Zdeněk Bäumelt

Advanced Methods and Models for Employee Timetabling Problems

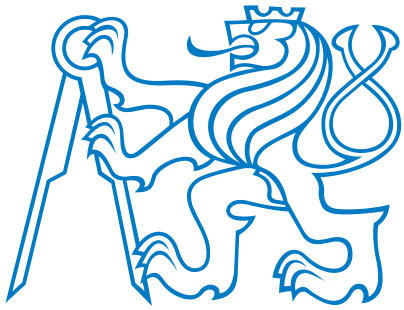
February 2015

CZECH TECHNICAL UNIVERSITY IN PRAGUE
Faculty of Electrical Engineering
Department of Control Engineering

Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Control Engineering

Advanced Methods and Models for Employee Timetabling Problems

Doctoral Thesis



Zdeněk Bäumelt

Prague, February 2015

Ph.D. Programme: Electrical Engineering and Information Technology
Branch of study: Control Engineering and Robotics

Supervisor: prof. Dr. Ing. Zdeněk Hanzálek
Supervisor specialists: Ing. Přemysl Šůcha, PhD.

To my wife and family members, with love.

Declaration

This doctoral thesis is submitted in partial fulfillment of the requirements for the degree of doctor (Ph.D.). The work submitted in this dissertation is the result of my own investigation, except where otherwise stated. I declare that I worked out this thesis independently and I quoted all used sources of information in accord with Methodical instructions about ethical principles for writing academic thesis. Moreover I declare that it has not already been accepted for any degree and is also not being concurrently submitted for any other degree.

Czech Technical University in Prague
Prague, February 2015 *Zdeněk Bäumelt*

Acknowledgments

I would like to give my great thanks to my thesis advisors prof. Dr. Ing. Zdeněk Hanzálek and Ing. Přemysl Šůcha, PhD. for their excellent guidance and continuous support throughout my PhD. studies.

A special thanks belong to my colleague Ing. Jan Dvořák who cooperated on the first model of the algorithm for the Nurse Rerostering Problem in his master thesis under my leadership.

I am grateful to all the colleagues from our group for created working conditions and a friendly atmosphere I could work in.

I would like to express a gratitude to my parents, siblings, wife and our children for their provided support, encouragement, patience and happiness that enabled me making this thesis.

Thanks to everyone who has helped me during the PhD. studies.

I would like to gratefully acknowledge the support of NVIDIA® Corporation with the donation of the GPUs.

This work was supported by the ARTEMIS initiative funded by the European Commission and the Ministry of Education of the Czech Republic under the project DEMANES 295372. This work was also supported by the Grant Agency of the Czech Republic under the Project GACR P103/12/1994 and by the Technology Agency of the Czech Republic under the Centre for Applied Cybernetics TE01020197.

Czech Technical University in Prague
Prague, February 2015 *Zdeněk Bäumelt*

Abstract

This thesis is focused on the design of efficient models and algorithms for employee timetabling problems (ETPs). From our point of view, there are two significant gaps in the current state of the art.

The first one, also important in practice, concerns the ETP with strongly varying workforce demand. Unlike the classical Nurse Rostering Problem (NRP) this problem considers dozens of shift types that can cover the demand more precisely than early, late and night shift type used in NRP. In this work we call this problem the Employee Timetabling Problem with a High Diversity of shifts (ETPHD). It comes as no surprise that the exact methods like Integer Linear Programming are not able to find its solution in reasonable time. Therefore, a transformation of ETPHD based on mapping of shift types to shift kinds was proposed. The transformation allows one to design a multistage approach (MSA). The aim of the first two stages is to find an initial ETPHD solution, where a rough position of assigned shifts is determined. This proved to be substantial for the last stage of MSA, where the solution is consequently improved in terms of its quality. In order to verify the MSA performance, a cross evaluation methodology was proposed. It is based on the comparison of the performance provided by more approaches on more combinatorial problems. Therefore, real life ETPHD instances from an airport ground company and also standard benchmark NRP instances were considered. The experiments confirmed the better or equal performance of our approach in the most of the cases.

The second gap in the literature is an absence of parallel algorithms for ETPs. We focused on the Nurse Rerostering Problem (NRRP) that appears when a disruption in the roster occurs, e.g., when one of the employees becomes sick. For this purpose, the parallel algorithm solving NRRP was proposed in order to shorten needed computational time. This algorithm was designed for a Graphics Processing Unit (GPU) offering a massive parallelization. To the best of our knowledge, this is the first usage of GPU for ETPs. The performance of the GPU parallel algorithm was tested on the real life NRRP benchmark instances and evaluated from two points of view. Firstly, the quality of the results was compared to the known results from the state of the art. Secondly, the speedup achieved by the parallel algorithm related to the sequential one was verified. In average, the parallel algorithm is able to provide the results of the same quality 15 times faster than the sequential one.

Abstrakt

Tato disertační práce je zaměřena na návrh modelů a algoritmů pro efektivní řešení problémů týkajících se rozvrhování v oblasti lidských zdrojů. Na základě provedené analýzy aktuálního stavu souvisejících prací v této oblasti byly identifikovány dva kombinatorické problémy, jimž v literatuře není věnována dostatečná pozornost.

Prvním z nich je problém rozvrhování lidských zdrojů, kde je velká variabilita požadavků na pracovní sílu. Narozdíl od známéno problému rozvrhování zdravotních sester (Nurse Rostering Problem, NRP), kde jsou pouze směny ranní, odpolední a noční, jsou v tomto problému uvažovány desítky až stovky různých směn za účelem přesnějšího pokrytí požadavků na pracovní sílu. Tento problém je nazván jako Employee Timetabling Problem with a High Diversity of shifts (ETPHD). Není překvapivé, že exaktní metody, jako je např. celočíselné programování, jsou pro řešení ETPHD nepoužitelné. Pro řešení tohoto problému byla navržena jeho transformace založená na mapování směn na typy směn. Tato transormace umožnila návrh třífázového algoritmu (MSA). Cílem prvních dvou fází je získat počáteční řešení, kde jsou již dány přibližné pozice bloků směn. Toto se ukázalo podstatné pro poslední fází, kdy je počáteční řešení zlepšováno. Pro ověření výkonnosti MSA byla navržena křížová metodika ohodnocení. Základní myšlenkou je porovnání výkonosti více algoritmů na různých kombinatorických problémech. V našem případě bylo pro experimenty použito 30 reálných ETPHD instancí ze společnosti z oblasti letecké dopravy a dále také 5 standardních NRP instancí z oblasti rozvrhování rozvrhování sester. Experimenty vyhodnocené touto metodikou ověřily, že naše výsledky jsou v drtivé většině případů lepší či alespoň stejně dobré.

Druhým problémem disertační práce z oblasti rozvrhování lidských zdrojů je problém přerozvržení přiřazených směn zdravotních sester (Nurse Rerostering Problem, NRRP). Tento problém nastává např. při onemocnění některé ze sester, kdy její směny musí být přiřazeny sestře jiné. V práci byl navržen paralelní algoritmus za účelem co nejvíce zkrátit potřebnou dobu k řešení NRRP. Paralelní algoritmus byl navíc navržen pro grafickou kartu (GPU), která umožňuje masivní paralelizaci. Dle našeho nejlepšího vědomí je to první použití GPU zaměřené na řešení NRRP. Výkonnost paralelního algoritmu byla testována na NRRP instancích z reálného života, výsledky byly vyhodnoceny ze dvou hledisek. Prvním hlediskem byla kvalita, kde byly výsledky našeho paralelního algoritmu porovnány s nejlepšími známými výsledky z literatury. Druhé porovnání bylo zaměřeno na zkrácení doby výpočtu. Paralelní algoritmus vyžaduje výrazně kratší dobu výpočtu pro získání stejné kvality NRRP řešení jako algoritmus sekvenční, a to v průměru 15 krát kratší.

Contents

List of Figures

List of Tables

Nomenclature

List of Variables and Constants

e	index for employees (nurses)
d, t, τ	indices for days within the planning horizon
s, k	indices for shifts and shift kinds
x	general index
$\alpha, \beta, \gamma, \delta$	positive weights of the criteria in the objective function
Z,Z^E	objective function, objective function of the evolutionary algorithm
$\mathrm{A}_{ev}(\dots)$	evaluated approach expressed by a metric ...
$\mathrm{A}_{ref}(\dots)$	reference approach expressed by a metric ...
$\mathrm{A}_{max}(\dots)$	maximal value expressed by a metric ... given by $\max\{\mathrm{A}_{ref}(\dots), \mathrm{A}_{ev}(\dots)\}$
$\Delta^{\mathrm{A}_{ev}}_{\mathrm{A}_{ref}}(\dots)$	relative difference of an evaluated approach A_{ev} related to a reference approach A_{ref} expressed by a metric ... in percents
S	set of shifts
$S(d)$	set of shifts required on day $d \in D$
E	set of employees (nurses)
D	set of days within the planning period
K	set of shift kinds, i.e., early, late, night, etc.
K_W, K_F	set of working shift kinds, set of non-working shift kinds
L_k	average shift length of shift kind k
b_{min}, b_{max}	minimal block length in days, maximal block length in days
br_{min}	minimal block rest length in days
M	big M – big integer number
W	vector of employees workloads
R	binary matrix representing the roster, where $R_{eds} = 1$ iff shift $s \in S$ is assigned to employee $e \in E$ on day $d \in D$
RS	matrix of requested shifts, where $RS_{sd} = 1$ iff shift $s \in S$ is requested on day $d \in D$
SP	matrix of the shift precedences, so that $SP_{s_1 s_2} = 1$ iff shift $s_1 \in S$ can be followed by shift $s_2 \in S$ on the consecutive day

Q	matrix defining skills, so that $Q_{es} = 1$ iff shift $s \in S$ can be assigned to employee $e \in E$
$R^{(\mathcal{K})}$	binary matrix representing the roster, where $R^{(\mathcal{K})}_{edk} = 1$ iff the shift of kind $k \in K$ is assigned to employee $e \in E$ on day $d \in D$
$RS^{(\mathcal{K})}$	matrix of the requested shifts, where $RS^{(\mathcal{K})}_{kd}$ is the number of the required shifts of kind $k \in K$ on day $d \in D$
$SP^{(\mathcal{K})}$	matrix representing the feasibility of two consecutive shift kinds precedence
$Q^{(\mathcal{K})}$	matrix defining skills, so that $Q_{ek} = 1$ iff shift kind $k \in K$ can be assigned to employee $e \in E$
p	index for the gene position that represents the group of the days
$R^{(\mathcal{K})}_{[e,p]}$	submatrix of $R^{(\mathcal{K})}$
$\mathcal{P}$	population in the evolutionary algorithm
$\#pop$	number of populations of an evolutionary algorithm
$pSize, pSize_0$	size of the population, size of the initial population
$offspringCount$	count of the offsprings bred in each population of EA
$\mathcal{I}$	individual of the population $\mathcal{P}$
$\mathcal{I}_{e,p}$	gene corresponding to a submatrix of the roster $R^{(\mathcal{K})}_{[e,p]}$
l	length of gene in days
$p_{(\mathcal{I})}$	survival probability of an individual $\mathcal{I}$
p_X	probability of crossover
p_M	probability of mutation
$CSE(d)$	count of the shifts that can be assigned to the employees on day $d \in D$
$SAE(d)$	binary matrix representing which shift can be assigned to which employee on day $d \in D$
G_d	bipartite graph related to day $d \in D$
$\mathbb{V}(G_d)$	set of vertices of G_d
$\mathbb{E}(G_d)$	set of edges of G_d
c, ϵ	weights of the edges $E(G_d)$
RBC	mathematical complexity of an instance computed by a tool Roster Booster (Burke and Curtois (2012))
t_{cpu}, t_{max}	computational time, time limit of the computational time
S^U	set of unassigned shifts
C^I	count of isolated days-on/days-off
R^0	matrix representing the original roster, $R^0_{e,d}$ contains shift $s \in S$ assigned to employee $e \in E$ on day $d \in D$
R_{prev}	matrix representing the previous original roster (before R^0)
$\tilde{R}, \tilde{R}^{best}$	matrix representing the modified roster, the best found modified roster

RS	matrix of the requested shifts, $RS_{s,d}$ expresses how many times is shift $s \in S$ requested on day $d \in D$
$minDaysOff$	minimal number of the days-off in each 7 consecutive days
A	matrix of absences in the original roster R^0
RP	random permutation of the roster positions corresponding to shifts to be assigned to all employees $e \in E$
RPP	random permutation of the roster positions corresponding to shifts to be assigned to a particular employee $e \in E$
RPP^{best}	random permutation of the roster positions that leads to the best found solution
i	index for the position in RP and RPP
pen	penalization function for the shift assignments
$Z(\tilde{R}), Z(\tilde{R}^{best})$	objective function, the best value of the objective function
run	counter of the runs of the algorithm
$maxRuns$	maximum number of runs of the algorithm
m	number of parallel instances of the algorithm
m_b	number of parallel instances of the algorithm per one CUDA block
q	index of the instance of the algorithm launched in one CUDA block
$isOccupied$	binary matrix of the already assigned roster positions, iff $isOccupied_{e,d} = 1$, no shift can be assigned to $\tilde{R}_{e,d}$
$unassigned$	vector of roster positions that cannot be assigned to $\tilde{R}$ to the original nurses given by R^0
$unassignedRP$	one roster positions (e, d) that cannot be assigned to $\tilde{R}$ to the original nurses given by R^0
$firstRun$	boolean to distinguish the first run from others
$firstRule$	boolean representing the mode of the current run of the algorithm instance
$feasible$	boolean representing the feasibility of the NRRP solved by one instance of the algorithm, i.e., feasibility of $\tilde{R}$
$terminateAlg$	boolean representing the flag to terminate the execution of the entire parallel algorithm
$applyLS$	boolean used to generate a new RPP by a local search applied on RPP^{best}
p_{LS}	value of the probability to swap two items in the list RPP (when the local search is applied)
$runsOfSameProb$	number of runs with probability p_{LS} (when the local search is applied)
$runsNoSuccess$	number of runs without improvement of $Z(\tilde{R}^{best})$ (when the local search is applied)

$applyBT$	boolean representing the flag whether the backtrack will be applied or not
$backtrack$	counter of backtracks made
$maxBacktracks$	maximal number of backtracks
t_{seq}	computational time of the sequential algorithm
t_{hom}, t_{het}	computational time of the homogeneous and the heterogeneous model of the parallel algorithm
$speedup_{hom}$	speedup of the homogeneous model of the parallel algorithm
$speedup_{het}$	speedup of the heterogeneous model of the parallel algorithm

Abbreviations

List of Abbreviations

VLSN	very large-scale neighborhood search
LNS	large neighborhood search
VNS	variable neighborhood search
HPC	high performance computing
AMD®	AMD® corporation
OR	operations research
HW	hardware
PC	personal computer, personal computing
A	approach
ETP	employee timetabling problem, personnel scheduling problem
ETPHD	employee timetabling problem with a high diversity of shifts
ETPHD$^{(\mathcal{K})}$	ETPHD transformed to shift kinds
TSA	tabu search algorithm
MA	memetic algorithm
EA	evolutionary algorithm
MWMA	maximal weighted matching in a bipartite graph algorithm
SIA	skill based initialization algorithm
VDSA	variable depth search algorithm
MSA	multi stage approach
CMPA$_x$	comparison approach x
NRRP	nurse rerostering problem
NRP	nurse rostering problem
TSP	traveling salesman problem
KP	knapsack problem
PFSP	permutation flowshop scheduling problem
RCPSP	resource-constrained project scheduling problem
CPU	central processing unit
GPU	graphical processing unit
GPGPU	general-purpose computation on graphics processing unit
CUDA	compute unified device architecture

NVIDIA®	NVIDIA® corporation
SIMT	single instruction multiple data
SM	streaming multiprocessor
AISA	artificial immune system algorithm
ILP	integer linear programming
SAA	simulated annealing algorithm
D19	dataset of the NRRP instances with 19 nurses (Pato and Moz (2013))
D32	dataset of the NRRP instances with 32 nurses (Pato and Moz (2013))

List of EA operators

GIP	generation of the initial population $\mathcal{P}_0$
RG	random generation of $\mathcal{P}_0$
WHG	generation of $\mathcal{P}_0$ based on the working hours of the genes
LEG	generation of $\mathcal{P}_0$ based on the lack of employees
LEWHG	generation of $\mathcal{P}_0$ as a combination of WHG and LEG
X	crossover operator
EBTSX	employee based tournament selection crossover operator
DBOPX	day based one point crossover operator
MUT	mutation operator
RM	random mutation operator
EBRM	employee based random mutation operator

Goals and Objectives

This thesis is focused on the domain of the employee timetabling problems. Its goals were set as follows:

1. To describe the basic terms, the classification and the categorization of the employee timetabling within the context of the operations research. Consequently, to identify the other goals of the thesis that reveal from significant gaps in the domain.

2. To propose and describe an approach capable to solve large instances of the employee timetabling problem having a high diversity of shifts (shift types).

3. To consider new architectures that can be exploited by a parallel algorithm applied in the domain of the employee timetabling in order to accelerate the solution of the chosen problem.

4. To verify the proposed models, algorithms and approaches on the state of the art benchmark instances and, if possible, on the real life instances.

Chapter 1

Introduction

The importance of *Operations Research* (OR) is rapidly growing since its origin in 1930s. Similarly, the combinatorial problems related to the *employee timetabling* domain, firstly mentioned in 1970s, e.g., by (Baker (1976)), have become increasingly relevant for companies having an irregular workload. The best way how to expressed the impact of the employee timetabling is to focus on its benefits, which can be divided to the employer and employee points of view as illustrated in Figure 1.1.

Firstly, the employees benefits are presented. The original motivation to solve *employee timetabling problems* (ETPs) computationally became from their solution by hand, which has been very tiresome and time consuming task for *planners* (employees designing timetables). Planners have to respect number of scheduling requirements, constraints and requests which can rarely be fully satisfied in manual design of timetables taking into account limited amount of time. Therefore, the algorithmic solutions for designing timetables are used by planners. Some manual corrections can be also made by planners manually and it is possible to alternate repeatedly these both ways of design as needed. Algorithms help at least to evaluate every change in the timetable with respect to all considered constraints and their violations can be visualized to planners. In summary, the direct benefit for planners consists in the comfortable time-saving design of timetables and, therefore, planners can be utilized efficiently for other tasks in the company. This is significantly remarkable when regular rescheduling is required due to timetable disruptions caused by e.g., sickness or other unpredictable reasons. Another valuable benefit is that more preferences of employees can be satisfied, i.e., algorithmic employee timetabling allows employees to work with respect to their free time activities and obligations, e.g., requests for a day-off, time-off within a day or a preferred shift for a given day etc. In general, timetables produced as was described above are reaching certainly better quality, since they can be better balanced

in terms of the fairness among employees, i.e., the balance of the assigned workload per employee, the number of night shifts per employee, the number

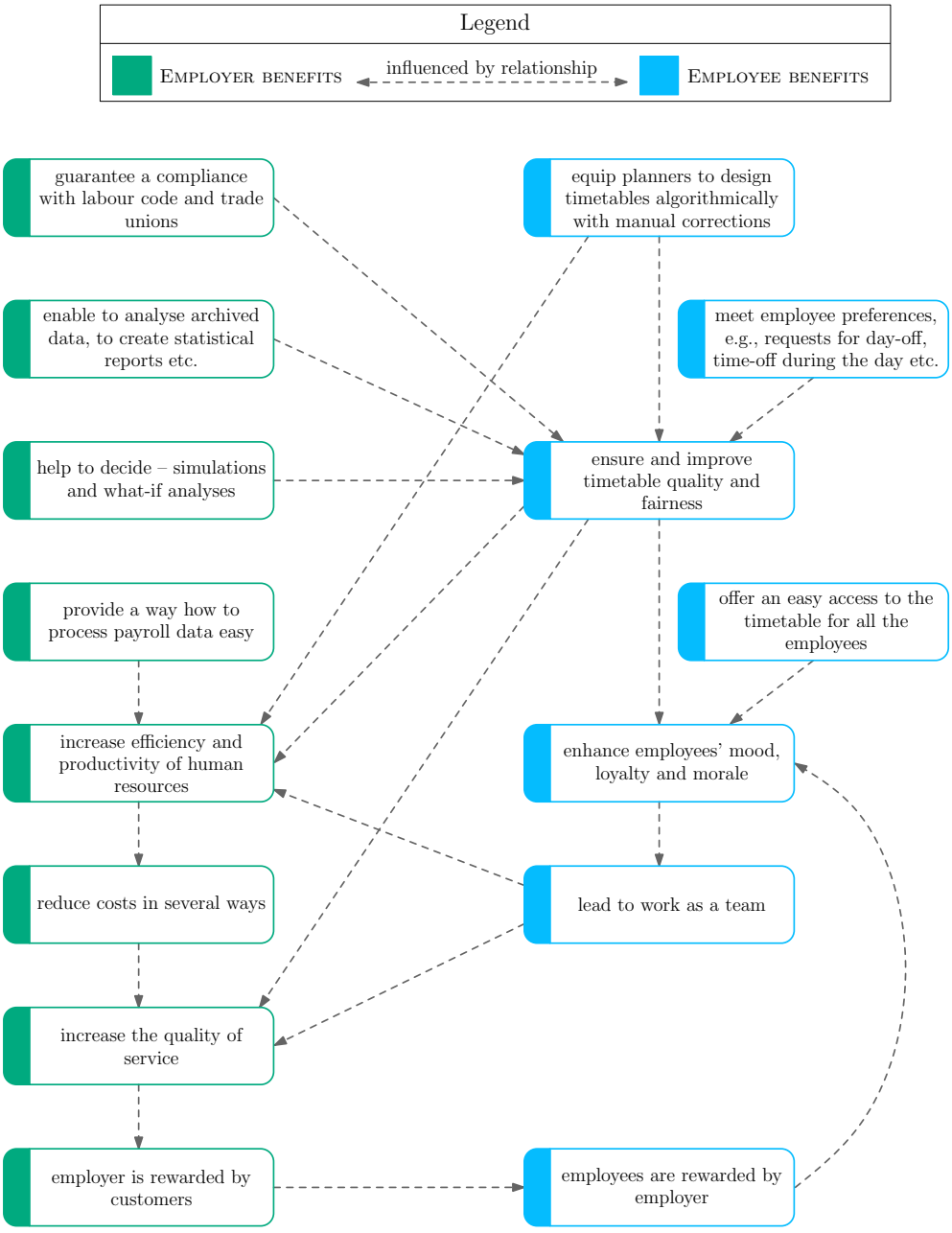


Figure 1.1: Benefits of the employee timetabling from the employer and employee points of view

of unfavorable shifts sequences per employee, the number of satisfied requests per employee etc. Moreover, every employee has an opportunity to look to the timetable and/or to insert his/her personal request as needed in case when the algorithm is a part of the employee timetabling web application. All above mentioned benefits contribute to an image of the flexible employer with friendly timetables. Therefore, the pleasant mood among employees is spread and they can be easier motivated by employer to work as a team. Naturally, employees can be more strongly motivated by the reward of their employer when the company growth is caused by the employee timetabling.

Secondly, in terms of the employer, the most essential benefit is the guarantee a compliance with the labour code and its trade unions, since there are many constraints related to the timetable design. Moreover, algorithms can be also exploited as the decision support system as follows. Employer can process statistical analyses of archived data in order to reuse experiences gained in the past, e.g., the timetable design for the planning horizon during December in the previous year, where the vacations were taken by the most of employees. Similarly, employer can analyse some what-if scenarios, e.g., related to the employee turnover, change in contracts of employees, covering the workload demand by different set of shifts, simulation of new constraints that will/should be taken into account etc. Furthermore, the payroll data can be extracted very easy from timetables. All these employer benefits help to increase the efficiency and productivity of the human resources utilization, since time needed to process following tasks can be remarkably shortened – planners designing timetables, the payroll department preparing payroll data and, finally, employees covering the workload demand more efficiently. Logically, the total cost of the employer can be significantly reduced. Consequently, the employer has an opportunity to invest saved costs in order to increase the quality of the provided service, which can be also raised by employees directly (see Figure 1.1). Consequently, the employer will be rewarded by his customers in the near future.

As you can see, there are many reasons why to dedicate some effort in solving employee timetabling problems algorithmically. With respect to this fact, this thesis deals with two real life employee timetabling problems and it is organized as follows. Firstly, the nomenclature used in this thesis is presented. Subsequently, the stated goals and objectives are summarized. Chapter 1 provides a brief motivation why to solve the employee timetabling problems algorithmically. Chapter 2 contains the necessary theoretical background related to the employee timetabling problems in order to target goals and objectives of this thesis. The employee timetabling problem having the large variety of the shifts is described and solved in Chapter 3. Namely, there are dozens of shifts in order to cover the workload demand as precisely as possible. The following Chapter 4 is focused on the timetable rescheduling in case of unexpected disruptions, which should be resolved as soon as possible. Therefore,

our objective was to accelerate its solutions with regards to known approaches. Moreover, the acceleration is performed by the parallel algorithm on a Graphics Processing Unit, which is innovative approach for this problem. Finally, the achieved results are evaluated and concluded in Chapter 5 with respect to the stated goals of the thesis.

Chapter 2

Theoretical Background

This chapter contains the necessary knowledge base, where the timetabling problematic is described within a context of operational research. A categorization and a basic terminology of the timetabling problems are presented. Subsequently, the most significant gaps in the workflow of the employee timetabling are identified.

2.1 Basic Terminology

Except the *timetabling*, other terms as a *rostering* and a *scheduling* are used for the very similar combinatorial problems and these terms are very often substituted by each other. A relationship among these terms is depicted in Figure 2.1 and their formal descriptions were defined by (Wren (1996)) as follows.

'*Scheduling* is the allocation, subject to constraints, of resources to objects being placed in space-time, in such a way as to minimize the total cost of some set of the resources used.' Namely, the classical production scheduling problems belong under this term, e.g., a Job Shop Scheduling Problem (Van Laarhoven et al. (1992)). Another example is Vehicle Routing Problem (Pisinger and Ropke (2007)) where the number of vehicles resp. drivers is minimized in combination with minimizing the total cost of the delivery.

'*Timetabling* is the allocation, subject to constraints, of given resources to objects being placed in space-time, in such a way as to satisfy as nearly as possible a set of desirable objectives.' E.g., the academic timetabling (Rudová et al. (2011)) and some of the personnel allocation problems (Van den Bergh et al. (2013)) refer to this term.

'*Rostering* is the placing, subject to constraints, of resources into slots in a pattern. One may seek to minimize some objective, or simply to obtain a feasible allocation. Often the resources will rotate through a roster.' This term

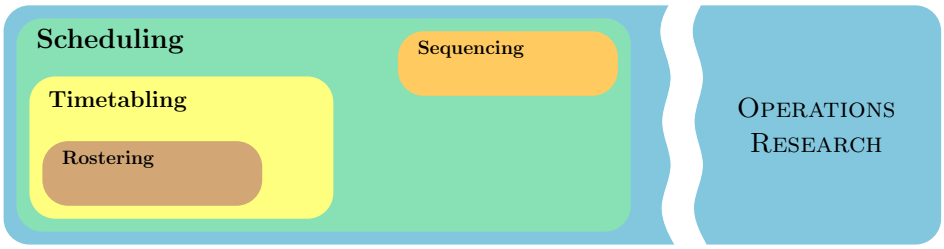


Figure 2.1: The terms related to the timetabling within a context of OR

is the most distinguishable from the others due to the well known Nurse Rostering Problem (Burke et al. (2004b)). The resources can rotate such that the schedule is cyclic, i.e., all the resources have the same schedule that is shifted according the given resource, see (Musliu (2006)) or (Rocha et al. (2013)).

Finally, one more term is defined by (Wren (1996)). '*Sequencing* is the construction, subject to constraints, of an order in which activities are to be carried out or objects are to be placed in some representation of a solution.' The examples of sequencing are a Flow Shop Scheduling Problem (Reeves (1995)), where the execution time of the schedule (makespan), given by a permutation of jobs, is minimized, or the Traveling Salesman Problem (Applegate et al. (2007)), where a sequence of the cities to be visited is required as a solution.

Nevertheless, you can notice that the terms mentioned above do not correspond to the names of the problems completely, e.g., the Flow Shop Scheduling Problem belongs to the sequencing problems. In general, the relationship among the terms is not respected strictly in the literature and the personnel scheduling and the employee timetabling are used as synonyms very often.

2.2 Categorization of Timetabling

The domain of timetabling problems belongs to the huge domain of the combinatorial optimization/operations research problems. However, the timetabling problem domain is still very wide and can be further categorized from the different point of views described in the following subsections.

2.2.1 Application Areas

The timetabling problems can be divided into several categories according to the area, where the research is applied, as illustrates Figure 2.2.

The category of an *Academic Timetabling* (see surveys (Lewis (2008); Schaerf (1999))) deals with scheduling problems occurring in an education system. Namely, a *courses timetabling*, where the goal is to construct the schedule of the lectures with respect to the students and the teachers

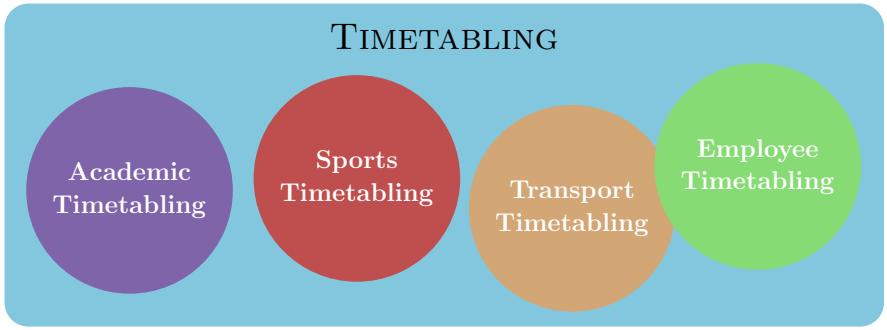


Figure 2.2: The timetabling application areas

(Rudová et al. (2011)), or an *exam timetabling* solving when the examinations of the courses should be planned in order to avoid overlapping of them for the shared students (Qu et al. (2009); Burke et al. (2001a)). Moreover, a *school timetabling* resolves the timetables for students of elementary and high schools (Cambazard et al. (2005)).

The category of a *Sports Timetabling* handles the problems of sport tournaments (Ribeiro and Urrutia (2007)) and leagues organization (Bartsch et al. (2006)). The progress in this area during the last 30 years is mapped by a comprehensive survey (Rasmussen and Trick (2008)) in order to set the unified terminology. The objective of these problems is e.g., the minimization of the traveled distance or the minimization of the breaks in the schedule.

The category of a *Transport Timetabling* tackles problems arising in the large transport systems such as railway companies (Cordeau et al. (1998)) or airline services (Gopalakrishnan and Johnson (2005)). These problems are very often called as the crew scheduling, e.g., the train crew scheduling (Ernst et al. (2001)) or the airline crew planning (Klabjan et al. (2001)), since the employees have to move from one place to another, typically in a periodical way. Naturally, the cost needed to provide the services by the crew is minimized.

A *Employee Timetabling* partially overlaps with the previous category of the Transport Timetabling, because the crew scheduling problems consider the location of the employees together with satisfying the employees' workload. The Employee Timetabling is described in more detail in Section 2.3, since it is the main application area addressed in this thesis.

2.2.2 Applied Approaches

The timetabling problems can be also organized according to the used algorithms, methods and approaches, which are illustrated in Figure 2.3. In general, there are two possibilities how to solve timetabling problems, either by an

exact method producing an optimal solution or to obtain a solution by a *sub-optimal approach* – a *heuristic*. If the heuristic is not proposed specifically to the solved problem, i.e., it can be applied due to its generality to an arbitrary combinatorial optimization problem, then it is called a *metaheuristic*.

The exact methods are frequently based on the *mathematical modeling*. An *Integer Linear Programming* (ILP), already introduced by (Warner (1976)), is very often used to formulate and solve Employee Timetabling Problems (ETPs). E.g., the workforce scheduling problems were solved in (Seçkiner et al. (2007); Thompson and Pullman (2007)), while a Nurse Rerostering Problem was formulated as an integer network flows model in (Moz and Pato (2004)). Similarly, a *Goal Programming* can be applied on ETPs having the multiple, usually conflicting objectives, see (Azaiez and Al Sharif (2005); Topaloglu and Ozkarahan (2004)). The ETPs can be also handled by a technique called a *Column Generation*, e.g., to create the timetables of the employees as in (Bard and Purnomo (2005); Al-Yakoob and Sherali (2006)). The next technique to find a solution is a well known *Branch and Bound* (Srimathy (2008)), that is able to reduce the solution space, represented by a tree, by cutting its branches that does not contain a better solution than the found best. A *Branch and Price* is based on the combination of the Branch and Bound and the Column Generation applied in each node of the solution space. The examples are shown in (Beliën and Demeulemeester (2008)), where the integrated problem of the nurse and the surgery scheduling is described, in (Freling et al. (2004)) dealing with the crew scheduling problem or in (Maenhout and Vanhoucke (2010a)). Finally, a *Constraint Programming*, which is a form of declarative programming, is utilized in e.g., (Triska and Musliu (2011)) to find a rotary schedules for the employees or (Stølevik et al. (2011)), where the constraint programming was combined together with the local search based methods to solve ETPs.

Unfortunately, the most of the employee timetabling problems are NP-

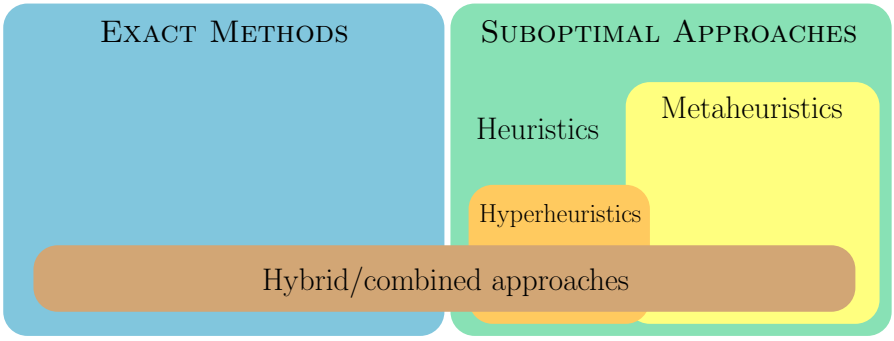


Figure 2.3: The overview of the approaches used in employee timetabling

hard (see in (Brucker et al. (2011))), i.e., no method providing a solution in a polynomial amount of time is known yet. Therefore, the exact methods can be applied on the smaller instances of the problems only and are useless on the real life instances of the problems usually. This drawback can be eliminated by suboptimal approaches that are based on the search of the solution space, which can not be fully scanned due to its size. Therefore, the space of the solutions is usually reduced by applying the intelligence gained by some expert knowledge of the solved problem. In general, in the case of the suboptimal approaches one has to find a balance between a quality of the found solution and the needed computational time.

The category of the *metaheuristics* contains the methods which were inspired by analogies from the real life or, at least, their usage was proven experimentally. One of them is a *Simulated Annealing Algorithm* (SAA), based by (Kirkpatrick et al. (1983)) and applied on ETPs e.g., in (Lučić and Teodorović (1999)). Its name is taken from an analogy of the annealing of a crystalline solid that is heated in order to obtain the most regular crystal configuration by slow cooling. This process is controlled by a temperature such that, the longer computational time, the lower temperature and the lower probability of accepting the solution having the higher objective function value. In other words, the algorithm is able to escape from the local optima by this mechanism less and less often during the execution of SAA.

Another metaheuristic called a *Tabu Search Algorithm* (TSA) was introduced by (Glover and Laguna (1989)). The key idea of TSA is to use the local search methods (adapted specifically to the solved problem) to escape from the local optimum. Moreover, the algorithm temporarily marks the moves applied to avoid the cycling in the search space of the solutions. This metaheuristic was applied on ETPs, e.g., in (Burke et al. (2004a); Bäumelt et al. (2007)).

The next group of metaheuristics is a *Very Large Scale Neighborhood search* (VLSN), see a survey (Ahuja et al. (2002)). The key idea is that the searching a very large neighborhood leads to finding a better quality local optima. The drawback of these methods is their time consumption and, therefore, the techniques to filter the solution space have to be used. The first example of VLSN metaheuristics is a *Large Neighborhood Search* (LNS) introduced by (Shaw (1998)). In this case, the neighborhood is given by heuristics in order to (partially) destroy and repair the current solution repeatedly and, consequently, to improve its quality. The second example of VLSN metaheuristics is a *Variable Neighborhood Search* (VNS) proposed by (Mladenović and Hansen (1997)). A size of the neighborhood can be adapted in order to find a local optimum in an intensification phase and to escape from the local optimum in a diversification phase of the algorithm.

Another suboptimal method dealing with ETPs becomes from the artificial intelligence. There are the decision support systems based on expert knowledge

(Beddoe et al. (2009)) gained by learning from the examples of a personnel manager behavior, which is then applied on ETPs to solve them. However, the success and the robustness of these methods are limited by the structure and the spectrum of the training ETP instances.

Subsequently, there is a huge subcategory of the metaheuristics that are biologically/nature inspired. The most known are *Evolutionary Algorithm* proposed by (Rechenberg (1971)), *Genetic Algorithms* introduced in (Holland (1975)), or *Memetic Algorithm* defined by (Moscato (1989)) that are based on an evolution of the population of individuals, where each individual corresponds to one solution. A basic idea is shared for all of them, the population is evolving in the algorithm analogously as in the nature, i.e., there are some operators for a recombination (crossover), a mutation and a selection of the individuals. These algorithms are very popular for solving ETPs, e.g., see (Landa-Silva and Le (2008); Aickelin and White (2004)), however, their typical drawback is the time consumption. This is caused by hundreds or thousands evolved generations of the population that are needed to achieve the good solutions. The second reason of longer computational times is that the population also consists of hundreds individuals usually in order to keep the diversity of the individuals in the population during the evolution. On the contrary, another evolutionary metaheuristic, called a *Scatter Search*, explores the solution space by evolving a set of reference points in order to operate a small set of solutions only. This algorithm was described in (Glover (1977)) firstly and applied on ETPs e.g., in (Burke et al. (2010)).

Moreover, there are more biological inspired algorithms that can be applied on ETPs. Namely, an *Ant Colony Optimization* in (Gutjahr and Rauner (2007)), an *Artificial Immune System* in (Maenhout and Vanhoucke (2013a)), a *Particle Swarm Optimization* in (Günther and Nissen (2010)), an *Artificial Bee Colony Algorithm* in (Buyukozkan and Sarucan (2014)), a *Harmony Search* in (Hadwan et al. (2013)) and *Neural Networks* in (Hao et al. (2004)).

Finally, you can see two more groups of the approaches in Figure 2.3. The main principle of the first one, *hyperheuristics*, is to automate, using the artificial intelligence, the process selecting or, moreover, combining simpler heuristics. Their main objective is to be applicable on more problems than on a single problem. The competition addressed to hyperheuristics was hold in 2011, see (Ochoa et al. (2012)) for more information. The second group contains *hybrid approaches* which can be combined by more of the heuristics or by an exact and suboptimal approach, e.g., (Burke et al. (2008)).

2.2.2.1 Applicability and Scalability of the Approaches

The applicability and scalability of the approaches is very interesting and important, however, these two measures are not always discussed in the ETPs

papers. Nevertheless, the survey (Van den Bergh et al. (2013)) categorizes papers according to the applicability of the used approach to four categories as follows: i) no tested – 3 papers ii) tested on the artificial data – 76 papers iii) tested on the real data – 196 papers and, finally, iv) applied in practice – 46 papers. One can see that the most common method to test the new approach is to apply it on the real life data, which is also one of the objectives of this thesis.

Unfortunately, survey (Van den Bergh et al. (2013)) did not bring the categorization in terms of the scalability provided by the used approach. In other words, there is no comparison of the approaches according to the size of the ETP instance, e.g., a number of employees, a number of shift types, etc. On the contrary, the older survey (Burke et al. (2004b)) made at least a comparison of the approaches considering the number of employees involved in the timetabling. The maximal number of employees over all approaches is up to 30, except the cyclical schedules in (Chan and Weil (2001)), where the number of employees is up to 150. To the best of our knowledge, there is no paper dealing with bigger ETP instances from the real life, i.e., having a large number of employees (e.g., one hundred) and more than a few shift types (e.g., up to 100 shift types) that are assigned to the employees. In this case, the solution space grows rapidly due to the complexity issues and it is more and more difficult to find any feasible solution and, moreover, to reach a sufficient solution quality.

2.2.3 Used Architectures

Papers dealing with combinatorial problems can be also organized according to the *hardware* (HW) used for solving the problem (see Figure 2.4). Regardless to the high increase in the progress of massively parallel devices, which has occurred during the last decade, the most common way to find a solution of ETPs remains still applying a sequential algorithm. It is executed on a single core of the *Central Processing Unit* (CPU) of a *Personal Computer* (PC).

Due to the growing number of CPU cores, the researchers have started to propose parallel algorithms on multiple core CPUs in order to exploit their full computational capability. Naturally, design of parallel algorithms is usually much more difficult than design of a sequential one, however, the operations research is strongly motivated to explore the larger part of the solution space, or to obtain the solution of the same quality within the shorter computational time. The results and the benefits of the parallel algorithms applied on the combinatorial optimization are summarized in (Talbi (2006)). Unfortunately, the application of parallel algorithms in combinatorial optimization is still less common than in other science branches, e.g., in the image processing, the simulations of chemical processes etc.

New computational architectures stated under the term *High Performance*

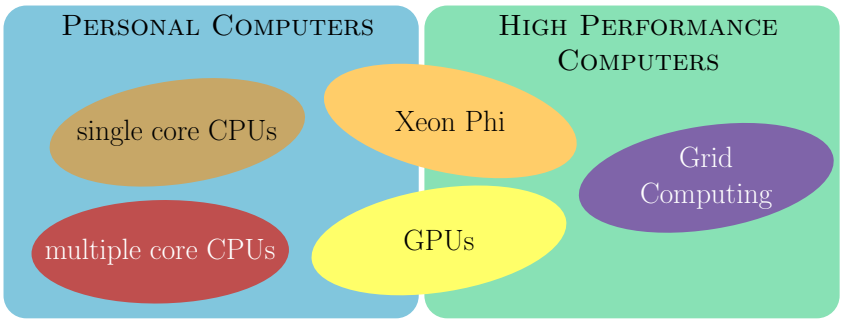


Figure 2.4: Used architectures

Computing (HPC) have started to appear in early 1990s. Due to its progress in the recent years, it seems that their role will become more and more important, see survey (Brodtkorb et al. (2013a)).

The most known and the oldest one is a *grid computing*, where the high performance is achieved by execution of the algorithm on a set of computational resources in order to reach a common goal. It is conditioned by the fact that you have to own or hire a grid, i.e., one has to deal with the budget issues in the case of owning it, or the security issues of the data in the case of hiring it. In general, there are only a few papers in parallel/grid computing domain related to OR, e.g., (Anstreicher et al. (2002); Mezmaz et al. (2014)). Grid computing is rather exploited in the academic/scientific sphere to e.g., the simulations of the problems from physics or bio-informatics.

The newest HPC HW is *Xeon Phi* (Intel Corporation (2012)) developed by Intel Corporation. This device supports a parallel computing on a single chip with multiple independent cores, in a significantly more massive way than in standard multicore CPUs. However, the price of this device is still not affordable and, probably, that is the main reason, why this device is not commonly utilized for solving the combinatorial problems yet.

Another HPC device is a *Graphics Processing Unit* (GPU). There are two manufacturers on the market, NVIDIA®, see (NVIDIA (2014)) and AMD®, see (AMD (2014)). This device is much financially accessible than Xeon Phi and, moreover, there is a support in the form of the freely available programming libraries that makes the development of the parallel algorithms on such a device easier. The number of papers utilizing GPUs is significantly higher than in the case of Xeon Phi, however, it is still rapidly lower than papers exploiting multicore CPUs, especially in the domain of the operational research.

2.3 Employee Timetabling

Historically, the first mention about employee timetabling was in 1950s and the first survey was published by (Baker (1976)). Over the passed decades, the employee timetabling has became more and more important in the same way as the entire OR. The following subsections provides a brief overview of the combinatorial problems belonging to the employee timetabling domain.

2.3.1 Application Areas

The most known ETP is the Nurse Rostering Problem occurring in hospitals, e.g., solved by (Burke and Curtois (2014)). Nevertheless, ETPs are as well addressed in other application areas, e.g., call centers (Gans et al. (2003)), protection and emergency services (Erdoğan et al. (2010)), postal services (Bard et al. (2003)), home health care services (Eveborn et al. (2006)), military services (Safaei et al. (2011)), transport services (de Matta and Peters (2009)) and also in all other areas where irregular workload is spread among the employees.

The overview of all the ETPs application areas was summarized in several surveys (Baker (1976); Cheang et al. (2003); Ernst et al. (2004); Burke et al. (2004b); Van den Bergh et al. (2013)).

2.3.2 Classification of ETPs

(De Causmaecker and Vanden Berghe (2010)) introduced the $\alpha|\beta|\gamma$ classification similar to the classical scheduling classification (Graham et al. (1979); Brucker (2007)), looking on the nurse rostering problems from three different point of views.

Firstly, α stands for a *personnel environment*, i.e., the personnel constraints (e.g., availability) and skills of the employees (e.g., whether the skills are time variant or invariant, individual etc.). Secondly, β represents a *work characteristics*, i.e., the coverage constraints (related to the coverage of the personnel demand) and shift types (properties of the shift set). Finally, γ contains specifics of the *optimization objective*, e.g., whether the optimization is multiobjective, whether it is focused on the personnel and coverage constraints etc. More examples are published in (De Causmaecker and Vanden Berghe (2011)), where some papers from the recent decade are classified.

2.3.3 Workflow of the Employee Timetabling

Independently on the ETP area, there is a large variety of the scheduling problems related to the workflow of the employee timetabling depicted in Figure 2.5. It is split into five *phases* that are executed in four *terms* – the long term, the

mid term, the short term and in real time. The content of the particular phases is described in the following subsections.

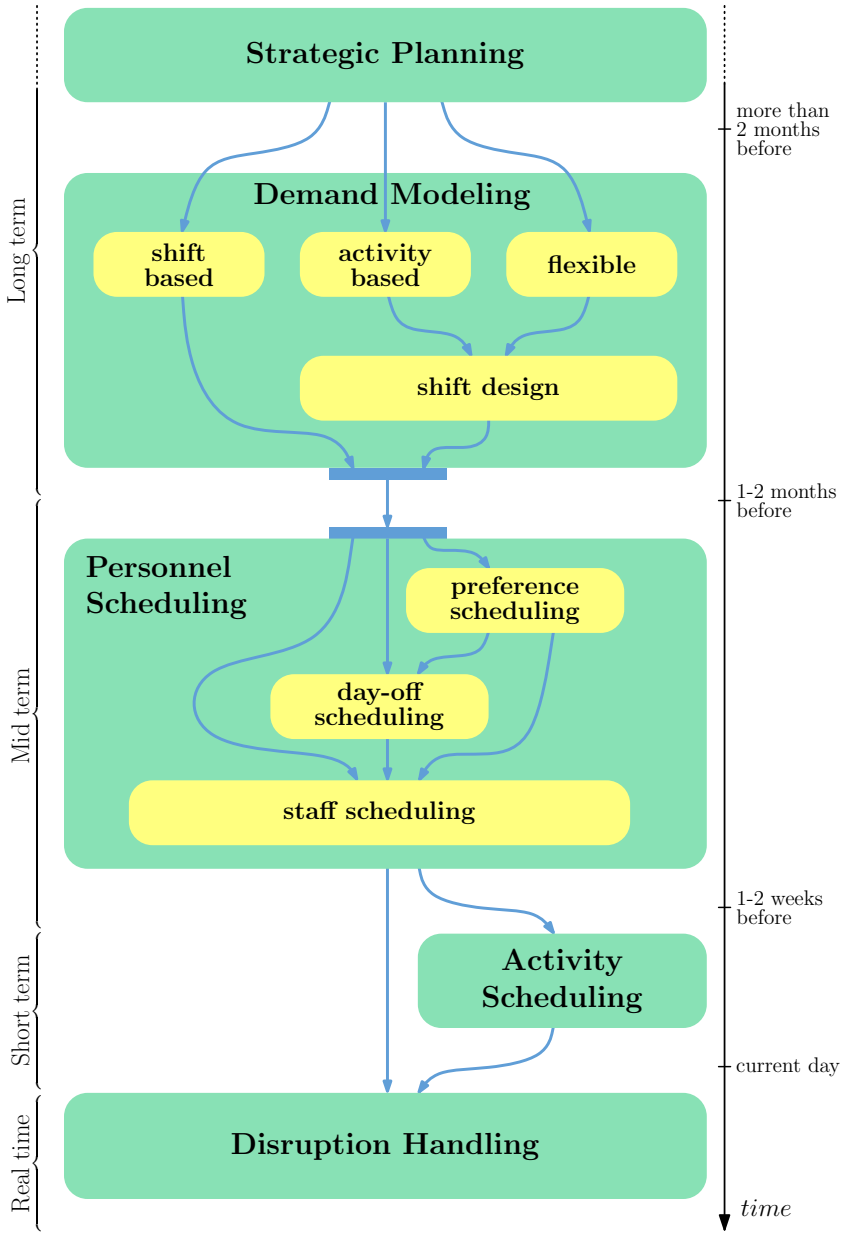


Figure 2.5: The workflow of the employee timetabling

2.3.3.1 Strategic Planning

At the beginning of the long term, some decisions based on a forecasting as well as a budgeting and a hiring of new employees have to be made by the human resources management in the phase of a *Strategic Planning*. In (Komarudin et al. (2013)) an estimation how many human resources will be approximately needed, called as *staffing*, is discussed. Moreover, the benefits of the integrated methodology on staffing and staff scheduling over the traditional iterative staffing and scheduling approach are assessed in (Maenhout and Vanhoucke (2013b)).

Another example of the Strategic Planning objectives is to decide, whether the *length of the planning horizon* will be constant (usually in the Nurse Rostering Problem) or variable (can be more effective in ETPs with the flexible demand – see Section 2.3.3.2). If the length of the planning horizon is constant, an option to make it rolling can be considered, see (Stolletz and Zamorano (2014)). In case of the variable planning horizon, settings of its length is a very sensitive decision having a significant impact on the following phases of the workflow. On one hand, a longer planning horizon makes the processes performed in all the following phases more time consuming (e.g., finding a feasible employee timetable). On the other hand, a longer planning horizon provides more options to resolve the problems such as balancing the worked hours among the employees within the timetable.

2.3.3.2 Demand Modeling

A following *Personnel Demand Modeling* phase from the long term is used to express the total workload needed for the given planning horizon. Basically, there are three kinds of the personnel demand modeling that are illustrated in Figure 2.6.

First one, a *shift based demand*, is expressed by the number of employees needed for each shift type in the given planning horizon. This model is usually used in ETPs where the demand can be easily determined according to the requested services measures. E.g., in the Nurse Rostering Problem (Azaiez and Al Sharif (2005); Burke et al. (2001b)), the number of nurses is related to the capacity of the hospital, since the nurses have to be able provide the sufficient healthcare with respect to the maximal number of the patients.

Second one, an *activity based demand*, is modeled by a list of activities that have to be completed within a given time window requiring a given skill. In this case, activities are usually grouped into sequences that can be assigned to one employee, see (Elahipanah (2012); Smet et al. (2014)). This type of the demand modeling is, e.g., used in a transport timetabling problems such as a crew scheduling (Maenhout and Vanhoucke (2010c)) or scheduling in postal

services (Bard and Wan (2006)).

The last one is a *flexible demand* that is modeled by forecasting techniques applied on the statistical data from the previous planning horizons. This is common in the transport timetabling, e.g., in airport companies, where the transport is varying during the different seasons and days in the week and, moreover, it is also dependent on bank holidays etc., see (Bäumelt et al. (2014); Örmeci et al. (2014); Green et al. (2007)).

In the case of the activity based and flexible demand, a *Shift Design* (sometimes called as *shift scheduling* as well) is processed in order to transform the personnel demand somehow into a *set of shifts*. This problem is called as a Shift Design Problem, solved e.g., in (Musliu et al. (2004)), where the set of

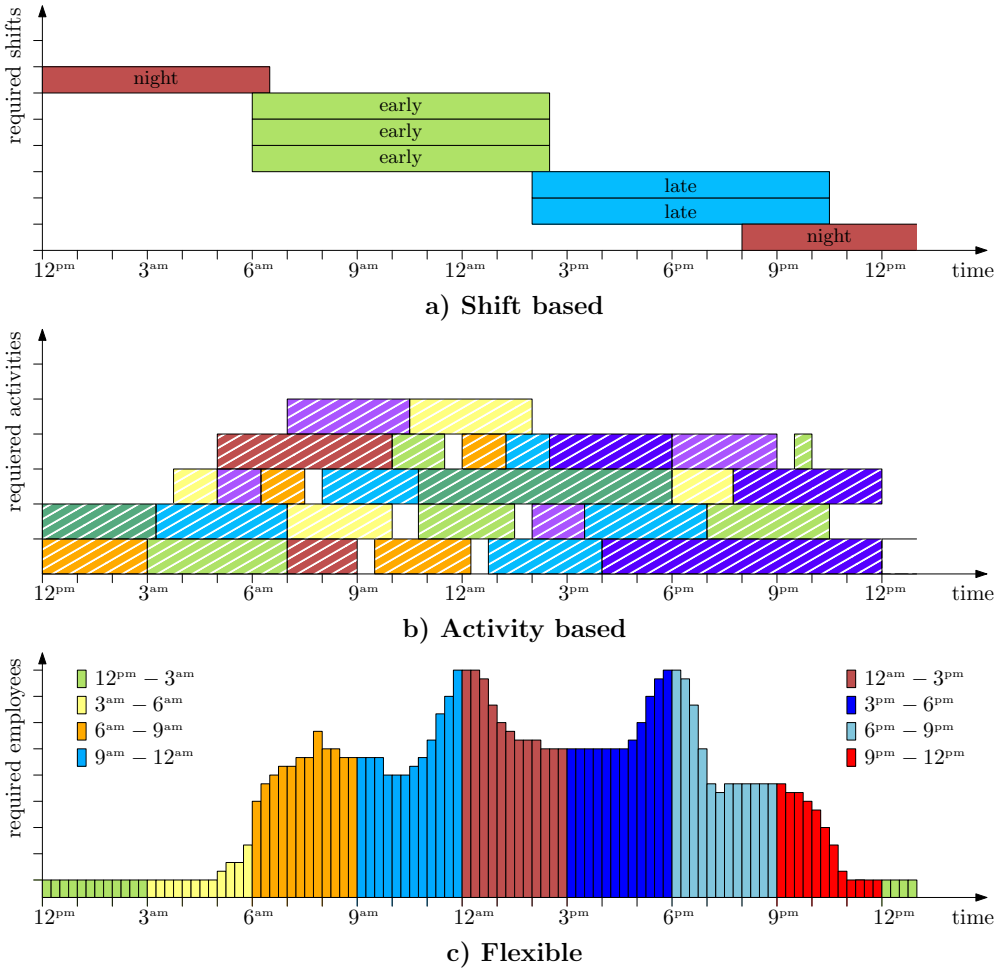


Figure 2.6: Three kinds of the personnel demand

shift types was found by the Tabu Search Algorithm. A related problem is a Minimal Shift Design Problem (Gaspero et al. (2007)), which considers as the objective to find the minimal number of shifts. When the personnel demand is modeled directly by the shifts, the Shift Design phase is skipped, since the set of shifts is already given.

2.3.3.3 Personnel Scheduling

The objective of a *Personnel Scheduling* phase, belonging to the mid term, is to assign the total workload typically expressed by the set of shifts to the employees. This phase can be further split into three parts.

The first one is a *preference scheduling*, where the main goal is to satisfy the requests of employees as much as possible, e.g., (Hanne et al. (2009); Bard and Purnomo (2005)). Employees have more types of requests, e.g., to take holiday, to go to the doctors, to have a free every second Thursday from 3pm to 8pm due to their free time activities etc. Some of the requests can be already incorporated into the timetable in this part, e.g., to assign the requested day-off to the timetable. On the contrary, some of the preferences are better to consider right in the last part of the personnel scheduling – the process of creating the timetable, e.g., one specific employee does not want to be on shifts together with another specific employee. Typically, the preference scheduling is handled manually, i.e., one has to decide whether the preference of the employee can be met or not with respect to its priority and, mainly, to the capacity of the human resources for the given planning horizon. Consequently, the requests are expressed as a part of the input data for the following two parts.

The second part is called a *days-off scheduling* and its objective is to schedule the working and non working days in the given planning horizon, see (Alfares (1998, 2001)). This part reduces the state space needed to explore to find an initial feasible solution.

The last part, a *staff scheduling*, is used in order to set the working times of the employees during the working days. The goal of this part is to find a feasible solution of the solved ETP, and, moreover, to reach the best value of the objective function with respect to the considered constraints. The majority of the ETP problems is related to this part and, therefore, they are discussed in many surveys (Ernst et al. (2004); Burke et al. (2004b); Cheang et al. (2003)).

2.3.3.4 Activity Scheduling

An *Activity Scheduling* phase follows the Personnel Scheduling phase in order to specify the particular activities within the assigned shifts, e.g., in (Quimper and Rousseau (2010); Lequy et al. (2012a,b)). Naturally, this phase

is optional, i.e., it is processed in those application areas where it makes a sense, see (Elahipanah (2012)). On the contrary, this phase is typically skipped in the ETPs such the Nurse Rostering Problem, since the activities done during the shift are created in a strongly operational way and cannot be planned one week before the planning horizon starts.

Moreover, a *tour scheduling* described thoroughly in survey (Alfares (2004)) appears in the employee timetabling papers. The tour scheduling is not illustrated in Figure 2.5 in order to keep it clear, however, this term integrates the phases of the demand modeling and personnel scheduling. Namely, the activity based personnel demand followed by the shift design is combined with the day-off scheduling in order to construct the *tours*. The main reason to call it the tour is that the activities ordered in the tour can have different locations. Consequently, the employees move among the locations in order to satisfy the sequences of the activities. This problem is addressed e.g., in (Isken (2004); Stolletz (2010); Brunner and Stolletz (2014)).

2.3.3.5 Disruption Handling

The last phase in the workflow called a *Disruption Handling* starts after publishing the final timetable. Subsequently, the published timetable has to be naturally modified with respect to the unexpected circumstances, e.g., when an employee get sick. However, the timetable cannot be completely rebuilt, since the employees have already planned some private activities during their days-off and before or after their assigned shifts. This problem is called a Nurse Rerostering Problem (NRRP) and it appears very often in the healthcare. However, compare to the Personnel Scheduling phase, there are only few papers dealing with NRRP, e.g., (Moz and Pato (2007); Maenhout and Vanhoucke (2010b, 2013a)). Due to this reason, there is no support to solve it by computers very often and it is usually solved in a manual way. However, this phase is executed operationally and should be finished as soon as possible in order to shorten these stressful situations.

2.4 Summary

The aim of this chapter was to identify significant gaps in the literature. Based on that, we decided to focus on some opened issues from the workflow of the employee timetabling (see Figure 2.5).

In the case of the Personnel Scheduling, we describe how to deal with the large ETP instances from the real life having the large variety of the shift types given by the flexible demand (called as ETPHD) in Chapter 3. The aim of the ETPHD research was to consider the dozens of shifts during the

timetable design in order to minimize the overstaffing and, consequently, to reduce personnel costs.

In the case of the Disruption Handling, we decided to exploit the GPU on NRRP (see Chapter 4) for two reasons. Firstly, Section 2.2 confirms that exploiting this hardware device to solve combinatorial problems is still not common, however, the published results from other scientific fields, e.g., physics, bio-informatics or image processing, are very promising. Secondly, our main motivation was to minimize computational time consumed by solving NRRP, since this problem belongs to the operational phase of the employee timetabling workflow requiring fast reactions.

Chapter 3

Employee Timetabling Problem with a High Diversity of Shifts

3.1 Introduction

This chapter deals with a scheduling problem at the airport belonging to the domain of employee timetabling problems (ETPs), also called employee rostering problems or personnel scheduling problems. The main difference between this problem and the most known problem from this domain, the Nurse Rostering Problem (NRP) (Burke et al. (2004b)), lies in the number of different shifts needed to satisfy a *personnel demand*, i.e., the number of employees needed at the specific time interval of the day. An illustrated example for both problems is shown in Figure 3.1, where the upper one is typical for the NRP and the lower one corresponds with the ETP typical for airports. The personnel demand, given by the statistical data from the previous planning horizons, is represented by the gray area. One is able to satisfy the personnel demand expressed by the number of required shifts to be assigned to the employees on the certain day. This ensures a *shift coverage model* (see (Burke et al. (2006))) depicted by a bold black line that corresponds to the coverage by {early, late, night} shifts in Figure 3.1 for both problems.

On the other hand, the personnel demand of the ETP from the transport services, e.g., at airports, is usually more dynamic (see the bottom chart in Figure 3.1). This is caused by the traffic peaks that are even different on various days during the week. The personnel demand is, very often, expressed in time intervals of the day in order to cover it as precise as possible. This *time interval coverage* model (see (Burke et al. (2006))) is depicted by white bars bordered by a black line. There are two ways how to deal with the coverage given by

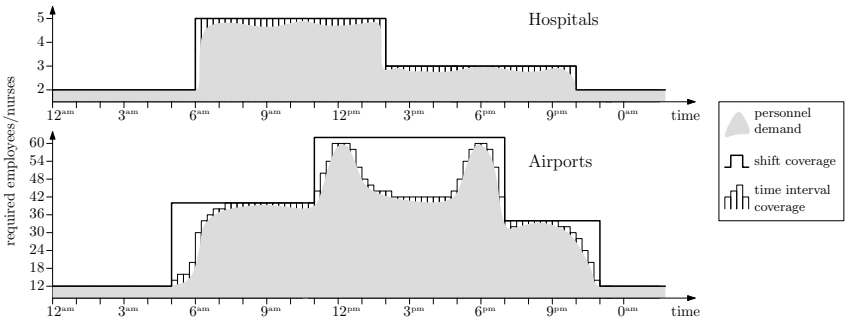


Figure 3.1: Different personnel demand models for employee timetabling problems in hospitals and airports

time intervals. Either the time intervals can be considered as independent tasks, or these tasks can be joined together and modeled as shifts, for more details see Section 3.1.1. In our case, we have an already given set of shifts (created on the base of the time interval coverage model) which allows one to cover the personnel demand very accurately (see bottom of Figure 3.1). This set not only consists of shifts with different start and finish times, but also contains split shifts and on-call shifts. The split shift facilitates covering the traffic peaks during the day, while the on-call shift is used as an alternative for employees' sick leaves and other unanticipated causes.

The objective of this chapter is to solve the employee timetabling problem with a fixed and enlarged set of shifts. We denote this problem as the Employee Timetabling Problem with a High Diversity of shifts (ETPHD). The ETPHD is not only specific by a large variety of shifts, but also by its set of constraints. The constraints that make this problem more complex are the so called *block constraints*. These constraints (described in detail in Section 3.2.1) are a generalization of the restrictions limiting the number of consecutive working days.

3.1.1 Related Works

The problem addressed in this chapter belongs to the employee timetabling area. Summaries of the approaches for solving problems from the employee timetabling/rostering domain are published in (Burke et al. (2004b); Ernst et al. (2004)). The most studied part of ETPs belongs to the health care branch (Hung (1995); Cheang et al. (2003)). In terms of the NRP classification proposed in (De Causmaecker and Vanden Berghe (2010)), our ETPHD can be categorized by ASBI|TVNO|PLGM.

The following two paragraphs are focused on the methods used to handle the common NRPs, where the shift coverage model is used. These problems can be solved optimally by an *Integer Linear Program-*

ming (ILP) (Azaiez and Al Sharif (2005)). However, this method provides the solution in a reasonable amount of time in the case of small instances only, i.e., a very limited set of shifts, a tiny set of employees and a simple set of constraints. Unfortunately, these assumptions are not usually kept for real data instances. Therefore, the ILP is used more often for simplified nurse rostering subproblems (Klinz et al. (2006)). This problem can also be modeled as a Constraints Satisfaction Problem, solved by constraint programming techniques (Cheng et al. (1997)). A hybrid approach from the domain of the declarative programming was presented in (Wong and Chun (2003)) on a simplified NRP where the authors proposed an automatically implied constraint generation. By this hybrid technique, the ratio of the solved NRPs can be increased.

The optimal approaches are usually unable to obtain the final solution in a reasonable amount of time when more difficult NRPs are considered. In this case, heuristic approaches are applied or the solved NRP is separated into its subproblems. These subproblems can be solved by different approaches (optimal or heuristic) to attain suboptimal solutions of the problem. One of the most applied metaheuristic approaches for NRPs is a *Tabu Search Algorithm* (TSA). A two stage approach to this problem is described in (Burke et al. (1999)) where, in the first step, a feasible solution with respect to hard constraints is found and, in the second step, a TSA based optimization is used. Similar stage separation is described in (Vanden Berghe (2002)) where the comparison of two approaches (TSA and *Memetic Algorithm* (MA)) for the optimization stage was presented. In a general way, TSA is faster than MA, but its computation time depends considerably on the previous initialization stage.

The papers relevant specifically to our ETPHD (not only to NRP) are described in this paragraph. There are two models from the coverage constraints point of view discussed beneath Figure 3.1. The first one is the shift coverage model used typically in most of the NRPs. The second one is the time interval coverage model considering a time scale smaller than days, e.g., hours, minutes. This model occurs in ETPs having the highly dynamic services from the time point of view, e.g., call centers (see a complex survey (Gans et al. (2003); Helber and Henken (2007))). In this case a *tour/sub daily scheduling problem* (Cezik et al. (2001); ASAP (2013)) or a *break scheduling problem* (Aykin (1996)) is handled. A tour is designed like a group of consecutive tasks (e.g., different types of the work, breaks). The arbitrary tasks in the tour can be moved or swapped. Nevertheless, in the case of our ETPHD the shifts are assigned to the roster in an atomic way, i.e., the shift cannot be separated into its tasks. Also, the time interval coverage model is considered in a *(minimal) shift design problem* (Gaspero et al. (2007); Musliu et al. (2004)). The objective is to determine how to design a set of shifts in order to cover the personnel demand (see the motivation Figure 3.1) as precise as possible. However, this

problem is not a part of our ETPHD described in this chapter (the output of the shift design problem is used as the input of the ETPHD).

The comparison of different coverage models (shift coverage and time interval coverage) is presented in (Burke et al. (2006)) and confirms that the time coverage model is more efficient from the overcoverage point of view than the shift coverage model. The time coverage model is transformed to the shift coverage model so that the personnel demand is fulfilled by different combinations of shifts, i.e., the shift design problem is solved. The main dissimilarity to our work lies in the number of shifts, which is up to 10 shifts in (Burke et al. (2006)), while our ETPHD takes into account the strictly given set of shifts enlarged to dozens or hundreds of shifts.

3.1.2 Contribution and Outline

In this chapter, we introduced a multistage approach for handling ETPHD. The basic idea lies in a transformation of the 'enlarged' set of shifts to a simpler one. The transformed timetable is initialized by an evolutionary algorithm (the first stage) and the problem instance is transformed back by an algorithm based on matching in the bipartite graph (the second stage). The objective of these stages is to determine the rough position of the blocks of shifts. The final roster is obtained during the optimization based on the TSA (the third stage). This stage uses our adaptation of the TSA suggested in (Vanden Berghe (2002)). The contributions of the chapter are: a) a transformation, based on a mapping of the shifts into the group of shifts, allowing one to solve the ETPHD described in Section 3.3 and Section 3.5, b) an ILP model of the ETPHD presented in Section 3.3, c) an algorithm for the first stage based on an evolutionary algorithm (EA) shown in Section 3.4 and d) a proposed cross evaluation methodology used to verify the contribution of the particular stages used in the different approaches applied on the different personnel scheduling problems (described in Section 3.6.4).

The chapter is organized as follows: Section 3.2 outlines the motivation problem at the airport. Section 3.3 explains the problem transformation to the problem with a reduced set of shifts and shows its ILP model. The transformed problem is solved in Section 3.4 by an EA. The inverse transformation is described in Section 3.5. The experiments and performance evaluation are summarized in Section 3.6 and the last section concludes the work.

3.2 Problem Statement

The problem solved in this chapter is inspired by a real ETPHD from the transport services. This problem is interesting through its 'enlarged' set of shifts where the shifts differ, not only, in the starting and finishing times.

There are also different split shifts to cover peaks during the day. Another interesting feature of this problem is that all mandatory constraints of the ETPHD, given by the labor code and the collective agreement, makes the problem over-constrained. Due to the problem complexity, a notation glossary is included in the beginning of the thesis.

The goal of the ETPHD is the same as in NRP, to assign the requested shifts from the set of shifts to the employees with respect to the given constraints that are discussed below in detail. From the complexity point of view, ETPHD is NP-hard, since NRP is NP-hard (Osogami and Imai (2000)) and ETPHD is an extension of NRP.

3.2.1 Constraints

From the employer point of view, the constraints considered in ETPHD are divided into two groups. The first group is stated as *hard constraints* that have to be satisfied. On the other hand, *soft constraints* can be violated, but their non-fulfillment is penalized in the objective function. From the algorithm point of view, this categorization is not very useful since different algorithm stages are focused on different constraints and have different objectives. The hard constraints considered in this problem are:

(c_1) An employee cannot be assigned to more than one shift per day.

(c_2) Shifts requiring a certain skill (grade) have to be covered by employees with this skill.

(c_3) Over coverage of shifts is not allowed.

(c_4) Under coverage of shifts is not allowed.

(c_5) The minimal time gap of free time between two shifts must be kept.

(c_6) Personnel requests must be considered – like fixed shift assignment, day-off requests, partial day-off requests (e.g., an employee is able to work to 5pm).

(c_7) The maximum number of consecutive days-on and maximum working hours in one block have to be kept.

(c_8) The minimal block rest between the blocks has to be fulfilled.

(c_9) Valid blocks of shifts must be respected, e.g., no more than one split shift is allowed in the block.

The constraint (c_5) defines a *preferred time gap* between two shifts equal to 12 hours. This preferred time gap can be shortened down to 10 hours (*minimal time gap*) subject to a condition that the following preferred time gap will be

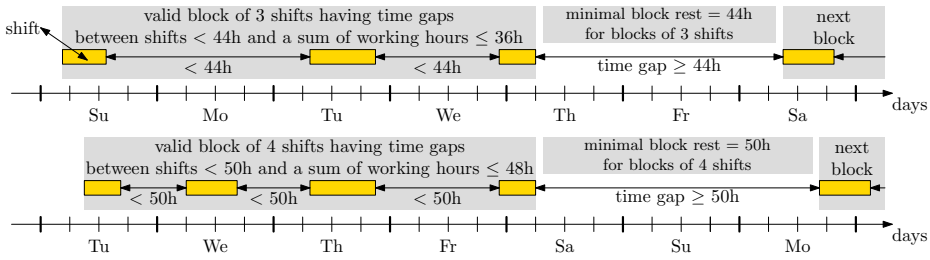


Figure 3.2: Block constraints

prolonged by the time equal to the previous shortage. The hard constraint (c_6) keeps the shifts *fixed* in the roster, e.g., a planned business trip or holiday assignments must be respected.

The hard constraints (c_7)–(c_9) dealing with the so called *block of shifts* make this ETPHD problem more difficult. The block of shifts is defined as a sequence of consecutive shifts where all time gaps between every two shifts in the block do not exceed the defined *minimal block rest* covered by (c_8), e.g., 44 hours (see Figure 3.2). In addition, this number depends on the number of shifts in the block. The hard constraint (c_7) defines that the count of working shifts in each *block* is less than or equal to the maximal shift count. Likewise, the number of working hours in the block is bounded, e.g., 36 hours for the block of 3 shifts. Furthermore, the block constraint (c_9) limits the number of certain shifts in the block.

These block constraints make the situation more complex since the position of the blocks is crucial for the quality of the resulting schedule. Therefore, in our opinion, it rules out the majority of the single stage approaches since the rough position of the block should be determined in the first stage respecting the fixed shifts in the roster (e.g., planned holidays, planned business trips, etc.). According to our experiments, it has a positive influence on the quality of resulting schedule.

The soft constraints considered in ETPHD are:

(c_{10}) Overtime hours should be balanced according to the employee's workload.

(c_{11}) The number of isolated days-on and isolated days-off should be minimized.

(c_{12}) The number of blocks having the length smaller than the given number should be minimized.

3.2.2 Problem Statement

This section describes the problem representation in a rigorous way due to several reasons. Firstly, this problem is expressed by an ILP model in the

following section. Furthermore, a transformation used in order to simplify the solved problem is a part of this model. Finally, the problem complexity can be computed with respect to the precise description in Section 3.3.2.

Let E be a *set of employees*, D represents a *set of days* from the whole planning period and S denotes the *set of shifts*. Consequently, the *roster* is represented by R, a binary matrix such that $\forall e \in E, \forall d \in D, \forall s \in S$

$$R_{eds} = \begin{cases} 1, & \text{shift } s \text{ is assigned to employee } e \text{ on day } d \\ 0, & \text{otherwise} \end{cases} \tag{3.1}$$

When the roster contains a *fixed shift* s, defined due to (c_6), the corresponding $R_{eds} = 1$ is a constant and another shift cannot be assigned to employee e on day d. Furthermore, the fact that shift s can only be assigned to employee e having a certain *skill* is given by matrix Q so that

$$Q_{es} = \begin{cases} 1, & \text{shift } s \text{ can be assigned to employee } e \\ 0, & \text{otherwise} \end{cases} \tag{3.2}$$

The coverage constraints (c_3) and (c_4) from Section 3.2.1 are expressed by a binary matrix RS where $RS_{sd} = 1$ iff shift $s \in S$ is required on day $d \in D$. Subsequently, in relationship to constraint (c_5), we can define a binary matrix of *shift precedences* SP so that

$$SP_{s_1 s_2} = \begin{cases} 1, & \text{shift } s_1 \text{ can be followed by shift } s_2 \text{ on the subsequent day} \\ 0, & \text{otherwise} \end{cases} \tag{3.3}$$

where $s_1, s_2 \in S$.

The shifts of the set S can be joined into groups given by a mapping $\mathcal{M} : S \longmapsto K$ where $K = \{\mathcal{F}, \mathcal{H}, \mathcal{E}, \mathcal{L}, \mathcal{N}, \mathcal{S}, \mathcal{O}\}$ is a *set of shift kinds*. The set of shift kinds consists of {free $\mathcal{F}$, required free or holiday $\mathcal{H}$, early shifts $\mathcal{E}$, late shifts $\mathcal{L}$, night shifts $\mathcal{N}$, split shifts $\mathcal{S}$ and on-call shifts $\mathcal{O}$}. Let K_W and K_F be subsets of K defined as follows

$$\begin{aligned} K_W &= \{\mathcal{E}, \mathcal{L}, \mathcal{N}, \mathcal{S}, \mathcal{O}\} \\ K_F &= \{\mathcal{F}, \mathcal{H}\}. \end{aligned} \tag{3.4}$$

i.e., K_W is the subset of *working shift kinds*, K_F represents *free shift kinds*. Furthermore, for each $k \in K$, let L_k be an *average shift length* of kind k so that $L_k = \text{avg}_{s \in S \ | \mathcal{M}(s)=k} |s|$ where $|s|$ is the length of shift $s \in S$. Finally, *workloads* of all employees E are defined by a non-negative vector W according to the length of the planning period. In order to simplify orientation in the chapter, the used nomenclature is summarized in the beginning of the thesis.

3.3 Transformation of the Problem and its Mathematical Model

The goal of the first stage of the algorithm is to determine the rough position of the blocks, i.e., a placement of the shift kinds with respect to the hard and soft constraints. The *rough blocks* of shift kinds can be modeled as blocks of days-on separated by days-off. In addition, the output of the first stage determines which kind of shift $k \in K$ should be assigned to the given employee on the given day in the block. Two approaches of the first stage are described in Section 3.3 and 3.4. This section presents a mathematical model based on a transformation used in the first stage.

3.3.1 Transformation $\mathcal{SK}$

Let $\mathcal{SK}$ be the transformation resulting from the mapping $\mathcal{M} : S \longmapsto K$. The $\mathcal{SK}$ transforms ETPHD to ETPHD$^{(\mathcal{K})}$, specifically roster R to $R^{(\mathcal{K})}$ where $R^{(\mathcal{K})}_{edk} = 1$ iff shift kind k is assigned to employee e on day d. In the same way, RS becomes $RS^{(\mathcal{K})}$ where $RS^{(\mathcal{K})}_{kd}$ is the number of required shifts of kind k for day d. Finally, SP becomes $SP^{(\mathcal{K})}$ such that $SP^{(\mathcal{K})}_{k_1k_2}$ expresses, whether the shifts of kind k_1 can be followed by the shifts of kind k_2. The transformation is defined by equations (3.5)–(3.8).

$\forall e \in E, \forall d \in D, \forall k \ \in K$:

$$R^{(\mathcal{K})}_{edk} = \begin{cases} 1, & \exists s \ \in S \ \mid R_{eds} = 1 \wedge \mathcal{M}(s) = k \\ 0, & \text{otherwise} \end{cases} \tag{3.5}$$

$\forall k \ \in K, \forall d \in D$:

$$RS^{(\mathcal{K})}_{kd} = \sum_{s \in S \ \mid \mathcal{M}(s)=k} RS_{sd} \tag{3.6}$$

$\forall e \in E, \forall k \ \in K$:

$$Q^{(\mathcal{K})}_{ek} = \begin{cases} 0, & \exists s \ \in S \ \mid Q_{es} = 0 \wedge \mathcal{M}(s) = k \\ 1, & \text{otherwise} \end{cases} \tag{3.7}$$

$\forall k_1, k_2 \in K$:

$$SP^{(\mathcal{K})}_{k_1k_2} = \begin{cases} 0, & \exists s_1, s_2 \in S \ \mid SP_{s_1s_2} = 0 \wedge \mathcal{M}(s_1) = k_1 \wedge \mathcal{M}(s_2) = k_2 \\ 1, & \text{otherwise} \end{cases} \tag{3.8}$$

3.3.2 Integer Linear Programming Model of ETPHD$^{(\mathcal{K})}$

Design of blocks in the roster can be formulated by an ILP model, where a multicriteria objective function Z is defined by a linear combination of the constraints (c_3), (c_4) and (c_{10}) fulfillment, where $\alpha, \beta > 0$ are weights of the criteria. These criteria are evaluated by the piecewise linear functions (e.g., absolute value function) penRS and penW. The penRS function reflects the over and under coverage of the assigned shift kinds, while the penW function corresponds to the coverage of the employees' workloads. These functions are represented in the ILP model by a set of auxiliary variables that are not incorporated into equations (3.9)–(3.16) in order to make the model more readable.

$$\min Z = \min \left\{ \alpha \cdot \sum_{k \in K_W} \sum_{d \in D} \text{penRS}\left(RS_{kd}^{(\mathcal{K})} - \sum_{e \in E} R_{edk}^{(\mathcal{K})} \right) + \beta \cdot \sum_{e \in E} \text{penW}\left(W_e - \sum_{d \in D} \sum_{k \in K} L_k \cdot R_{edk}^{(\mathcal{K})} \right) \right\} \tag{3.9}$$

subject to

$\forall e \in E, \forall d \in D$:

$$\sum_{k \in K} R_{edk}^{(\mathcal{K})} = 1 \tag{3.10}$$

$\forall e \in E, \forall d = \langle 1, |D| - 1 \rangle, \forall k_1, k_2 \in K$:

$$R_{edk_1}^{(\mathcal{K})} + R_{e,d+1,k_2}^{(\mathcal{K})} - SP_{k_1 k_2}^{(\mathcal{K})} \leq 1 \tag{3.11}$$

$\forall e \in E, \forall d \in D | Q_{ek}^{(\mathcal{K})} = 0$:

$$R_{edk}^{(\mathcal{K})} = 0 \tag{3.12}$$

$\forall e \in E, \forall t = \langle 1, |D| - b_{max} \rangle$:

$$\sum_{d=t}^{t+b_{max}} \sum_{k \in K_W} R_{edk}^{(\mathcal{K})} \leq b_{max} \tag{3.13}$$

$\forall e \in E, \forall d = \langle 2, b_{min} \rangle, \forall d = \langle 1, |D| - t \rangle$:

$$\sum_{k \in K_W} \left(R_{edk}^{(\mathcal{K})} - R_{e,d+1,k}^{(\mathcal{K})} + R_{e,d+t,k}^{(\mathcal{K})} \right) \geq 0 \tag{3.14}$$

$$\forall e \in E, \forall d = \langle 2, br_{min} \rangle, \forall d = \langle 1, |D| - t \rangle : \sum_{k \in K_W} \left(R^{(\mathcal{K})}_{edk} - R^{(\mathcal{K})}_{e,d+t-1,k} + R^{(\mathcal{K})}_{e,d+t,k} \right) \leq 1 \tag{3.15}$$

$$\forall e \in E, \forall \tau = \langle b_{min}, b_{max} \rangle, \forall t = \langle 1, |D| - \tau \rangle : \sum_{d=t}^{t+\tau} R^{(\mathcal{K})}_{ed(k=\mathcal{S})} \leq 1 + M \cdot \sum_{d=t}^{t+\tau} \sum_{k \in K_F} R^{(\mathcal{K})}_{edk} \tag{3.16}$$

The constraints of the ILP model are stated by equations (3.10)–(3.16). The first constraint equation (3.10) corresponds to the constraint (c_1), i.e., one shift kind is assigned per day. Similarly, equation (3.11) matches the constraint (c_5) represented by $SP^{(\mathcal{K})}_{k_1 k_2}$ and equation (3.12) stands for the skills of employees (c_2). The constraints (c_7), (c_{12}), (c_8) are given by (3.13), (3.14), (3.15). A maximal block length b_{max} of the working shift kinds is constrained by (3.13), while the following equation (3.14) considers the minimal length of the blocks b_{min}. The last inequality from the block constraints (3.15) defines the minimal block rest length br_{min} between the block of shifts. Since the ILP model of the first stage is formulated on shift kinds, constraints (3.13)–(3.15) limit the number of consecutive shift kinds only instead of the sum of hours as is defined by block constraints. But it is sufficient since the objective of the first stage is just to determine the rough position of blocks. The values for $(b_{max}, b_{min}, br_{min})$ used in the solved ETPHD$^{(\mathcal{K})}$ are fixed and given by $(5, 3, 2)$. In the last stage of our approach these constraints (c_7), (c_8) are reflected in the complete form. The last equation (3.16) stands for (c_9) to avoid more shifts of the same kind in one block, e.g., it is not feasible to have more than one

Table 3.1: Number of constraints and binary and continuous variables in the ILP model

type	**number**
binary variables	$\|E\| \cdot \|D\| \cdot \|K\|$
continuous variables	$\|E\| + \|D\| \cdot \|K\|$
constraints (3.10)	$\|E\| \cdot \|D\|$
constraints (3.11)	$\|E\| \cdot (\|D\| - 1) \cdot \|K\|^2$
constraints (3.13)	$\|E\| \cdot (\|D\| - b_{max})$
constraints (3.14)	$\|E\| \cdot (\|D\| - 2 + \ldots + \|D\| - b_{min})$
constraints (3.15)	$\|E\| \cdot (\|D\| - 2 + \ldots + \|D\| - br_{min})$
constraints (3.16)	$\|E\| \cdot (\|D\| - b_{min} + \ldots + \|D\| - b_{max})$

split shift, i.e., $k = \mathcal{S}$, in one block. The term $M \cdot \sum_{d=t}^{t+\tau} \sum_{k \in K_F} R_{edk}^{(\mathcal{K})}$ on the right side of (3.16) eliminates the equation in effect, when t consecutive days contain a free shift kind $k \in K_F$, i.e., it is not a block of the consecutive shifts. M is a big integer number.

The size of the ILP model stated by (3.5)–(3.16) is summarized in Table 3.1. This table considers the worst case, when no variable is relaxed, e.g., by skills expressed in (3.12). For an instance with $|E| = 100$ employees, the planning horizon $|D| = 30$ days and shift kinds $|K| = 7$ the total number of binary variables equals 21000 while the number of continuous variables is 310. When we consider $b_{max} = 5$, $b_{min} = 3$ and $br_{min} = 2$, the ILP model contains 166500 constraints.

3.4 Solution of the First Stage by an Evolutionary Algorithm

A solution of ETPHD$^{(\mathcal{K})}$ is the output of the algorithm's first stage. Due to enormous size of the ILP model (see Table 3.1) we need a faster way to find an initial schedule. Therefore, the solution of the first stage is found heuristically, namely by an evolutionary algorithm (EA), outlined in Algorithm 1, and discussed in this section.

Algorithm 1: An evolutionary algorithm pseudo-code

Input : ETPHD instance
Output: Roster $R^{(\mathcal{K})}$

ETPHD$^{(\mathcal{K})}$ $\leftarrow$ `Preprocessing(`$ETPHD$`)`
$\mathcal{P}_0 \leftarrow$ `GeneratePopulation(`$ETPHD^{(\mathcal{K})}$, $pSize_0$`)`
foreach $\mathcal{I} \in \mathcal{P}_0$ **do** `Evaluate(`$\mathcal{I}$`)`
$\mathcal{P} \leftarrow \mathcal{P}_0$; $q \leftarrow 0$
while $q < \#pop$ **do**
  $\mathcal{P} \leftarrow$ `Select(`$\mathcal{P}, pSize$`)` `// select the` $pSize$ $\mathcal{I} \in \mathcal{P}$
  $\mathcal{P}_N \leftarrow \emptyset$ `// clear population` $\mathcal{P}_N$
  while $|\mathcal{P}_N| < offspringCount$ **do** `// breed` $offspringCount$ `offsprings`
    $[\mathcal{I}_1, \mathcal{I}_2] \leftarrow$ `ChooseParents(`$\mathcal{P}$`)`
    $\mathcal{I}_N \leftarrow$ `Crossover(`$\mathcal{I}_1, \mathcal{I}_2$`)`
    $\mathcal{I}_N \leftarrow$ `Mutate(`$\mathcal{I}_N$`)` with probability p_M
    $\mathcal{P}_N \leftarrow \mathcal{P}_N \cup \mathcal{I}_N$ `// add offspring` $\mathcal{I}_N$
  foreach $\mathcal{I} \in \mathcal{P}_N$ **do** `Evaluate(`$\mathcal{I}$`)` `// evaluate` $\mathcal{P}_N$
  $\mathcal{P} \leftarrow \mathcal{P} \cup \mathcal{P}_N$ `// merge populations`
  $q \leftarrow q + 1$
$R^{(\mathcal{K})} \leftarrow \mathcal{I} \in \mathcal{P}$ with the lowest value of Z^E
return $R^{(\mathcal{K})}$

3.4.1 Encoding

A direct value encoding is used to represent the rosters $R^{(\mathcal{K})}$ as individuals $\mathcal{I}$. The shift kinds assigned to the fixed number of consecutive days constitute a *gene*. The number of days representing one gene is called a *gene length* denoted as l. The genes are indexed by two indices (see Figure 3.3). The first one is the index of employee e and the second one is the index of the gene position p such that $p = \{\lfloor \frac{d}{l} \rfloor \mid \forall d \in D\}$. Thereafter, a submatrix of $R^{(\mathcal{K})}$ given by $R^{(\mathcal{K})}_{[e,p]} = R^{(\mathcal{K})}_{edk} \mid (p-1) \cdot l < d \leq p \cdot l$ corresponds to a *gene* $\mathcal{I}_{e,p}$. Each gene $\mathcal{I}_{e,p}$ is encoded to an integer value given by $\sum_{d=p\cdot l}^{(p+1)\cdot l-1} \left(|K|^{d-p\cdot l} \cdot \sum_{k=0}^{|K|-1} \left(k \cdot R^{(\mathcal{K})}_{edk} \right) \right)$. Naturally, the count of genes is reduced by (3.8), e.g., a sequence of shift kinds $[\mathcal{N}, \mathcal{E}, \mathcal{F}, \mathcal{F}]$ is excluded.

3.4.2 Preprocessing

The `Preprocessing` function contains the described transformation $\mathcal{SK}$ (Section 3.3.1). Moreover, this function initializes static parts in order to accelerate the evaluation of the rosters. More specifically, all permutations with a repetition of shift kinds $k \in K$ of length l are generated and evaluated with respect to the considered hard constraints (c_1), (c_5), (c_6) (infeasible permutations are excluded). Similarly, the precedences of genes are assessed with respect to the block constraints (c_7)–(c_9), (c_{12}) presented in (3.11)–(3.16), i.e., the infeasible gene precedences are forbidden. On the other hand, the feasible gene precedences are penalized according to their shift kinds on the boundary of the genes (see Section 3.4.4).

3.4.3 Generation of the Initial Population (GIP)

The initial population $\mathcal{P}_0$ is accomplished by the `GeneratePopulation` function, i.e., $pSize_0$ individuals are created. Generally, the individual $\mathcal{I} \in \mathcal{P}$, constituted by $R^{(\mathcal{K})}_{[e,p]}$, represents one roster $R^{(\mathcal{K})}$. For each individual $\mathcal{I} \in \mathcal{P}_0$, the roster is created step by step for each employee separately. Employee e is given so that the employees are sorted by the descending order of the shift kinds that cannot be moved to another employee, i.e., the number of the fixed shift kinds, e.g., holiday.

Furthermore, four different heuristics RG, WHG, LEG and LEWHG were proposed to construct the individuals. Firstly, the order in which the positions are initialized is given by the *order of gene positions* (see the second column in Table 3.2). Secondly, the order in which genes are assigned is given by the *order of feasible genes* (see the third column in Table 3.2).

The order of the gene positions in the heuristics RG and WHG reflects the count of the feasible genes to the given position. In other words, genes having

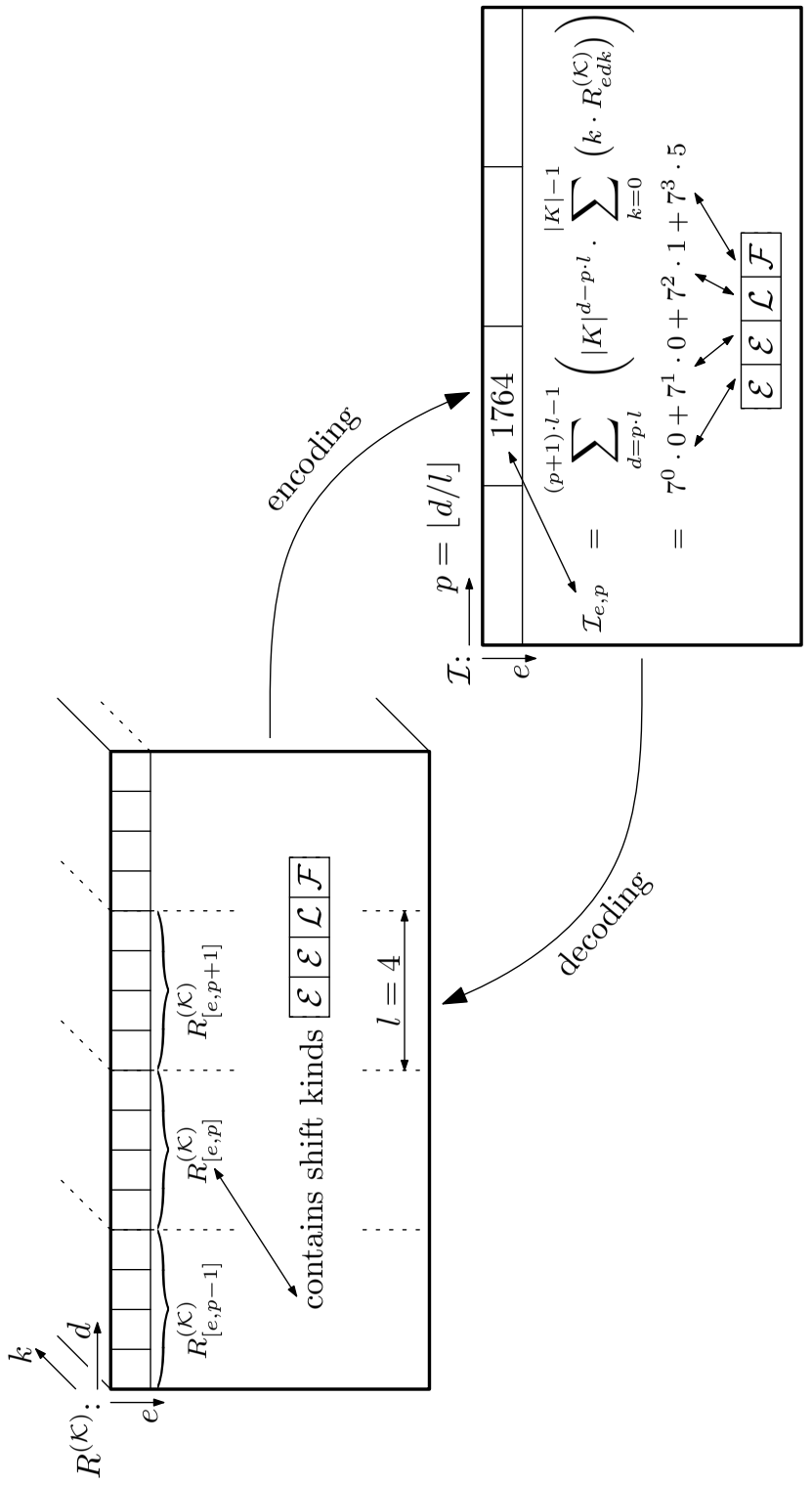


Figure 3.3: The gene representation

Table 3.2: Different heuristics for the generation of the initial population $\mathcal{P}_0$

type of GIP	order of gene positions	order of feasible genes
RG	count of feasible genes asc.	random
WHG	count of feasible genes asc.	count of working hours desc.
LEG	lack of employees desc.	random
LEWHG	lack of employees desc.	count of working hours desc.

more fixed shifts are assigned first. For the heuristics LEG and LEWHG the gene positions with the maximal lack of employees are preferred. The goal of this strategy is to cover the requested shift kinds $RS^{(\mathcal{K})}$ as precisely as possible.

The order of feasible genes remain to be determined for each gene position $p \in \{0, \ldots, \lfloor |D| / l \rfloor\}$. Feasible values for the given gene are restricted to the ones respecting precedences, i.e., the forbidden combinations of the consecutive genes are ignored. Either the order can be generated in a random way (heuristics RG and LEG) or the feasible genes with the higher number of working hours are preferred (heuristics WHG and LEWHG). This strategy is used in order to improve the employees' workload fulfillment.

3.4.4 Objective Function Z^E

The multicriteria objective function Z^E(3.17) is used in the `Evaluate` method in Algorithm 1. The following constraints are taken into account as penalties in Z^E.

The first two elements correspond to the objective function (3.9), i.e., penalties of the under and over coverage of the required shift kinds (constraints (c_3) and (c_4)) and penalties of the unbalanced workload of the employees (c_{10}). The next two terms follow from the roster encoding. The third element of Z^Epenalizes the quality of $R^{(\mathcal{K})}_{[e,p]}$ w.r.t block constraints (c_7)–(c_9), (c_{12}) expressed in (3.13)–(3.16).

The last element is focused on the gene precedences penalties. It is necessary to take the borders of the genes after the recombination into account, i.e., the kind precedences $SP^{(\mathcal{K})}$ and the block constraints (c_7)–(c_9), (c_{12}) related to the neighborhood genes have to be checked and penalized.

$$\min Z^E = \min \left\{ \alpha \cdot \sum_{k \in K_W} \sum_{d \in D} \text{penRS} \left(RS_{kd}^{(\mathcal{K})} - \sum_{e \in E} R_{edk}^{(\mathcal{K})} \right) + \right.$$
$$\beta \cdot \sum_{e \in E} \text{penW} \left(W_e - \sum_{d \in D} \sum_{k \in K} L_k \cdot R_{edk}^{(\mathcal{K})} \right) +$$
$$\gamma \cdot \sum_{e \in E} \sum_{p \in GS} \text{penGene} \left(R_{[e,p]}^{(\mathcal{K})} \right) +$$
$$\left. \delta \cdot \sum_{e \in E} \sum_{p \in \langle 1, |GS|-2 \rangle} \text{penPrec} \left(R_{[e,p]}^{(\mathcal{K})}, R_{[e,p+1]}^{(\mathcal{K})}, R_{[e,p+2]}^{(\mathcal{K})} \right) \right\} \tag{3.17}$$

3.4.5 Selection (SEL)

The *rank based wheel selection* similar to (James et al. (2007)) is used in the `Select` function to reduce the number of individuals of $\mathcal{P}$ to $pSize$. Firstly, the individuals are ordered by their quality measured by Z^E. The sorted individuals are labeled by a *rank* function as follows: The best individual gets the rank $|\mathcal{P}|$ (equal to the total count of individuals in the population before the selection is applied), the second one is marked with the rank $|\mathcal{P}| - 1$, the third one with $|\mathcal{P}| - 2$, ..., and the last rank is equal to 1. Finally, the ranks are transformed to the survival probabilities $p_{\mathcal{I}}$ of particular individuals by equation

$$p_{(\mathcal{I})} = \frac{2 \cdot rank(\mathcal{I})}{|\mathcal{P}| \cdot (|\mathcal{P}| + 1)}, \qquad \forall \mathcal{I} \in \mathcal{P}. \tag{3.18}$$

These values of survival probabilities are normalized between 0 and 1. Finally, the elitism set keeps the best $\mathcal{I} \in \mathcal{P}$ alive.

3.4.6 Crossover Operators (X)

All crossover operators X are applied in the breeding process of the offspring individual $\mathcal{I}_N$ with a probability $p_X = 0.85$. There are more possibilities, how to perform the `Crossover` function in ETPHD$^{(\mathcal{K})}$. However, we decided to apply the *employee based tournament selection crossover* (EBTSX) presented in (Maenhout and Vanhoucke (2008)). The basic idea is as follows: For given parent individuals $\mathcal{I}_1, \mathcal{I}_2$ the better roster of employee e with respect to $Z^E$is chosen. The choice of a better roster of each employee e is executed with the certain probability which was experimentally set to 0.6 for the best performance. An advantage of EBTSX is that there is no need to apply any repair

operators for this type of the crossover, because the whole rosters of the employees are copied to the offspring, i.e., all constraints related to precedences (kind precedence, gene precedence) are preserved. The influence of the crossover to the constraints evaluated for each day across all employees, e.g., shift kinds coverage, is incorporated to the objective function Z^E.

On the contrary, for the crossover based on the combination of the rosters of the particular employees, it is better to apply the *day based one point crossover* (DBOPX) (Maenhout and Vanhoucke (2008)) instead of the uniform one. Generally, the objective function Z^E increases rapidly for each point of such crossover due to precedence constraints violations. Moreover, if some of the hard constraints are violated after the DBOPX execution, e.g., the number of assigned shift kinds in two consecutive genes is greater than b_{max}, then the repair mechanism has to be applied to these violations at the point of the crossover. It is realized by a repeatedly applied mutation in order to fix all the violated precedences of the genes in the employee's roster. This repair mechanism is applied to each employee where the gene precedence is violated. However, DBOPX is noticeably more time consuming in comparison to the rest of the evolutionary algorithm and for that reason it is not used in the presented experiments.

3.4.7 Mutation Operators (MUT)

All mutation operators MUT are applied to the offspring $\mathcal{I}_N$ with a probability p_M after the application of the crossover. The best performance of the algorithm is reached by $p_M = 0.4$ since there is no other possibility to change the particular employees' rosters inside.

Two different mutation operators were tested. The first one is a *random driven mutation* RM. Firstly, the number of employees for the mutation is selected randomly from the interval of $\langle \lfloor 0.25 \cdot |E| \rfloor ; \lceil 0.75 \cdot |E| \rceil \rangle$. Then, for each employee chosen in a random way from all the employees determined for the mutation, the count of genes to be mutated has to be determined. Subsequently, for each specific gene picked in a random way the randomly selected feasible gene is assigned.

The second type of mutation operator, the *employee based random mutation* EBRM is identical to the RM until the employee is randomly chosen. Thereafter, the roster of the selected employee is cleared and the RG (see Section 3.4.3) is applied to obtain the new employee's roster.

3.5 The Second Stage – Inverse Transformation $\mathcal{KS}$

The objective of the first stage is to determine the rough position of the blocks and to assign a shift kind $k \in K$ to each position of the block. The second

stage transforms the roster back, i.e., the shift kinds K are substituted by the required shifts. This inverse transformation is based on the maximal weighted matching algorithm (MWMA) in a bipartite graph G_d where $d \in D$ is the index of the day.

For the given day $d \in D$, let G_d be a bipartite graph with bipartition $\mathbb{V}(G_d) = S(d) \cup E$, where E is a set of employees and $S(d)$ is a set of shifts required for day d, so that $S(d) = \{s \in S \mid RS_{sd} = 1\}$. Furthermore, let $c : \mathbb{E}(G_d) \to \mathbb{R}$ be the weights on the edges. There is an edge $(e, s) \in \mathbb{E}(G_d)$ with $c\big((e, s)\big) = 1$ iff the shift kind assigned in the first stage is equal to the shift kind of s and shift s doesn't violate shift precedences (3.3), i.e., $\big(R^{(\mathcal{K})}_{edk} = 1 \mid \mathcal{M}(s) = k \wedge k \in K_W\big) \wedge \big(SP_{s_{prev},s} = 1 \mid s_{prev} \in S(d-1) \wedge R_{e,d-1,s_{prev}} = 1\big)$ where s_{prev} is a shift assigned to $e \in E$ on the previous day.

Moreover, there are also edges $(e, s) \in \mathbb{E}(G_d)$ with a lower weight $c\big((e, s)\big) = \epsilon(s, s_{prev}) < 1$ iff $\big(\exists k \in K_W \mid R^{(\mathcal{K})}_{edk} = 1\big) \wedge \big(SP_{s_{prev},s} = 1 \mid s_{prev} \in S(d-1) \wedge R_{e,d-1,s_{prev}} = 1\big)$. These edges represent an assignment which is still possible but it is not preferred, i.e., $\mathcal{M}(s)$ is not the kind assigned to this position in the block. Thereafter, the weight $\epsilon(s, s_{prev})$ reflects how shift s fits into the block when s_{prev} is placed on the day before.

The algorithm of the inverse transformation consecutively, for $d = 1$ to $d = |D|$, generates graphs G_d and finds the maximal weighted matching. When this matching contains (e, s), shift $s \in S(d)$ is assigned to employee $e \in E$ on day d, i.e., $R_{eds} = 1$.

The result of the second stage (roster R) is further optimized in the third stage which can be based on common techniques, e.g., a Tabu Search algorithm (Vanden Berghe (2002)) or other heuristic approaches.

3.6 Experiments and Evaluation

This section is organized as follows: Firstly, the experiments with ILP are summarized in Section 3.6.1. Subsequently, the performance of EA is evaluated in Section 3.6.2. The description of the approaches (Vanden Berghe (2002); Burke et al. (2007)) used for comparison to MSA is in Section 3.6.3. All approaches are evaluated by a *crossover evaluation methodology* in Section 3.6.4. Finally, Section 3.6.5 summarizes and discusses the obtained results. All approaches were implemented in C# and all experiments were executed on a PC with Intel Core 2 at 2.4 GHz, 2GB RAM.

3.6.1 The ILP Model

Even though the original problem ETPHD was transformed to ETPHD$^{(\mathcal{K})}$ and several modifications of the ILP model were tuned to solve ETPHD$^{(\mathcal{K})}$, the size

of the transformed model is still huge (discussed in Section 3.3.2). Therefore, the feasible solution of the first stage can be found in a reasonable amount of time for up to $|E| \leq 3$ using the non-commercial ILP solver GLPK and up to $|E| \leq 5$ using CPLEX. Since the number of employees $|E|$ of our instances is about one hundred the first stage cannot be directly solved by an ILP.

3.6.2 The Evolutionary Algorithm

This section is focused on the appraisal of the proposed evolutionary algorithm. The stop condition of EA is the number of populations in the case of all of the following experiments. We tuned some parameters during the experiments and fixed them for the final tests. One of them is the length of the gene l. The best results were reached with a gene length $l = 4$. This value correlates to the constants of the block constraints b_{max}, b_{min} and br_{min}. We fixed the population size $pSize$ equal to 50 as the best compromise between the solution quality and the algorithm performance. Similarly, the initial population size $pSize_0$ was set to $2 \cdot pSize$. In the case of the crossover operator X, we decided to apply the crossover operator EBTSX in order to keep EA as fast as possible without the need of any repair operator. The crossover operator DBOPX is significantly worse from the computation time point of view and slightly better according to the obtained quality. Therefore, the DBOPX crossover operator was not used.

3.6.2.1 Performance of the Evolutionary Algorithm

This section is focused on the first and the second stage of our multistage approach (described in Section 3.4 and 3.5). The results are presented in Table 3.3, each line was executed repeatedly 50 times. The real-data instances were tested with different parameters of EA: the number of populations $\#pop$, type of creation of the initial population GIP and the mutation operator MUT. These parameters were varied to look for the best performance of EA. Furthermore, it is necessary to pick the appropriate metric to evaluate the solution: the value of the objective function for the initial population Z_0^E, the final value of the objective function Z^E, the number of the unassigned shifts for the whole planning horizon $|S^U|$ and the average computational time $\bar{t}_{cpu}$, as well. The most relevant metric out of all of these is the objective function Z^Ein combination with $|S^U|$ (the hard constraints (c_3) and (c_4) are taken into account as soft constraints to minimize the overcoverage and undercoverage of shifts).

In Table 3.3, you can notice that for $\#pop = 10^3$ the best results (in bold) are acquired by the combination of LEWHG and EBRM. For the instances with $\#pop = 10^4$ and $\#pop = 10^5$ this combination is outperformed by LEG and RM. Furthermore, we can observe from the results that the mutation operator

Table 3.3: The evolutionary algorithm performance after the 2nd stage (each line executed 50 times)

EA parameters			Z_0^E	Z^E			$\lvert S^U \rvert$			$t_{cpu}\,[s]$
#pop	GIP	MUT	avg	min	max	avg	min	max	avg	avg
10^3	RG	RM	8485	667	994	841	48	78	61.7	8.3
	RG	EBRM	7801	543	742	643	19	39	29.1	12.4
	WHG	RM	16171	723	1104	861	39	71	59.6	7.9
	WHG	EBRM	16352	567	822	663	19	46	29.2	12.6
	LEG	RM	16598	692	1014	841	25	58	40.1	8.1
	LEG	EBRM	15923	544	749	648	9	50	27.2	12.2
	LEWHG	RM	20018	920	1232	1049	37	59	49.7	8.3
	LEWHG	EBRM	19400	514	765	**637**	14	32	**24.7**	12.2
10^4	RG	RM	8533	111	249	181	13	32	22.1	81.2
	RG	EBRM	8831	273	416	347	8	26	16.1	122.5
	WHG	RM	16708	135	278	193	14	31	22.3	82.2
	WHG	EBRM	15248	245	456	337	4	28	15.1	126.6
	LEG	RM	16509	62	188	**117**	6	22	**12.1**	82.3
	LEG	EBRM	16787	276	446	345	6	23	14.0	123.3
	LEWHG	RM	19872	61	194	140	9	23	14.7	80.2
	LEWHG	EBRM	19514	233	450	331	3	19	12.7	121.9
10^5	WHG	RM	15820	18	67	43	5	15	10.1	795.0
	WHG	EBRM	16645	178	254	212	4	16	9.8	1221.8
	LEG	RM	16174	24	55	**40**	6	13	**8.7**	791.6
	LEG	EBRM	18068	179	290	227	5	16	9.5	1218.9
	LEWHG	RM	20230	27	73	49	5	13	9.5	789.2
	LEWHG	EBRM	19952	174	248	214	6	16	9.4	1208.2

EBRM is about 50 % more time consuming than RM independently to the other parameters. The last remark to this set of tests is that the relation between the $\#pop$ and $\bar{t}_{cpu}$ is approximately linear.

3.6.2.2 Performance of the Evolutionary Algorithm inside the Multistage Approach

This section evaluates EA as a part of the MSA consisting of 3 stages. The last stage is TSA that considers the same constraints (see Section 3.2) with the same weights in the objective function in all experiments. Furthermore, the TSA (Vanden Berghe (2002)) was extended by one special mechanism. The unplaced shifts can be assigned by TSA under a condition that the value of Z does not increase. Naturally, the unassigned shifts can occur in the solutions after the initialization stages. Therefore, the hard constraints (c_3) and (c_4) are modeled in TSA as soft constraints with a higher weight.

The experiments were performed on real data instances of periods with 28 and 35 days. The MSA was executed 30 times (see Table 3.4) on each problem instance in order to achieve the average values for the comparison. The quality of the solutions are reflected by the number of unassigned shifts $|S^U|$ and the value of the objective function Z after the final stage TSA. You can notice that the best performance of MSA is achieved by the combinations {WHG, EBRM @ $\#pop = 10^4$} and {LEG, RM @ $\#pop = 10^5$} for both lengths of periods. Remarkably, the best performance of the EA does not imply the best performance of the MSA with this type of the EA (compare results of Table 3.3 and Table 3.4).

3.6.3 Comparison of the MSA to Other Approaches

This section describes all approaches applied on the ETPHD and the NRP (the overview in Table 3.5). These approaches can be separated into two groups.

The first group consists of MSA, CMPA_1 and CMPA_2. Each approach of this group contains TSA in the last stage. The MSA (described in Section 3.6.2.2) contains the EA with the following parameters: $pSize = 50$, $pSize_0 = 100$, $\#pop = 10^4$, GIP=WHG, X=EBTSX and MUT=EBRM. The execution was bounded by a time limit $t_{max} = 600$ s. Therefore, according to the results mentioned in Table 3.4, the time remaining for the TSA in the last stage of MSA is approximately 200 s. The spent time is used in the last stage of TSA to improve the quality of the solution as much as possible in the case of the approaches consuming no time (CMPA_1) or less time (CMPA_2) than MSA in the first two stages. The CMPA_1 (explained in this section) is similar to our MSA, the only difference is that the first two stages are skipped. The shifts are assigned one by one by the mechanism of the TSA described in Section 3.6.2.2.

Table 3.4: The EA performance as a part of the multistage approach (each line evaluated 30 times)

inst.	EA parameters			Z			$\lvert S^U \rvert$			$t_{cpu}\,[s]$
$\lvert D \rvert$	$\#pop$	GIP	MUT	min	max	avg	min	max	avg	avg
28	10^4	WHG	RM	287	356	313	3	13	6.8	273.6
		WHG	EBRM	284	349	**307**	0	5	**2.8**	336.2
		LEG	RM	305	351	322	0	7	5.4	297.6
		LEG	EBRM	298	359	324	0	8	5.1	328.1
		LEWHG	RM	303	367	329	2	9	5.4	315.7
		LEWHG	EBRM	301	356	325	3	11	5.9	310.0
	10^5	WHG	RM	256	332	291	0	8	5.9	771.9
		WHG	EBRM	264	327	294	2	5	4.1	1136.1
		LEG	RM	213	308	**258**	0	2	**1.3**	675.1
		LEG	EBRM	253	319	281	1	4	2.4	1084.1
		LEWHG	RM	294	338	279	0	5	2.9	760.2
		LEWHG	EBRM	273	349	287	1	6	4.9	974.3
35	10^4	WHG	RM	334	381	357	6	14	10.1	382.2
		WHG	EBRM	332	385	**355**	1	9	**5.8**	406.6
		LEG	RM	338	399	361	3	12	6.9	385.3
		LEG	EBRM	325	389	364	2	9	6.3	423.3
		LEWHG	RM	348	394	364	5	12	7.1	380.2
		LEWHG	EBRM	335	409	357	3	14	7.3	421.9
	10^5	WHG	RM	302	355	318	4	10	7.2	1095.0
		WHG	EBRM	307	353	322	0	6	4.5	1421.8
		LEG	RM	265	322	**295**	0	3	**2.1**	964.6
		LEG	EBRM	299	342	315	0	5	3.2	1458.9
		LEWHG	RM	297	346	312	3	7	4.8	989.2
		LEWHG	EBRM	303	351	325	2	7	4.9	1408.2

Table 3.5: The overview of the compared approaches

approach	stages			problems	
name	1st stage	2nd stage	3rd stage	ETPHD	NRP
MSA	EA	MWMA	TSA	✓	✓
$CMPA_1$	–	–	TSA	✓	✓
$CMPA_2$	SIA (Burke et al. (2006))	–	TSA	✓	–
$CMPA_3$	EA	MWMA	VDSA (Burke et al. (2007))	✓	✓
$CMPA_4$	–	–	VDSA (Burke et al. (2007))	✓	✓
$CMPA_5$	EA	MWMA	TSA + VDSA (Burke et al. (2007))	–	✓

The second comparison approach CMPA_2 (explained in this section too) contains the TSA preceded by the initialization algorithm from (Vanden Berghe (2002)) (pages 139–160). This heuristic is built on the assignment of the shifts to the employees that are separated into groups by their skills. Therefore, this heuristic denoted as SIA – a skill based initialization algorithm. The goal of SIA is to find any solution of the ETPHD with respect to the constraints $(c_1), (c_2), (c_5)$ and (c_6). The SIA is described in full detail in Appendix A.

The second group (CMPA_3, CMPA_4 and CMPA_5) employs in the last stage a Variable Depth Search Algorithm VDSA (Burke et al. (2007)) that is a part of the tool Roster Booster (Burke and Curtois (2012)) (Roster Booster ver. 3.2.6 including VDSA ver. 3.6.). The CMPA_3 is the same as MSA except the last stage, where TSA is replaced by VDSA. The first two stages of CMPA_3 are skipped in CMPA_4. Finally, the last approach CMPA_5 contains TSA and VDSA that are executed in a sequential manner, i.e., the time determined to the third stage is spread between TSA and VDSA. How these approaches are applied and compared is discussed in Section 3.6.4.

3.6.4 Cross Evaluation Methodology

According to the complexity and the originality of the ETPHD, it is not easy to compare MSA to other approaches. Therefore, we propose a *cross evaluation methodology*. This methodology is inspired by the method used in (Burke et al. (2006)). The authors compare two different approaches applied on the same problem, but represented by two different models (more about (Burke et al. (2006)) in Section 3.1.1), e.g., four combinations of approaches and models were compared by a metric relevant to the used model. However, the aim of our methodology is more general, to verify the contribution of the particular stages in the used approaches (namely EA) applied at least on two problems (in our case the ETPHD and the NRP (ASAP (2013))).

The cross evaluation methodology used in this chapter is schematically depicted in Figure 3.4. The approaches are represented by labels at the top of this figure. Each approach consists of the stages affected by the line starting from its label and is applied to the problems that the lines are routed to. Finally, the obtained results are compared. Namely, the approaches colored by bold red (MSA, CMPA_1, CMPA_3 and CMPA_4) participate on the cross evaluation. In addition, the CMPA_2 is applied to the ETPHD and the CMPA_5 is employed on the NRP.

The metrics for the ETPHD are Z and $|S^U|$ as in Section 3.6.2. Furthermore, the third metric C^I stands for the number of the isolated days-on and days-off, i.e., the violations of the constraint (c_{11}). The metrics Z and $|S^U|$ are only taken into account in NRP, since the third metric C^I depends on the constraints of the given NRP instance. In order to compare the results by one

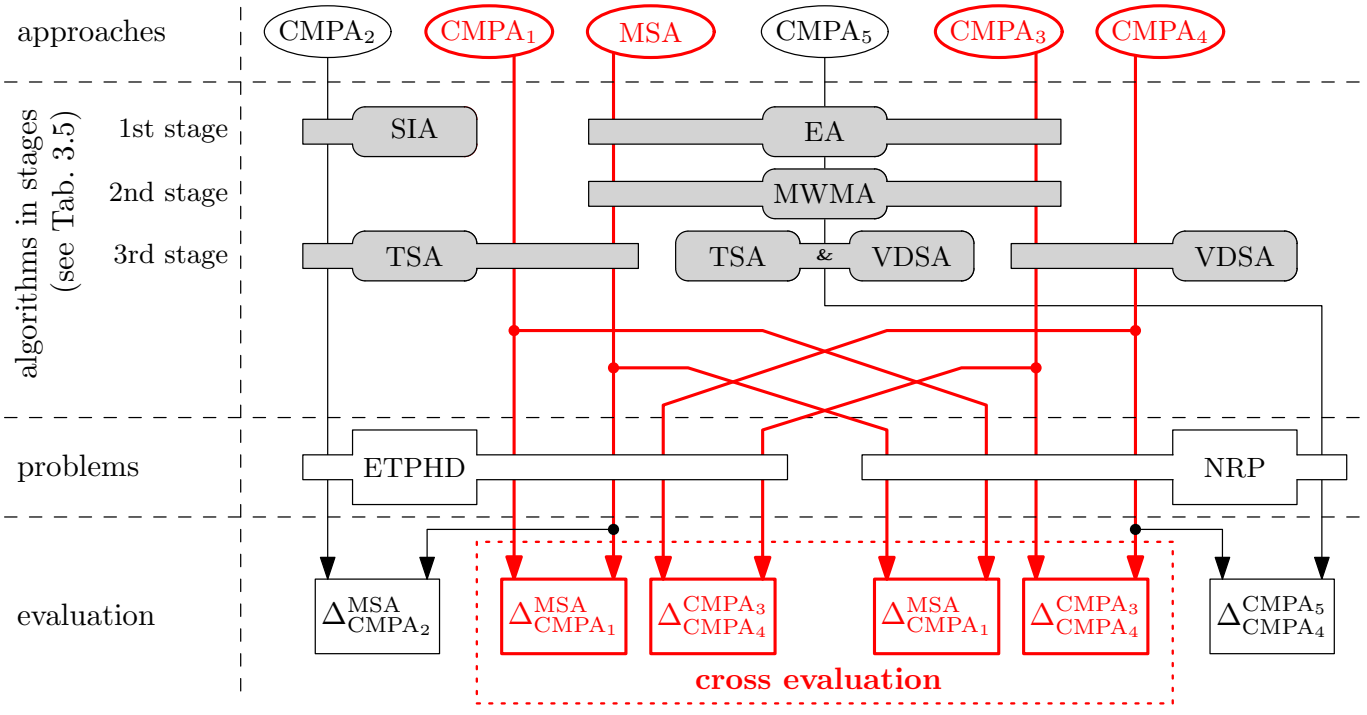


Figure 3.4: The cross evaluation methodology used to verify the contribution of EA

of these metrics, a relative difference is defined as follows. Let A_{ref} be the reference approach used to evaluate the approach A_{ev} by the metric Z, i.e., the approach A_{ref} is evaluated by $\mathrm{A}_{ref}(Z)$ and the approach A_{ev} by $\mathrm{A}_{ev}(Z)$. Then, the *relative difference* $\Delta^{\mathrm{A}_{ev}}_{\mathrm{A}_{ref}}(Z)$ is given by

$$\Delta^{\mathrm{A}_{ev}}_{\mathrm{A}_{ref}}(Z) = \begin{cases} \big(\mathrm{A}_{ref}(Z) - \mathrm{A}_{ev}(Z)\big)/\mathrm{A}_{max}(Z) \cdot 100, & \mathrm{A}_{max}(Z) \neq 0 \\ 0, & \text{otherwise} \end{cases} \tag{3.19}$$

where $\mathrm{A}_{max}(Z) = \max\{\mathrm{A}_{ref}(Z), \mathrm{A}_{ev}(Z)\}$. Consequently, $\Delta^{\mathrm{A}_{ev}}_{\mathrm{A}_{ref}}(Z) \in \langle -100, 100\rangle$ [%]. The relative differences $\Delta^{\mathrm{A}_{ev}}_{\mathrm{A}_{ref}}(|S^U|)$ and $\Delta^{\mathrm{A}_{ev}}_{\mathrm{A}_{ref}}(C^I)$ are defined similarly as $\Delta^{\mathrm{A}_{ev}}_{\mathrm{A}_{ref}}(Z)$.

3.6.4.1 Comparison of the Approaches Evaluated on the ETPHD Problem

This section demonstrates the performance of the approaches applied on the ETPHD (see Table 3.5). All approaches were tested on the set of the instances presented in Table 3.6, where the basic sizes of each instance are mentioned – the number of days $|D|$, the number of employees $|E|$, the number of shifts $|S|$ and the ratio of the fixed shifts F_R. This ratio is defined as the number of days containing fixed shifts to the number of days in the roster (i.e., the number of employees times the length of the planning horizon). The last column

Table 3.6: ETPHD instances description

inst.	$\lvert D\rvert$	$\lvert E\rvert$	$\lvert S\rvert$	RBC	inst.	$\lvert D\rvert$	$\lvert E\rvert$	$\lvert S\rvert$	RBC
p01	94	28	118	10^{7846}	p16	91	28	96	10^{7593}
p02	95	35	79	10^{8299}	p17	88	28	96	10^{7343}
p03	89	28	102	10^{7439}	p18	93	35	124	10^{9556}
p04	90	35	124	10^{9247}	p19	95	28	118	10^{7933}
p05	92	28	96	10^{7677}	p20	92	28	102	10^{7693}
p06	93	35	124	10^{9556}	p21	96	28	118	10^{8020}
p07	91	28	102	10^{7609}	p21	97	35	79	10^{8486}
p08	92	35	79	10^{8020}	p22	89	28	96	10^{7427}
p09	90	28	102	10^{7524}	p23	97	28	118	10^{8107}
p10	87	28	102	10^{7270}	p24	93	28	96	10^{7760}
p11	94	35	79	10^{8206}	p25	92	35	124	10^{9453}
p12	88	35	124	10^{9042}	p26	93	35	79	10^{8113}
p13	96	35	79	10^{8393}	p27	91	35	124	10^{9350}
p14	89	35	124	10^{9145}	p29	90	28	96	10^{7510}
p15	92	35	124	10^{9453}	p30	88	28	102	10^{7355}

RBC stands for the complexity of the instance computed by the tool Roster Booster (Burke and Curtois (2012)).

In general, MSA outperformed CMPA_1 and CMPA_2 on all instances on the level of Z in Table 3.7. On the other hand, CMPA_2 is 'a little bit more successful' from the $|S^U|$ point of view. It is necessary to say that the testing instances were chosen in order to be on the edge of the solvability, i.e., it is very hard to minimize $|S^U|$, C^I and Z as much as possible simultaneously. However, all achieved values of $|S^U|$ are relatively low, if we consider that the number of all of the shifts that have to be assigned is $\sum_s \sum_d RS_{sd}$, i.e., approximately 1300 (1600) shifts within the period with $|D| = 28\,(35)$ days. Furthermore, the solutions where $|S^U| > 0$ are still very valuable for personnel schedulers in transport services, since they are able to assign this small amount of the shifts in a manual way to the employees who usually have business trips that are more or less floating. The CMPA_2 was defeated by MSA on the level of C^I, where MSA acquired approximately 40 % better results.

Remarkably, almost all metrics presented in Table 3.8 are better for CMPA_3, where our EA with MWMA is used. Nevertheless, you can notice a rapid increase of Z in Table 3.8 in comparison to Z in Table 3.7. This is caused by the problem of modeling the ETPHD instances in the Roster Booster. All instances were modeled as precisely as possible, but the Roster Booster does not guarantee the satisfaction of the hard constraints, which are modeled by soft constraints with higher weights. Therefore, the results of the MSA, CMPA_1 and CMPA_2 are not directly comparable to CMPA_3, CMPA_4 and CMPA_5. Nevertheless, the cross evaluation does not directly compare the approaches in this way, its goal lies in the evaluation of the contribution of the particular stages.

3.6.4.2 Comparison of the Approaches Evaluated on the NRP Problem

This section evaluates the relevant approaches (see Table 3.5) on the set of benchmark instances of NRP (ASAP (2013)). Namely, the smaller (Millar, Gpost) and the bigger instances (SINTEF, Valouxis, WHPP) were chosen, as you can notice in Table 3.9. The metric Z was used to evaluate these instances. The metric $|S^U|$ is not mentioned in Table 3.9 since all shifts were always assigned, i.e., $|S^U| = 0$ for each instance executed by all relevant approaches. The value of Z^* represents the objective function of the best known solution so far without any time limit. Our time limit t_{max} was set to 15 s. The parameters of EA were changed with respect to t_{max}, e.g., $\#pop$ was decreased.

In this case, MSA totally outperformed CMPA_1. Nevertheless, a different situation occurs in the case of CMPA_3 and CMPA_4. The negative relative difference $\Delta^{\text{CMPA}_3}_{\text{CMPA}_4}$ appears for the Gpost and Valouxis instances. This is caused

Table 3.7: The cross evaluation on ETPHD – TSA in the last stage
(average values, each line evaluated 30 times, time limit 600 s)

instance	MSA (Z)	MSA $(\lvert S^U \rvert)$	MSA (C^l)	$\text{CMPA}_1(Z)$	$\text{CMPA}_1(\lvert S^U \rvert)$	$\text{CMPA}_1(C^l)$	$\text{CMPA}_2(Z)$	$\text{CMPA}_2(\lvert S^U \rvert)$	$\text{CMPA}_2(C^l)$	$\Delta^{\text{MSA}}_{\text{CMPA}_1}(Z)$ [%]	$\Delta^{\text{MSA}}_{\text{CMPA}_1}(\lvert S^U \rvert)$ [%]	$\Delta^{\text{MSA}}_{\text{CMPA}_1}(C^l)$ [%]	$\Delta^{\text{MSA}}_{\text{CMPA}_2}(Z)$ [%]	$\Delta^{\text{MSA}}_{\text{CMPA}_2}(\lvert S^U \rvert)$ [%]	$\Delta^{\text{MSA}}_{\text{CMPA}_2}(C^l)$ [%]
p01	613.4	6.2	38.9	721.7	50	44	699.9	5	72	15.0	87.7	11.5	12.4	-18.8	45.9
p02	634.1	7.0	49.6	772.9	58	57	934.1	9	95	18.0	87.9	13.0	32.1	21.9	47.8
p03	305.5	1.6	41.2	381.8	23	46	398.5	0	45	20.0	93.2	10.4	23.3	-100.0	8.4
p04	974.3	14.5	40.4	1035.9	49	54	1255.3	5	126	5.9	70.4	25.2	22.4	-65.5	67.9
p05	555.4	2.8	42.6	635.4	41	59	677.8	1	84	12.6	93.3	27.9	18.1	-63.8	49.3
p06	1013.2	2.1	50.4	1165.2	44	86	1122.9	0	75	13.0	95.3	41.4	9.8	-100.0	32.8
p07	355.5	0.0	35.2	433.0	18	32	454.1	0	66	17.9	100.0	-9.1	21.7	0.0	46.6
p08	617.9	19.9	55.4	685.6	76	64	1060.5	25	104	9.9	73.8	13.4	41.7	20.2	46.7
p09	324.4	0.4	35.9	395.1	17	34	460.9	0	56	17.9	97.4	-5.2	29.6	-100.0	36.0
p10	274.0	2.2	38.0	345.9	37	34	443.3	0	56	20.8	93.9	-10.5	38.2	-100.0	32.1
p11	644.8	10.3	72.8	696.4	55	69	902.8	11	98	7.4	81.2	-5.2	28.6	6.1	25.8
p12	926.5	26.0	50.2	1014.4	73	62	1179.8	29	192	8.7	64.4	19.0	21.5	10.4	73.8
p13	657.1	1.6	59.8	767.1	45	71	942.3	5	91	14.3	96.5	15.8	30.3	68.4	34.3
p14	976.7	21.0	48.2	1021.7	64	49	1136.3	13	102	4.4	67.2	1.6	14.0	-38.1	52.7
p15	1007.3	1.7	58.6	1093.0	42	71	1107.7	0	78	7.8	95.9	17.4	9.1	-100.0	24.9
p16	729.8	15.3	44.6	781.0	51	42	896.8	5	104	6.6	70.0	-5.7	18.6	-67.3	57.2
p17	560.2	24.7	59.7	612.3	73	67	734.6	18	106	8.5	66.2	11.0	23.7	-27.1	43.7
p18	1013.0	2.5	55.4	1164.7	44	86	1123.0	0	75	13.0	94.4	35.6	9.8	-100.0	26.2
p19	572.9	5.7	42.2	712.9	39	51	723.2	5	54	19.6	85.3	17.3	20.8	-12.6	21.9
p20	380.5	0.0	35.4	469.7	18	32	510.6	0	69	19.0	100.0	-9.5	25.5	0.0	48.7
p21	624.1	4.4	30.0	726.0	34	36	736.8	5	59	14.0	87.2	16.7	15.3	12.8	49.2
p22	697.8	1.9	59.2	790.9	38	75	984.8	5	98	11.8	94.9	21.1	29.1	61.3	39.6
p23	520.0	19.7	51.0	563.8	53	57	747.9	10	113	7.8	62.8	10.5	30.5	-49.3	54.9
p24	669.7	4.2	35.9	761.2	33	45	737.7	5	67	12.0	87.4	20.1	9.2	16.8	46.3
p25	578.0	2.8	54.7	624.6	29	47	669.6	0	85	7.5	90.5	-14.1	13.7	-100.0	35.6
p26	989.6	4.4	47.1	1093.1	42	71	1107.2	0	78	9.5	89.4	33.7	10.6	-100.0	39.6
p27	647.9	14.7	62.0	694.8	59	83	1027.9	16	101	6.8	75.1	25.2	37.0	8.0	38.6
p28	1002.9	6.6	61.6	1068.3	51	65	1125.5	1	87	6.1	87.1	5.3	10.9	-84.8	29.2
p29	521.0	10.2	44.9	585.0	49	57	663.1	4	91	10.9	79.2	21.2	21.4	-60.7	50.7
p30	314.8	4.2	39.9	395.6	42	45	421.4	0	49	20.4	89.9	11.4	25.3	-100.0	18.6

Table 3.8: The cross evaluation on ETPHD – VDSA in the last stage
(average values, each line evaluated 30 times, time limit 600 s)

instance	$\mathrm{CMPA}_3(Z)$	$\mathrm{CMPA}_3(\lvert S^U\rvert)$	$\mathrm{CMPA}_3(C^1)$	$\mathrm{CMPA}_4(Z)$	$\mathrm{CMPA}_4(\lvert S^U\rvert)$	$\mathrm{CMPA}_4(C^1)$	$\Delta^{\mathrm{CMPA}_3}_{\mathrm{CMPA}_4}(Z)$ [%]	$\Delta^{\mathrm{CMPA}_3}_{\mathrm{CMPA}_4}(\lvert S^U\rvert)$ [%]	$\Delta^{\mathrm{CMPA}_3}_{\mathrm{CMPA}_4}(C^1)$ [%]
p01	2487.3	4.9	118.8	2621.6	3.9	142.0	5.1	-20.0	16.3
p02	3398.9	33.5	121.4	3538.4	34.0	158.5	3.9	1.5	23.4
p03	2440.9	6.4	82.5	2723.0	10.8	96.5	10.4	40.7	14.5
p04	3980.4	14.7	138.4	4142.9	14.3	115.0	3.9	-2.5	-16.9
p05	2486.9	6.3	122.5	2683.1	5.4	130.3	7.3	-13.7	6.0
p06	3930.1	9.6	144.2	4247.9	11.0	182.4	7.5	12.9	20.9
p07	2471.2	5.9	109.4	2553.7	5.4	125.1	3.2	-7.7	12.6
p08	3476.1	31.0	89.0	3555.1	32.0	117.4	2.2	3.3	24.2
p09	2408.4	5.2	120.1	2666.3	12.0	134.7	9.7	56.7	10.8
p10	2403.9	8.3	93.6	2932.8	13.3	90.6	18.0	37.6	-3.2
p11	3622.4	32.6	130.3	3729.8	40.8	127.1	2.9	20.1	-2.5
p12	4281.5	15.3	79.9	4845.1	20.7	128.8	11.6	26.0	38.0
p13	3486.1	30.3	134.3	3490.1	34.3	135.8	0.1	11.7	1.1
p14	4018.2	12.2	103.4	4237.1	13.0	109.5	5.2	6.2	5.6
p15	3964.8	8.8	192.2	4045.8	8.6	164.9	2.0	-1.9	-14.2
p16	3004.6	9.2	113.2	3102.6	9.5	110.3	3.2	3.3	-2.6
p17	2998.8	6.4	80.7	3035.0	19.3	98.2	1.2	66.8	17.8
p18	4191.0	7.4	136.9	4203.1	11.4	182.9	0.3	35.1	25.2
p19	2397.4	5.2	81.7	2453.0	4.4	87.4	2.3	-15.7	6.6
p20	2454.1	5.9	149.6	2723.7	6.1	131.4	9.9	2.8	-12.2
p21	2551.1	2.3	146.4	2507.1	2.3	160.9	-1.7	-1.0	9.0
p22	3398.4	30.1	136.7	3445.6	31.3	164.6	1.4	3.8	17.0
p23	2868.5	7.3	83.9	3212.6	9.8	81.3	10.7	25.2	-3.1
p24	2320.7	3.1	131.1	2579.8	3.4	163.3	10.0	8.8	19.7
p25	3029.5	4.6	95.5	3154.3	4.1	116.3	4.0	-11.0	17.9
p26	3780.0	3.4	151.5	4101.9	8.0	165.3	7.8	57.8	8.3
p27	3803.3	33.6	113.5	3969.7	39.0	124.5	4.2	13.9	8.8
p28	3906.7	15.6	128.0	4080.7	13.1	141.0	4.3	-16.1	9.2
p29	2692.8	6.4	95.5	2870.5	12.9	116.6	6.2	50.4	18.1
p30	2290.5	7.4	110.5	2612.8	7.5	111.7	12.3	1.4	1.1

by the fact, that EA is not able to not satisfy all constraints of these instances, but consumes an important part of t_{max}. Thus, less time is available for the VDSA and the roster is not improved very much. In the case of Gpost, the constraint limiting consecutive weekends is not considered by EA. The Valouxis instance was hard to solve for EA with respect to satisfying the 2- and 3-length stretches of the night shifts. On the contrary, a comparable quality was reached on the Millar and WHPP instances and CMPA_3 exceeded on the SINTEF instance. For the complex evaluation, the CMPA_5 was compared to CMPA_4, because the constraints of the problematic instances Gpost and Valouxis can be modeled much better in the TSA than in EA itself. You can notice that CMPA_5 reached the best results overall.

Table 3.9: The cross evaluation on the NRP
(average values, each line evaluated 30 times, time limit 15 s)

instance	RBC	Z^*	$\text{MSA}(Z)$	$\text{CMPA}_1(Z)$	$\text{CMPA}_3(Z)$	$\text{CMPA}_4(Z)$	$\text{CMPA}_5(Z)$	$\Delta^{\text{MSA}}_{\text{CMPA}_1}(Z)$ [%]	$\Delta^{\text{CMPA}_3}_{\text{CMPA}_4}(Z)$ [%]	$\Delta^{\text{CMPA}_5}_{\text{CMPA}_4}(Z)$ [%]
SINTEF	10^{392}	0	8.2	1560	3.5	13.0	0.0	99.5	73.1	100.0
Gpost	10^{107}	5	6660.6	13430	269.3	96.5	14.6	50.4	-64.2	84.9
Millar	10^{53}	0	0.0	100	0.0	0.0	0.0	100.0	0.0	0.0
Valouxis	10^{270}	20	739.7	4800	259.7	160.0	79.1	84.6	-38.4	50.6
WHPP	10^{253}	5	4021.3	10023	2021.1	2014.0	1016.7	59.9	-0.4	49.5

3.6.5 Summary of Experiments

This section summarizes all results of both tackled problems, ETPHD and NRP. Average values and standard deviations of the Δ results from Section 3.6.4 are summed up in Table 3.10. The cross evaluation is reflected in six columns containing $\Delta^{\text{MSA}}_{\text{CMPA}_1}$ and $\Delta^{\text{CMPA}_3}_{\text{CMPA}_4}$. The positive differences are emphasized by bold. Two negative values appear and the reasons of their occurrence are mentioned in Section 3.6.4.1 and Section 3.6.4.2. In summary, the cross evaluation confirms that the application of our EA leads significantly to better or equal results in 9 out of 10 cases. Additionally, the performance of our multistage approach MSA was shown by $\Delta^{\text{MSA}}_{\text{CMPA}_2}$ and $\Delta^{\text{CMPA}_5}_{\text{CMPA}_4}$ (MSA is included in CMPA_5). In this case, the MSA was successful in 3 out of 4 cases.

Table 3.10: The summary of the cross evaluation and the comparison to other approaches

		cross evaluation						comparison to other approaches			
problem		$\Delta^{\text{MSA}}_{\text{CMPA}_1}(Z)$ [%]	$\Delta^{\text{MSA}}_{\text{CMPA}_1}(\lvert S^U \rvert)$ [%]	$\Delta^{\text{MSA}}_{\text{CMPA}_1}(C^I)$ [%]	$\Delta^{\text{CMPA}_3}_{\text{CMPA}_4}(Z)$ [%]	$\Delta^{\text{CMPA}_3}_{\text{CMPA}_4}(\lvert S^U \rvert)$ [%]	$\Delta^{\text{CMPA}_3}_{\text{CMPA}_4}(C^I)$ [%]	$\Delta^{\text{MSA}}_{\text{CMPA}_2}(Z)$ [%]	$\Delta^{\text{MSA}}_{\text{CMPA}_2}(\lvert S^U \rvert)$ [%]	$\Delta^{\text{MSA}}_{\text{CMPA}_2}(C^I)$ [%]	$\Delta^{\text{CMPA}_5}_{\text{CMPA}_4}(Z)$ [%]
ETPHD	avg.	**12.2**	**85.1**	**12.3**	**5.7**	**19.0**	**9.6**	**21.8**	-41.7	**40.8**	–
	st.dev.	5.0	11.6	14.3	4.4	21.7	12.5	9.2	54.7	14.3	–
NRP	avg.	**78.9**	**0.0**	–	-7.5	**0.0**	–	–	–	–	**71.2**
	st.dev.	22.8	0.0	–	51.9	0.0	–	–	–	–	38.6

3.7 Conclusion

In this chapter, we introduced a three stage heuristic algorithm for the employee timetabling domain. The solved problem, motivated by a real employee rostering problem at the airport, is characteristic by the personnel demand typically having two peaks per a day. In order to satisfy the coverage requirements and to minimize the personnel expenses it is necessary to cover the requirements by the 'enlarged' set of shifts. In our case, the set of shifts is fixed and it contains more than one hundred shifts. This fact together with the complex set of the roster constraints, given by the collective agreement, makes the employee rostering very difficult. Therefore, the problem solution was decomposed into three stages, where the first one determines a rough position of the shift kinds in the roster (i.e., early shifts, late shifts, etc.), the second stage assigns shifts into the roster and the last stage fine-tunes the final roster.

According to the complexity and the originality of ETPHD, it is not easy to compare MSA to other approaches. Therefore, a cross evaluation methodology was proposed. Moreover, one is able to verify the contribution of the particular stages, namely EA. The experiments reflecting this methodology confirmed that approaches using EA provide better or equal solutions in 12 out of 14 cases (summarized in Table 3.10). These experiments were performed on ETPHD and NRP problems of the Roster Booster complexity (Burke and Curtois (2012)) from 10^{50} to 10^{10000} for the complex and fair evaluation.

Chapter 4

GPU based Parallel Algorithm for the Nurse Rerostering Problem

4.1 Introduction

This chapter is focused on an NP-hard combinatorial problem that occurs in healthcare (Clark et al. (2012)). The shifts have to be assigned to the nurses in a given planning horizon to create a *roster*. However, the roster usually has to be changed during the planning horizon, e.g., when one of the nurses gets sick. Then the original roster has to be modified in order to ensure sufficient healthcare service. Typically, the roster is not completely rebuilt, since the nurses affected by the arising changes have to cancel or reschedule their already planned free time activities, which is very unpopular. Moreover, it may also significantly increase the personnel costs, since the overtime hours of employees have to be rewarded. Therefore, the criterion of this problem typically involves a number of changes in comparison to the original roster. Minimization of this objective with respect to all considered constraints may lead to long computational times that are unacceptable in these stressful rescheduling processes. The problem is known as the *Nurse Rerostering Problem* (NRRP) and the aim of this chapter is to propose a parallel solution solving the NRRP which is faster and provides the same quality as conventional sequential approaches.

4.1.1 Related Works

The existing works related to this topic can be split into two categories. The first one, summarized in the first subsection, contains the literature related to the NRRP. The second one (see Section 4.1.1.2) refers to the *operation*

research (OR) domain in the context of the Graphic Processing Unit (GPU) computing.

4.1.1.1 The NRRP Literature Overview

The rerostering problem belongs to the human resources/personnel scheduling domain, which has been summarized in several surveys (Clark et al. (2012); Ernst et al. (2004); Burke et al. (2004b); Van den Bergh et al. (2013)). Although NRRP occurs in hospitals very often, the number of papers addressing this problem is minor in comparison to the Nurse Rostering Problem (NRP), e.g., the survey (Clark et al. (2012)) from the year 2012 presents 8 papers dealing with NRRP only. Most of these papers were published by Moz and Pato (see (Moz and Pato (2003, 2004, 2007); Pato and Moz (2008))). The first two (Moz and Pato (2003)), (Moz and Pato (2004)) are based on the multi-commodity network flows models that are formulated as an Integer Linear Programming (ILP) problem. Naturally, this approach has the disadvantage common to exact methods, i.e., the time needed to obtain a solution grows considerably for larger instances. In order to eliminate this drawback Moz and Pato introduced a heuristic (Moz and Pato (2007)) minimizing the number of changes in the original roster. This heuristic is based on a construction of the roster by the iterative assignments of the shifts in a given order. In order to improve the quality of the solutions, this constructive heuristic was encapsulated by a genetic algorithm that was applied to shuffle the order of the shifts to be assigned. This extension improved 10 % of the solutions in terms of the quality, however it is outweighed by the significant increase of the computational time. This paper was followed by (Pato and Moz (2008)) where a bi-objective rerostering problem was solved by the Pareto genetic heuristic. In addition to the number of changes, the deviation from the number of shifts originally assigned to a given nurse is considered as the second objective. The quality of the solutions was investigated mainly in (Pato and Moz (2008)). In comparison to (Moz and Pato (2007)), the computational times were longer and unfortunately, only 31 out of 68 NRRP instances were tested.

Kitada et al. propose in (Kitada et al. (2011)) methods to find an optimal schedule of the NRRP with a minimum number of changes. These methods are based on a recursive search algorithm to generate feasible solutions of the NRRP, however, they are proposed and evaluated on the instances with a single-day absence only. This drawback is partially eliminated in (Kitada and Morizawa (2013)) which describes a method for solving the NRRP with an absence of nurses for several consecutive days. Firstly, the consecutive days of absences are separated into a set of single-day absences, which generates a set of single-day NRRPs. Secondly, these single-day NRRPs are resolved one by one using the method described in (Kitada et al. (2011)).

Nevertheless, this decomposition does not take into account all absent days at once and in general, this may lead to a suboptimal solution of the problem.

The previous works were addressed to NRRP within one department of the nurses, i.e., hiring nurses from other departments is not possible. On the contrary, the different absence scenarios were considered in (Lilleby et al. (2012)) to show how successfully one deals with NRRP using more departments. However, the main contribution of (Lilleby et al. (2012)) is the stochastic model in order to improve nurse utilization, decrease the personnel cost and, at last, but not least, build the robust competence of the nurses to make them substitutable across departments.

Other papers tackling NRRP were published by Maenhout and Vanhoucke. The latest one (Maenhout and Vanhoucke (2013a)) describes an evolutionary algorithm inspired by the theory of immunology called an Artificial Immune System Algorithm (AISA, see (De Castro and Timmis (2002))). AISA provides results having approximately the same quality as an Evolutionary Algorithm (EA) from their previous paper (Maenhout and Vanhoucke (2010b)) (overall, AISA outperforms EA by 0.04 %). However, the experiments in (Maenhout and Vanhoucke (2013a)) verified that some NRRP instances are better to be solved by AISA instead of EA from (Maenhout and Vanhoucke (2010b)) and vice versa. The previous paper (Maenhout and Vanhoucke (2010b)) combines EA with local search methods using network flows. The solution of the NRRP is represented by the best individual, which is organized in the nurse-day view (Cheang et al. (2003)). The local search methods are applied after the recombination operators (a crossover and a mutation) to improve the quality of the individuals. The different strategies of the algorithm were tested on 1000 generated NRRP instances. The average computational times were from 13 to 268 seconds per the NRRP instance according to the strategy used. The results achieved by EA in (Maenhout and Vanhoucke (2010b)) outperformed the results of Maenhout's and Vanhoucke's implementation of the algorithm presented by (Moz and Pato (2007)). Both algorithms were applied on the dataset from their paper (Maenhout and Vanhoucke (2010b)). However, from our point of view, the problems in (Maenhout and Vanhoucke (2010b)) and (Moz and Pato (2007)) cannot be compared to each other in a straightforward manner since there are obvious dissimilarities in the problem statement (see Sec. 2 in (Moz and Pato (2007)) and Sec. 3 in (Maenhout and Vanhoucke (2010b))), e.g., the different definitions of the disruption in the original roster. A disruption in (Moz and Pato (2007)) is defined such that a nurse *cannot* be assigned to any shift on the same day, except a day-off. On the contrary, the disruption in (Maenhout and Vanhoucke (2010b)) is defined such that a nurse *can* be assigned to any shift except the absented one on the same day, e.g., a nurse is absent for the early shift, however he/she can be assigned to the late or the night shift on the same day. Moreover, the different hard constraints and objec-

tives are considered in these two papers (e.g., the objective in (Moz and Pato (2007)) is the minimal number of changes only, versus three objectives in (Maenhout and Vanhoucke (2010b)) – the minimal number of changes, the effort to meet the preferences of the nurses and the balance of the workload among the nurses). Furthermore, the results of (Maenhout and Vanhoucke (2010b)) are presented in a condensed form only and, therefore, one cannot compare the results of the particular instances and their execution times.

4.1.1.2 The GPU Computing Literature Overview

In the last decade, there has been a growing interest to utilize GPUs for non-graphic applications, which is confirmed by the surveys (Owens et al. (2007); Brodtkorb et al. (2013a,b); Schulz et al. (2013)). A few operations research problems have already been solved on GPU. The most important metric of the GPU algorithm performance is its *speedup* defined as a ratio of the *computational time* needed for the sequential and parallel version of the algorithm.

The authors of (Janiak et al. (2008)) solved two combinatorial problems using the Tabu Search Algorithm (TSA). Firstly, TSA was applied on the Traveling Salesman Problem (TSP), however, only the solutions of the instances having more than 50 cities ensure a *speedup*. Smaller instances are better solved on a CPU, since there is a large overhead of data preparation for a GPU. Unfortunately, the maximal speedup, achieved on the instances with 100 cities, was only 12 %. Secondly, the Permutation Flowshop Scheduling Problem (PFSP) was handled by the TSA, where solutions were found on a GPU approximately 4 times faster than on a CPU.

The same problem solved again by the TSA is presented in (Czapinski and Barnes (2011)). In this case, the GPU significantly outperforms the CPU (speedup up to 89 times). However, the solutions found were not evaluated in terms of their quality. The same authors also applied the GPU based TSA on the Quadratic Assignment Problem in (Czapinski (2013)), where each instance of the TSA has slightly different parameters to ensure the diversity of the solutions. The overhead caused by the communication between the CPU and the GPU is eliminated strongly, since the entire algorithm is launched on the GPU and this leads to a speedup of up to 70 times. In comparison to (Czapinski and Barnes (2011)), the quality performance was evaluated with respect to the best known solutions and the parallel algorithm in (Czapinski (2013)) was capable of providing very good quality solutions (often optimal or the best known).

Another paper dealing with the OR problem on the GPU (see (Boyer et al. (2012))) describes the dynamic programming applied on the Knapsack Problem (KP). The achieved speedup for the KP instances considering from 40 to 100 thousand objects was about 26, while the smaller instances reveal a speedup of

about 20. The Resource Constrained Project Scheduling Problem (RCPSP) is solved on a GPU in (Bukata and Šůcha (2013)). The experiments show that the GPU outperforms the CPU version in both performances – the speedup and the quality of solutions. That is possible thanks to an effective schedule evaluation and a GPU-optimized TSA. In addition, the required data transfers are reduced to a minimum since the entire algorithm runs on a GPU. The quality of the solutions is comparable with existing algorithms solving the RCPSP.

In general, all papers mentioned above except (Bukata and Šůcha (2013)) have one common feature. The combinatorial problems solved on the GPU have either plain data representation or a very simple evaluation of the criterion. However, NRRP does not fulfill these assumptions. Moreover, the results of these works are evaluated by the self-comparison very often. To prove the applicability of GPU computing for OR, more work needs to be done, i.e., the achieved results from the related works have to be considered.

4.1.2 Contribution and Outline

To the best of our knowledge there is no paper focused on a NRRP solved on a GPU, moreover, there is no paper dealing with a parallel approach applied on an NRRP at all. In order to design the first parallel algorithm solving an NRRP, which preserves the quality of the solution of the sequential approach and reduces the computational time needed to obtain this solution, the current papers mentioned above were taken into account. As it will be shown, the algorithm from (Moz and Pato (2007)) is highly appropriate to be an inspiration for the design of a new parallel algorithm. Furthermore, the benchmark instances from this paper were published in (Pato and Moz (2013)), so we are able to compare the quality of the sequential and the parallel version in a fair way.

In general, a design of a parallel algorithm for an NRRP is a non-trivial task due to the GPU's features. Moreover, the design of our algorithm had to be adjusted to different sizes of the NRRP instances and to be robust with respect to the number of disruptions in the original roster. Therefore, we show a unique problem decomposition allowing an efficient parallelization. Our work also contains a comparison of two models of the parallel algorithm – when the entire algorithm runs on a GPU (a *homogeneous model*) and when the algorithm is partially solved on a CPU and partially on a GPU (a *heterogeneous model*). Overall, the homogeneous model provides the speedup 12.7 (17.7) for the NRRP datasets with 19 (32) nurses in comparison to the same algorithm performed sequentially. This is significantly higher than the speedup of the heterogeneous model – 2.3 (2.4) for the same datasets. Our results were compared to the best known solutions of the benchmark instances presented

in the related works and our parallel algorithms provide the same quality of the solutions to most of the benchmark instances within a significantly shorter computational time.

The chapter is organized as follows: The principles of the GPU computing are outlined in Section 4.2. The NRRP is formally defined in Section 4.3. The sequential algorithm is presented in Section 4.4 while the design of the homogeneous and heterogeneous model of the parallel algorithms are discussed in Section 4.5 and 4.6. Subsequently, the performance of both models is verified with respect to the related works in Section 4.7 on the NRRP benchmark datasets with 19 and 32 nurses. The work is concluded in Section 4.8 and the nomenclature used in this chapter is summarized in the beginning of this thesis.

4.2 Computing on a Graphics Processing Unit

Computing on a GPU has become more and more powerful due to the fast evolving hardware devices over the last decade (see (Brodtkorb et al. (2013a))). However, compared to the CPU, the GPU has several specific features highlighted in this section. *Compute Unified Device Architecture* (CUDA) is a parallel computing platform and programming model created by NVIDIA® corporation supported by their GPUs (see a detailed description in (NVIDIA Corporation (2013))). CUDA is based on a *Single Instruction Multiple Threads* (SIMT) parallelization, which means that the threads are initiated on a GPU to process different data by the same code called a *kernel*. The threads are grouped into CUDA *blocks*. Furthermore, these blocks are organized into a *grid* of the CUDA blocks executing the same kernel. From the hardware point of view (see Figure 4.1), the blocks of threads are launched on *streaming multiprocessors* (SM). More CUDA blocks can be executed on one SM simultaneously if the capacity of the resources, e.g., the memory size, is sufficient. The kernels are executed on each SM in batches of 32 threads called *warps*. The order of the executed warps from all CUDA blocks in the grid cannot be affected by a programmer and it is fully handled by a built-in device scheduler.

The design of the GPU algorithms has to also respect that the warps within a CUDA block can be executed asynchronously, although the warps can be synchronized by an instruction `__syncthreads()` at the place of this instruction. This mechanism is called a barrier synchronization. Nevertheless, the excessive usage of `__syncthreads()` leads to a decrease in the performance. It is necessary to take into account together with the SIMT thread mapping, which has also a major relationship to the rest of the limitations of the GPU architecture, e.g., a *branch divergence* of the code. It occurs when threads in a warp

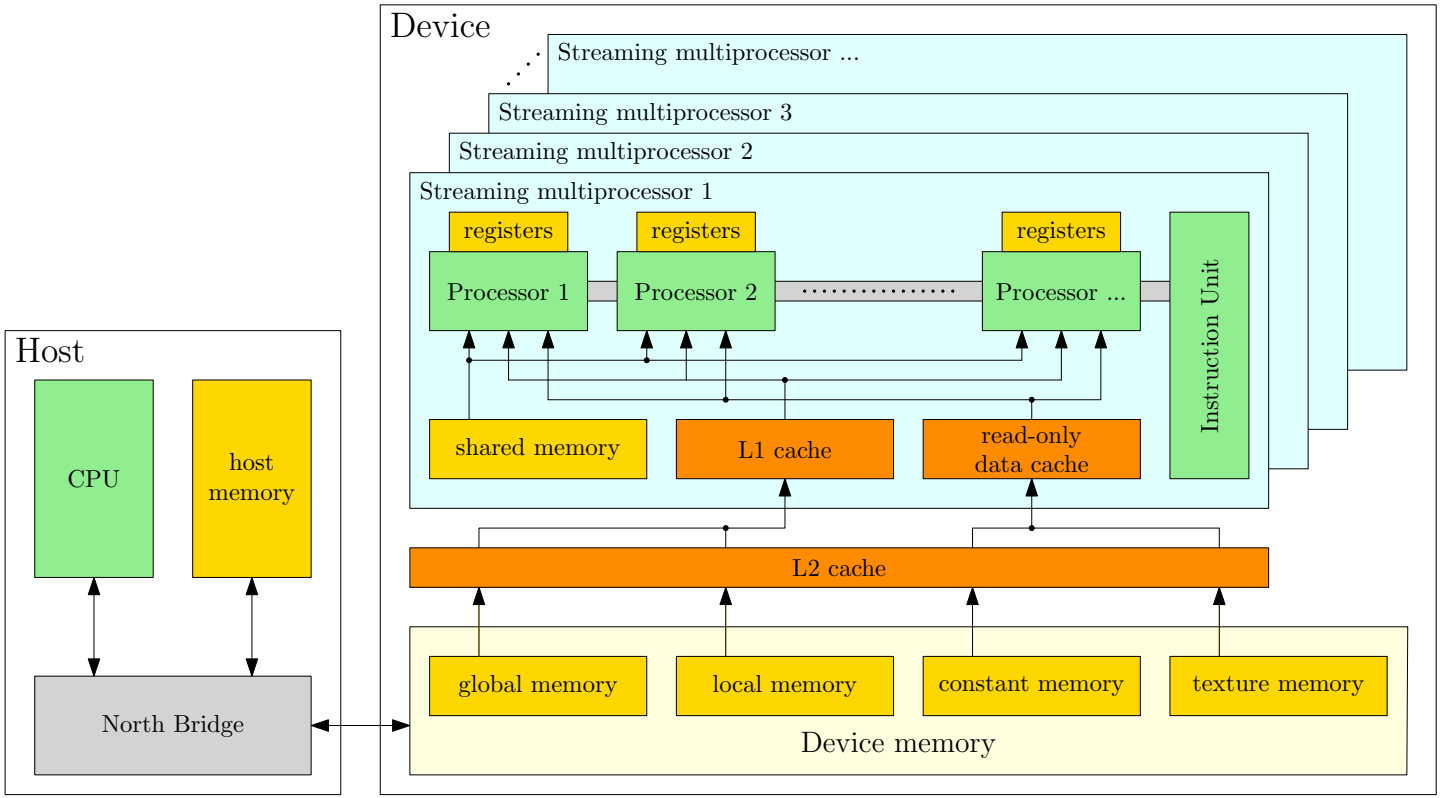


Figure 4.1: The NVIDIA® GPUs architecture (Kepler's generation)

hit an instruction which cannot be executed in parallel by all the threads, e.g., an `if-else` statement. At this moment these two branches are processed in a sequential way, firstly, the threads for the `if` branch in parallel and then the remaining threads for the `else` branch in parallel. On that account, the general aim is to minimize the branch divergence as much as possible in order to eliminate the decrease of the speedup.

The GPU algorithm can be designed using two different models, namely a *heterogeneous* or a *homogeneous* model (see comparison in Figure 4.2). In the homogeneous computing model all computations are performed on a *device* (GPU), while the heterogeneous model has the main logic of the algorithm on a *host* (CPU) and only computationally-intensive parts are accelerated by the GPU, which is executed repeatedly very often (illustrated by the reverse arrow in Figure 4.2). Usually, the heterogeneous model is used when the entire algorithm can not be executed in parallel. The performance of the heterogeneous model is affected by the communication between the host and the device and for that reason, this overhead should be minimized.

During a GPU algorithm design, the *memory model* of the algorithm has to be proposed in an efficient way, since one can store the data to different types of the memory on the GPU. The first one is a *global memory* which is used for the communication between the host and the device. Its advantages are accessibility for read/write operations by all threads and its size, e.g., 1 GB in case of the NVIDIA® GTX 650 Ti. On the other hand, the access to this memory can be outweighed by its huge latency. Basically, there are two ways how to shorten the huge latencies. The first one is a technique called a *coalescing*, which is based on the multiple access of the threads within a CUDA block to read from/write to the global memory, when the multiple requests are joined into a single one. This technique can be used for the specifically organized data only (the consecutive addresses of the memory). The second mechanism to accelerate the memory access is using an L1, L2 *cache* (illustrated in Figure 4.1). Moreover, the huge latencies can be partially hidden by a computation of another CUDA block, while the first block is waiting for the data. Other types of *device memory* are a *constant memory* and a *texture memory*. These memories are optimized to maintain data in a specific format, i.e., constants and 2D arrays (such as images), which are shared and accessible by all of the threads. Both are read-only memories, where the latencies can be shortened in the case, when the data is cached. A *shared memory* is very fast and it is used to exchange the shared data among all threads within one CUDA block. Nevertheless, the size of this memory is very limited, e.g., 16-48 kB per SM on the NVIDIA® GTX 650 Ti. *Registers* are used to store elementary local variables needed by each thread in a CUDA block. However, the size of registers is strongly limited. Therefore, the thread specific data structures that cannot fit into registers are stored in a *local memory*. The local memory has

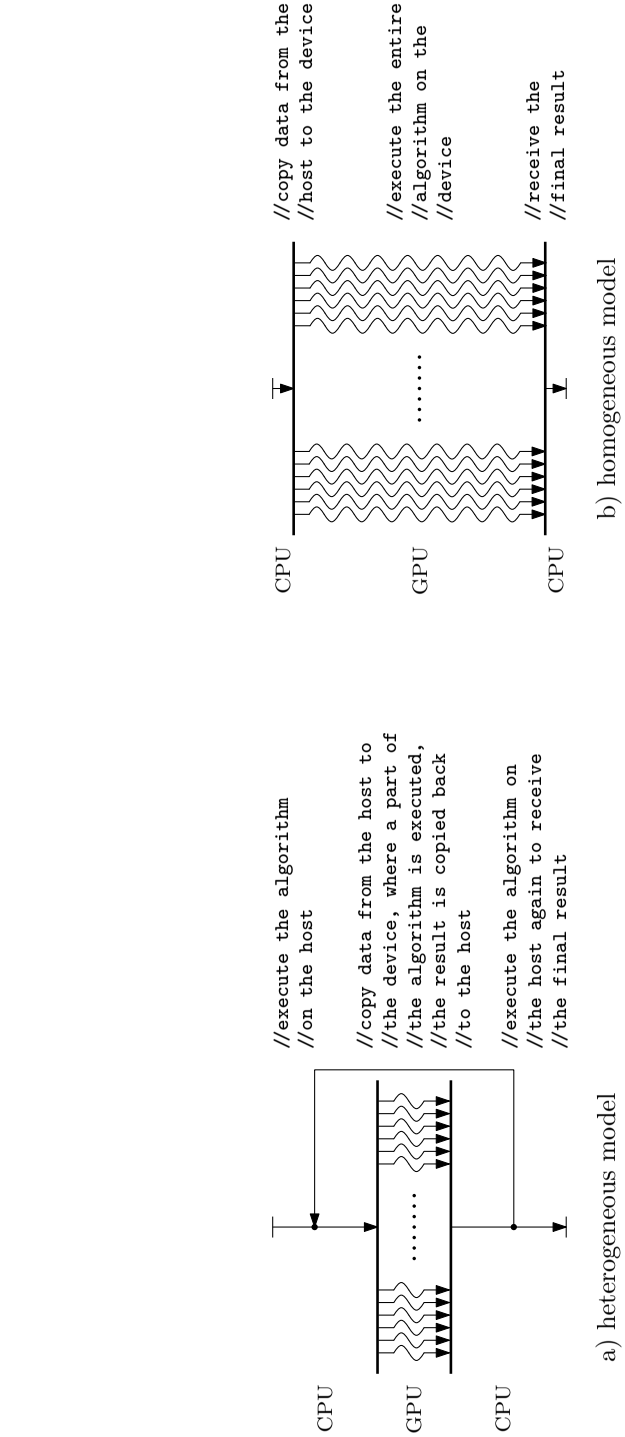


Figure 4.2: An execution of the heterogeneous and the homogeneous model of the algorithm on the GPU

similar properties as the global memory except the thread scope. All memory limitations are critical for the design of a GPU parallel algorithm, since the distribution of the data to the particular memory types has an essential impact on its performance.

4.3 The Nurse Rerostering Problem Statement

The Nurse Rerostering Problem considered in this chapter containing all used symbols) is the same as in (Moz and Pato (2007)) and it is defined as follows: Let E be a set of the scheduled human resources, i.e., the nurses of one department/unit in a hospital. The set of employees is fixed, i.e., one is not able to hire another nurse from a different department in order to solve NRRP. Each employee $e \in E$ has to met some requirements for the days off. Let $minDaysOff$ be a vector of the length $|E|$ corresponding to the minimal number of days off in every 7 consecutive days of the roster for each nurse $e \in E$. The *planning horizon* is given by a set of days D. The healthcare provided by the nurses is organized into several shifts. A set of shifts S consists of these types of shifts: *early*, *late* and *night* marked as $\mathcal{E}, \mathcal{L}, \mathcal{N}$ and, naturally, a *day off*, denoted as $\mathcal{F}$ (considered as a virtual shift). All shifts except $\mathcal{F}$ takes 8 hours. These shifts are assigned to an *original roster* R^0 of a size $|E| \cdot |D|$, which is created for the planning horizon. Each element of $R^0_{e,d} = s$ denotes that nurse $e \in E$ has shift $s \in S$ on day $d \in D$. However, in NRRP this roster is disrupted by unexpected circumstances. These disruptions are formally defined by a set of absences A, where absence $a \in A$ is given by a couple (e, d), where $e \in E$ and $d \in D$. The absence means that nurse e is not able to serve any of the shifts except the day off on day d. Various restrictions are taken into account in the case of NRP, e.g., the restrictions given by a labor code, a collective agreement, the contracts and the preferences of the nurses. However, not all of these constraints are considered in NRRP since the main, and the most important, goal of NRRP is to keep the original roster as much as possible. Therefore, the *modified roster* $\tilde{R}$ must fulfill the following set of hard constraints only:

(c_1) A nurse cannot be assigned to more than one shift per day.

(c_2) Nurses must have the minimal number of days off in every 7 consecutive days of the roster according to their average workload (35 or 42 hours per week). At least 2 days off for nurses with the average workload of 35 hours per week and 1 day off for nurses with the average workload of 42 hours per week have to be met.

(c_3) Nurses must have a minimal rest of at least 16 hours between two consecutive shifts, i.e., the sequences of consecutive shifts $\mathcal{LE}, \mathcal{NE}, \mathcal{NL}$ are forbidden.

(c_4) The set of absences A has to be respected.

(c_5) The roster $\tilde{R}$ cannot be modified before the first (earliest) absence over all nurses.

(c_6) The minimal number of requested shifts defined by matrix RS has to be provided.

In order to satisfy constraints such as (c_2) or (c_3) at the start of the planning horizon, the roster of the previous planning horizon R_{prev} has to be known. Furthermore, the required number of shifts assigned on a given day to the nurses has to be given in order to guarantee sufficient healthcare coverage. For this purpose, let RS be a matrix of requested shifts so that $RS_{s,d}$ is equal to the minimal number of shifts s assigned on day d in the modified roster. Finally, the objective function Z corresponds to the number of changes of the modified roster $\tilde{R}$ in comparison to the original roster R^0.

4.4 A Sequential Algorithm

This section describes the sequential approach which forms the basis of the parallel algorithm described in the next section. The sequential algorithm is a list based constructive heuristic. Let RP be a *list of roster positions* indexed by a *position* i. Each *roster position* RP_i contains a unique pair (e, d) where nurse $e \in E$ and day $d \in D$ and it refers to shift s such that $R^0_{e,d} = s$.

The basic idea of Algorithm 2 is the following: At the beginning of the algorithm, the modified roster $\tilde{R}$ is initialized by function `initAll` such that it contains only the shifts from R^0 before the earliest absence over all nurses (see constraint (c_5)). The value of position i is also set by this function due to the same constraint. The indices of all the roster positions from the original roster R^0 are randomly ordered to the list of the original roster positions so that $RP = (RP_i)_{i \in \langle 0, |E| \cdot |D| - 1 \rangle}$. Subsequently, the shifts are assigned back to $\tilde{R}$ one by one according to the hierarchical rules Rule1 – Rule 4 specified in Step 2 in the order given by RP, i.e., for all i in ascending manner. Solution $\tilde{R}$ is found when all the shifts given by RP are assigned. Processing of one RP (line 4-15 in Algorithm 2) is called a *run* and the number of the performed runs is stored in an auxiliary variable *run*. Each run considers a different RP list and for that reason, each one can obtain a different modified roster. The best modified roster over all runs is stored in $\tilde{R}^{best}$. The stopping condition is the number of performed runs which is given by constant $maxRuns$.

When Rule 1 – Rule 4 cannot be applied to assign shift s from RP_i, the `doBacktrack` function is called. This backtrack is based on taking out of some shifts from $\tilde{R}$ and swapping of the roster positions in RP. Thereby the

algorithm increases the probability that a feasible solution will be found in this run.

4.5 A Homogeneous Model of the Parallel Algorithm

Unfortunately, the algorithm explained in the previous section does not provide enough parallelism to be directly implemented on a GPU. An even worse issue is the branch divergence in Step 2. For example, if there are two shifts to be assigned in parallel and one is assigned immediately after applying Rule 1 and the second one has to try all four rules then the execution of the algorithm

Algorithm 2: A constructive heuristic for NRRP

Input : NRRP instance given by $\{R^0, RS, A, maxRuns\}$
Output: Best found modified roster $\tilde{R}^{best}$

$\tilde{R}^{best} \leftarrow null$; $run \leftarrow 0$;
while $run < maxRuns$ **do**
  Step 1:
  $RP \leftarrow$ `randomInit(`R^0`);` `// initialize an order of the roster positions`
  $[\tilde{R}, i] \leftarrow$ `initAll(`R^0, A`);` `// initialize modified roster and the position counter`
  while $i < |E| \cdot |D|$ **do** `// assign shifts given by` RP `iteratively`
    Step 2:
    assign shift s given by RP_i to the modified roster $\tilde{R}$ satisfying (c_1) - (c_6) according to hierarchical rules:
    - *Rule 1:* assign s to a nurse who was scheduled on this shift in R^0; **if** assigned **then goto** Step 3;
    - *Rule 2:* assign s to a nurse who was not scheduled on this shift in R^0, but to whom the assignment satisfies (c_3) wrt. R^0 from both sides ($R^0_{e,d-1}$ and $R^0_{e,d+1}$); **if** assigned **then goto** Step 3;
    - *Rule 3:* assign s to a nurse who was not scheduled on this shift in R^0, but to whom the assignment satisfies (c_3) wrt. R^0 from one side ($R^0_{e,d-1}$ or $R^0_{e,d+1}$); **if** assigned **then goto** Step 3;
    - *Rule 4:* assign s to an arbitrary nurse (without a constraint violation in $\tilde{R}$); **if** assigned **then goto** Step 3;

    $[RP, \tilde{R}, i] \leftarrow$ `doBacktrack(`RP`,` $\tilde{R}$`,` i`);` `// backtrack when` s `has not been assigned`
    if $i < 0$ **then break else goto** Step 2; `// no feasible` $\tilde{R}$ `for given` RP `found`
  Step 3:
  $i \leftarrow i + 1$;
  Step 4:
  if $Z(\tilde{R}) < Z(\tilde{R}^{best})$ **then** $\tilde{R}^{best} \leftarrow \tilde{R}$; `// choose better roster from` $\tilde{R}$ `and` $\tilde{R}^{best}$
  $run \leftarrow run + 1$;
return $\tilde{R}^{best}$; `// return` $\tilde{R}^{best}$ `having the minimal number of changes in comparison to` R^0

can not continue until both shifts are not assigned. Due to the SIMT (Single Instruction Multiple Threads) thread mapping and long latency of the global memory, such a naive parallelization can not provide the expected speedup of the parallel algorithm. This section explains how to deal with these crucial issues and shows the design of the homogeneous parallel algorithm which is the main contribution of this chapter.

4.5.1 Problem Decomposition for Parallelization

A simple basic rule for the design of an efficient GPU algorithm is to exploit as many threads as possible. In our case, the smallest independent data element to be processed by a single thread is the modified roster of one nurse. The key idea of this problem decomposition lies in the unique mapping of an individual thread to each nurse (described in this section) and an efficient processing of hierarchical Rule 1 – Rule 4 from Step 2 in Algorithm 2 (described in Section 4.5.2 and 4.5.3).

First of all, the list of roster positions RP from the sequential algorithm has to be replaced in the parallel algorithm by several partial lists stored in RPP (see Figure 4.3). The RPP is an array of $|E|$ *partial lists of roster positions* $RPP_e = (RPP_{e,i})_{i \in \langle 0,|D|-1 \rangle}$. Each element of the partial list $RPP_{e,i} = d$ refers to element $R^0_{e,d}$ in the original roster. These elements in RPP_e are organized in the same order as in the RP. Practically, RPP splits the original RP into lists dedicated to each particular nurse $e \in E$ as is illustrated in Figure 4.3.

However, $|E|$ threads cannot fully utilize the GPU still, or at least, to hide the memory access latency. For that reason, we need to find another way how to exploit the computational power of the GPU more effectively. Since a single run of the main `while` loop in Algorithm 2 is independent of the other runs, all the runs can be executed concurrently, each of them with its own RPP. Then we refer to one instance of Algorithm 2 as an *instance of the algorithm* and m is the *number of instances of the algorithm* executed at the same time. It means that $m \cdot |E|$ threads are used to solve one NRRP instance. Consequently, m_b denotes the *number of instances of the algorithm per one CUDA block* ($m_b \leq m$). Then each CUDA block contains $m_b \cdot |E|$ threads. The values of m and m_b are chosen with respect to the size of the NRRP instance and the number of SMs on the used GPU.

4.5.2 Algorithm Reorganization

Thanks to the decomposition described above we have enough threads to be executed. Unfortunately, the decomposition itself does not guarantee full utilization of the GPU due to the branch divergence, which is mostly appreciable in Step 2 of Algorithm 2. Rule 1 decides, whether shift s given by RP_i can be

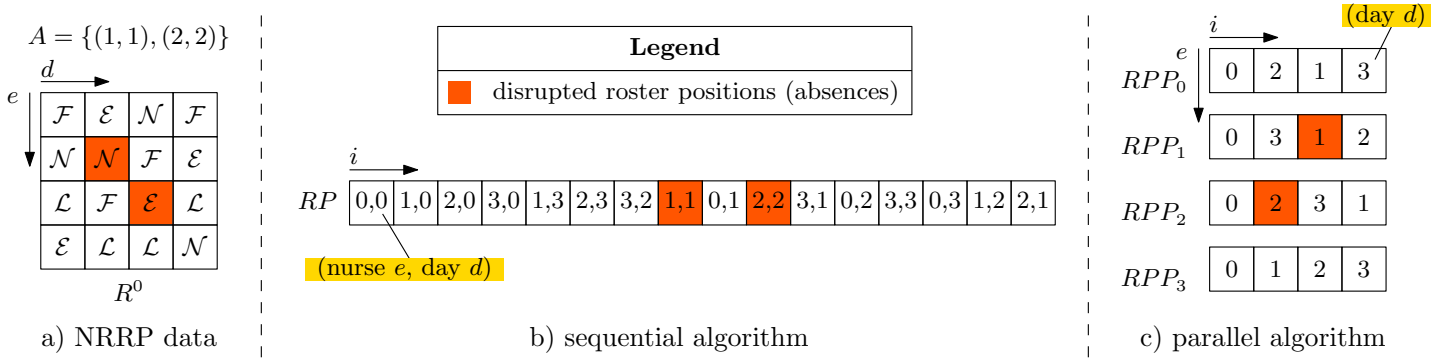


Figure 4.3: The comparison of RP and RPP for the sequential and the parallel algorithm on example

assigned to the original employee e on day d. On the other hand, the remaining rules (Rule 2 – Rule 4 in Algorithm 2) search for an employee satisfying the given rule. It means that s can be assigned directly or, in the worst case, the algorithm has to evaluate the assignment of shift s to $|E|$ nurses for Rule 2 – Rule 4. From the computational time point of view, there is a significant imbalance between the execution of Rule 1 and Rule 2 – Rule 4, which is not appropriate for parallel algorithms on the GPU. As a consequence, Algorithm 2 had to be reorganized due to the high branch divergence. A schematic reorganization of the original sequential algorithm (Algorithm 2) and the algorithms proposed in this chapter are depicted in Figure 4.4.

The original sequential algorithm (Figure 4.4a) was reorganized to the modified one (Figure 4.4b) and, consequently, to the heterogeneous (Figure 4.4d) and the homogeneous model (Figure 4.4c) of the parallel algorithm as follows. Step 1 used for the initialization of the list of roster positions is marked as **Part Init**. Subsequently, Step 2 was split into two parts. The first one called **Part A** reads the shifts to be assigned. Secondly, **Part B** represents the evaluation of these shift assignments with respect to the hierarchical rules from Step 2. The branch divergence in Step 2 is eliminated in **Part B** by a smart switching of the hierarchical rules explained in Section 4.5.3. Consequently, these shifts are assigned to the nurses in **Part C**, which comprises Step 3 and Step 4 of Algorithm 2. The backtrack is realized in this part also. According to this algorithm reorganization, one run of the modified sequential algorithm consists of **Part Init** and {**Part A, B, C**} in a `while` loop.

4.5.3 Minimization of Branch Divergence

The parallel algorithm in Figure 4.4c) in comparison to the modified sequential one in Figure 4.4b) has an extra **Part Alloc**, where the needed memory space is allocated at the beginning of the algorithm. Subsequently, **Part Init** in the parallel algorithm is similar to the sequential one, while instead of initializing a single RP we initialize $m \cdot |E|$ partial lists of the roster positions RPP_e. In order to minimize the branch divergence, the parallel algorithm considers two *modes*. Let $firstRule$ be a boolean variable defining in which mode the algorithm is, i.e., how the threads of the given run process the shifts to be assigned. If **Part B** is in the $firstRule$ mode, hierarchical Rule 1 is used, otherwise the algorithm evaluates Rule 2 – Rule 4 simultaneously. Both modes are illustrated in Figure 4.5 (solving the NRRP instance from the example in Figure 4.3). The figure shows one run of one instance of the parallel algorithm. The header of the figure (the row labeled by $i = 0$ in the upper left corner) contains the initial state of the NRRP instance given by **Part Init** (the shifts that are before the first (earliest) absence are fixed with respect to constraint (c_5)). Each row of the figure illustrates a single execution of the sequence

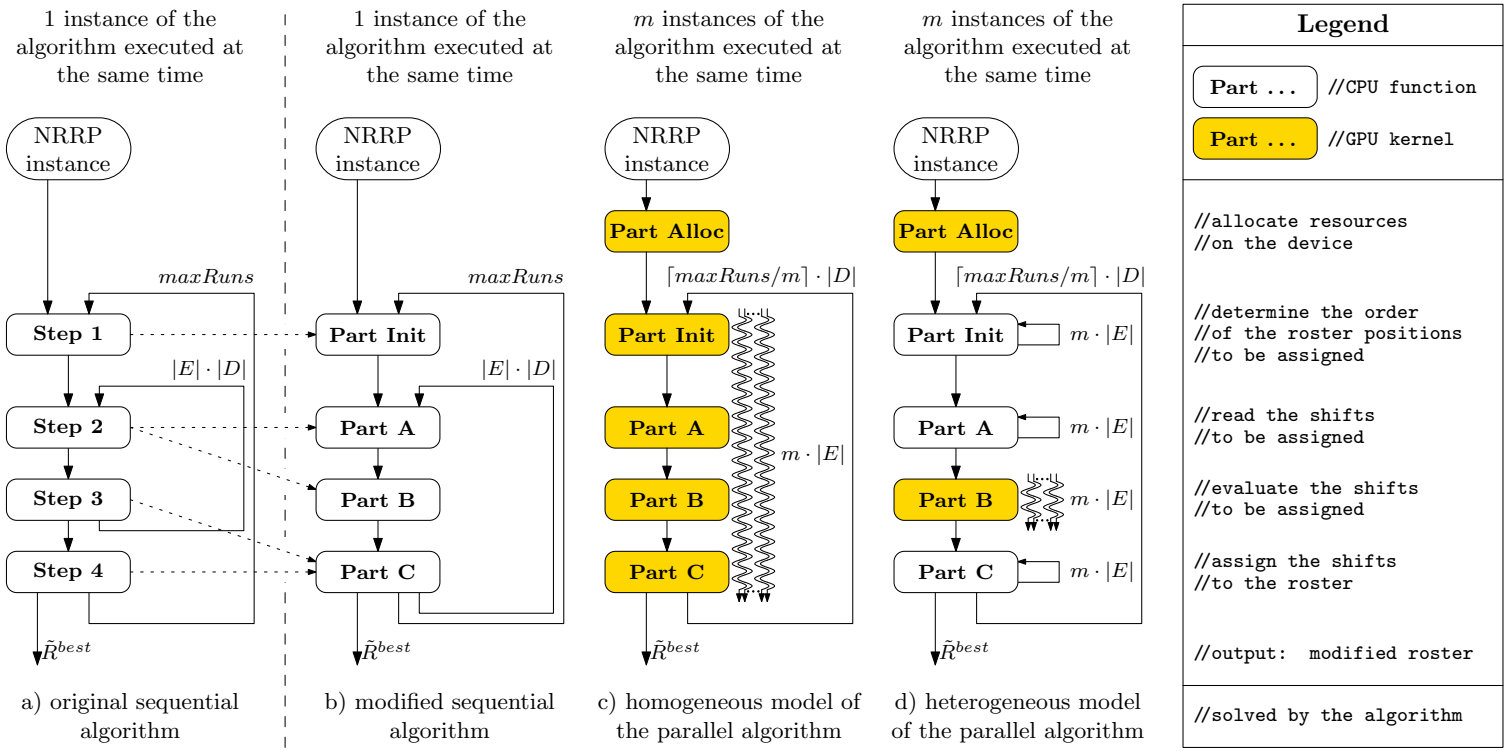


Figure 4.4: The overview of the sequential and the parallel algorithms

{**Part A, B, C**}. The left side of each row shows which roster positions are processed in the current sequence of {**Part A, B, C**}, while the current state of modified roster $\tilde{R}$ after finishing **Part C** is presented on the right.

The one instance of the algorithm starts on position $i = 1$ in the $firstRule$ mode in Figure 4.6. Firstly, for each thread **Part A** takes the appropriate $RPP_{e,i}$ and determines shift s to be assigned. Secondly, each thread in **Part B** evaluates the shift assignment of shift s to employee e on day d. Each assignment is evaluated according to the penalization function

$$pen = \begin{cases} 0, & \text{iff Rule 1 succeeded} \quad (\text{occurs in the } firstRule \text{ mode}) \\ 1, & \text{iff Rule 2 succeeded} \quad (\text{occurs in the } \neg firstRule \text{ mode}) \\ 2, & \text{iff Rule 3 succeeded} \quad (\text{occurs in the } \neg firstRule \text{ mode}) \\ 3, & \text{iff Rule 4 succeeded} \quad (\text{occurs in the } \neg firstRule \text{ mode}) \\ \infty, & \text{otherwise} \quad (\text{occurs in both modes}) \end{cases} \tag{4.1}$$

Since the $firstRule$ mode is active, possible evaluations are $pen = 0$ or $pen = \infty$. Thirdly, the last **Part C** in the $firstRule$ mode assigns the shifts having $pen = 0$ and the others are marked as unassigned (i.e., shift $\mathcal{E}$ on roster position $R_{2,2}$). If all the shifts are assigned to the original employees and the $firstRule$ mode is active, the instance of the algorithm remains in this mode and position i is incremented. If at least one of the shifts is not assigned (see $pen = \infty$ of thread 2 in the row with $i = 1$ and the $firstRule$ mode), the instance of the algorithm continues in the $\neg firstRule$ mode executing the same sequence {**Part A, B, C**} (see the row with the $\neg firstRule$ mode and position $i = 1$ in Figure 4.5). **Part A** in the $\neg firstRule$ mode loads shift s which has not been previously assigned to all the nurses. **Part B** evaluates the assignment of shift s to all the nurses concurrently using Eq. (4.1). Finally, shift s is assigned in **Part C** to the employee having the minimal penalty (in our example shift $\mathcal{E}$ is assigned to nurse 1). Subsequently (see the row with position $i = 2$ in Figure 4.5), either the instance of the algorithm continues in the $\neg firstRule$ mode if there is another unassigned shift on the current position i or in our case, it is switched back to the $firstRule$ mode and position i is increased by one. In summary, the branch divergence was reduced very strongly by this efficient model, since the code executed in both modes is practically the same, independently on position i and the current run.

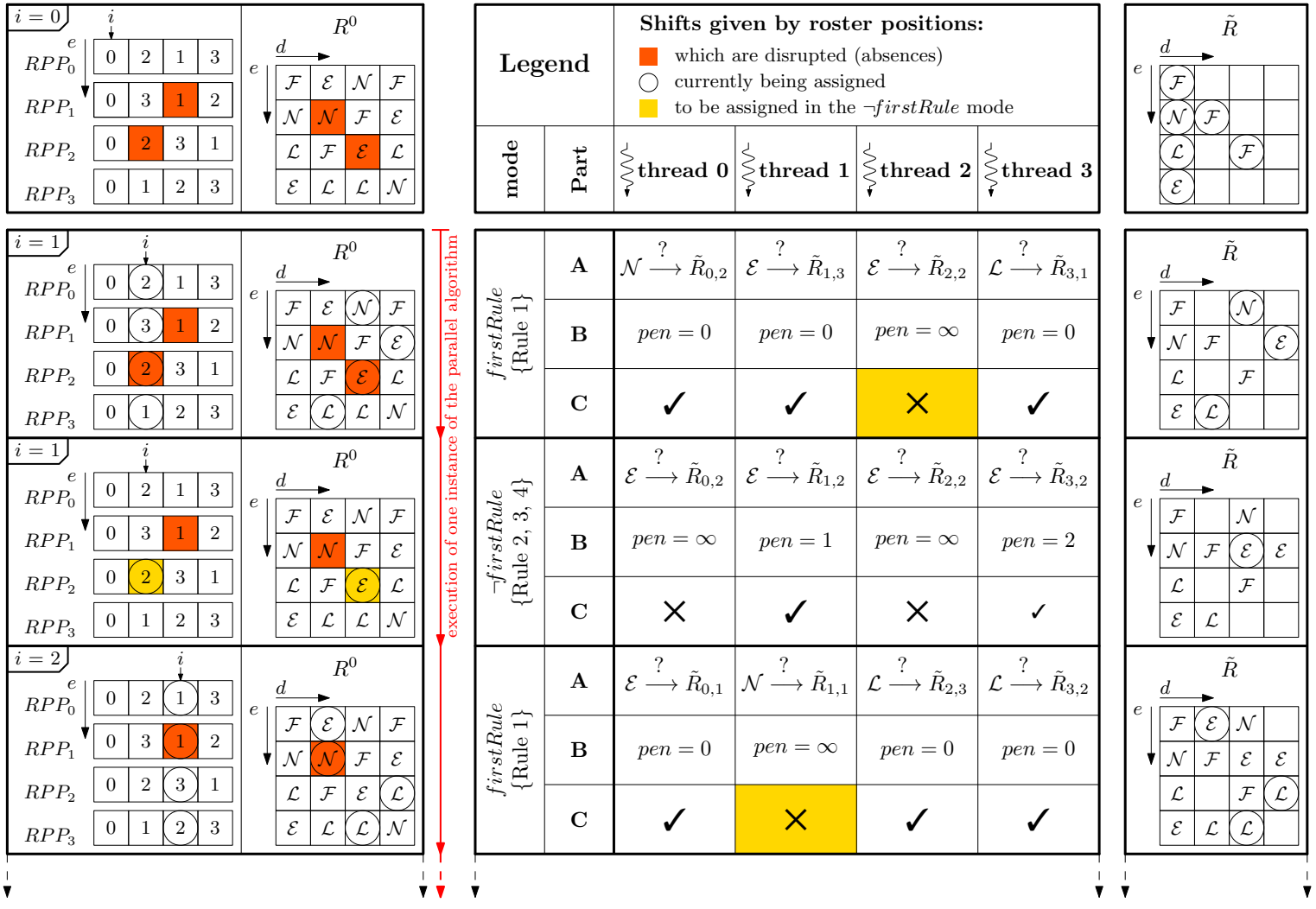


Figure 4.5: Example of the execution of 1 instance of the parallel algorithm in two modes given by the $firstRule$ flag

4.5.4 Detailed Description of the Homogeneous Model of the Parallel Algorithm

This section describes, in detail, the design of the homogeneous model of the parallel algorithm that is illustrated by Algorithm 3 in **one thread executed in one instance of the parallel algorithm** point of view. **Part Alloc** (line 1–5 in Algorithm 3) determines the number of instances of the algorithm m and allocates the memory space needed by function `allocateGPUResources` with respect to the size of the NRRP instance and the number of available streaming multiprocessors on the GPU. At the beginning of the algorithm, the necessary variables are initialized. Firstly, the variables that are common to all the instances of the algorithm (see line 2) and, secondly, the specific variables for one instance of the algorithm (line 3). The thread mapping is determined by the function `mapThreads`, since it is the same during the entire execution of the instance of the algorithm.

Algorithm 3: The homogeneous model of the parallel algorithm – 1 thread of 1 instance of the algorithm

Input : NRRP instance $\{R^0, RS, A, maxRuns\}$
Output: Best found modified roster $\tilde{R}^{best}$

Part Alloc:
// determine the number of parallel instances of the algorithm
1 $m \leftarrow$ `allocateGPUResources`$(|E|, |D|, maxRuns)$;
// initialize variables common to all instances of the algorithm
2 $\tilde{R}^{best} \leftarrow$ `null`; $run \leftarrow 0$; $applyLS \leftarrow$ `false`; $terminateAlg \leftarrow$ `false`;
// initialize variables for this instance of the algorithm
3 $firstRun \leftarrow$ `true`; $applyBT \leftarrow$ `false`; $backtrack \leftarrow 0$;
// set the employee index for the current thread
4 $e \leftarrow$ `mapThreads`$(|E|)$;
5 `__syncthreads()`;

// perform m parallel instances of the algorithm, each exploits $|E|$ threads
6 **while** `true` **do**
  Part Init:
7 **if** $firstRun \parallel i = |D| \parallel \neg feasible$ **then**
8 **if** $run < maxRuns$ **then**
  // initialize data for the new run
9 $firstRun \leftarrow$ `false`; $feasible \leftarrow$ `true`; $i \leftarrow 0$; $firstRule \leftarrow$ `true`;
10 $[\tilde{R}, isOccupied, unassigned, i] \leftarrow$ `initAll`(e, R^0, A);
11 **if** $\neg applyBT$ **then** $RPP \leftarrow$ `generatePartialRosterPositions`$(e, R^0, applyLS)$ **else** $applyBT \leftarrow$ `false`;
12 **else**
13 $terminateAlg \leftarrow$ `true`; // stop condition, part # 1
14 `__syncthreads()`;

  // perform the sequence of Parts A, B and C (see Algorithm 4)
15 `performSequenceOfPartsABC()`;
16 **return** $\tilde{R}^{best}$;

Algorithm 4: Part A, B, C of Algorithm 3

Input : By reference – NRRP instance data from Algorithm 3, line 15
Output: By reference – NRRP instance data to Algorithm 3, line 15

Part A:

1 **if** *terminateAlg* **then exit** Algorithm 3; `// stop condition, part # 2`
2 **if** *firstRule* **then** `// read the shift`
3 $d \leftarrow RPP_{e,i}$; $s \leftarrow R^0_{e,d}$;
4 **else**
5 $d \leftarrow d \in unassignedRP$; $s \leftarrow R^0_{unassignedRP}$;
6 `__syncthreads();`

Part B:

7 $pen \leftarrow$ `evaluateShiftAssignment`$(e, d, s, \tilde{R}, R^0, R_{prev}, RS, isOccupied, minDaysOff)$;
8 `__syncthreads();`

Part C:

9 **if** *firstRule* **then**
10 **if** $pen = 0$ **then** `// either assign the shift or mark it as unasssigned by Rule 1`
11 $R_{e,d} \leftarrow s$; $isOccupied_{e,d} \leftarrow$ `true`;
12 **else**
13 $unassigned_e \leftarrow RPP_{e,i}$;
14 **else**
15 **if** $e = 0$ **then** `// either assign the shift or set this run as infeasible`
16 $e_{min} \leftarrow \text{argmin}_{e \in E}(pen)$;
17 **if** $pen_{e_{min}} < \infty$ **then**
18 $R_{e_{min},d} \leftarrow s$; $isOccupied_{e_{min},d} \leftarrow$ `true`; $unassigned_{e_{min}} \leftarrow$ `null`;
19 **else**
20 $[applyBT, RPP_e] \leftarrow$ `doModifiedBacktrack`(RPP_e, i); $feasible \leftarrow$ `false`;
21 `__syncthreads();`
22 **if** $e = 0$ && *feasible* **then** `// set mode of this run wrt. unassigned shifts`
23 $unassignedRP \leftarrow$ `pickUnassignedRosterPosition`$(unassigned)$;
24 **if** $unassignedRP =$ `null` **then**
25 $firstRule \leftarrow$ `true`; $i \leftarrow i + 1$;
26 **else**
27 $firstRule \leftarrow$ `false`;
28 `__syncthreads();`
29 **if** $i = |D|$ && *feasible* **then** `// evaluate the finished feasible run`
30 $Z_e(\tilde{R}) \leftarrow$ `computeObjectiveFunction`$(e, \tilde{R}, R^0)$;
31 `__syncthreads();`
32 **if** $e = 0$ **then**
33 **if** $i = |D| \parallel \neg feasible$ **then**
34 **if** *feasible* **then** `// critical section: keep the best found solution`
35 $[Z(\tilde{R}^{best}), \tilde{R}^{best}, RPP^{best}] \leftarrow$ `chooseBetter`$(\sum_{\forall e \in E} Z_e(\tilde{R}), Z(\tilde{R}^{best}))$;
36 **else**
37 $applyLS \leftarrow$ `updateLocalSearchParameters()`; `// update for the next run`
38 $run \leftarrow run + 1$; `// critical section: increment the counter of runs`
39 `__syncthreads();`

Part Init (line 7–14) initializes the next run and checks the stop condition (described in Section 4.5.4.3). Initialization of a new run is performed by function `initAll`. Firstly, this function clears modified roster $\tilde{R}$. Secondly, this function resets all the elements of a binary matrix *isOccupied* to `false`. This matrix has the same dimensions as $\tilde{R}$ and keeps information whether the roster position is already occupied by any shift or is not in order to ensure one assignment of one shift to one roster position (constraint (c_1)). Thirdly, this function also clears the vector *unassigned*, which is used to store all the roster positions of the unassigned shifts from the $firstRule$ mode. Fourthly, position i is determined with respect to absences A and constraint (c_5), i.e., position i is set to the first element of RPP that can be assigned to the different roster position in $\tilde{R}$ in comparison to R^0. Function `generatePartialRosterPositions` creates partial lists RPP in two ways, either by the local search methods (see Section 4.5.4.2) or in a standard manner. In this case, the roster positions from RPP are permutated randomly. Each RPP_e is generated by a different thread e, i.e., RPP is generated in parallel. Consequently, the roster positions that can not be modified with respect to absences A and constraint (c_5) are shifted left as much as possible in each partial list, since these roster positions have to contain the day-off. Subsequently, the sequence of {**Part A, B, C**} is executed by function `performSequenceOfPartsABC` (line 15 in Algorithm 3), whose body is described by Algorithm 4.

Part A (see line 1–6 in Algorithm 4) reads shift s to be assigned for each thread. Either shift s is given by $RPP_{e,i}$ (in the $firstRule$ mode) or it is given by $unassignedRP$ (in the $\neg firstRule$ mode), which is one chosen element from vector *unassigned*.

Part B (see line 7–8 in Algorithm 4), represented by function `evaluateShiftAssignment`, returns the penalization defined by Eq. (4.1) with respect to the input data. Namely, the shift assignment given by employee e, shift s and day d, the current modified roster $\tilde{R}$, original roster R^0 and R_{prev} and matrix *isOccupied*.

Part C (line 9–15 in Algorithm 4) tries to assign shift s to the modified roster $\tilde{R}$ and prepares all variables for the next sequence {**Part A, B, C**}. One can notice, that the behavior of **Part C** is dependent on the mode given by flag $firstRule$ (see line 9–21 in Algorithm 4). On one hand, in the $firstRule$ mode, the value of pen can be either equal to 0 or to ∞. If $pen = 0$, shift s is assigned to $\tilde{R}$ to the original roster position. In the case of $pen = \infty$, shift s can not be assigned to the original roster position and, therefore, this roster position is stored to *unassigned* to process it in the $\neg firstRule$ mode later. On the other hand, the $\neg firstRule$ mode assigns shift s to the employee having the minimal penalization over all employees and removes this roster position from *unassigned*. If more employees have the same minimal penalty, a random one is chosen. When the minimal penalty is ∞, shift s can not be assigned to

any employee and the current run of the instance of the algorithm is marked as infeasible.

The following instructions of **Part C** (line 22–15 in Algorithm 4) are executed in order to update position i, flag $firstRule$ and counter run. If there is any unassigned roster position in $unassigned$, this roster position is chosen by function `pickUnassignedRosterPosition`, stored into $unassignedRP$ and the mode is switched to $\neg firstRule$. If all roster positions are assigned, the instance of the algorithm continues in the $firstRule$ mode and position i is incremented. The relation among the modes, particular parts of the parallel algorithm and the unassigned shifts are illustrated in Appendix B.

When the incremented position i exceeds the length of RPP_e (equal to the number of days $|D|$) and the current run is feasible (see line 29 in Algorithm 4), then a new solution was found. In this case the objective function $Z_e(\tilde{R})$ related to employee e is enumerated by function `computeObjectiveFunction`. Subsequently, the current objective function value given by $\sum_{\forall e \in E} Z_e(\tilde{R})$ is compared to $Z(\tilde{R}^{best})$ and the better one is preserved by function `chooseBetter`. Finally, run is incremented in order to start execution of the new run.

One can notice, that some sections of the code for one instance of Algorithm 4 are executed only if $e = 0$, e.g., line 32–38. These sections are executed by a single thread, since it does not make sense to execute them in parallel. Nevertheless, you can observe the symbol of a lock in the same section of the code (namely, line 35 and line 38). This symbol presents the critical sections of the code, where the access of the multiple instances of the algorithm have to be controlled via locks (see more in Section 4.5.5.1). Finally, it is also important to stress that the execution of the threads within one CUDA block have to be synchronized in order to preserve the data consistency. This is called a barrier synchronization realized by instruction `__syncthreads()` (e.g., line 21 in Algorithm 4).

Algorithm 3 and Algorithm 4 together with the description above explains how one instance of the algorithm works. To illustrate the relationship between the runs and multiple instances of the algorithm, Figure 4.6 shows an example considering an NRRP instance with $|E| = 4$ and $|D| = 4$. Each row corresponds to one of m instances of the algorithm. The numbers in the circles in each row represent different values of position i. One can notice that the run in the first instance of the algorithm is resolved within 5 iterations of the sequence {**Part A, B, C**} and then, the next run is started (gray circle). The second row illustrates an instance of the algorithm that executes the run terminated at position $i = 1$ returning no solution. The last row shows an instance of the algorithm that repeats position $i = 1$ three times. Firstly, the $firstRule$ mode produces two unassigned roster positions that are stored into $unassigned$. Then these shifts are assigned by two consequent sequences of {**Part A, B, C**}

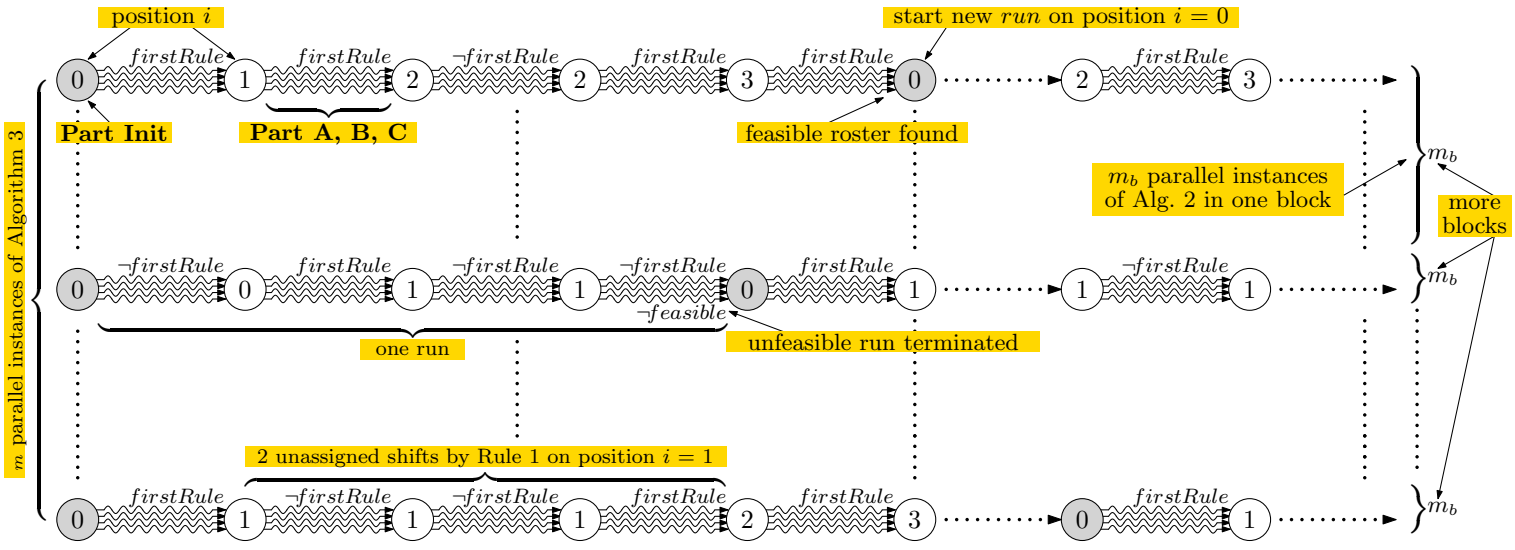


Figure 4.6: The parallel execution of m instances of the parallel algorithm for the NRRP instance with $|E| = 4$, $|D| = 4$

in the $\neg firstRule$ mode. In general, the parallel execution of the instances of the parallel algorithm can be executed asynchronously, i.e., the different positions i can be explored by the different instances of the parallel algorithm in one moment.

4.5.4.1 Backtrack

There are two mechanisms how to improve the quality of the solutions. The first one is the short-term mechanism discussed in this subsection – the backtrack. This mechanism improves the convergence of the algorithm to feasible solutions. The second one is the long-term mechanism – the local search, introduced in Section 4.5.4.2. Its aim is to improve the quality of the best solution found.

With respect to the GPU limitations, we were not able to apply the backtrack originally used in Algorithm 2, since it is memory consuming and it increases the branch divergence of the parallel algorithm. Therefore, we propose the modified backtrack algorithm (see Algorithm 5, function `doModifiedBacktrack`). It is called from the main Algorithm 3 (see line 20) whenever it is not possible to assign shift s in the $\neg firstRule$ mode. The input parameters of function `doModifiedBacktrack` are position i and the list of roster positions RPP_e, which is an output parameter as well. The output value of this function is also a flag $applyBT$ representing whether the backtrack can be applied or not. The backtrack itself is based on the idea of swapping the roster positions in RPP such that the unassigned roster position $RPP_{e,i}$ is swapped with the previous one $RPP_{e,i-1}$ and such a modified RPP_e is returned. The backtrack is performed till the position $i > 0$ and the counter of backtrack steps $backtrack$ does not exceed the maximal number of backtrack steps $maxBacktracks$. The resulting $applyBT$ is taken into account in Algorithm 3 on line 11. When the backtrack is applied ($applyBT$ = `true`), the generation of RPP by the function `generatePartialRosterPositions` is

Algorithm 5: Function `doModifiedBacktrack`

Input : List of roster positions RPP_e, position i
Output: Flag $applyBT$, whether the backtrack will be applied or not, updated list of roster positions RPP_e

```
1 if i > 0 && backtrack < maxBacktracks then
2     applyBT ← true;
3     backtrack ← backtrack + 1;
4     swapRosterPositions(RPP_{e,i-1}, RPP_{e,i});
5 else
6     applyBT ← false;
7     backtrack ← 0;
8 return [applyBT, RPP_e];
```

skipped since RPP is already given by the previous run that updates RPP_e in the function `doModifiedBacktrack`.

4.5.4.2 Local Search

The original sequential algorithm Algorithm 2 may be extended by an auxiliary genetic algorithm in order to permutate the list of roster positions RPP^{best}. However, a direct implementation of this extension would increase the branch divergence significantly and, therefore, we propose our local search, which eliminates these drawbacks. The key idea of the local search is to explore the state space near the local optima $\tilde{R}^{best}$ represented by RPP^{best}.

The local search is controlled by a flag $applyLS$ set by function `updateLocalSearchParameters` (line 37). At the beginning of the algorithm the local search is switched off ($applyLS \leftarrow$ `false`). When a new feasible solution is stored, counter $runsNoSuccess$, representing the number of runs without improvement of $Z(\tilde{R}^{best})$, is reset to zero. If the feasible solution is not found, or it is found and $Z(\tilde{R}) \geq Z(\tilde{R}^{best})$, the counter $runsNoSuccess$ is incremented. In case that its value exceeds a given threshold and $\tilde{R}^{best} \neq$ `null`, the local search is switched on ($applyLS \leftarrow$ `true`). The local search generates the RPP in the function `generatePartialRosterPositions` (line 11) as a modification of RPP^{best} in the following parallel way. Firstly, RPP_e^{best} is copied to RPP_e. Subsequently, each position i of RPP_e is considered to be swapped with another random position $i' \in D$. The swap of the elements $RPP_{e,i}$ and $RPP_{e,i'}$ is carried out with probability p_{LS} (common to all the instances of the algorithm) such that each thread is responsible to make the swap in its RPP_e. Probability p_{LS} is set to its initial value at the start of the local search by the function `updateLocalSearchParameters`. When the local search is applied and a better solution than $\tilde{R}^{best}$ has not been found, the same function decreases gradually (after the given number of runs – $runsOfSameProb$) probability p_{LS} to its half. When p_{LS} is below the threshold that is close to zero, the local search is stopped ($applyLS \leftarrow$ `false`) and the RPP of the next run is generated in the usual manner (see Section 4.5.4, **Part Init**).

4.5.4.3 Stop Condition

The runs are executed in parallel by m instances of the algorithm till the total number of runs is smaller than $maxRuns$. The stop condition implemented by a $terminateAlg$ flag is split into two parts, as you can notice in Algorithm 3, line 13 and line 1. When it would be only executed by line 6 or by line 13, a deadlock happens, since one of the instances of the algorithm passes the stop condition and leaves the `while` loop. However, the rest of the instances of the algorithm will be caught by the barrier synchronization in line 14, waiting on

the instance of the algorithm that has been already terminated.

4.5.5 Memory Model

There are more types of memory on the GPU device (described in detail in Section 4.2) and one has to decide which part of the data will be stored in which memory, since the final speedup is closely dependent on this decision as is illustrated in the experimental part of this chapter (Section 4.7.2). A naive idea is that the maximum speedup could be obtained by the distribution of all the data to the registers, the shared memory or the data covered by the cache, which have significantly shorter latencies than the global memory in general. However, we have two different algorithm workspaces, the global one to keep the global data belonging to the algorithm itself (e.g., Z^{best}) and a local workspace to store the specific data for each instance of the algorithm (e.g., position i). Nevertheless, the distribution of the data in the memory is not so straightforward (see Table 4.1), since it depends on the size of an NRRP instance, naturally, on the parameters of the used GPU and the frequency of the data access. In our case, a part of the specific data has to be stored in the global memory because of its size ($pen, unassigned$). On the contrary, some local copies of the global data (e.g., R^0, A, R_{prev}) can be put into the shared memory to accelerate the execution of the algorithm.

Also, the coalescing and the caching of the data has to be taken into account in order to shorten the memory latencies. Unfortunately, the behavior of the threads within one CUDA block is not deterministic (the number of the threads executed in the $firstRule$ and $\neg firstRule$ mode is not known) and for that reason, the coalescing cannot be applied except for copying data from the global memory to the shared memory by the `allocateGPUResources` function. The only way to accelerate the access to the global memory is to use the cache and, therefore, our aim is to minimize the amount of data in the global memory as much as possible.

The memory model is summarized in Table 4.1, which is organized as follows: Each row contains one variable name, its data type, its memory type, its size in Bytes (B) and a short variable description. There are five data types, `uint` for unsigned 32 bit integers, `bool[]` for binary arrays, `bool` for booleans represented on the GPU as `uint`, `float` for a single precision floating point representation and so called roster arrays labeled as `roster[]`. Each item of `roster[]` corresponds to one shift. Each shift can be represented by its `uint` index consuming 4 B. Nevertheless, the number of shifts $|S|$ is bounded in the NRRP instance and the shifts can be binary encoded such that $\mathcal{F} \mapsto 00, \mathcal{E} \mapsto 01, \mathcal{L} \mapsto 10$ and $\mathcal{N} \mapsto 11$. Consequently, one is able to compress the roster arrays c times, where $c \leftarrow \lfloor 32/\lceil \log_2 |S| \rceil \rfloor$. In our case $c = 16$, since 16 consecutive shifts can be compressed to 4 B. If you focus on Table 4.1, to

Table 4.1: The overview of the memory model for the homogeneous model of the parallel algorithm

Memory	**Variable**	**Type**	**Size** [Bytes]	**Purpose**
constant	$\|E\|_{max}$	`uint`	4	max{$\|E\|$ of all NRRP instances launched in one batch on the GPU}
	m	`uint`	4	total number of the instances of the algorithm
	m_b	`uint`	4	number of the instances of the algorithm per one CUDA block
global	pen	`uint`	$4 \cdot m \cdot \|E\|$	penalization used to evaluate a shift assignment
	$unassigned$	`uint`	$4 \cdot m \cdot \|E\|$	vector of roster positions that have not been assigned by Rule 1
	$Z(\tilde{R}^{best})$	`uint`	4	best value of the objective function
	$\tilde{R}^{best}$	`roster[]`	$4 \cdot \|E\| \cdot \|D\|/c$	best modified roster found
	RPP^{best}	`uint`	$4 \cdot \|E\| \cdot \|D\|$	list of the roster positions which corresponds to $\tilde{R}^{best}$
	$lock^{best}$	`bool`	4	lock protecting Z^{best}
	run	`uint`	4	counter of runs, which have been started
	$maxRuns$	`uint`	4	maximal number of runs
	$firstRun$	`bool`	$4 \cdot m$	flag to initialize the first run of each instance of the algorithm
	$terminateAlg$	`uint`	4	flag to terminate the execution of the algorithm
	RS	`uint`	$4 \cdot \|S\| \cdot \|D\|$	matrix expressing the number of requested shifts per day
	$applyLS$	`bool`	4	flag to apply the local search around the solution based on RPP^{best}
	$runsNoSuccess$	`uint`	4	number of the runs without improvement of Z^{best}
	p_{LS}	`float`	4	value of the probability to swap two items in the RPP list
	$runsOfSameProb$	`uint`	4	number of the runs with probability p_{LS}
shared	$applyBT$	`bool`	$4 \cdot m_b$	flag whether the backtrack will be applied or not
	$backtrack$	`uint`	$4 \cdot m_b$	counter of made backtracks
	$maxBacktracks$	`uint`	4	maximal number of backtracks
	Z	`uint`	$4 \cdot m_b$	objective function
	$feasible$	`bool`	$4 \cdot m_b$	feasibility flag
	$firstRule$	`bool`	$4 \cdot m_b$	flag determining whether shifts are assigned according to Rule 1 or other rules
	RPP	`uint`	$4 \cdot m_b \cdot \|E\|_{max} \cdot \|D\|$	lists of partial roster positions
	i	`uint`	$4 \cdot m_b$	position in RPP
	s	`uint`	$4 \cdot m_b \cdot \|E\|_{max}$	shift to be assigned (shift from R^0 at the roster position given by $RPP_{e,i}$)
	$unassignedRP$	`uint`	$4 \cdot m_b$	one unassigned roster position (e, d) by Rule 1
	$\tilde{R}$	`roster[]`	$4 \cdot m_b \cdot \|E\|_{max} \cdot \|D\|/c$	modified roster
	$isOccupied$	`bool[]`	$m_b \cdot \|E\|_{max} \cdot \|D\|/8$	binary array determining which cell is occupied in the roster
	R^0	`roster[]`	$4 \cdot \|E\|_{max} \cdot \|D\|/c$	original roster
	R_{prev}	`roster[]`	$4 \cdot \|E\|_{max}/c$	end of the previous original roster
	A	`bool[]`	$\|E\|_{max} \cdot \|D\|/8$	binary array of absences
	$minDaysOff$	`uint`	$4 \cdot \|E\|_{max}$	minimal number of the days-off in each (random) 7 consecutive days
local and registers	$maxRuns$	`uint`	4	maximal number of the runs
	$\|E\|$	`uint`	4	number of the employees in the NRRP instance
	$\|D\|$	`uint`	4	number of the days in the NRRP instance
	e	`uint`	4	index of the employee inside 1 instance of the algorithm
	q	`uint`	4	index of the instance of the algorithm launched in 1 CUDA block

the region of the shared memory, specifically to the size of the variables, you can simply distinguish what is shared over all instances of the algorithm in the CUDA block and what is specific for each one. In the second case, the size is a multiplied by m_b (the number of the instances of the algorithm per one CUDA block).

4.5.5.1 Critical Sections

In the case of the parallel design, one has to consider the parallel access to the shared data which should be accessed exclusively only by 1 thread in order to modify the global data maintained over all instances of the algorithm, e.g., $\tilde{R}^{best}$. The instructions which need a mechanism for protecting the access to the shared resource via locks are marked in Algorithm 3 by the symbol 🔒. The first one is the instruction for updating the best NRRP solution (line 35). The second one is the instruction for incrementing the total runs counter *run* (line 38), where the access is controlled via atomic operations provided by the CUDA framework itself.

4.6 A Heterogeneous Model of the Parallel Algorithm

This section describes the design of the heterogeneous model of the parallel algorithm as an alternative to the homogeneous one. In the case of the heterogeneous model, one has to decide which part of the algorithm will be executed in a parallel way on the GPU and which will stay on the CPU. In our case, **Part B** was moved to GPU, since it consumes 76 % of the total computing time. Therefore, the upper bound of the speedup is $100/(100 - 76) \sim 4.17$ times, if zero computational time of the GPU execution is assumed. In order to compare the heterogeneous model of the parallel algorithm to the homogeneous one (see Algorithm 3), the complete heterogeneous pseudo-code is stated in Appendix C.

The biggest bottleneck of this model from the speedup point of view is the communication between the host and the device. The basic idea how to minimize the communication in our algorithm is to perform it in a batch way. The rosters prepared for the evaluation by **Part A** are accumulated on the host as long as the maximal size of data is not reached. Afterwards, the data is copied from the CPU memory space to the GPU memory space and the kernel of **Part B** is launched. When the parallel evaluation is finished, the resulting *pen* for all shift assignments are copied back to the host and **Part C** assigns the shifts to $\tilde{R}$.

4.7 Experiments & Evaluation

4.7.1 Experimental Setup

The experiments were performed on a PC with the AMD Phenom II X4 945, GHz, 8 GB of RAM and on the GPU device NVIDIA® GTX 650 Ti. Nevertheless, this is just a common GPU for playing games equipped by 768 cores and 1 GB of global memory (a full specification of this GPU is in (NVIDIA Corporation (2014))). The algorithm was developed in the CUDA framework version 5.5 in the Microsoft Visual Studio 2010.

The performance of our approach was verified from the speedup and the quality point of view on two datasets (Pato and Moz (2013)). Dataset D19 contains 32 instances with 19 nurses, while dataset D32 consists of 36 instances with 32 nurses. These datasets are consistent with the NRRP problem statement described in Section 4.3. The number of absences in the particular NRRP instances differs from 1 up to 59. Moreover, the NRRP instances are organized into groups indicated by Roman numerals in their names. The Roman numeral corresponds to the number of weeks from the end of the planning horizon which can be modified, e.g., instance II.5_19 has the absences in the third and the fourth week, i.e., the two weeks can be modified with respect to constraint (c_5); instance III.2_32 has the absences in the second, the third and the fourth week, etc. In addition, dataset D32 contains one extra group of instances V.1_32 – V.4_32 that have the larger set of absences.

To eliminate the randomness of the algorithms given by the `generatePartialRosterPositions` function, all of the following experiments were evaluated over 50 *samples*. One sample is performed by the execution of all algorithms that have the following parameters: $maxRuns = 100 \cdot 10^3, maxBacktracks = 5 \cdot 10^3$. The parameters relevant for the parallel algorithms were set experimentally to $m = 1200, m_b = 2$.

4.7.2 Tuning the Memory Model

To determine the most efficient version of the parallel algorithm from the *speedup* point of view that was used for the final experiments, the memory model was tuned. The speedup is defined as the ratio of the time consumed by the modified sequential algorithm (Figure 4.4b) to the time consumed by the parallel algorithm, either for the homogeneous model (Figure 4.4c) or for the heterogeneous model (Figure 4.4d), i.e., $speedup_{hom} = t_{seq}/t_{hom}$ and $speedup_{het} = t_{seq}/t_{het}$.

The experiments depicted in Figure 4.7 show the relationship between the changes of the memory model and the gained speedup. The speedup of each memory model was compared relatively to the final memory model (described in Section 4.5.5, which is illustrated in figures by *black*). This memory model

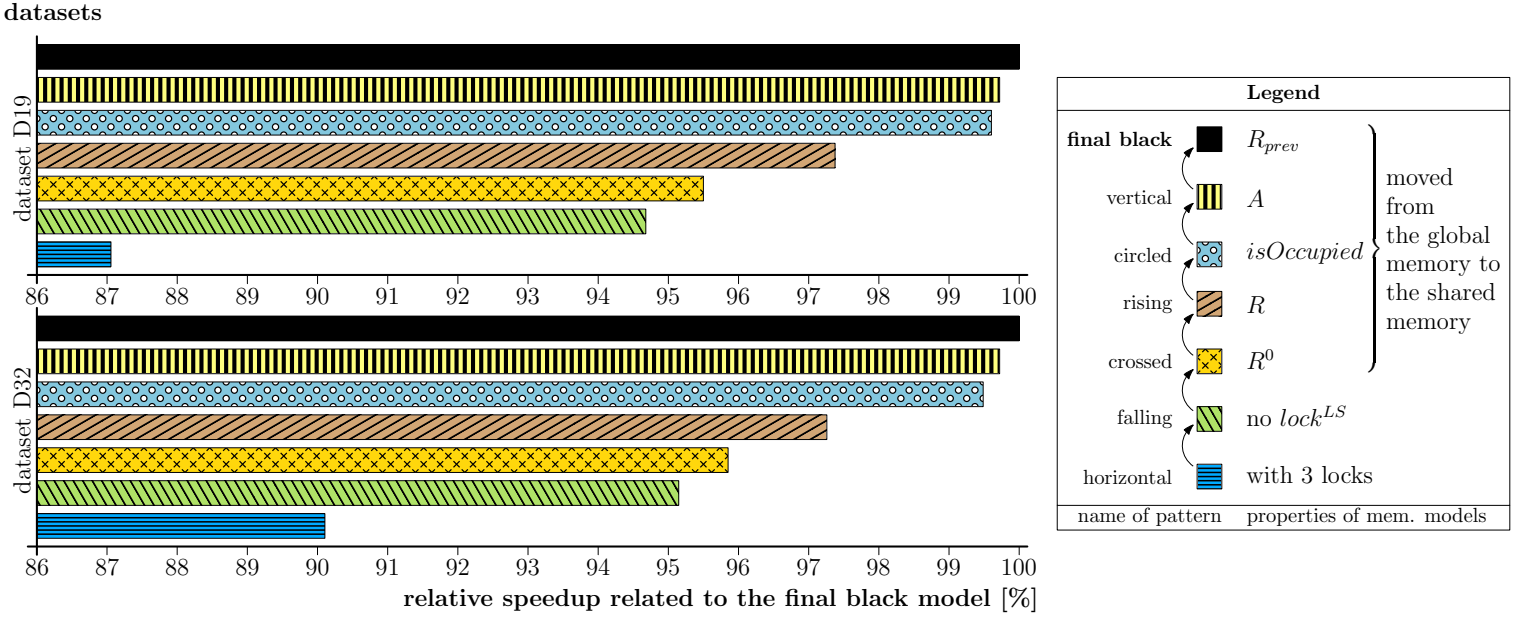


Figure 4.7: The memory model tunning for the homogeneous model

was the best one for the homogeneous and the heterogeneous model for both datasets. Seven different memory models presented in Figure 4.7 were compared such that the results are averaged over the NRRP instances belonging to the particular dataset. Each memory model was made from the previous one (illustrated by arrows between patterns in the legend), e.g., the *falling* one was based on the *horizontal* one, etc. Our first memory model (depicted by *horizontal* pattern) considered three locks. The first one controls the access to the best solution (line 35 in Algorithm 3), the second one protects the incrementation of *run* (line 38), while the third one, called as $lock^{LS}$ (line 37), was used to correctly update the local search parameters, e.g., $runsNoSuccess$. However, this line of the code in the algorithm is not marked by the symbol 🔒 in Algorithm 3, since $lock^{LS}$ was omitted in all the memory models except the *horizontal* one. This is still acceptable, since in the worst case the local search will be not switched on and off after an exact number of runs, but approximately after a given number of runs due to the concurrent access of the instances of the parallel algorithm. One can notice that this change leads to a very significant increase of the speedup in the *falling* memory model (approximately by 5 % without any influence to the quality of the solutions). The next 5 % was gained by moving the variables from the global memory to the shared memory. Namely, R^0 in the *cross* model, $\tilde{R}$ in the *rising* model, *isOccupied* in the *circle* model, A in the *vertical* model and at last, R_{prev} in the *final black* model.

This final memory model described in Section 4.5.5 is used in Section 4.7.3 and Sec.4.7.4.

4.7.3 Speedup Evaluation

These experiments were performed in order to prove the expected speedup of the parallel algorithms. The results are summarized in Table 4.2 (dataset D19) and Table 4.3 (dataset D32), where the rows correspond to the particular NRRP instances evaluated by 50 samples. For each NRRP instance, the computational times t_{seq}, t_{hom} and t_{het} provided by the modified sequential CPU algorithm (Figure 4.4b), the homogeneous parallel GPU algorithm (Figure 4.4c) and the heterogeneous parallel GPU algorithm (Figure 4.4d) were compared, since all of them are based on the same code presented by Algorithm 3. Consequently, the derived speedups $speedup_{hom}$ and $speedup_{het}$ are presented on the right in the tables.

One can notice some relationships among the particular results. Firstly, for the computational times, it holds that $t_{seq} > t_{het} > t_{hom}$ and, therefore, $speedup_{hom} > speedup_{het}$, as we expected. Namely, dataset D19 was accelerated on average by $speedup_{hom} = 12.71$ and $speedup_{het} = 2.36$, while dataset D32 achieves $speedup_{hom} = 17.70$ and $speedup_{het} = 2.26$. Secondly, one can

Table 4.2: The comparison of computational times and speedups – dataset D19, 50 samples per instance

dataset D19	Figure 4.4b	Figure 4.4c	Figure 4.4d	speedups	
NRRP instance	**sequential** CPU t_{seq} [s]	**homogeneous** GPU t_{hom} [s]	**heterogeneous** GPU t_{het} [s]	$speedup_{hom}$ t_{seq}/t_{hom} [$-$]	$speedup_{het}$ t_{seq}/t_{het} [$-$]
I.1_19	3.42	0.44	3.34	7.75	1.02
I.2_19	7.37	0.75	8.17	9.78	0.90
I.3_19	6.11	0.71	3.71	8.62	1.65
I.4_19	7.04	0.61	6.62	11.48	1.06
I.5_19	5.81	0.68	3.22	8.57	1.80
I.6_19	6.26	0.63	3.26	10.01	1.92
I.7_19	6.71	0.70	3.30	9.54	2.03
I.8_19	7.12	0.62	4.99	11.46	1.43
II.1_19	7.84	0.61	6.00	12.81	1.31
II.2_19	9.50	0.67	5.96	14.09	1.59
II.3_19	7.00	0.57	3.64	12.18	1.92
II.4_19	9.02	0.71	3.73	12.77	2.42
II.5_19	7.40	0.65	3.63	11.33	2.04
II.6_19	10.26	0.93	4.09	11.05	2.51
II.7_19	7.98	0.79	3.48	10.10	2.30
II.8_19	7.61	0.53	3.39	14.25	2.25
III.1_19	9.37	0.66	3.63	14.22	2.58
III.2_19	8.13	0.66	3.10	12.36	2.62
III.3_19	8.68	0.61	3.39	14.16	2.56
III.4_19	12.62	0.77	4.53	16.29	2.79
III.5_19	13.43	0.96	4.51	14.01	2.98
III.6_19	11.55	1.20	3.98	9.64	2.90
III.7_19	10.20	0.76	3.83	13.42	2.66
III.8_19	11.84	0.78	4.02	15.23	2.95
IV.1_19	10.76	0.61	3.40	17.51	3.16
IV.2_19	10.76	0.64	3.35	16.70	3.21
IV.3_19	11.51	0.83	3.60	13.83	3.20
IV.4_19	11.92	0.83	4.09	14.39	2.92
IV.5_19	15.75	1.05	4.80	15.07	3.28
IV.6_19	11.42	0.84	4.07	13.59	2.81
IV.7_19	15.22	1.09	4.52	13.91	3.37
IV.8_19	15.55	0.93	4.56	16.75	3.41
avg. of D19	9.54	0.75	4.19	12.71	2.36
st.dev. of D19	2.96	0.17	1.11	2.54	0.72

Table 4.3: The comparison of computational times and speedups – dataset D32, 50 samples per instance

dataset D32	Figure 4.4b	Figure 4.4c	Figure 4.4d	speedups	
NRRP instance	**sequential** CPU t_{seq} [s]	**homogeneous** GPU t_{hom} [s]	**heterogeneous** GPU t_{het} [s]	$speedup_{hom}$ t_{seq}/t_{hom} [$-$]	$speedup_{het}$ t_{seq}/t_{het} [$-$]
I.1_32	13.88	1.11	13.11	12.54	1.06
I.2_32	14.77	1.12	13.52	13.16	1.09
I.3_32	18.79	1.35	14.24	13.88	1.32
I.4_32	16.51	1.03	12.89	16.00	1.28
I.5_32	19.51	1.46	13.92	13.37	1.40
I.6_32	15.94	1.40	9.97	11.36	1.60
I.7_32	15.52	1.30	11.64	11.92	1.33
I.8_32	15.81	1.32	11.78	11.95	1.34
II.1_32	18.95	1.11	12.29	17.07	1.54
II.2_32	19.58	1.11	12.12	17.58	1.62
II.3_32	21.85	1.26	12.65	17.38	1.73
II.4_32	28.12	1.57	14.20	17.92	1.98
II.5_32	28.41	1.91	13.23	14.87	2.15
II.6_32	25.45	1.92	9.41	13.25	2.70
II.7_32	23.26	1.42	12.66	16.43	1.84
II.8_32	24.89	1.41	12.84	17.62	1.94
III.1_32	27.66	1.45	12.95	19.08	2.13
III.2_32	27.73	1.42	12.49	19.55	2.22
III.3_32	30.74	1.60	13.20	19.20	2.33
III.4_32	31.74	1.53	13.55	20.80	2.34
III.5_32	36.33	2.15	14.63	16.88	2.48
III.6_32	37.35	2.31	10.42	16.19	3.58
III.7_32	37.72	2.08	14.72	18.17	2.56
III.8_32	36.67	2.30	12.98	15.93	2.83
IV.1_32	32.97	1.42	12.76	23.23	2.58
IV.2_32	33.71	1.51	13.21	22.31	2.55
IV.3_32	34.18	1.48	13.00	23.03	2.63
IV.4_32	39.96	1.91	13.41	20.97	2.98
IV.5_32	39.15	1.93	14.48	20.26	2.70
IV.6_32	30.16	1.28	12.35	23.55	2.44
IV.7_32	39.17	1.87	14.38	20.98	2.72
IV.8_32	48.74	2.55	16.89	19.12	2.88
V.1_32	30.14	1.30	10.05	23.23	3.00
V.2_32	30.15	1.47	8.45	20.55	3.57
V.3_32	32.93	1.65	9.04	19.92	3.64
V.4_32	46.24	2.57	14.95	17.98	3.09
avg. of D32	28.46	1.60	12.73	17.70	2.26
st.dev. of D32	9.09	0.41	1.75	3.46	0.71

observe that $speedup_{het}$ is approximately the same for both datasets. This is caused by the main bottleneck of the heterogeneous model, in other words, the communication between the host and the device. However, $speedup_{hom}$ of dataset D32 is on average higher than $speedup_{hom}$ of dataset D19, since the bigger NRRP instance we have for our algorithm, the higher the speedup can be expected in comparison to the sequential algorithm. The last observation is that there is a correlation between the NRRP instances and the achieved speedup for both datasets. One can see that the speedup is growing with the number of weeks that are modified (see the description of the datasets in Section 4.7.1).

4.7.4 Quality Evaluation

This section is focused on evaluating the quality of the solutions produced by our algorithms, specifically the parallel one. For this purpose, the values of the objective function were compared in Table 4.4 (dataset D19) and Table 4.5 (dataset D32). Each row in the tables contains the results for one NRRP instance arranged from left to right as follows – the optimal value of Z given by ILP (Moz and Pato (2007)), the results of the best sequential algorithm from (Moz and Pato (2007)) (labeled as HH_{PMX}) and our results. Namely, the results of the original sequential algorithm implemented by ourselves in order to eliminate the influence of the used hardware (see Figure 4.4a), the modified sequential algorithm (see Figure 4.4b) and the homogeneous and the heterogeneous model of the parallel algorithm (corresponds to Figure 4.4c and Figure 4.4d). The results are given by a pair of values – the value of the objective function Z and the computational time needed to achieve this quality. These results are summarized in the footer of the tables such that # feasible (# optimal) stands for the total number of feasible (optimal) solutions of instances over the entire dataset.

In order to make a fair comparison from the CPUs point of view, each sequential algorithm had more computational time than the parallel one. Namely, t_{seq} was determined for each NRRP instance such that $t_{seq} = t_{hom} \cdot speedup_{hom} = t_{het} \cdot speedup_{het}$. The experiments proved that the algorithm reorganization made in the modified sequential algorithm (see Figure 4.4b) has no influence on the quality of the results in comparison to the performance of the original sequential algorithm (see Figure 4.4a). Furthermore, the values of the objective function Z are almost the same for all our algorithms (Figure 4.4a – Figure 4.4d) evaluated on both datasets. There are some small differences caused by the randomness of the algorithm, which were eliminated by the evaluation over 50 samples.

You can observe the huge difference in the computational times among the sequential algorithms (HH_{PMX} from (Moz and Pato (2007)) and Figure 4.4a,

Table 4.4: The comparison of the objective function values – dataset D19, 50 samples per instance

dataset D19	(Moz and Pato (2007))			Figure 4.4a		Figure 4.4b		Figure 4.4c		Figure 4.4d	
NRRP instance	CPU ILP $Z[-]$	CPU HH_{PMX} $Z[-]$	$t[s]$	CPU **sequential** $Z[-]$	$t_{seq}[s]$	CPU **sequential** $Z[-]$	$t_{seq}[s]$	GPU **homogeneous** $Z[-]$	$t_{hom}[s]$	GPU **heterogeneous** $Z[-]$	$t_{het}[s]$
I.1_19	3	3	163.50	3	3.56	3	3.42	3	0.44	3	3.34
I.2_19	2	2	186.66	2	7.80	2	7.37	2	0.75	2	8.17
I.3_19	9	9	279.98	9	6.60	9	6.11	9	0.71	9	3.71
I.4_19	2	2	355.70	2	7.20	2	7.04	2	0.61	2	6.62
I.5_19	20	20	453.79	20	6.01	20	5.81	20	0.68	20	3.22
I.6_19	8	9	380.75	8	6.60	8	6.26	8	0.63	8	3.26
I.7_19	20	21	717.40	21	7.32	23	6.71	22	0.70	∞	3.30
I.8_19	2	2	354.50	2	7.07	2	7.12	2	0.62	2	4.99
II.1_19	1	1	420.24	1	8.40	1	7.84	1	0.61	1	6.00
II.2_19	0	0	384.42	0	9.60	0	9.50	0	0.67	0	5.96
II.3_19	5	5	485.90	5	7.20	5	7.00	5	0.57	5	3.64
II.4_19	12	12	644.55	12	9.00	12	9.02	12	0.71	12	3.73
II.5_19	6	6	437.51	6	7.80	6	7.40	6	0.65	6	3.63
II.6_19	17	17	624.03	17	10.80	17	10.26	17	0.93	17	4.09
II.7_19	∞	∞	2199.36	∞	8.61	∞	7.98	∞	0.79	∞	3.48
II.8_19	5	5	608.02	5	7.59	5	7.61	5	0.53	5	3.39
III.1_19	7	7	729.30	7	9.60	7	9.37	7	0.66	7	3.63
III.2_19	12	12	1002.30	13	8.40	12	8.13	12	0.66	13	3.10
III.3_19	13	13	1138.48	14	9.00	14	8.68	14	0.61	14	3.39
III.4_19	7	7	917.48	7	12.59	7	12.62	7	0.77	7	4.53
III.5_19	27	28	1570.93	33	13.82	30	13.43	29	0.96	34	4.51
III.6_19	27	28	1729.36	27	12.04	29	11.55	29	1.20	35	3.98
III.7_19	19	19	912.13	20	10.76	19	10.20	19	0.76	19	3.83
III.8_19	11	11	891.56	11	11.99	11	11.84	11	0.78	11	4.02
IV.1_19	9	9	1044.69	9	11.39	9	10.76	9	0.61	9	3.40
IV.2_19	12	12	1140.94	12	10.80	12	10.76	12	0.64	12	3.35
IV.3_19	10	10	1197.91	10	12.00	10	11.51	10	0.83	10	3.60
IV.4_19	34	37	2528.40	38	12.06	39	11.92	37	0.83	∞	4.09
IV.5_19	18	18	1485.12	19	16.14	19	15.75	20	1.05	23	4.80
IV.6_19	23	25	2509.28	27	11.42	25	11.42	25	0.84	30	4.07
IV.7_19	9	9	1271.72	9	15.57	9	15.22	9	1.09	9	4.52
IV.8_19	10	10	1307.64	10	15.60	10	15.55	10	0.93	10	4.56
# feasible	31	31	-	31	-	31	-	31	-	29	-
# optimal	31	25	-	23	-	24	-	24	-	21	-

Table 4.5: The comparison of the objective function values – dataset D32, 50 samples per instance

dataset D32	(Moz and Pato (2007))			Figure 4.4a		Figure 4.4b		Figure 4.4c		Figure 4.4d	
NRRP instance	CPU ILP $Z[-]$	CPU HH_{PMX} $Z[-]$	$t[s]$	CPU **sequential** $Z[-]$	$t_{seq}[s]$	CPU **sequential** $Z[-]$	$t_{seq}[s]$	GPU **homogeneous** $Z[-]$	$t_{hom}[s]$	GPU **heterogeneous** $Z[-]$	$t_{het}[s]$
I.1_32	5	5	210.74	5	14.23	5	13.88	5	1.11	5	13.11
I.2_32	5	5	241.89	5	15.60	5	14.77	5	1.12	5	13.52
I.3_32	8	8	372.27	8	19.20	8	18.79	8	1.35	8	14.24
I.4_32	3	3	499.36	3	16.80	3	16.51	3	1.03	3	12.89
I.5_32	10	10	376	10	19.80	10	19.51	10	1.46	10	13.92
I.6_32	15	15	348.78	16	16.20	15	15.94	15	1.40	15	9.97
I.7_32	9	9	261.49	9	15.60	9	15.52	9	1.30	9	11.64
I.8_32	10	10	264.32	10	16.20	10	15.81	10	1.32	10	11.78
II.1_32	5	5	626.37	5	19.20	5	18.95	5	1.11	5	12.29
II.2_32	5	5	633.84	5	19.80	5	19.58	5	1.11	5	12.12
II.3_32	7	7	707.69	7	22.20	7	21.85	7	1.26	7	12.65
II.4_32	13	13	921.69	13	28.20	13	28.12	13	1.57	13	14.20
II.5_32	20	20	900.99	20	28.80	20	28.41	20	1.91	20	13.23
II.6_32	24	24	725.85	24	25.80	24	25.45	24	1.92	24	9.41
II.7_32	10	10	782.62	10	23.39	10	23.26	10	1.42	10	12.66
II.8_32	11	11	923.21	11	25.20	11	24.89	11	1.41	11	12.84
III.1_32	13	13	1122.19	13	27.60	13	27.66	13	1.45	13	12.95
III.2_32	13	13	1183.45	13	28.20	13	27.73	13	1.42	13	12.49
III.3_32	15	15	1250.79	15	31.20	15	30.74	15	1.60	15	13.20
III.4_32	14	14	1390.03	14	31.80	14	31.74	14	1.53	14	13.55
III.5_32	25	25	1177.45	25	36.00	25	36.33	25	2.15	25	14.63
III.6_32	44	44	1499.19	45	37.81	45	37.35	47	2.31	53	10.42
III.7_32	25	25	1601.16	25	37.79	25	37.72	25	2.08	25	14.72
III.8_32	30	30	1169.97	30	36.60	30	36.67	30	2.30	31	12.98
IV.1_32	14	14	1613.55	14	33.00	14	32.97	14	1.42	14	12.76
IV.2_32	15	15	1701.5	15	34.20	15	33.71	15	1.51	15	13.21
IV.3_32	14	14	1785.02	14	34.20	14	34.18	14	1.48	14	13.00
IV.4_32	20	20	2049.01	20	40.80	20	39.96	20	1.91	20	13.41
IV.5_32	21	21	1726.8	21	39.00	21	39.15	21	1.93	21	14.48
IV.6_32	11	11	1620.95	11	30.60	11	30.16	11	1.28	11	12.35
IV.7_32	20	20	1786.5	20	39.60	20	39.17	20	1.87	20	14.38
IV.8_32	30	30	1928.47	30	48.60	30	48.74	30	2.55	30	16.89
V.1_32	14	14	2071.26	14	30.00	14	30.14	14	1.30	14	10.05
V.2_32	27	27	2278.8	27	31.20	27	30.15	27	1.47	27	8.45
V.3_32	28	28	2281.24	28	33.00	28	32.93	28	1.65	28	9.04
V.4_32	117	128	7397.17	133	47.57	∞	46.24	127	2.57	∞	14.95
# feasible	36	36	-	36	-	35	-	36	-	35	-
# optimal	36	35	-	33	-	33	-	34	-	32	-

Figure 4.4b), which achieved practically the same results in terms of the quality. The sequential algorithm HH_{PMX} from (Moz and Pato (2007)) had only 20 runs, however, it was extended by a genetic algorithm, where each individual of a population can correspond to one of our instances of the algorithm. Moreover, the population consisting of 400 individuals was evolved over 2000 generations and this is the main reason of the higher computational time consumed by HH_{PMX}.

Finally, it flows from the comparison of the sequential and the parallel algorithms that their quality is on average the same and is achieved within different computational times according to the speedup presented in the previous subsection.

4.8 Conclusion

This work provides, up to our knowledge, the first parallel approach solving the Nurse Rerostering Problem (NRRP). The basic ideas, principles and limitations used to design the parallel algorithm on GPU for NRRP are explained. Furthermore, we describe two models (the homogeneous and the heterogeneous one) of the parallel algorithm and the differences in their design. Our algorithms were evaluated from the speedup and the quality point of view on the NRRP benchmark instances (Pato and Moz (2013)). The experiments show that we are able to achieve the same quality of the results within significantly shorter computational time in comparison to the same sequential algorithm. Namely in the case of the homogeneous model of the parallel algorithm, the NRRP instances with 19 nurses were accelerated almost 13 times in average, while the instances having 32 nurses were solved almost 18 times faster.

Looking to the future, one can expect the rapid progress in the GPUs performance compared to the performance of the common CPUs limited by their architecture. The authors of (Brodtkorb et al. (2013b)) concluded their survey focused on GPU computing in OR that it is clear that GPUs will play an important role in all of computational science in the nearest future and it is important to consider how to utilize these new kinds of architectures. Nevertheless, it is still not common to apply GPUs on the OR problems, which was a main motivation for us to publish this work.

Chapter 5

Conclusion

This chapter is summarizing the achieved contributions of the thesis.

5.1 Achieved Contributions

Firstly, the state of the art related to employee timetabling was presented. Consequently, objectives of the thesis were set to significant gaps revealed from the literature. Namely, two combinatorial problems were addressed in the thesis. The first one is so called Employee Timetabling Problem with a High Diversity of shifts (ETPHD), while the second one is focused on the Nurse Rerostering Problem (NRRP).

The first contribution related to ETPHD is that this problem is rigorously described by the ILP model. However, the problem complexity does not allow us to apply exact methods to solve it and, therefore, we designed a multistage approach (MSA) able to solve large instances of this problem. Its success is mainly given by a proposed problem transformation, which is based on the mapping of the shifts into the groups of shifts. Consequently, the rough positions of assigned shifts are determined by the first stage, while the second stage is represented by the inverse transformation based on network flows. These first two stages provide the initial solution of the problem. Finally, a Tabu Search algorithm is performed in the last stage to improve the quality of the solution.

In order to evaluate MSA by comparing to other approaches, we proposed a cross evaluation methodology (see Section 3.6.4). It is based on cross applying all the considered approaches on different ETPs and comparing their results. Namely, ETPHD was presented by 30 real life instances from an airport company and, furthermore, 5 standard benchmark instances (ASAP (2013)) of the Nurse Rostering Problem (NRP) were evaluated. The cross evaluation methodology was used, since there were no benchmark instances of ETPHD and it confirmed that the approaches providing the initial solution based on

the transformation perform better or equal solutions in 12 out of 14 cases (see Table 3.10). One case corresponded to a comparison of the results of two different approaches applied on all the instances of one combinatorial problem within the given time limit repeated 30 times to eliminate the randomness of the approaches.

In the case of NRRP, a parallel algorithm that helps to accelerate NRRP solution was designed. Furthermore, it exploits not so common hardware device used for the solution of combinatorial problems - a Graphics Processing Unit (GPU). To the best of our knowledge, this is the first parallel algorithm for NRRP and the first parallel algorithm for ETPs performed on GPU. The thesis brings the detailed description of all the issues to be solved in order to achieve the final speedup of the parallel algorithm in comparison to the sequential version of the algorithm executed on Central Processing Unit (CPU). Moreover, two models of the parallel algorithm were compared, the homogeneous and the heterogeneous one. The homogeneous one is completely executed on GPU, while the heterogeneous one is executed on GPU (the computationally intensive parts of the algorithm) and on CPU as well (the rest of the algorithm). The experiments were elaborated on 68 real life standard benchmark NRRP instances (Pato and Moz (2013)) tested 50 times per instance. These experiments verified that the quality of the results provided by the parallel algorithm is comparable to the sequential ones and to the best known ones. In terms of the speedup, the heterogeneous model of the parallel algorithm was in average 2.3 times faster in comparison to the sequential one. However, the speedup of the homogeneous model of the parallel algorithm was more significant, since the results of the same quality were provided in 15 times shorter computational time in average.

In general, the results of the research made are summarized by the list of publications at the end of the thesis. To the most outstanding publications belong impacted journal papers. Namely, (Bäumelt et al. (2014)) was accepted in the Computers & Operations Research, where the proposed approach dealing with ETPs from Chapter 3 is described. Furthermore, the publication (Bäumelt et al. (2015)) related to the proposed parallel algorithm from Chapter 4 is in major revision to be submitted to European Journal of Operational Research.

5.2 Fulfillment of Stated Goals and Objectives

The fulfillment of the stated goals and the objectives is revised below.

1. The goal was satisfied in Chapter 2. The basic terms, the classification and the categorization of the employee timetabling within the context of the operations research was described. Consequently, other goals of

the thesis were targeted to the significant gaps of the domain that reveal from the presented state of the art review.

2. This goal was achieved in Chapter 3. Namely, the multistage approach based on a transformation of ETPHD was proposed, where the problem solution was decomposed into three stages. The first one works with the transformed problem in order to determine a rough position of shifts. The second one contains an inverse transformation that assigns shifts to employees. The last stage improves the quality of the final timetable. The experimental part showed that approaches based on our transformation, e.g., MSA, outperformed other approaches in the most of the cases (see details in goal 4 below).

3. This goal was met by Chapter 4. Namely, the parallel algorithm for the Graphics Processing Unit solving the Nurse Rerostering Problem was designed. The issues solved in order to gain the speedup of the algorithms are described and their influence on the quality of the solutions and the speedup is discussed. Moreover, two different models of the parallel algorithm, i.e., the homogeneous and the heterogeneous one, were compared and their advantages and drawbacks were explained. The significant speedup to the sequential version of the algorithm was attained by the homogeneous model, the computational times were 15 times shorter in average.

4. This goal was filled by the experimental parts of the both handled problems (see Section 3.6 for ETPHD and Section 4.7 for NRRP).

   To evaluate approaches applied on ETPHD, the cross evaluation methodology (Section 3.6.4) was introduced. It was used to verify the contribution of the implemented multistage approach MSA based on a transformation of ETPHD. The experiments were tested on 30 real life ETPHD benchmark instances (see Table 3.6) and on 5 standard NRP benchmark instances (ASAP (2013)) as well. The experiments demonstrated that the approaches having this transformation included were better or equal to other approaches in 12 out of 14 cases in terms of the quality of the solutions. One case responses to a comparison of the results of two different approaches applied on all the instances of one combinatorial problem within the given time limit repeated 30 times to eliminate the randomness of the approaches. Finally, MSA is being used to solve ETPHDs in an airport company.

   To evaluate the parallel algorithm dealing with NRRP, the standard

NRRP benchmark set (Pato and Moz (2013)) containing the real life instances from the hospital was used. This set consists of two datasets having 19 and 32 nurses. In order to make a fair comparison, we decided to verify (except the gained speedup) the quality of the solutions provided by the parallel algorithm also. In general, the comparison of the homogeneous (see Section 4.5) and the heterogeneous model (Section 4.6) demonstrates the GPUs power. Their speedups related to the sequential version of the algorithm were in average 15 times (homogeneous) and 2.3 times (heterogeneous) to obtain the NRRP solution with no influence to its quality.

5.3 Concluding Remarks

Conclusions that flow from the thesis and can be further investigated in order to continue contributing to the employee timetabling domain and, generally, to the Operations Research, are pointed out at this place. Firstly, a way how to deal with Employee Timetabling Problems with a High Diversity of shifts more efficiently was showed and we believe that new methods based on ours will appear for this problem in the future. Secondly, the cross evaluation methodology was introduced. It is suitable to evaluate the performance of the designed algorithm fairly, because the results provided by more algorithms applied on more combinatorial problems are compared. The methodology was used, since there were no benchmark instances of ETPHD and, therefore, no results of any other algorithm to be compared to. The methodology can be used by the domain community for other cases having the same characters. Finally, the chapter focused on the design of the parallel algorithm on GPU for the Nurse Rerostering Problem demonstrates that the usage of such a new architectures is very promising and opens wide range of opportunities to accelerate the solution of combinatorial problems by its parallelization. Since the companies producing GPUs concern to the research and the development related to the parallelization very intensively, the rapid progress of GPU performance can be expected in the future, which is very promising for further research.

Bibliography

Ahuja, R.K., Ergun, O., Orlin, J.B., Punnen, A.P., 2002. A survey of very large-scale neighborhood search techniques. Discrete Applied Mathematics 123, 75–102.

Aickelin, U., White, P., 2004. Building better nurse scheduling algorithms. Annals of Operations Research 128, 159–177.

Al-Yakoob, S.M., Sherali, H.D., 2006. A column generation approach for an employee scheduling problem with multiple shifts and work locations. Journal of the Operational Research Society 59, 34–43.

Alfares, H.K., 1998. An efficient two-phase algorithm for cyclic days-off scheduling. Computers & Operations Research 25, 913–923.

Alfares, H.K., 2001. Efficient optimization of cyclic labor days-off scheduling. OR-Spektrum 23, 283–294.

Alfares, H.K., 2004. Survey, categorization, and comparison of recent tour scheduling literature. Annals of Operations Research 127, 145–175.

AMD, 2014. AMD Accelerated Parallel Processing OpenCL Programming Guide. http://developer.amd.com/wordpress/media/2013/07/AMD_Accelerated_Parallel_Processing_OpenCL_Programming_Guide-rev-2.7.pdf. Accessed: 2014-10-01.

Anstreicher, K.M., Brixius, N.W., Goux, J., Linderoth, J., 2002. Solving large quadratic assignment problems on computational grids. Mathematical Programming 91, 563–588.

Applegate, D.L., Bixby, R.E., Chvátal, V., Cook, W.J., 2007. The Traveling Salesman Problem: A Computational Study (Princeton Series in Applied Mathematics).

ASAP, 2013. Automated Scheduling, Optimisation and Planning research group. Staff Rostering Benchmark Data Sets. http://www.cs.nott.ac.uk/~tec/NRP/. Accessed: 2014-10-01.

Aykin, T., 1996. Optimal shift scheduling with multiple break windows. Management Science 42, 591–602.

Azaiez, M.N., Al Sharif, S.S., 2005. A 0-1 goal programming model for nurse scheduling. Computer & Operations Research 32, 491–507.

Baker, K.R., 1976. Workforce allocation in cyclical scheduling problems: A survey.

Bard, J.F., Binici, C., De Silva, A.H., 2003. Staff scheduling at the united states postal service. Computers & Operations Research 30, 745–771.

Bard, J.F., Purnomo, H.W., 2005. Preference scheduling for nurses using column generation. European Journal of Operational Research 164, 510–534.

Bard, J.F., Wan, L., 2006. The task assignment problem for unrestricted movement between workstation groups. Journal of Scheduling 9, 315–341.

Bartsch, T., Drexl, A., Kröger, S., 2006. Scheduling the professional soccer leagues of austria and germany. Computers & Operations Research 33, 1907 – 1937. Special Issue: Operations Research in Sport Special Issue: Operations Research in Sport.

Bäumelt, Z., Dvořák, J., Šůcha, P., Hanzálek, Z., 2015. A Novel Approach for the Nurse Rerostering Problem based on a Parallel Algorithm. European Journal of Operational Research In major revision.

Bäumelt, Z., Šůcha, P., Hanzálek, Z., 2007. Nurse Scheduling Web Application, in: 26th Workshop of the UK Planning and Scheduling Special Interest Group, Charles University, Prague, Czech Republic. pp. 120–123.

Bäumelt, Z., Šůcha, P., Hanzálek, Z., 2014. A multistage approach for an employee timetabling problem with a high diversity of shifts as a solution for a strongly varying workforce demand. Computers & Operations Research 49, 117–129.

Beddoe, G., Petrovic, S., Li, J., 2009. A hybrid metaheuristic case-based reasoning system fornurserostering. Journal of Scheduling 12, 99–119.

Beliën, J., Demeulemeester, E., 2008. A branch-and-price approach for integrating nurse and surgery scheduling. European Journal of Operational Research 189, 652 – 668.

Van den Bergh, J., Beliën, J., De Bruecker, P., Demeulemeester, E., De Boeck, L., 2013. Personnel scheduling: A literature review. European Journal of Operational Research 226, 367–385.

Boyer, V., Baz, D.E., Elkihel, M., 2012. Solving knapsack problems on GPU. Computers & Operations Research 39, 42–47. Special Issue on Knapsack Problems and Applications.

Brodtkorb, A.R., Hagen, T.R., Saetra, M.L., 2013a. Graphics processing unit (GPU) programming strategies and trends in GPU computing. Journal of Parallel and Distributed Computing 73, 4–13.

Brodtkorb, A.R., Hagen, T.R., Schulz, C., Hasle, G., 2013b. GPU computing in discrete optimization. part i: Introduction to the GPU. EURO Journal on Transportation and Logistics 2, 129–157.

Brucker, P., 2007. Scheduling Algorithms. Springer-Verlag New York, Inc.. 5rd edition.

Brucker, P., Qu, R., Burke, E.K., 2011. Personnel scheduling: Models and complexity. European Journal of Operational Research 210, 467–473.

Brunner, J.O., Stolletz, R., 2014. Stabilized branch and price with dynamic parameter updating for discontinuous tour scheduling. Computers & Operations Research 44, 137–145.

Bukata, L., Šůcha, P., 2013. A GPU Algorithm Design for Resource Constrained Project Scheduling Problem, in: Parallel, Distributed and Network-Based Processing (PDP), 2013 21st Euromicro International Conference on, pp. 367–374.

Burke, E., Bykov, Y., Petrovic, S., 2001a. A multicriteria approach to examination timetabling, in: Burke, E., Erben, W. (Eds.), Practice and Theory of Automated Timetabling III. Springer Berlin Heidelberg. volume 2079 of *Lecture Notes in Computer Science*, pp. 118–131.

Burke, E., Kendall, G., Soubeiga, E., 2004a. A tabu-search hyperheuristic for timetabling and rostering. Journal of Heuristics 9, 451–470.

Burke, E.K., Cowling, P., De Causmaecker, P., Vanden Berghe, G., 2001b. A memetic approach to the nurse rostering problem. Applied Intelligence 15, 199–214.

Burke, E.K., Curtois, T., 2012. Staff roster solutions – roster booster.

Burke, E.K., Curtois, T., 2014. New approaches to nurse rostering benchmark instances. European Journal of Operational Research 237, 71–81.

Burke, E.K., Curtois, T., Post, G., Qu, R., Veltman, B., 2008. A hybrid heuristic ordering and variable neighbourhood search for the nurse rostering problem. European Journal of Operational Research 188, 330–341.

Burke, E.K., Curtois, T., Qu, R., Vanden Berghe, G., 2007. A time pre-defined variable depth search for nurse rostering.

Burke, E.K., Curtois, T., Qu, R., Vanden Berghe, G., 2010. A scatter search methodology for the nurse rostering problem. Journal of the Operational Research Society 61, 1667–1679.

Burke, E.K., De Causmaecker, P., Petrovic, S., Vanden Berghe, G., 2006. Metaheuristics for handling time interval coverage constraints in nurse scheduling. Applied Artificial Intelligence 20, 743–766.

Burke, E.K., De Causmaecker, P., Vanden Berghe, G., 1999. A hybrid tabu search algorithm for the nurse rostering problem, in: Selected papers from the Second Asia-Pacific Conference on Simulated Evolution and Learning on Simulated Evolution and Learning, pp. 187–194.

Burke, E.K., De Causmaecker, P., Vanden Berghe, G., Van Landeghem, H., 2004b. The state of the art of nurse rostering. Journal of Scheduling 7, 441–499.

Buyukozkan, K., Sarucan, A., 2014. Applicability of artificial bee colony algorithm for nurse scheduling problems. International Journal of Computational Intelligence Systems 7, 121–136.

Cambazard, H., Demazeau, F., Jussien, N., David, P., 2005. Interactively solving school timetabling problems using extensions of constraint programming, in: Burke, E., Trick, M. (Eds.), Practice and Theory of Automated Timetabling V. Springer Berlin Heidelberg. volume 3616 of *Lecture Notes in Computer Science*, pp. 190–207.

Castiñeiras, I., Sáenz-Pérez, F., 2013. Applying CP(F D), CLP(F D) and CFLP(F D) to a Real-life Employee Timetabling Problem. Procedia Computer Science 18, 531–540. URL http://www.sciencedirect.com/science/article/pii/S1877050913003608, Accessed: 2015-01-10.

Cezik, T., Günlük, O., Luss, H., 2001. An integer programming model for the weekly tour scheduling problem.

Chan, P., Weil, G., 2001. Cyclical staff scheduling using constraint logic programming, in: Burke, E., Erben, W. (Eds.), Practice and Theory of Automated Timetabling III. Springer Berlin Heidelberg. volume 2079 of *Lecture Notes in Computer Science*, pp. 159–175.

Cheang, B., Li, H., Lim, A., Rodrigues, B., 2003. Nurse rostering problems – a bibliographic survey. European Journal of Operational Research 151, 447–460.

Cheng, B.M.W., Lee, J.H.M., Wu, J.C.K., 1997. A nurse rostering system using constraint programming and redundant modeling. Information Technology in Biomedicine, IEEE Transactions on 1, 44–54.

Clark, A.R., Moule, P., Topping, A., Serpell, M., 2012. Rescheduling nursing shifts: scoping the challenge and examining the potential of mathematical model based tools. Journal of Nursing Management .

Cordeau, J.F., Toth, P., Vigo, D., 1998. A survey of optimization models for train routing and scheduling. Transportation Science 32, 380–404.

Czapinski, M., 2013. An effective parallel multistart tabu search for quadratic assignment problem on CUDA platform. Journal of Parallel and Distributed Computing 73, 1461–1468.

Czapinski, M., Barnes, S., 2011. Tabu search with two approaches to parallel flowshop evaluation on CUDA platform. Journal of Parallel and Distributed Computing 71, 802–811. Special Issue on Cloud Computing.

De Castro, L.N., Timmis, J., 2002. Artificial immune systems: A novel paradigm to pattern recognition, in: Artificial Neural Networks in Pattern Recognition, Springer Verlag, University of Paisley, UK. pp. 67–84.

De Causmaecker, P., Vanden Berghe, G., 2010. Towards a reference model for timetabling and rostering. Annals of Operations Research Online, 1–10.

De Causmaecker, P., Vanden Berghe, G., 2011. A categorisation of nurse rostering problems. Journal of Scheduling 14, 3–16.

Elahipanah, M., 2012. Task Scheduling and Activity Assignment to Work Shifts with Schedule Flexibility and Employee Preference Satisfaction. Ph.D. thesis. École Polytechnique de Montréal.

Erdoğan, G., Erkut, E., Ingolfsson, A., Laporte, G., 2010. Scheduling ambulance crews for maximum coverage. Journal of the Operational Research Society 61, 543–550.

Ernst, A.T., Jiang, H., Krishnamoorthy, M., Nott, H., Sier, D., 2001. An Integrated Optimization Model for Train Crew Management. Annals of Operations Research 108, 211–224.

Ernst, A.T., Jiang, H., Krishnamoorthy, M., Sier, D., 2004. Staff scheduling and rostering: A review of applications, methods and models. European Journal of Operational Research 153, 3–27.

Eveborn, P., Flisberg, P., Rönnqvist, M., 2006. Laps Care – an operational system for staff planning of home care. European Journal of Operational Research 171, 962–976.

Freling, R., Lentink, R., Wagelmans, A., 2004. A decision support system for crew planning in passenger transportation using a flexible branch-and-price algorithm. Annals of Operations Research 127, 203–222.

Gans, N., Koole, G., Mandelbaum, A., 2003. Telephone call centers: Tutorial, review, and research prospects. Manufacturing & Service Operations Management 5, 79–141.

Gaspero, L.D., Gärtner, J., Kortsarz, G., Musliu, N., Schaerf, A., Slany, W., 2007. The minimum shift design problem. Annals OR 155, 79–105.

Glover, F., 1977. Heuristics for integer programming using surrogate constraints. Decision Sciences , 156–166.

Glover, F., Laguna, M., 1989. Tabu search – part i. ORSA Journal on Computing , 190–206.

Gopalakrishnan, B., Johnson, E., 2005. Airline crew scheduling: State-of-the-art. Annals of Operations Research 140, 305–337.

Graham, R.L., Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G., 1979. Optimization and approximation in deterministic sequencing and scheduling: a survey. Annals of Discrete Mathematics 5, 287–326.

Green, L.V., Kolesar, P.J., Whitt, W., 2007. Coping with time-varying demand when setting staffing requirements for a service system. Production and Operations Management 16, 13–39.

Günther, M., Nissen, V., 2010. Particle swarm optimization and an agent-based algorithm for a problem of staff scheduling, in: Applications of Evolutionary Computation. Springer Berlin Heidelberg. volume 6025 of *Lecture Notes in Computer Science*, pp. 451–461.

Gutjahr, W.J., Rauner, M.S., 2007. An {ACO} algorithm for a dynamic regional nurse-scheduling problem in austria. Computers & Operations Research 34, 642–666. Logistics of Health Care Management Part Special Issue: Logistics of Health Care Management.

Hadwan, M., Ayob, M., Sabar, N.R., Qu, R., 2013. A harmony search algorithm for nurse rostering problems. Information Sciences 233, 126–140.

Hanne, T., Dornberger, R., Frey, L., 2009. Multiobjective and preference-based decision support for rail crew rostering, in: IEEE Congress on Evolutionary Computation, IEEE. pp. 990–996.

Hao, G., Lai, K., Tan, M., 2004. A neural network application in personnel scheduling. Annals of Operations Research 128, 65–90.

Helber, S., Henken, K., 2007. Profit-oriented shift scheduling of inbound contact centers with skills-based routing, impatient customers, and retrials. Diskussionspapiere der Wirtschaftswissenschaftlichen Fakultät der Leibniz Universität Hannover dp-379. Leibniz Universität Hannover, Wirtschaftswissenschaftliche Fakultät.

Holland, J.H., 1975. Adaptation in Natural and Artificial Systems. University of Michigan Press. Second edition, 1992.

Hung, R., 1995. Hospital nurse scheduling. Journal of Nursing Administration 25, 21–23.

Intel Corporation, 2012. Intel Xeon Phi. http://www.intel.com/content/www/us/en/processors/xeon/xeon-phi-detail.html. Accessed: 2014-10-01.

Isken, M.W., 2004. An implicit tour scheduling model with applications in healthcare. Annals of Operations Research 128, 91–109.

James, T.L., Brown, E.C., Keeling, K.B., 2007. A hybrid grouping genetic algorithm for the cell formation problem. Computers & Operations Research 34, 2059–2079.

Janiak, A., Janiak, W., Lichtenstein, M., 2008. Tabu search on GPU. Journal of Universal Computer Science 14, 2416–2427.

Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P., 1983. Optimization by simulated annealing. SCIENCE 220, 671–680.

Kitada, M., Morizawa, K., 2013. A heuristic method for nurse rerostering problem with a sudden absence for several consecutive days. International Journal of Emerging Technology and Advanced Engineering 3, 353–361.

Kitada, M., Morizawa, K., Nagasawa, H., 2011. A heuristic method for nurse rerostering problem with a sudden absence of nurses, in: Proceedings of the 11th Asia Pacific Industrial Engineering & Management Systems, pp. 1219–1227.

Klabjan, D., Johnson, E.L., Nemhauser, G.L., Gelman, E., Ramaswamy, S., 2001. Solving Large Airline Crew Scheduling Problems: Random Pairing Generation and Strong Branching. Computational Optimization and Applications 20, 73–91.

Klinz, B., Pferschy, U., Schauer, J., 2006. Ilp models for a nurse scheduling problem, in: OR, pp. 319–324.

Komarudin, Guerry, M.A., De Feyter, T., Vanden Berghe, G., 2013. The roster quality staffing problem a methodology for improving the roster quality by modifying the personnel structure. European Journal of Operational Research 230, 551–562.

Kumara, B.T.G.S., Perera, A.A.I., 2011. Automated System For Nurse Scheduling Using Graph Coloring. Indian Journal of Computer Science and Engineering 2, 476–485. URL http://www.ijcse.com/docs/IJCSE11-02-03-089.pdf, Accessed: 2015-01-10.

Ladier, A.L., Alpan, G., Penz, B., 2011. Optimisation séquentielle des emplois du temps dans une plateforme logistique, in: 12th Annual Congress of the French National Society of Operations Research and Decision Science, Saint-Éttiene, France. p. 627. URL http://uma.ensta-paristech.fr/work/labo_work/files/diam/docro/roadef_2011/VERSION-ELECTRONIQUE/roadef2011_submission_449.pdf, Accessed: 2015-01-10.

Landa-Silva, D., Le, K.N., 2008. A simple evolutionary algorithm with self-adaptation for multi-objective nurse scheduling, in: Cotta, C., Sevaux, M., Srensen, K. (Eds.), Adaptive and Multilevel Metaheuristics. Springer Berlin Heidelberg. volume 136 of *Studies in Computational Intelligence*, pp. 133–155.

Lequy, Q., Bouchard, M., Desaulniers, G., Soumis, F., Tachefine, B., 2012a. Assigning multiple activities to work shifts. Journal of Scheduling 15, 239–251.

Lequy, Q., Desaulniers, G., Solomon, M.M., 2012b. A two-stage heuristic for multi-activity and task assignment to work shifts. Computers and Industrial Engineering 63, 831–841.

Lewis, R., 2008. A survey of metaheuristic-based techniques for university timetabling problems. OR Spectrum 30, 167–190.

Lilleby, H.E.S., Schittekat, P., Nordlander, T.E., Hvattum, L.M., Andersson, H., 2012. Competence building with the use of nurse re-rostering, in: Luangpaiboon, P., Moz, M., Dedoussis, V. (Eds.), 4th International Conference on Applied Operational Research, Proceedings, Tadbir Operational Research Group Ltd.. pp. 70–77.

Lučić, P., Teodorović, D., 1999. Simulated annealing for the multi-objective aircrew rostering problem. Transportation Research Part A: Policy and Practice 33, 19 – 45.

Maenhout, B., Vanhoucke, M., 2008. Comparison and hybridization of crossover operators for the nurse scheduling problem. Annals OR 159, 333–353.

Maenhout, B., Vanhoucke, M., 2010a. Branching strategies in a branch-and-price approach foramultiple objective nurse scheduling problem. Journal of Scheduling 13, 77–93.

Maenhout, B., Vanhoucke, M., 2010b. An evolutionary approach for the nurse rerostering problem. Computers & Operations Research In Press, Corrected Proof.

Maenhout, B., Vanhoucke, M., 2010c. A hybrid scatter search heuristic for personalized crew rostering in the airline industry. European Journal of Operational Research 206, 155–167.

Maenhout, B., Vanhoucke, M., 2013a. An artificial immune system based approach for solving the nurse re-rostering problem, in: Middendorf, M., Blum, C. (Eds.), Evolutionary Computation in Combinatorial Optimization. Springer Berlin Heidelberg. volume 7832 of *Lecture Notes in Computer Science*, pp. 97–108.

Maenhout, B., Vanhoucke, M., 2013b. An integrated nurse staffing and scheduling analysis for longer-term nursing staff allocation problems. Omega 41, 485–499.

de Matta, R., Peters, E., 2009. Developing work schedules for an inter-city transit system with multiple driver types and fleet types. European Journal of Operational Research 192, 852–865.

Mezmaz, M., Mehdi, M., Bouvry, P., Melab, N., Talbi, E.G., Tuyttens, D., 2014. Solving the three dimensional quadratic assignment problem on a computational grid. Cluster Computing 17, 205–217.

Mladenović, N., Hansen, P., 1997. Variable neighborhood search. Computer Operations Research 24, 1097–1100.

Moscato, P., 1989. On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts: Towards Memetic Algorithms. Technical Report C3P 826. California Institute of Technology.

Moz, M., Pato, M.V., 2003. An integer multicommodity flow model applied to the rerostering of nurse schedules. Annals OR 119, 285–301.

Moz, M., Pato, M.V., 2004. Solving the problem of rerostering nurse schedules with hard constraints: New multicommodity flow models. Annals OR 128, 179–197.

Moz, M., Pato, M.V., 2007. A genetic algorithm approach to a nurse rerostering problem. Computers & OR 34, 667–691.

Musliu, N., 2006. Heuristic methods for automatic rotating workforce scheduling. International Journal of Computational Intelligence Research 2, 309–326.

Musliu, N., Schaerf, A., Slany, W., 2004. Local search for shift design. European Journal of Operational Research 153, 51–64.

NVIDIA, 2014. NVIDIA Kepler's Generation. http://www.nvidia.com/object/nvidia-kepler.html. Accessed: 2014-10-01.

NVIDIA Corporation, 2013. NVIDIA CUDA C Programming Guide. http://docs.nvidia.com/cuda/pdf/CUDA_C_Programming_Guide.pdf. Accessed: 2014-10-01.

NVIDIA Corporation, 2014. GeForce GTX 650 Ti. http://www.geforce.com/hardware/desktop-gpus/geforce-gtx-650ti/specifications. Accessed: 2014-10-01.

Ochoa, G., Hyde, M., Curtois, T., Vazquez-Rodriguez, J., Walker, J., Gendreau, M., Kendall, G., McCollum, B., Parkes, A., Petrovic, S., Burke, E., 2012. Hyflex: A benchmark framework for cross-domain heuristic search, in: Hao, J.K., Middendorf, M. (Eds.), European Conference on Evolutionary Computation in Combinatorial Optimisation(EvoCOP 2012), Springer. pp. 136–147.

Örmeci, E.L., Salman, F.S., Yücel, E., 2014. Staff rostering in call centers providing employee transportation. Omega 43, 41–53.

Osogami, T., Imai, H., 2000. Classification of Various Neighborhood Operations for the Nurse Scheduling Problem. Technical Report. IBM Tokyo Research Laboratory 242-8502 Kanagawa Japan.

Owens, J.D., Luebke, D., Govindaraju, N., Harris, M., Krüger, J., Lefohn, A., Purcell, T.J., 2007. A survey of general-purpose computation on graphics hardware. Computer Graphics Forum 26, 80–113.

Pato, M.V., Moz, M., 2008. Solving a bi-objective nurse rerostering problem by using a utopic pareto genetic heuristic. Journal of Heuristics 14, 359–374.

Pato, M.V., Moz, M., 2013. The dataset of the nurse rerostering problem instances.

Pisinger, D., Ropke, S., 2007. A general heuristic for vehicle routing problems. Computers & Operations Research 34, 2403 – 2435.

Qu, R., Burke, E., McCollum, B., Merlot, L., Lee, S., 2009. A survey of search methodologies and automated system development for examination timetabling. Journal of Scheduling 12, 55–89.

Quimper, C.G., Rousseau, L.M., 2010. A large neighbourhood search approach tothe multi-activity shift scheduling problem. Journal of Heuristics 16, 373–392.

Rasmussen, R.V., Trick, M.A., 2008. Round robin scheduling - a survey. European Journal of Operational Research 188, 617–636.

Rechenberg, I., 1971. Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution. Ph.D. thesis. Technical University of Berlin, Department of Process Engineering.

Reeves, C.R., 1995. A genetic algorithm for flowshop sequencing. Computers & Operations Research 22, 5–13.

Reyes, F.d.J.P., 2011. Uso de algoritmos evolutivos para resolver el problema de asignación de horarios escolares en la Facultad de Psicología de la Universidad Veracruzana. Master's thesis. Artificial Intelligence, Universidad Veracruzana. Xalapa, Mexico. URL http://www.lania.mx/~emezura/util/files/tesis_FerminFinal.pdf, Accessed: 2015-01-10.

Ribeiro, C.C., Urrutia, S., 2007. Heuristics for the mirrored traveling tournament problem. European Journal of Operational Research 179, 775 – 787.

Rocha, M., Oliveira, J.F., Carravilla, M.A., 2013. Cyclic staff scheduling: optimization models for some real-life problems. Journal of Scheduling 16, 231–242.

Rudová, H., Müller, T., Murray, K.S., 2011. Complex university course timetabling. Journal of Scheduling 14, 187–207.

Safaei, N., Banjevic, D., Jardine, A.K., 2011. Workforce-constrained maintenance scheduling for military aircraft fleet: a case study. Annals of Operations Research 186, 295–316.

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

Schulz, C., Hasle, G., Brodtkorb, A.R., Hagen, T.R., 2013. GPU computing in discrete optimization. part ii: Survey focused on routing problems. EURO Journal on Transportation and Logistics 2, 159–186.

Seçkiner, S.U., Gökçen, H., Kurt, M., 2007. An integer programming model for hierarchical workforce scheduling problem. European Journal of Operational Research 183, 694 – 699.

Shaw, P., 1998. Using constraint programming and local search methods to solve vehicle routing problems, in: Maher, M., Puget, J.F. (Eds.), Principles and Practice of Constraint Programming CP98. Springer Berlin Heidelberg. volume 1520 of *Lecture Notes in Computer Science*, pp. 417–431.

Smet, P., Wauters, T., Mihaylov, M., Vanden Berghe, G., 2014. The shift minimisation personnel task scheduling problem: A new hybrid approach and computational insights. Omega 46, 64–73.

Srimathy, M., 2008. Scheduling part-time personnel with availability restrictions and preferences to maximize employee satisfaction. Mathematical and Computer Modelling 48, 1806 – 1813.

Stølevik, M., Nordlander, T.E., Riise, A., Frøyseth, H., 2011. A hybrid approach for solving real-world nurse rostering problems, in: Lee, J. (Ed.), Principles and Practice of Constraint Programming CP 2011. Springer Berlin Heidelberg. volume 6876 of *Lecture Notes in Computer Science*, pp. 85–99.

Stolletz, R., 2010. Operational workforce planning for check-in counters at airports. Transportation Research Part E: Logistics and Transportation Review 46, 414–425.

Stolletz, R., Zamorano, E., 2014. A rolling planning horizon heuristic for scheduling agents with different qualifications. Transportation Research Part E: Logistics and Transportation Review 68, 39–52.

Sukstrienwong, A., 2012. Genetic Algorithm for Forming Student Groups Based on Heterogeneous Grouping, in: Recent Advances in Information Science: Proceedings of the 3rd European Conference of Computer Science, pp. 92–97. URL http://www.wseas.us/e-library/conferences/2012/Paris/ECCS/ECCS-14.pdf, Accessed: 2015-01-10.

Talbi, E.G., 2006. Parallel Combinatorial Optimization. Wiley-Interscience.

Tein, L.H., Ramli, R., 2010. Recent Advancements of Nurse Scheduling Models and a Potential Path, in: Proceedings of 6th IMT-GT Conference on Mathematics, Statistics and its Applications, pp. 395–409. URL http://research.utar.edu.my/CMS/ICMSA2010/ICMSA2010_Proceedings/files/statistics/ST-Lim.pdf, Accessed: 2015-01-10.

Thompson, G.M., Pullman, M.E., 2007. Scheduling workforce relief breaks in advance versus in real-time. European Journal of Operational Research 181, 139 – 155.

Topaloglu, S., Ozkarahan, I., 2004. An implicit goal programming model for the tour scheduling problem considering the employee work preferences. Annals of Operations Research 128, 135–158.

Triska, M., Musliu, N., 2011. A constraint programming application for rotating workforce scheduling, in: Mehrotra, K.G., Mohan, C., Oh, J.C., Varshney, P.K., Ali, M. (Eds.), Developing Concepts in Applied Intelligence. Springer Berlin Heidelberg. volume 363 of *Studies in Computational Intelligence*, pp. 83–88.

Van Laarhoven, P.J.M., Aarts, E.H.L., Lenstra, J.K., 1992. Job shop scheduling by simulated annealing. Operations Research 40, 113–125.

Vanden Berghe, G., 2002. An Advanced Model and Novel Meta-heuristic Solution Methods to Personnel Scheduling in Healthcare. Ph.D. thesis. University of Gent.

Warner, D.M., 1976. Scheduling nursing personnel according to nursing preference: A mathematical programming approach. Operations Research 24, 842–856.

Wong, G.Y.C., Chun, H.W., 2003. Nurse rostering using constraint programming and meta-level reasoning, in: Proceedings of the 16th international conference on Developments in applied artificial intelligence, Springer Springer Verlag Inc. pp. 712–721.

Wren, A., 1996. Scheduling, timetabling and rostering A special relationship?, in: Burke, E., Ross, P. (Eds.), Practice and Theory of Automated Timetabling. Springer Berlin Heidelberg. volume 1153 of *Lecture Notes in Computer Science*, pp. 46–75.

Appendices

Appendix A

Skill Based Initialization Algorithm

The skill based initialization algorithm (based on (Vanden Berghe (2002)) (pages 139–160)) is outlined in Algorithm 6. Its objective is to assign the shifts to the employees that are separated into groups by their skills and obtain the initial roster. Let $S(d)$ be a *list of requested shifts* that has to be assigned on each day d and the list is sorted in a descending order according to the 'difficulty' of the shift assignment to the roster, e.g., a night shift is assigned before an early shift to the roster. Furthermore, let $CSE(d)$ be a list of the counts of the shifts that can be assigned to employee e for each day d. This list is based on the skills of the employees (c_2), personnel requests of the employees (c_6) and block constraints (c_7)–(c_9), (c_{12}) with respect to the shifts assigned on the previous days. Finally, let $SAE(d)$ be a binary matrix expressing whether shift s can be assigned to employee e. Similarly, this matrix is a reflection of the same constraints as in $CSE(d)$. Moreover, the shift precedences stated by constraint (c_5) are considered with these constraints together.

For each day d, each requested shift $s \in S(d)$ is assigned to one of the employees $e \in E \,|\, CSE(d)_e > 0$ who is able to serve shift s on day d, i.e., $SAE(d)_{es}$ is ***true***. When such an employee e is found, it is necessary to update $CSE(d)$. Therefore, $CSE(d)_e$ is reset since employee e cannot be used for assignment of another shift on day d. The value of the $CSE(d)$ of all other employees must be decremented, because they were able to serve shift s on day d, but this shift has just been assigned. After each shift assignment, the list $CSE(d)$ is ordered by the count of shifts in an ascending order.

Algorithm 6: Skill based initialization algorithm (SIA) pseudo-code

Input : ETPHD instance

Output: Any feasible roster R respecting constraints $(c_1), (c_2), (c_5), (c_6)$

foreach *day* $d \in D$ **do**
  create and sort $S(d)$; `// 'difficult' shifts first`
  create $CSE(d)$; `// list with counts of assignable shifts to the employees on day` d
  create $SAE(d)$; `// binary matrix employees × shifts of assignable shift on day` d
  foreach *shift* $s \in S(d)$ **do**
    sort $CSE(d)$; `// employees with small count of assignable shifts first`
    foreach *employee* e ***in*** *list* $CSE(d) \,|\, CSE(d)_e > 0$ **do**
      if $SAE(d)_{es} ==$ *false* **then continue**;
      $R_{eds} \leftarrow 1; R_{eds'} \leftarrow 0, \forall s' \in \{S \setminus s\}$; `// assign shift` s `to employee` e `on day` d
      $CSE(d)_e \leftarrow 0$; `// reset` $CSE(d)$ `of employee` e
      $\forall e' \in \{E \setminus e\} : CSE(d)_{e'} \leftarrow CSE(d)_{e'} - 1$; `// decrement` $CSE(d)$ `for the rest`
      break;
return R

Appendix B

Different Modes in the Parallel Algorithm

Figure B.1 illustrates the relation among the modes, particular parts of the parallel algorithm and the unassigned shifts. This automata represents the behavior of the both models – homogeneous and heterogeneous one.

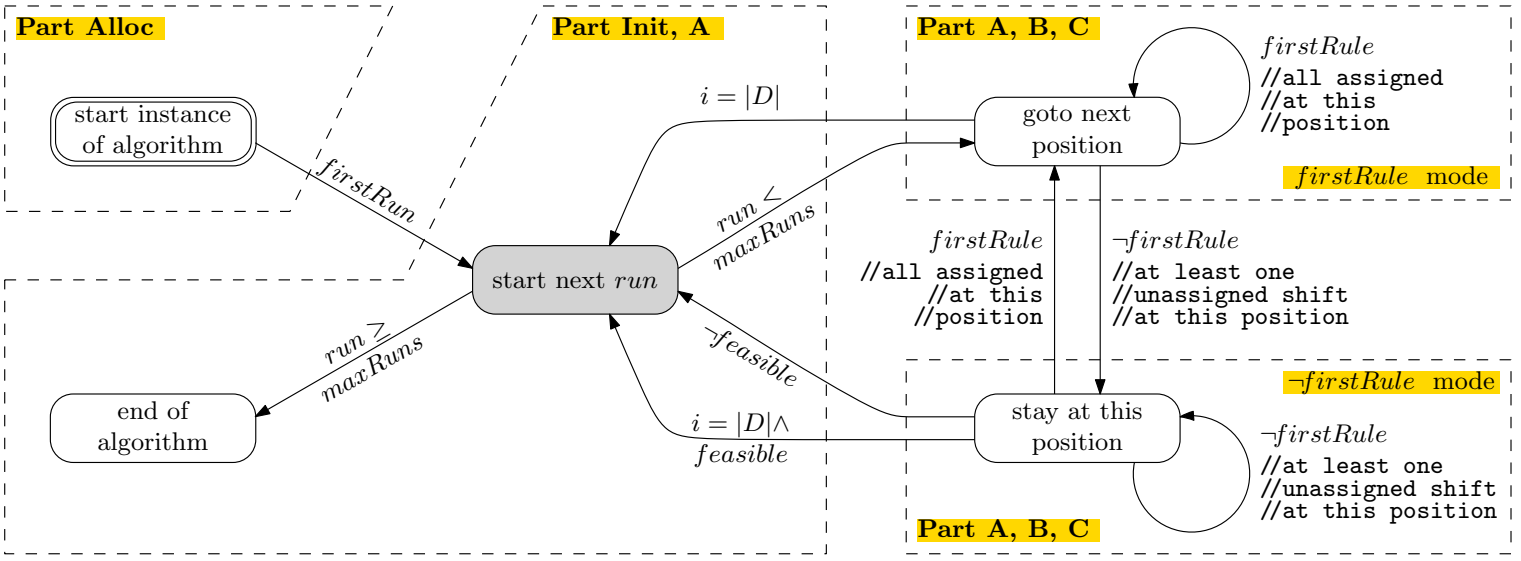


Figure B.1: The automata representing the relation among modes and parts of the algorithm's instance

Appendix C

Heterogeneous Model of the Parallel Algorithm

This appendix contains the pseudo-code of the heterogeneous model of the parallel algorithm explained in Chapter 4 that is represented by Algorithm 7 and Algorithm 8.

Algorithm 7: The heterogeneous model of the parallel algorithm – 1 thread of 1 instance of the algorithm

Input : NRRP instance $\{R^0, RS, A, maxRuns\}$
Output: Best found modified roster $\tilde{R}^{best}$

Part Alloc:

```
// determine the number of parallel instances of the algorithm
m ← allocateGPUResources(|E|, |D|, maxRuns);
// initialize variables common to all instances of the algorithm
R̃^best ← null; run ← 0; applyLS ← false;
// initialize variables for this instance of the algorithm
firstRun ← true; applyBT ← false; backtrack ← 0;
// set the employee index for the current thread
e ← mapThreads(|E|);
__syncthreads();
// perform m parallel instances of the algorithm, each exploits |E| threads
while run < maxRuns do                                          // stop condition
    Part Init:
    foreach instance of m parallel instances do
        if firstRun || i = |D| || ¬feasible then
            // initialize data for the new run
            firstRun ← false; feasible ← true; i ← 0; firstRule ← true;
            foreach e ∈ E do
                [R̃, isOccupied, unassigned, i] ← initAll(e, R^0, A);
                if ¬applyBT then RPP ← generatePartialRosterPositions(e, R^0, applyLS)
                else applyBT ← false;

    Part A:
    foreach instance of m parallel instances do
        foreach e ∈ E do
            if firstRule then                                   // read shift
                d ← RPP_{e,i};          s ← R^0_{e,d};
            else
                d ← d ∈ unassignedRP;   s ← R^0_{unassignedRP};

    Part B:
    cudaMemcpy('cudaMemcpyHostToDevice');
    pen ← evaluateShiftAssignment(e, i, s, R̃, R^0, R_prev, RS, isOccupied, minDaysOff);
    cudaMemcpy('cudaMemcpyDeviceToHost');

    // perform Part C (see Algorithm 8)
    performPartC();
return R̃^best;
```

Algorithm 8: Part C of Algorithm 7

Input : By reference – NRRP instance data from Algorithm 7, line 22
Output: By reference – NRRP instance data to Algorithm 7, line 22

Part C:

1 **foreach** instance of m parallel instances **do**
2 **if** *firstRule* **then**
3 **foreach** $e \in E$ **do**
4 **if** $pen = 0$ **then** `// either assign shift or mark it as unassigned by Rule 1`
5 $R_{e,d} \leftarrow s$; $isOccupied_{e,d} \leftarrow$ `true`;
6 **else**
7 $unassigned_e \leftarrow RPP_{e,i}$;
8 **else**
9 $e_{min} \leftarrow \text{argmin}_{e \in E}(pen)$; `// either assign shift or set this run as unfeasible`
10 **if** $pen_{e_{min}} < \infty$ **then**
11 $R_{e_{min},d} \leftarrow s$; $isOccupied_{e_{min},d} \leftarrow$ `true`; $unassigned_{e_{min}} \leftarrow$ `null`;
12 **else**
13 $applyBT \leftarrow$ `doModifiedBacktrack()`; $feasible \leftarrow$ `false`; `// set the flag for backtrack`
14 **if** $feasible$ **then**
15 $unassignedRP \leftarrow$ `pickUnassignedRosterPosition(`$unassigned$`)`; `// set mode of this run wrt unassigned shifts`
16 **if** $unassignedRP =$ `null` **then**
17 $firstRule \leftarrow$ `true`; $i \leftarrow i + 1$;
18 **else**
19 $firstRule \leftarrow$ `false`;
20 **if** $i = |D|$ && $feasible$ **then** `// compute objective function for feasible finished run`
21 $Z(\tilde{R}) \leftarrow \sum_{\forall e \in E}$ `computeObjectiveFunction(`$e, \tilde{R}, R^0$`)`;
22 **if** $i = |D|$ || $\neg feasible$ **then**
23 **if** $feasible$ **then**
24 $[Z(\tilde{R}^{best}), \tilde{R}^{best}, RPP^{best}] \leftarrow$ `chooseBetter`$Z(\tilde{R}), Z(\tilde{R}^{best})$ `// store best solution`
25 **else if** $\neg applyBT$ **then**
26 $applyLS \leftarrow$ `updateLocalSearchParameters`; `// or update local search related data for the next run`
27 $run \leftarrow run + 1$;

Curriculum Vitae

Zdeněk Bäumelt was born in Dvůr Králové nad Labem, Czech Republic, in 1982. He received his master of science degree in cybernetics and control engineering in Faculty of Electrical Engineering in Czech Technical University in Prague (CTU) in 2007, when he had defended his master thesis focused on the Nurse Rostering Problem. Subsequently, he was employed in CTU on the ARTIST project in order to extend the knowledge background of the algorithms design in the employee timetabling domain. Since 2008, he has started his Ph.D. studies on Advanced Methods and Models for Employee Timetabling Problems in the same university. During his studies, he participated on the projects Decision Making and Control for Manufacturing III, Centre for Applied Cybernetics, Demanes and Centre for Applied Cybernetics III. He is interested in combinatorial optimization, specifically to scheduling and timetabling.

His teaching activities in CTU involved courses of Logic Control, Computer Systems Structures and, mainly, Combinatorial Optimization, where he also was the co-author of the lectures. He has also supervised several students' projects and diploma theses.

Research results of Zdeněk Bäumelt were presented in several international conferences, e.g., in the top two of the employee timetabling domain – MISTA (Multi-disciplinary International Scheduling conference: Theory & Applications) and PATAT (international conference of the Practice And Theory of Automated Timetabling). Moreover, his results were published in the international reviewed journal Computers & Operations Research and are under review in European Journal of Operational Research. Finally, Zdeněk Bäumelt participated on the employee timetabling project with an airport ground company, where the impact of his research was proven.

Czech Technical University in Prague
Prague, February 2015 *Zdeněk Bäumelt*

List of Author's Publications

All of the author's publications are directly related to the topic of the thesis. They are separated into five groups as follows.

Publications in Journals with Impact Factor

Zdeněk Bäumelt, Přemysl Šůcha, and Zdeněk Hanzálek. A Multistage Approach for an Employee Timetabling Problem with a High Diversity of Shifts as a Solution for a Strongly Varying Workforce Demand. *Computers & Operations Research*, 49, pages 117–129, 2014. ISSN 0305-0548. doi: 10.1016/j.cor.2014.03.019. URL http://www.sciencedirect.com/science/article/pii/S0305054814000744. Accessed: 2015-01-10. **Coauthorship 50 %**.

Zdeněk Bäumelt, Jan Dvořák, Přemysl Šůcha, and Zdeněk Hanzálek. A Novel Approach for the Nurse Rerostering Problem based on a Parallel Algorithm. *European Journal of Operational Research*, 2015. In major revision. **Coauthorship 40 %**.

Publications in Reviewed Journals

Zdeněk Bäumelt, Přemysl Šůcha, and Zdeněk Hanzálek. A Multistage Approach for an Employee Timetabling Problem with a High Diversity of Shifts as a Solution for a Strongly Varying Workforce Demand. *Computers & Operations Research*, 49, pages 117–129, 2014. ISSN 0305-0548. doi: 10.1016/j.cor.2014.03.019. URL http://www.sciencedirect.com/science/article/pii/S0305054814000744. Accessed: 2015-01-10. **Coauthorship 50 %**.

Zdeněk Bäumelt, Jan Dvořák, Přemysl Šůcha, and Zdeněk Hanzálek. A Novel Approach for the Nurse Rerostering Problem based on a Parallel Algorithm. *European Journal of Operational Research*, 2015. In major revision. **Coauthorship 40 %**.

Patents

There are no patents related to the thesis.

Publications indexed in Web of Science

Zdeněk Bäumelt, Přemysl Šůcha, and Zdeněk Hanzálek. A Multistage Approach for an Employee Timetabling Problem with a High Diversity of Shifts as a Solution for a Strongly Varying Workforce Demand. *Computers & Operations Research*, 49, pages 117–129, 2014. ISSN 0305-0548. doi: 10.1016/j.cor.2014.03.019. URL http://www.sciencedirect.com/science/article/pii/S0305054814000744. Accessed: 2015-01-10. **Co-authorship 50 %.**

Other Publications

International Conference Papers

Zdeněk Bäumelt, Přemysl Šůcha, and Zdeněk Hanzálek. Nurse Scheduling Web Application. In *26th Workshop of the UK Planning and Scheduling Special Interest Group*, pages 120–123. Charles University, Prague, Czech Republic, 2007. URL http://ktiml.mff.cuni.cz/~bartak/PLANSIG2007/papers/paper07.pdf. Accessed: 2015-01-10. **Co-authorship 34 %, cited by** (Tein and Ramli (2010); Kumara and Perera (2011)).

Zdeněk Bäumelt, Přemysl Šůcha, and Zdeněk Hanzálek. Personnel Scheduling Problem with a High Diversity of Shifts. In *23rd European Conference on Operational Research – Book of Abstracts*, page 235. Bonn, Germany, The Association of European Operational Research Societies, Brussels, Belgium, 2009. URL https://www.yumpu.com/en/document/view/10243388/technical-sessions-monday-0800-0920-euro-is-the/235. Accessed: 2015-01-10. **Co-authorship 34 %.**

Zdeněk Bäumelt, Libor Waszniowski, Přemysl Šůcha, and Zdeněk Hanzálek. Integrated Vehicle Routing and Rostering in Home Health Care Services. In *36th International Conference of the EURO Working Group on Operational Research Applied to Health Services, focused on Operations Research for Patient-Centered Health Care Delivery*, pages 267–275. University of Genova, Genova, Italy, 2010. ISBN 978-88-568-2595-4. URL https://support.dce.felk.cvut.cz/pub/hanzalek/publications/Hanzalek10_169603.pdf. Accessed: 2015-01-10. **Co-authorship 25 %.**

Zdeněk Bäumelt, Přemysl Šůcha, and Zdeněk Hanzálek. An Evolutionary Algorithm in a Multistage Approach for an Employee Rostering Problem with a High Diversity of Shifts. In *8th International Conference on the Practice and Theory of Automated Timetabling*, pages 97–112. Queen's University of Belfast, Belfast, Northern Ireland, 2010. ISBN 08-538-9973-3. URL https://support.dce.felk.cvut.cz/pub/hanzalek/publications/Hanzalek10_169605.pdf. Accessed: 2015-01-10. **Co-authorship 40 %, cited by** (Ladier et al. (2011); Reyes (2011)), **cited by** (Castiñeiras and Sáenz-Pérez (2013)) – **indexed in Web of Science**.

Zdeněk Bäumelt, Přemysl Šůcha, and Zdeněk Hanzálek. A Genetic Algorithm for a Nurse Rerostering Problem. In *10th Workshop on Models and Algorithms for Planning and Scheduling Problems*, pages 70–72. Institute for Theoretical Computer Science, Charles University, Prague, Czech Republic, 2011. **Co-authorship 50 %, cited by** (Sukstrienwong (2012)).

Zdeněk Bäumelt, Přemysl Šůcha, and Zdeněk Hanzálek. A Parallel Approach for a Nurse Rerostering Problem on the GPU. In *5th Multidisciplinary International Conference on Scheduling: Theory and Applications*, pages 576–578. Phoenix, Arizona, USA, University of Nottingham, Nottingham, Czech Republic, 2011. URL http://expertise.hogent.be/files/9242367/MISTA_proceedings.pdf. Accessed: 2015-01-10. **Co-authorship 50 %**.

Zdeněk Bäumelt, Jan Dvořák, Přemysl Šůcha, and Zdeněk Hanzálek. An Acceleration of the Algorithm for the Nurse Rerostering Problem on a Graphics Processing Unit. In *5th International Conference on Applied Operational Research: Lecture Notes in Management Science*, pages 101–110. Technical University of Lisbon, Lisbon, Portugal, Tadbir Operational Research Group, Vancouver, Canada, 2013. URL http://orlabanalytics.ca/lnms/archive/v5/lnmsv5p101.pdf. Accessed: 2015-01-10. **Co-authorship 40 %**.

Remaining Publications

Zdeněk Bäumelt. The Air Navigators Rostering Problem. In *14th International Student Conference on Electrical Engineering*, pages 1–4. Faculty of Electrical Engineering, Czech Technical University in Prague, Prague, Czech Republic, 2009. **Co-authorship 100 %, awarded**.

Zdeněk Bäumelt. Preliminary Doctoral Thesis: Personnel Scheduling Problem with a High Diversity of Shifts. Faculty of Electrical Engineering, Czech Technical University in Prague, Prague, Czech Republic, 2010. **Co-authorship 100 %**.

Zdeněk Bäumelt. A Multistage Approach for an Employee Timetabling Problem with a High Diversity of Shifts as a Solution for a Strongly Varying Workforce Demand. In *Embedded Systems Colloquium*. Centre for Applied Cybernetics, Department of Control Engineering, Faculty of Electrical Engineering, Czech Technical University in Prague, Prague, Czech Republic, 2011. **Co-authorship 100 %**.

Zdeněk Bäumelt, Přemysl Šůcha, and Zdeněk Hanzálek. Personnel Scheduling System, 2013. Authorized software, see more on URL http://support.dce.felk.cvut.cz/pub/hanzalek/RAL/. Accessed: 2015-01-10. **Co-authorship 45 %**.

Czech Technical University in Prague
Prague, February 2015 *Zdeněk Bäumelt*

This thesis is focused on the domain of the employee timetabling problems. Its goals were set as follows:

1. To describe the basic terms, the classification and the categorization of the employee timetabling within the context of the operations research. Consequently, to identify the other goals of the thesis that reveal from significant gaps in the domain.

2. To propose and describe an approach capable to solve large instances of the employee timetabling problem having a high diversity of shifts (shift types).

3. To consider new architectures that can be exploited by a parallel algorithm applied in the domain of the employee timetabling in order to accelerate the solution of the chosen problem.

4. To verify the proposed models, algorithms and approaches on the state of the art benchmark instances and, if possible, on the real life instances.