Scheduler

eliminate some of them by his/her preferences. If, for example, we eliminate some of these sequences only some schedules will remain (see Figure 6.5. One of these schedules is presented in Figure 6.6.

In the system FCS, rotating workforce schedules are usually generated in a reasonable amount of the time. Moreover, these schedules are of high quality. The system provides interaction with the decision maker and, thus, including soft preferences through this interaction is possible. The system is also very flexible as it allows for the relaxation of constraints and the generation of rotating workforce schedules for a wide range of constraints. According to the consultants of the Ximes Corp. FCS can solve successfully 80 to 90% of problems of rotating workforce schedules that appear in practice.

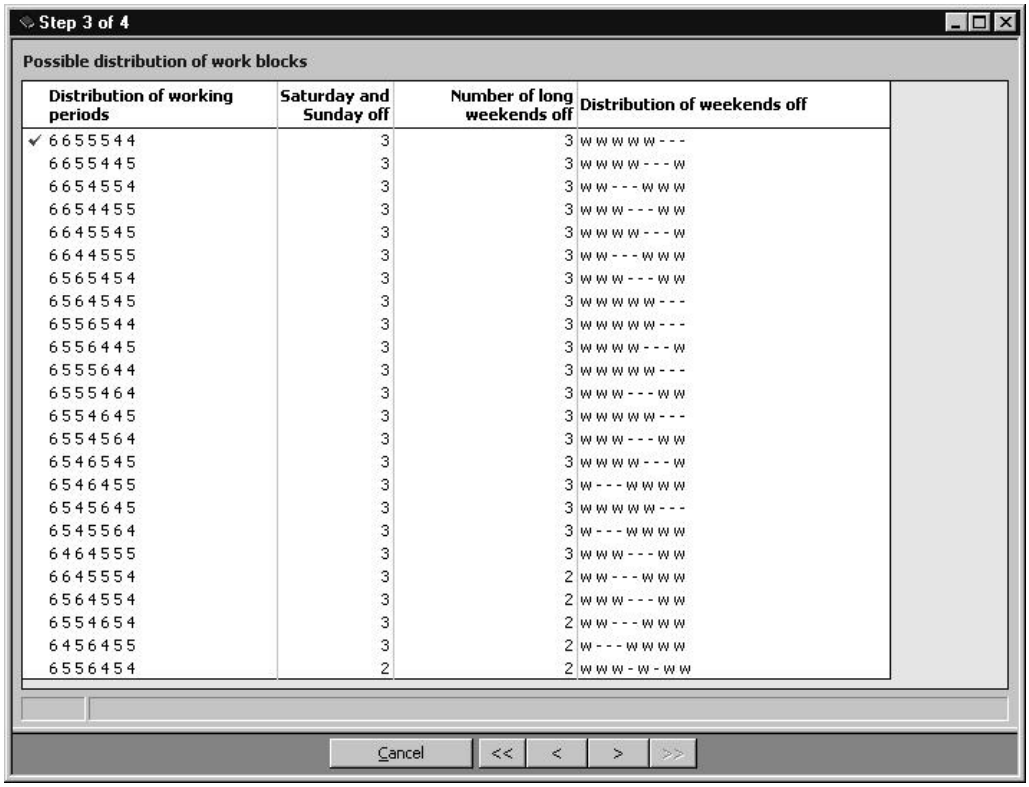


Figure 6.3: Selecting of possible distribution of work and day off blocks based on the weekend characteristics

6.2 Operating Hours Assistant

The generation of shifts is one of the main features of commercial product Operating Hours Assistant(OPA). This product includes also other tools, which will not be described here. For the generation of shifts the algorithms described in the Chapter 5 are used. The OPA is in use since early 2001 by the consultants of the Ximes Corp. and it has been successfully applied in several organizations. The system is also installed and works in several organizations like, for example, Eurostar, Semperit Reifen Ges.m.b.H. Fachchklinik Schleswig etc. In Figure 6.7 the screenshot of the OPA is given. Further, we describe the process of the generation of shifts in OPA.

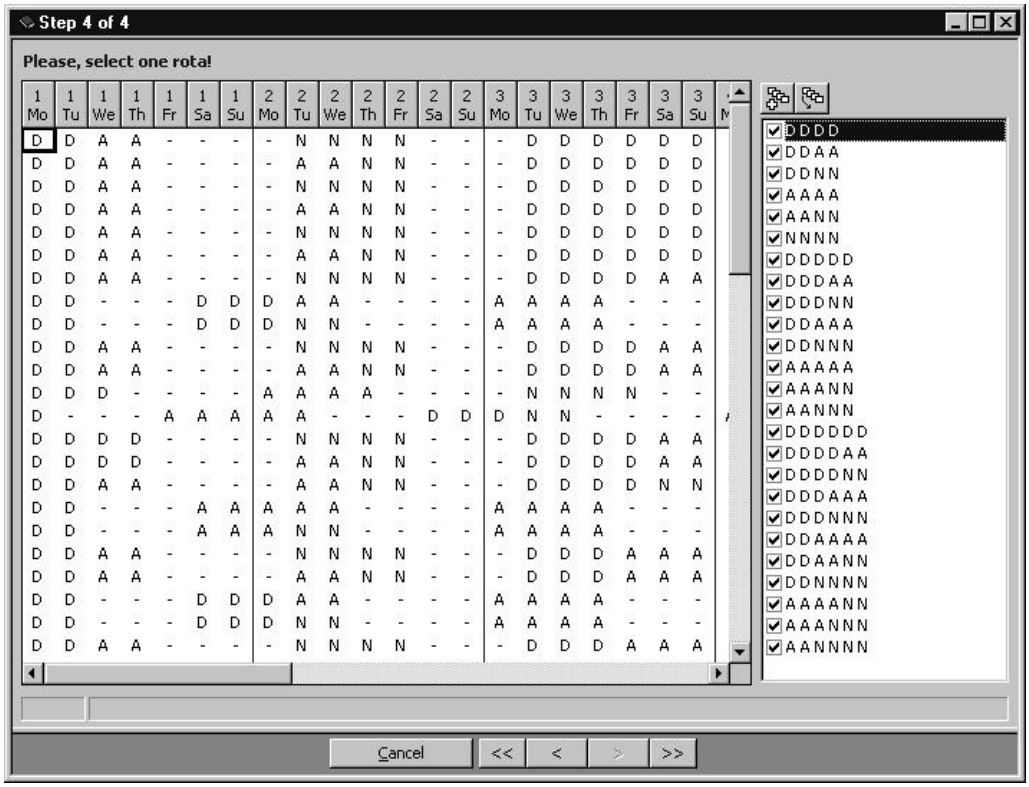


Figure 6.4: Generated schedules that fulfill all constraints

Definition of temporal requirements

As was described in Chapter 5, the first step for designing shifts is to define temporal requirements (see Figure 6.8). Typically, the temporal requirements would be given for a week, but they can also be given for one day or less then seven days. In our case, when the temporal requirements are given for a week, the cycle should be used in consideration (night shift that begins on Sunday at 23:00 is 8 hours long, impacts the first day of the week). In the example we show here, the temporal requirements are defined for one week.

Constraints regarding shift types

Shift types determine the possible start and length of the real shifts. In this case (see Figure 6.9), we define four shift types, morning shift, day shift, evening shift, and night shift. Shifts generated by algorithms should fulfill the criteria exposed

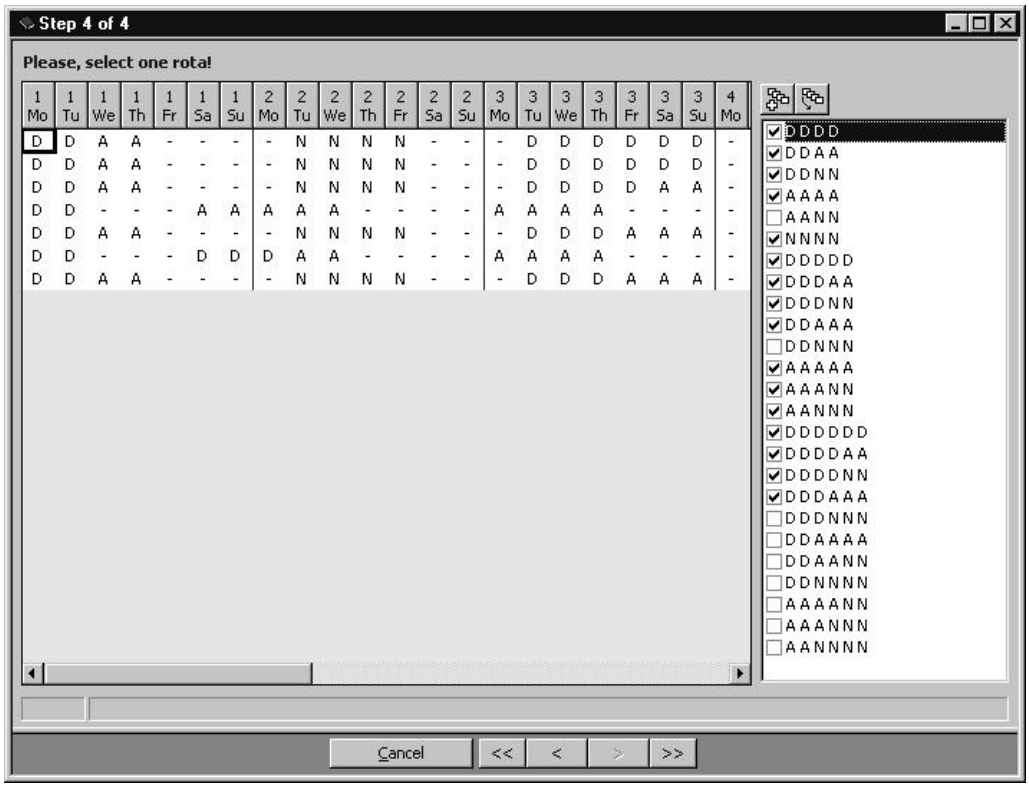


Figure 6.5: Remaining schedules after eliminating of some the shift sequences

	1 Mo	1 Tu	1 We	1 Th	1 Fr	1 Sa	1 Su
A	D	D	A	A			
B		N	N	N	N		
C		D	D	D	D	D	D
D			D	D	D	D	D
E	D			A	A	N	N
F	N				A	A	A
G	A	A				A	A
H	A	A	A				

Figure 6.6: Week representation of one schedule that fulfills all constraints

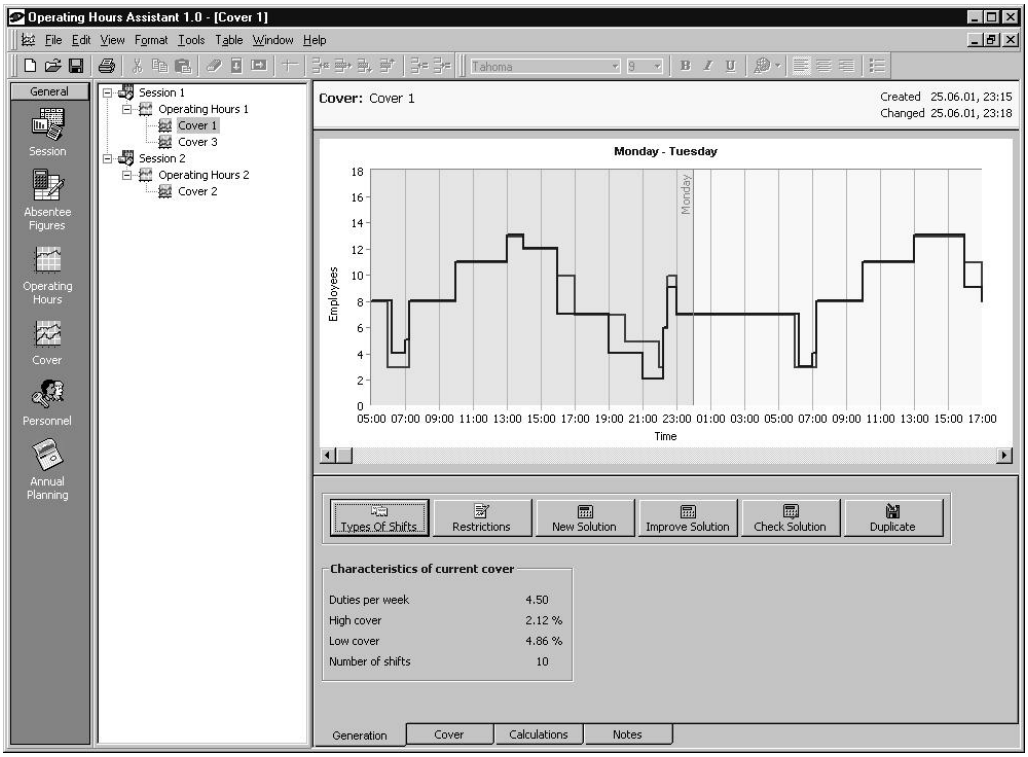


Figure 6.7: Screenshot of Operating Hours Assistant

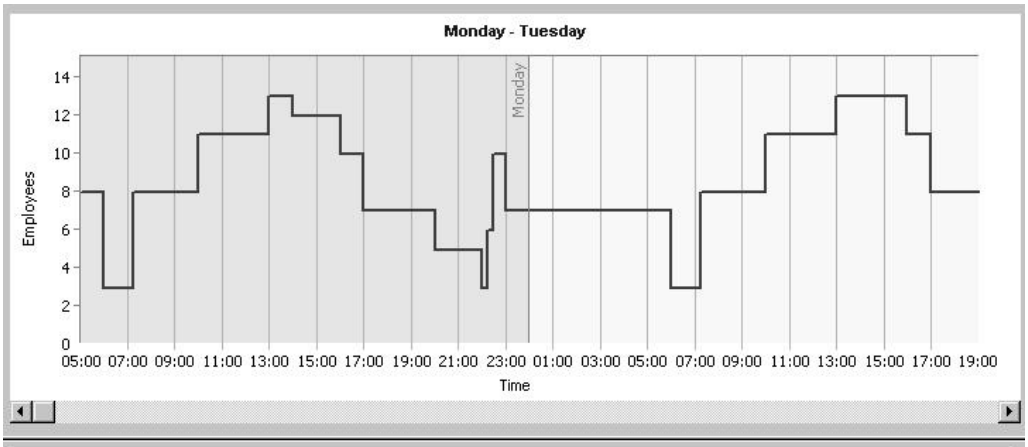
by the shift types. For example, morning shift can start between 5:00-8:00 and their length should be from 7-9 hours.

Weights of criteria

Just as we have described in Chapter 5, the solution to the shift design problem is evaluated with a scalar function, which combines four weighted criteria: excess in minutes, shortage in minutes, number of shifts and distance from average number of duties per week. OPA offers the possibility to change the importance of these criteria (Figure 6.10) that depend from his preferences easily.

Generation of shifts

After the constraints have been defined, the algorithm for the generation of shifts (Chapter 5) can be called. This algorithm iteratively improves the initial solution. The process of improvement is shown to the user (the graphical representation



From	To	Stand-by duty	Mon	Tue	Wed	Thu	Fri	Sat	Sun	Comment
10:00	17:00		3	3	3	3	3	3	3	
07:15	15:00		3	3	3	4	4	4	4	
07:15	16:00		2	2	2	3	3	3	3	
06:00	14:00		3	3	3	3	3	3	3	
13:00	20:00		2	2	1	1	1	1	1	
14:00	22:00		2	3	3	3	3	3	3	
15:00	23:00		3	3	3	3	3	3	3	
22:15	06:00		3	4	4	4	4	4	4	
22:30	06:00		4	4	4	4	4	4	4	

Figure 6.8: Definition of temporal requirements in Operating Hours Assistant

was implemented by the Ximes Corp. team). The algorithm can be stopped at any time and started all over again from any solution. In case the decision maker is not satisfied with the solution, the weights can be changed and attempts could be made to improve the current solution.

For the given requirements, constraints and weights (which we represented in the screenshots), algorithm which combine basic tabu search and guided search and starts with a good initial (see Chapter 5) generates the solution shown in Figure 6.11.

The OPA is suitable for generation of shifts in different areas. It has been already successfully used in call centers, hospitals etc.

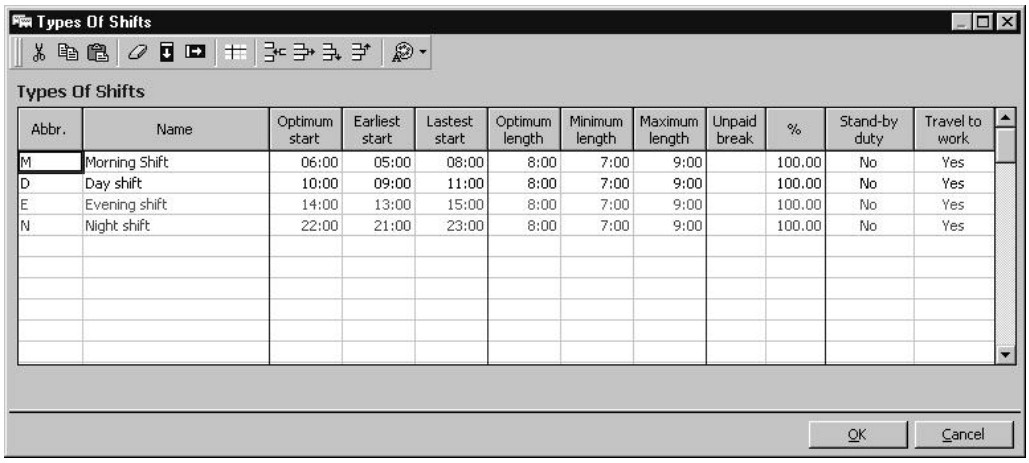


Figure 6.9: Definition of shift types in Operating Hours Assistant

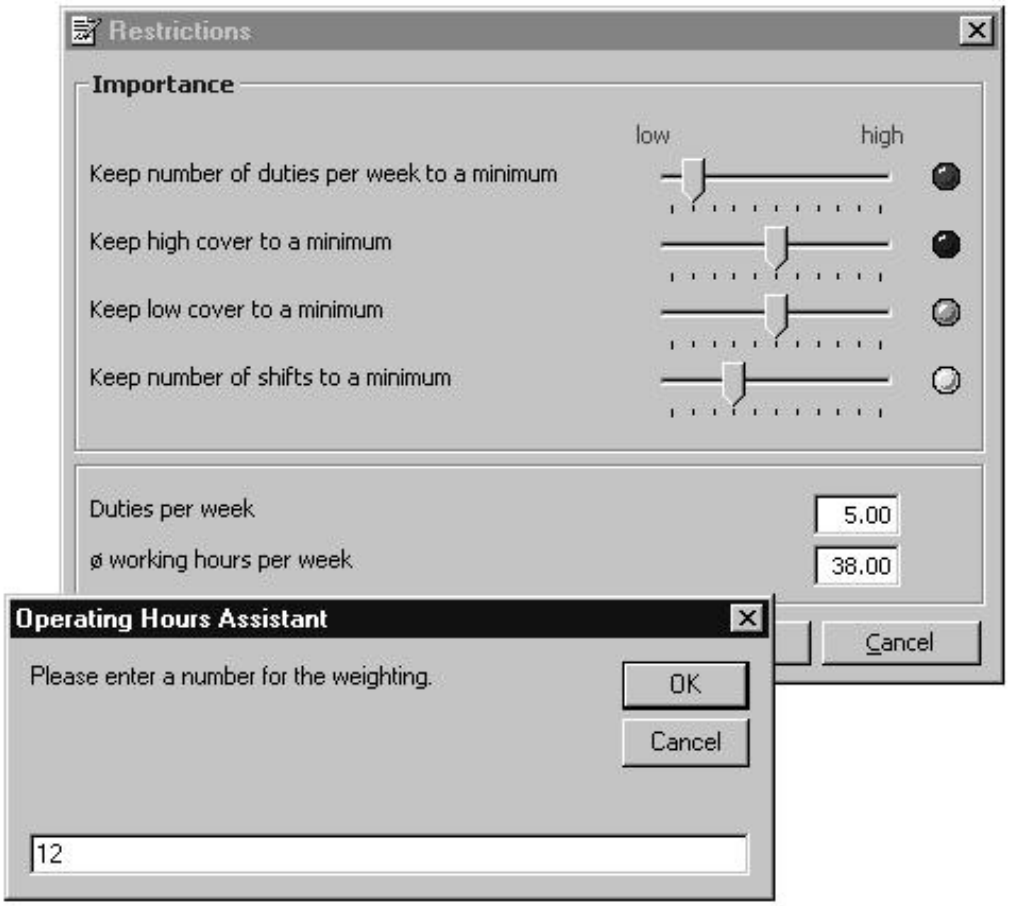
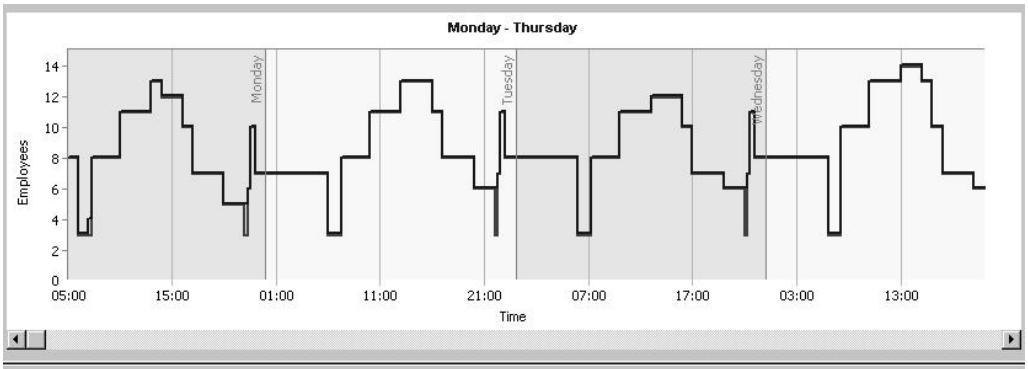


Figure 6.10: Definition of weights about the criteria in Operating Hours Assistant



Abbr.	Full name	Start	End	+ whole	U. breaks	%	Length in h	Stand-by duty	Travel to work	Mon	Tue	Wed	Thu	Fri	Sat	Sun	Sum
M1	Morning Shift	07:15	15:00		0	100.00	7:45	No	Yes	5	6	6	7	7	7	7	45
M2	Morning Shift	07:00	14:00		0	100.00	7:00	No	Yes	1							1
M3	Morning Shift	07:15	16:00		0	100.00	8:45	No	Yes	2	2	2	3	3	3	3	18
D1	Day shift	10:00	17:00		0	100.00	7:00	No	Yes	3	3	3	3	3	3	3	21
E1	Evening shift	13:00	20:00		0	100.00	7:00	No	Yes	2	2	1	1	1	1	1	9
E2	Evening shift	15:00	23:00		0	100.00	8:00	No	Yes	3	3	3	3	3	3	3	21
E3	Evening shift	15:00	22:30		0	100.00	7:30	No	Yes	2	3	3	3	3	3	3	20
N1	Night shift	22:30	06:00		0	100.00	7:30	No	Yes	3	4	4	4	4	4	4	27
N2	Night shift	22:30	07:15		0	100.00	8:45	No	Yes	3	3	3	3	3	3	3	21
N3	Night shift	22:15	06:00		0	100.00	7:45	No	Yes	1	1	1	1	1	1	1	7

Figure 6.11: Solution generated using guided search with an initial solution

Chapter 7

Conclusions

In this thesis, we used intelligent search methods to investigate the computerized generation of solutions for two specific problems in workforce scheduling: rotating workforce scheduling and shift design problems.

In Chapter 4 we proposed a new framework for solving the problem of rotating workforce scheduling. We showed that this framework is very powerful for solving real life problems. The main features of this framework are the possibility to generate high quality schedules through the interaction with the human decision-maker and to solve real cases in a reasonable amount of time. Apart from the fact that the generated schedules meet all hard constraints, it also allows to incorporate preferences of the human decision-maker regarding soft constraints that are more difficult to assess and to model otherwise. In step 1 an enhanced view of possible solutions subject to the length of work blocks is given. In step 2 preferred sequences of work blocks in connection with features of weekends off can be selected. In step 3, possible shift sequences for the chosen work blocks subject to shift change constraints and bounds on sequences of shifts are enumerated. Finally, in step 4 bounds for successive shifts and shift change constraints can be specified with much more precision because the decision maker has a complete view on terms (shift sequences) that are used to build the schedules. Step 2 of our framework can be solved very efficiently because of the existence of step 1. Furthermore, we showed that the assignment of shifts to employees in step 4 can be done very efficiently using backtracking algorithms even for large instances if sequences of shifts for work blocks are generated first. When the number of employees is very large they can be grouped into teams and thus this framework can still be applied.

In Chapter 5, we presented a local search approach for the shift design problem. We proposed basic and composite moves for the generation of the neigh-

borhood during every iteration. Experiments over randomly generated examples confirmed that composite moves improve the quality of the generated solutions. To avoid cycles during the search, we used basic principles of tabu search and experimented with different mechanisms for prohibiting the solutions to be descendant of the current solution. We also experimented with different lengths of tabu lists for each variant of tabu list. Experiments showed that the length of tabu list had much more impact in case where only basic moves were used. Furthermore, we proposed a procedure for the generation of good initial solution, which has shown to improve the results compared to the basic tabu search over the randomly generated examples. In order to make the search more effective, we proposed a method which exploits knowledge about the problem during the search. Using this method the neighborhood of a solution is explored only selectively during every iteration. Experimental results confirmed that this method improves the search effectiveness. Probably, local search still can be refined for this problem to give better results, but the aim here was to show that that knowledge about the problem can significantly contribute to the search by using it in combination with classical techniques. In Chapter 5, we also proposed set of randomly generated examples for the shift design problem, which can be used further by other researchers to compare their results with ours.

We have implemented an optimization framework for the generation of rotating workforce schedules in a software package called First Class Scheduler (FCS), which is part of a general shift scheduling package Shift-Plan-Assistant (SPA) of Ximes Corp. The package has been very well received internationally, and German, English, Finnish, and Dutch versions of it are in daily use throughout Europe. We implemented algorithms for the shift design problem in a software package called Operating Hours Assistant. This product is in early stages of use and has seen first application in several organizations.

Even though the framework for rotating workforce schedules is appropriate for most real cases, for large instances of problems optimal solution for weekends off cannot always be guaranteed because of the large size of the search space. One way to solve this problem more efficiently could be to stop backtracking when one solution that has the maximum number of weekends off — or close to the maximum — is found (for a given problem we always know the maximum number of weekends off from the temporal requirements). Once we have a solution with most weekends off, other search techniques like local search heuristics can be used to improve the distribution of weekends off. Finally we can extend this framework by introducing new constraints.

One further interesting point for research in the future is to consider shift design and assignment of the shifts and days-off to employees as one problem, and to solve both in coordination with each other.

Bibliography

[1] E. Aarts and J. Korst. *Simulated annealing and Boltzman Machines.* John Wiley and Sons, 1989.

[2] Emile Aarts and Jan Karl Lenstra, editors. *Local Search in Combinatorial Optimization.* Wiley, 1997.

[3] S. Abdennadher and H. Schlenker. Interdip - an interactive constraint based nurse scheduler. In *In PACLP-99: Proceedings of the International Conferenceon Practical Applications of Constraint Practical Applications Expo, London*, 1999.

[4] Hashem K. Alfares. Days-off employee scheduling over a three-week work cycle. In *Proceedings of the 3rd international conference on the practice and theory of automated timetabling Konstanze, Germany*, 2000.

[5] Harald Meyer auf'm Hofe. Solving rostering tasks as constraint optimization. In *Proceedings of the 3rd international conference on the practice and theory of automated timetabling Konstanze, Germany*, 2000.

[6] Turgut Aykin. Optimal shift scheduling with multiple break windows. *Managament Science*, 42(4):591–602, 1996.

[7] Turgut Aykin. A comparative evaluation of modeling approaches to the labor shift scheduling problem. *European Journal of Operational Research*, 125:381 –397, 2000.

[8] Nagraj Balakrishnan and Richard T. Wong. A network model for the rotating workforce scheduling problem. *Networks*, 20:25–42, 1990.

[9] Roberto Battiti. Reactive search: Toward self–tuning heuristics. In V. J. Rayward-Smith, I. H. Osman, C. R. Reeves, and G. D. Smith, editors, *Modern Heuristic Search Methods*, pages 61–83. John Wiley & Sons Ltd., Chichester, 1996.

[10] Jacobs-L.W. Bechtold, S.E. Implicit modeling of flexible break assignments in optimal shift scheduling. *Management Science*, 36(11):1339 –1351, 1990.

[11] BEST. Guidelines for shiftworkers. Bulletin of European Time Studies No. 3, European Foundation for the Improvement of Living and Working Conditions, 1991.

[12] J. Bitner. Backtracking programming techniques. *Communications of the ACM*, 18(11):651–656, 1975.

[13] Leonard Bolc and Jerzy Cytowski. *Search methods for Artificial Intelligence.* Academic Press Limited, 1992.

[14] J. Brusco and L. Jacobs. A simulated annealing approach to the cyclic staff-scheduling problem. *Naval Research Logistics*, 40:69–84, 1993.

[15] B. Butler. Computerized manpower scheduling. Master's thesis, University of Alberta, Canada, 1978.

[16] Tolga Çezik, Oktay Günlük, and Hanan Luss. An integer programming model for the weekly tour scheduling problem, to appear in Naval Res. Log.

[17] Marko Chiarandini, Andrea Schaerf, and Fabio Tiozzo. Solving employee timetabling problems with flexible workload using tabu search. In *Proceedings of the 3rd international conference on the practice and theory of automated timetabling 298–302, Konstanze, Germany*, 2000.

[18] Stephen A. Cook. The complexity of theorem-proving procedures. In *Conference Record of Third Annual ACM Symposium on Theory of Computing*, pages 151–158, Shaker Heights, Ohio, 3–5 1971 1971.

[19] G.B. Danzig. A comment on eddie's traffic delays at toll booths. *Operations Research*, 2:339 –341, 1954.

[20] K. Dowsland. Simulated annealing. chapter 2 in C.R. Reeves, editor, 1995.

[21] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* Freeman and Co., 1979.

[22] J. Gärtner, M. Kundi, S. Wahl, K. Hörwein, M. Janke, H. Conrad, I. Carlberg, *Handbuch Schichtpläne: Planungstechnik, Entwicklung, Ergonomie, Umfeld.* vdf, Hochschulverlag AG an der ETH Zürich, 1998.

[23] J. Gärtner and S. Wahl. The significance of rota representation in the design of rotas. *Scandinavian Journal of Work, Environment and Health*, 24(3):96–102, 1998.

[24] Johannes Gärtner and Stephen Popkin. Influence of law on shift schedule design: USA and Europe. *XIV International Symposium on Night and Shiftwork, Wiesensteig, Germany*, 1999.

[25] Fred Glover. Future paths for integer programming and links to artificial intelligence. *Computers & Operations Res.*, 5:533–549, 1986.

[26] Fred Glover. Tabu search–part I. *ORSA Journal on Computing*, 1(3):190–206, 1989.

[27] Fred Glover. Tabu search–part II. *ORSA Journal on Computing*, 2(1):4–32, 1989.

[28] Fred Glover and Manuel Laguna. *Tabu search.* Kluwer Academic Publishers, 1997.

[29] Fred Glover and Claude McMillan. The general employee scheduling problem: An integration of MS and AI. *Comput. Ops. Res.*, 13(5):563–573, 1986.

[30] D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley, 1989.

[31] P. Hansen. The steepest ascent mildest descent heuristic for combinatorial programming. In *Congress on Numerical Methods in Combinatorial Optimization*, 1986.

[32] R. Haralick and G. Elliot. Increasing tree seach efficiency for constraint satisfaction problems. *Artificial Intelligence*, 14:263–313, 1980.

[33] N. Heller, J. McEwen, and W. Stenzel. Computerized scheduling of police manpower. *St. Louis Police Department, St. Louis, MO*, 1973.

[34] J. H. Holland. *Adaptation in Natural and Artificial Systems.* University of Michigan Press, Ann Arbor, 1975.

[35] R. Hung. A three-day workweek multiple-shift scheduling model. *J Opl Res Soc*, (44):141–146, 1993.

[36] R. Hung. A multiple-shift workforce scheduling model under 4-day workweek with weekday and weekend labour demands. *J Opl Res Soc*, (45):1088–1092, 1994.

[37] W. Ken Jackson, William S. Havens, and Harry Dollard. Staff scheduling: A simple approach that worked. Technical Report CMPT97-23, 1997.

[38] Richard M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103, New York, 1972. Plenum Press.

[39] S. Kirkpatrick, C. D. Gelatt Jr., and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.

[40] P. Knauth. Designing better shift systems. *Appl Ergonom*, (27):39–44, 1996.

[41] Guy Kortsarz and Wolfgang Slany. The minimum shift design problem and its relation to the minimum edge-cost flow problem. Unpublished manuscript.

[42] P. Laarhoven and E. Aarts. *Simulated Annealing: Theory and Applications.* Kluwers Academic Publishers, Dordrecht, The Netherlands, 1987.

[43] G. Laporte. The art and science of designing rotating schedules. *Journal of the Operational Research Society*, 50:1011–1017, 1999.

[44] G. Laporte, Y. Nobert, and J. Biron. Rotating schedules. *Eur. J. Ops. Res.*, 4:24–30, 1980.

[45] Hoong Chuin Lau. Combinatorial aproaches for hard problems in manpower scheduling. *Journal of Operations Research Society of Japan*, 39(1):88–98, 1996.

[46] Hoong Chuin Lau. On the complexity of manpower scheduling. *Computers. Ops. Res.*, 23(1):93–102, 1996.

[47] Hoong Chuin Lau and Seet Chong Lau. Efficient multi-skill crew rostering via constrained sets. In *In Proceedings of the Second ILOG Solver and Scheduler Users Conference. Paris.*, 1997.

[48] K. Marriott and P. Stuckey. *Programming with Constraints.* MIT Press, 1998.

[49] A. Meisels and N. Lusternik. Experiments on networks of employee timetabling problems. *Lecture Notes in Computer Science*, 1408:130–141, 1998.

[50] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller. Equation of state calculations for fast computing machines. *Journal of Chemical Physics*, 21:1087–1092, 1953.

[51] Z. Michalewicz and B. F. Fogel. *How to solve it: modern heuristics.* Springer-Verlag, 2000.

[52] Steven Minton, Mark D. Johnston, Andrew B. Philips, and Philip Laird. Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problems. *Artificial Intelligence*, 58:161–205, 1992.

[53] L. Moondra. An lp model for work force scheduling for bank. *Journal of bank Research*, 7:299–301, 1976.

[54] J. Morris and M. Showalter. Simple approaches to shift, days-off and tour scheduling problems. *Management Science*, 29(8):942–951, 1983.

[55] Nysret Musliu, Johannes Gärtner, and Wolfgang Slany. Efficient generation of rotating workforce schedules. Technical Report DBAI-TR-2000-35, Institut für Informationssysteme der Technischen Universität Wien, 2000. http://www.arXiv.org/abs/cs.OH/0002018.

[56] R. Nanda and J. Browner, editors. *Introduction to Employee Scheduling.* Van Nostrand Reinhold, New York,, 1992.

[57] K. Nonobe and T. Ibaraki. A tabu search approach to the constraint satisfaction problem as a general problem solver. *European Journal of Operational Research*, 106:599–623, 1998.

[58] D.T. Pham and D. Karaboga. *Intelligent Optimisation Techniques.* Springer-Verlag, 2000.

[59] Colin R. Reeves, editor. *Modern Heuristic Techniques for Combinatorial Problems.* Halsted Pr., 1993.

[60] Stuart Russell and Peter Norvig. *Artifical Intelligence: A Modern Approach,.* Prentice-Hall, Englewood Cliffs, NJ, 1995.

[61] Andrea Schaerf and Amnon Meisels. Solving employee timetabling problems by generalized local search. In *Paper presented at AI*IA'99, Bologna, Italy*, 1999.

[62] Claude E. Shannon. Programming a computer for playing chess. *Philosophical Magazine*, (Series 7)(41):256–275, 1950.

[63] B. M. Smith and S. Bennett. Combining constraint satisfaction and local improvement algorithms to construct anaesthetists' rotas. In *Proc. of the Eighth Conference on Artificial Intelligence for Application CAIA-92*, pages 106–112, Monterey, CA, 1992.

[64] G. Thompson. Improved implicit modeling of the labor shift scheduling problem. *Management Science*, 41(4):595–607, 1995.

[65] G. Thompson. A simulated-annealing heuristic for shift scheduling using non continuously available employees. *Computers and Operations Research*, 23(3):275–288, 1996.

[66] James M. Tien and Angelica Kamiyama. On manpower scheduling algorithms. *SIAM Review*, 24(3):275–287, 1982.

[67] E. Tsang and C. Voudouris. Fast local search and guided local search and their application to british telecom's workforce scheduling problem. *Operations Research Letters*, 20:119–127, 1997.

[68] Edward Tsang. *Foundations of Constraint Satisfaction.* Academic Press, 1993.

[69] Georges Weil and Kamel Heus. Eliminating interchangeable values in the nurse scheduling problem formulated as a constraint satisfaction In *Paper presented at the workshop CONSTRAINT'95, Melbourne Beach, Florida, USA*, 1995.

Curriculum Vitae

Personal Data:

Name:	Nysret Musliu.
Born:	February 2, 1973, in Lebanë, Kosova.
Parents:	Avdi Musliu and Bahtije Musliu.
Brothers:	Xhevdet and Valdet Musliu.
Nationality:	Albanian (from Kosova).
Marital status:	Single.
Languages:	Albanian, English, German, Croatian.
Address:	Lorenz Müller Gasse 1a/18, A-1200 Wien, Austria
Email:	musliu@dbai.tuwien.ac.at

Education:

1978–1986:	Primary School "Nexhmi Mustafa", Besi, Kosova.
1986–1990:	Secondary school "Gjimnazi Sami Frasheri", Prishtinë, Kosova.
1990–1996:	The Faculty of Electrical Engineering (field of computer sciences and telecommunications), University of Prishtina, Kosova.
Summer 1996:	Graduation as a degree 'inxhinier i diplomuar' (equivalent with Dipl.-Ing).
since 1998:	Ph.D. student at the Vienna University of Technology.

Work Experience:

1996-1997:	Designer and Programmer in Faculty of Agriculture in Prishtina and Infotrade Corp. in Prishtina, Kosova.
since 1999:	Research assistant at the Vienna University of Technology.

Publications:

1. Nysret Musliu, Johannes Gärtner, Wolfgang Slany. Efficient generation of rotating workforce schedules. Discrete Applied Mathematics, to appear. Also in Proceedings of the 3rd international conference on the practice and theory of automated timetabling (PATAT 2000), pages 314-332, August 2000

2. Ruth Fingerlos, Johannes Gärtner, Nysret Musliu, and Wolfgang Slany. Zyklische Schichtplanung. KI Journal, to appear.

3. Johannes Gärtner, Nysret Musliu, and Wolfgang Slany. Rota: A research project on algorithms for workforce scheduling and shift design optimisation. Artificial Intelligence Communications, to appear.

4. Wahl S., Musliu N., Angelova R., Slany W., Herber G., Janke M. Shiftplanassistant 4.0 - State of development. XIV International Symposium on Night and Shiftwork, 1999 Wiesensteig, Germany

5. Nysret Musliu, Andrea Schaerf, Wolfgang Slany. Local search for shift design (extended abstract). MIC'2001 - 4th Metaheuristics International Conference Porto (Portugal), 2001 July 16-20

6. Nysret Musliu, Johannes Gärtner, Wolfgang Slany. Shift scheduling from a combinatorial optimization point of view (extended abstract). Third Alio-Euro Workshop on Applied Combinatorial Optimization, Erice-Italy, November 1999

7. Wolfgang Slany, Nysret Musliu, Guy Kortsarz, and Johannes Gärtner. Theory and practice of shift scheduling (invited paper). RIMS Kokyuroku of the Research Institute of Mathematical Sciences, Kyoto University, to appear.

Talks:

1. Shift scheduling from a combinatorial optimization point of view (extended abstract). Nysret Musliu, Johannes Gärtner, Wolfgang Slany. Third Alio-Euro Workshop on Applied Combinatorial Optimization, Erice-Italy, November 1999

2. Shiftplanassistant 4.0 - State of devolepment. Wahl S., Musliu N., Angelova R., Slany W., Herber G., Janke M. XIV International Symposium on Night and Shiftwork, 1999 Wiesensteig, Germany

3. Efficient generation of rotating workforce schedules. Nysret Musliu, Johannes Gärtner, Wolfgang Slany. 3rd international conference on the practice and theory of automated timetabling (PATAT 2000), pages 314-332, August 2000

4. Local search for shift design (extended abstract). Nysret Musliu, Andrea Schaerf, Wolfgang Slany.MIC'2001 - 4th Metaheuristics International Conference Porto (Portugal), 2001 July 16-20