

KATHOLIEKE UNIVERSITEIT
LEUVEN

Arenberg Doctoral School of Science, Engineering & Technology
Faculty of Science
Computer Science Campus Kortrijk

Heuristic approaches for real-world timetabling problems in education and health care

Peter Demeester

Dissertation presented in partial fulfillment of the requirements for the degree of Doctor in Science

March 2010

Heuristic approaches for real-world timetabling problems in education and health care

Peter Demeester

Jury:
Prof. Dr. Patrick De Causmaecker, promotor
Dr. Ir. Greet Vanden Berghe, co-promotor
Prof. Dr. Dirk Van Oudheusden
Prof. Dr. Dirk Roose
Prof. Dr. Nico Vandaele
Prof. Dr. Ir. Tom Holvoet
Prof. Dr. Edmund Burke
(University of Nottingham, UK)

Dissertation presented in partial fulfillment of the requirements for the degree of Doctor in Science

March 2010

© Katholieke Universiteit Leuven – Faculty of Science
Address, B-3001 Leuven (Belgium)

Alle rechten voorbehouden. Niets uit deze uitgave mag worden vermenigvuldigd en/of openbaar gemaakt worden door middel van druk, fotocopie, microfilm, elektronisch of op welke andere wijze ook zonder voorafgaande schriftelijke toestemming van de uitgever.

All rights reserved. No part of the publication may be reproduced in any form by print, photoprint, microfilm or any other means without written permission from the publisher.

D/2010/10.705/12
ISBN 978-90-8649-319-7

Voorwoord

Een doctoraat is eigenlijk een complex optimalisatieprobleem, met resources die efficiënt gepland en gebruikt moeten worden. Binnen een beperkte periode moeten er artikels, resultaten en een thesis gegenereerd worden. Zonder de hulp van de volgende mensen was me dat zeker niet gelukt.

Mijn 2 promotoren, Patrick en Greet, zou ik graag willen bedanken voor de aangename samenwerking en de kans die ze me geboden hebben om te doctoreren.

Ik kan ook niet anders dan de vele collega's en ex-collega's waarmee ik ooit een bureau of een project heb gedeeld en die voor de nodige sfeer zorg(d)en bedanken (in alfabetische volgorde): Annemie, Bart (2x), Björn, Burak, Dimitri, Filip, Geert, Geoffrey, Jannes, Joris, Jorn, Karel, Katja, Koen, Mieke, Mustafa, Philippe, Pieter, Steven, Stijn, Tahir, Tijs, Tony, Vincent, Wim, en Wouter.

Mijn laatste bureaucompagnon, Wouter, wil ik nog eens bedanken voor de hulp bij de wiskundige modellen en zijn aparte kijk op de dingen. Burak voor de introductie tot de wereld van hyperheuristieken en testen. Koen, Filip, Jannes en Tony voor de ondersteuning bij het gebruik van de tientallen computers in de computerklassen om mijn experimenten op uit te voeren. Door de vele experimenten ben ik waarschijnlijk mede-verantwoordelijk voor de stijging van de temperatuur op aarde en op KaHo Sint-Lieven in het bijzonder. Koen voor de hulp met LaTeX.

I would like to thank also Andrea Schaerf and Luca Di Gaspero from the Università di Udine for their hospitality during my one month visit.

Mijn juryleden zou ik graag willen bedanken voor hun nuttige opmerkingen. Tijdens mijn eerste doctoraatsseminarie hebben die me geholpen om mijn doctoraatsonderwerp beter af te lijnen. Daardoor is de kwaliteit van mijn thesis er op vooruit gegaan.

Mijn zus en ons pa wil ik bedanken voor hun nooit aflatende steun.

Abstract

In this thesis we describe, model, and solve several real-world timetabling problems from the health care and education domain. **Timetabling** problems are characterized by scarce and/or expensive resources that have to be scheduled into a space and time dimension taking into account several constraints. In timetabling problems, time is divided into discrete events called time slots. The goal is to assign the resources to time slots taking into consideration several restrictions. Characteristic for these resources, for example, is that they cannot be shared in the same time slot. This means that the necessary precautions should be taken to avoid that a resource is assigned more than once to the same time slot. Examples of timetabling problems are university course and examination timetabling, or employee timetabling. In the former, the resources are lecturers, students, and rooms and the goal is to assign lectures or exams to rooms and time slots, taking into account for example that no student can take more than lecture or exam in the same time slot. In the latter, personnel is assigned to time slots, taking into account, for example, work regulations. A well known example is nurse rostering in which nurses are assigned to shifts such that the required coverage per shift is satisfied and that the personal wishes and work regulations are taken into account.

Real-world problems from education and health care are the focus of this thesis. On the one hand, real-world problems are, because of their specific constraints, a scientific challenge to solve. Timetabling problems from the **education** and **health care** domain are chosen on the other hand, since these sectors feel more and more the pressure to reduce costs, and consequently using their available resources more efficiently. Because of their size and unusual constraints, real-world problems turn out to be very hard to solve to optimality. Actually, in practical cases, often not the optimal solution is wanted, but rather a good enough solution, that should be obtained in a reasonable amount of computation time. This is where **heuristics** come into play. Exact methods guarantee that if a feasible solution exists, that the result will be optimal; heuristics, however, do not guarantee this. However, on the other hand, obtaining an optimal solution for these real-world problems with an exact method may take very long, while a heuristic method can provide a solution in reasonable computation time. The craft is to construct these

heuristics such that the result of the search is good enough to have a workable solution. In practice, the result of the automatic search algorithm should be such that a human planner cannot improve the solution anymore. Of course, this is only possible if all constraints are explicitly known to the developer.

The idea of this thesis is to develop applications, in a short application development cycle, that tackle different real-world timetabling problems. All tackled problems are solved in a similar manner. In the first part of the procedure the domain model which describes the base concepts of the problem at hand is constructed. Next, the necessary parts of the heuristics are assembled. Essential ingredients are the representation of the search space and the neighbourhoods. Choosing the search space representation and neighbourhoods well, has a major impact on the algorithm's performance.

In the first part of the thesis we focus on real-world timetabling problems from the education domain. We successively tackle a curriculum-based university course timetabling at KaHo Sint-Lieven, a post-enrolment university course timetabling problem at K.U. Leuven and an examination timetabling problem at KaHo Sint-Lieven. We fully describe and model the problems and solve them with a token-ring based tabu search, a hybrid local search, and hyper-heuristics search algorithm respectively. Regarding the KaHo Sint-Lieven examination timetabling problem we need less time slots than the human planner to solve it. The following real-world health care problems are tackled in the second part of the thesis: a real-world nurse rostering problem and a patient admission scheduling problem. The first problem is solved with a tabu search algorithm, while the latter is tackled with a token-ring tabu search algorithm as well as with a hyper-heuristic approach. Regarding the patient admission scheduling problem, the hyper-heuristic approach outperforms the tabu search algorithm. To compare the performance of the developed heuristics with those of the literature, we have applied the exam timetabling hyper-heuristic to two well-known exam timetabling datasets from the literature. Regarding the first dataset we find for 5 out of 13 instances better or equal results. Concerning the second dataset, which was part of the Second International Timetabling Competition, we obtain results that can compete with the competition's top 5.

Nederlandstalige samenvatting

In deze doctoraatsthesis worden verscheidene reële timetablingproblemen uit het onderwijs- en gezondheidszorgdomein beschreven, gemodelleerd en opgelost. Timetablingproblemen worden gekarakteriseerd door schaarse en/of dure middelen die moeten toegewezen worden aan een tijdstip en/of plaats, daarbij rekening houdend met allerlei beperkingen. In de meeste timetablingproblemen is de tijd onderverdeeld in discrete tijdssloten. Typerend voor deze middelen is dat ze bijvoorbeeld niet gedeeld kunnen worden in hetzelfde tijdsslot. De nodige voorzorgen zullen dan ook moeten genomen worden om te vermijden dat een middel meer dan éénmaal toegewezen wordt aan een tijdsslot. Voorbeelden van timetablingproblemen zijn het automatisch opstellen van uur- en examenroosters of personeelsplanning. In het eerste geval zijn de middelen docenten, studenten, en lokalen en de doelstelling is om lessen of vakken toe te wijzen aan lokalen en tijdssloten, rekening houdend met het feit dat bijvoorbeeld geen enkele student meer dan één vak kan volgen of meer dan één examen kan afleggen in hetzelfde tijdsslot. In het laatste geval wordt personeel toegewezen aan tijdssloten rekening houdend met wettelijke regels en persoonlijke voorkeuren. Een heel bekend voorbeeld is nurse rostering waarin verpleegkundigen toegewezen worden aan shifts zodat aan de gewenste bezetting per shift en eveneens aan de persoonlijke wensen en wettelijke regels voldaan is.

De reële timetablingproblemen die we in deze scriptie behandelen komen uit het domein van de gezondheidszorg en het onderwijs. Enerzijds zijn reële problemen door hun specifieke beperkingen een wetenschappelijke uitdaging om op te lossen. Timetablingproblemen uit het onderwijs en de gezondheidszorg werden anderzijds gekozen daar deze sectoren alsmaar meer onder druk staan om hun beschikbare middelen efficiënter te gebruiken. Deze sectoren hebben met andere woorden veel baat bij een geautomatiseerde oplossing voor hun problemen. Enerzijds komen er (dure) middelen vrij die zich niet meer moeten bezighouden met het manueel opstellen van roosters en anderzijds kan er verwacht worden dat deze automatisch gegenereerde oplossing van een betere kwaliteit zal zijn dan de manueel opgestelde. Reële problemen blijken door hun grootte en ongewone beperkingen heel moeilijk optimaal op te lossen. Echter, in de realiteit wilt men niet altijd de meest optimale

oplossing, maar een voldoende goede oplossing die binnen een redelijke rekentijd gevonden kan worden. Dit zijn de ideale omstandigheden om heuristieken toe te passen. Exacte methodes garanderen dat als een 'feasible' (uitvoerbare) oplossing bestaat, dat deze zeker optimaal is; heuristieken, daarentegen, garanderen dit niet. Anderzijds, kan het vinden van een optimale oplossing met een exacte methode voor een complex timetablingprobleem heel lang duren, terwijl een heuristische methode een voldoende goede oplossing kan vinden in een redelijke rekentijd. De kunst is nu om deze heuristieken zodanig te construeren dat het resultaat van het zoekproces goed genoeg is om een uitvoerbare oplossing te bekomen. Het bekomen resultaat zou voldoende goed van kwaliteit moeten zijn opdat een menselijke planner de gevonden oplossing onmogelijk nog kan verbeteren. Dit is natuurlijk enkel mogelijk op voorwaarde dat alle beperkingen waarmee de menselijke planner rekening houdt gekend zijn bij de software-ontwikkelaar.

De bedoeling van deze thesis is dan ook om in zo'n korte mogelijke ontwikkelingstijd verscheidene reële timetablingproblemen op te lossen. Alle in deze thesis behandelde problemen worden op een zelfde manier aangepakt. Initieel wordt in deze procedure een domeinmodel opgesteld dat de basisconcepten van het bewuste probleem beschrijft. Daarna worden de verschillende onderdelen van de heuristieken ontwikkeld. Essentiële onderdelen zijn onder andere de voorstelling van de zoekruimte en de omgevingen. Een omgeving is de verzameling van oplossingen die kan bereikt worden vanuit de huidige oplossing door het uitvoeren van een zet. De keuze van zowel de voorstelling van de zoekruimte als de omgevingen hebben een grote invloed op de performantie van het zoekalgoritme.

In het eerste gedeelte van deze scriptie behandelen we reële timetablingproblemen uit het onderwijsdomein. We lossen achtereenvolgens een curriculumgebaseerd uurroosterprobleem aan de KaHo Sint-Lieven, een post-enrolment gebaseerd uurroosterprobleem aan de K.U. Leuven en een examenroosterprobleem eveneens aan de KaHo Sint-Lieven op. De drie problemen worden respectievelijk opgelost met een token-ring tabu search, een hybriede guided local search en hyperheuristiek algoritme. Voor het examenroosterprobleem vinden we een oplossing waarbij we alle examens rekening houdend met alle beperkingen kunnen plannen, daarbij gebruik makend van minder tijdssloten dan de oorspronkelijke oplossing bekomen door de menselijke planner. Het tweede deel van de scriptie behandelt twee reële problemen uit de gezondheidszorg: enerzijds een verpleegstersplanningsprobleem en het probleem van het automatisch toekennen van patiënten aan bedden. Het eerste probleem wordt oplost met een tabu search algoritme, terwijl het laatste probleem opgelost wordt met zowel een token-ring tabu search als een hyperheuristiek algoritme. Wat betreft het laatste probleem presteert de hyperheuristiek beter dan het tabu search algoritme. Om de performantie van onze heuristieken te vergelijken met de methodes uit de literatuur, hebben we de hyperheuristiek die we ontwikkeld hadden om de examenroosters van de KaHo Sint-Lieven automatisch te genereren, met enkele kleine wijzigingen toegepast op

twee welbekende datasets uit de literatuur. Voor de eerste dataset bekomen we met deze hyperheuristiek voor 5 van de 13 instanties betere oplossingen dan deze gevonden in de literatuur. Wat betreft de tweede dataset, die onderdeel was van de tweede International Timetabling Competition, bekomen we resultaten die kunnen concurreren met de top 5 van deze competitie.

Contents

Contents ix

1 Introduction 1

1.1 Objective 1

1.2 Contributions 4

1.3 Structure of the Thesis 5

2 Literature Overview 7

2.1 University Timetabling: Course and Examination Timetabling . . 7

2.1.1 University Course Timetabling 8

2.1.2 Examination Timetabling 12

2.2 Health Care: Personnel Planning and Patient Admission Scheduling 16

2.2.1 Artificial Intelligence and Operations Research in Health Care 16

2.2.2 Personnel Planning 16

2.2.3 Patient Admission Scheduling 17

3 Curriculum Based University Course Timetabling 25

3.1 Introduction 25

3.2 The KaHo Sint-Lieven Course Timetabling Problem 26

3.2.1 Problem Description 26

3.2.2 Formal Problem Description 29

3.2.3 Problem Size 30
3.3 Constructing a Local Search Algorithm for the KaHo Sint-Lieven Course Timetabling Problem 31
3.3.1 Solution Space 31
3.3.2 Neighbourhoods 32
3.3.3 Cost Function 34
3.3.4 The Local Search Algorithm and its Parameters 35
3.3.5 Real-World Issues 36
3.4 Decomposition Approach 36
3.4.1 From Tiles to Pillars 36
3.5 Solving the Problem in Different Stages 40
3.6 Test Results 42
3.7 Conclusion 47

4 Post Enrolment Based University Course Timetabling 49
4.1 Introduction 49
4.2 Dynamic Cost Function: Related Work 52
4.3 Problem Description 53
4.3.1 Constraints 54
4.3.2 Mathematical Formulation 54
4.4 A Hybrid GLS Algorithm 57
4.4.1 Cost Function 57
4.4.2 Dynamic Cost Function 57
4.4.3 Representation of a Solution 58
4.4.4 Neighbourhoods 59
4.4.5 Initial Solution 61
4.5 Experiments 61
4.5.1 First Phase: Independent Data Sets 63

4.5.2 Phase 2: Joint Data Sets 68
4.5.3 Summary 71
4.6 Conclusion 71

5 Examination Timetabling 81

5.1 Introduction 81
5.2 Examination Timetabling Problem Description 82
5.2.1 The KaHo Examination Problem 82
5.2.2 Constraints and Objective 83
5.2.3 Mathematical Formulation 84
5.2.4 Benchmarks versus KaHo Examination Problem 87
5.3 Solution Methods 87
5.3.1 Hyper-heuristics Framework: Acceptance Criteria and Heuristic Selection Method 87
5.3.2 Low-Level Heuristics 89
5.3.3 Three Low-Level Heuristics Sets for three Examination Timetabling Problems 93
5.4 Experiments 95
5.4.1 The KaHo Examination Timetabling Problem 96
5.4.2 Toronto Benchmark Problems 96
5.4.3 The ITC 2007 Examination Timetabling Benchmark Problems 100
5.5 Conclusion 101

6 Personnel Planning 115

6.1 Introduction 115
6.2 Problem Description 116
6.3 Search Algorithm 117
6.4 Linear Numberings Evaluation Method 118
6.4.1 Description of the Linear Numberings Method 118

6.5 Extended Linear Numberings 121
6.5.1 Extended Linear Numberings Representation 123
6.6 Results 123
6.6.1 A Sample Application 123
6.6.2 Efficiency 127
6.7 Conclusion 127

7 Patient Admission Scheduling 131

7.1 Introduction 131
7.2 Terminology 133
7.3 Problem Description and Generation of Test Data 134
7.3.1 Parameters and Variables 134
7.3.2 Constraints 135
7.3.3 Constraint Weights 136
7.3.4 Generation of the Data Set 137
7.4 Integer Programming Approach 138
7.4.1 Extra Constants 138
7.4.2 Decision Variables 138
7.4.3 Objective Function 139
7.4.4 Constraints 139
7.4.5 Results 140
7.5 A Tabu Search Algorithm for Assigning Patients to Beds 140
7.5.1 The Representation of a Solution 140
7.5.2 Neighbourhoods 141
7.5.3 Initial Solution 144
7.5.4 Cost Function 144
7.5.5 Experiments 145

7.6 A Hyper-heuristic Approach to Solve the Patient Admission Scheduling Problem . . . 153
7.6.1 Heuristics . . . 153
7.6.2 Hyper-heuristics . . . 154
7.7 Experiments . . . 155
7.7.1 Settings . . . 155
7.7.2 Results . . . 157
7.8 Conclusion . . . 158

8 Conclusions and Future Work **163**
8.1 Conclusions . . . 163
8.2 Future Research . . . 165
8.2.1 Education . . . 165
8.2.2 Health Care . . . 167

Bibliography **169**

Curriculum vitae **187**

Chapter 1

Introduction

1.1 Objective

Nowadays, the public sector is more and more forced to reduce costs and cut expenses. This means that the available resources should be used more efficiently. One of the possibilities to realize this, is automating repetitive, cumbersome, and time-consuming tasks. For example, semi-annual repetitive tasks such as the construction of an examination or course timetable can be automated, freeing resources that can be used more efficiently. The time spend to manually construct a good timetable ranges from days to even weeks. Head nurses who spend several hours constructing the monthly work roster can be assigned to other tasks, on the condition of course that they can dispose of an automated rostering tool that assists them during this process. This kind of problems are the subject of this thesis: time-consuming, repetitive tasks, which are executed (semi-)manually, that involve expensive resources, such as personnel, that need to be assigned to location-time slot combinations taking into account several constraints. We fully describe, model, and solve different ***real-world*** timetabling problems from education and health care. A substantial part of the current timetabling research concentrates on tackling one particular problem with tailor-made algorithms in order to obtain better results than the state-of-the-art. The focus of this thesis is on solving a wide range of real-world timetabling problems, since they are - because of their specific constraints - a true challenge to model and solve from a scientific point of view. The only prerequisite is that good enough solutions should be obtained in reasonable computation time, and maybe more importantly, in a short application development time. For this reason, all timetabling problems are tackled in this thesis with the same generic approach. This approach provides assistance and sufficient flexibility to create in a fast manner, a wide range of

timetabling applications. To illustrate the wide applicability of the approach, it is applied to different timetabling problems from the education and health care domain. Problems from health care and education are chosen, since on the one hand they have social relevance, and the other hand they constitute a substantial part of the timetabling research, of which several benchmark data sets exist. The latter allows comparing the approach presented in this thesis with the existing state-of-the-art.

- Concerning the educational part of the thesis, we solve the course and examination timetabling problem at the School of Engineering of the university college KaHo Sint-Lieven in Gent, Belgium. Both are curriculum-based timetabling problems: students are grouped into student groups according to the curriculum they take. Instead of generating individual timetables for every student, as is the case in post-enrolment based course timetabling, timetables are constructed according to the students' curricula. Furthermore, we also solve the course timetabling problem at the Belgian university K.U.Leuven. In essence, this is also a curriculum-based course timetabling problem, but since an increasing number of students nowadays take courses from different curricula, leading to conflicting timetables, we solve it as a post-enrolment based timetabling problem. The latter problem is solved with a hybrid tabu search algorithm, that is extended with a cost function of which the weights can be adapted dynamically. In order to have some comparison with the current state-of-the-art methods, we also solve two exam timetabling benchmarks.
- With respect to health care, we study, model and solve a nurse rostering problem in a Belgian hospital as well as a new problem that we have introduced to the operational research community as the patient admission scheduling problem. In the former problem, nurses are assigned to shifts taking into consideration *coverage* and *time-related* constraints. The coverage constraints express the number of employees required per skill category and per shift type, whereas the time-related constraints express the employees' preferences and the statutory rules. The patient admission scheduling problem consists of assigning patients to beds taking into account numerous medical and personal constraints.

According to Anthony Wren [146] '***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'. In most cases the resources are either scarce or expensive (or a combination of both) and can only be used once in a time unit. Examples of resources are personnel, such as nurses [31, 65] that have to be assigned shifts or students and lecturers that need to be assigned a limited set of time slot-room combinations in examination and course timetabling. Neither students nor teachers can be at two places at the same

time. In course timetabling, a room can only hold one course in the same time slot. This means that these resources can be used only once in the same unit of time.
Besides these impracticalities that are inherent to the type of problem, there are also other constraints that should be taken into consideration. Some of the constraints are rather universal, such as that a room cannot accommodate more students than there are seats available or that an employee cannot work more than a limited number of consecutive shifts, but others really depend on the individual situation, and this is what makes real-world timetabling complex. Constraints can be considered either *hard* or *soft*. Contrary to soft constraints, hard constraints cannot be violated at any time. A solution satisfying all hard constraints is called *feasible*. The more soft constraints a feasible solution obeys, the better it will be perceived.

Focusing on real-world problems can, as a by-product, give software developers and researchers new insights for tackling other problems, such as benchmark instances. These are used as a test bed for comparing the performance of different algorithms. Since developers are confronted with new and sometimes exotic constraints when considering real-world problems, they are challenged to come up with new approaches to tackle the problem. These insights may lead to improved solution methods to grapple with the benchmarks, as we will show in this thesis.

Due to their complexity, most real-world timetabling problems turn out to be very 'hard' to solve to optimality in a reasonable amount of time. Often in practical cases, not the optimal solution is wanted, but a good enough solution should be provided within a reasonable amount of computation time. Dependent on the situation, the concept of a reasonable amount of time can range from a few seconds or minutes to several weeks. An examination timetabling problem that only needs to be solved twice a year easily can take several weeks to generate a solution, while a solution for the patient admission scheduling problem should be obtained in a few minutes, since it has to be executed several times a day. Burke et al. [32] summarize this as 'good enough - soon enough - cheap enough' solutions for problems. If solutions should be provided in a reasonable amount of time, ***heuristics*** are a good option. In contrast to, for example, exact methods of which one can be certain that the solution, once it is found, is optimal, heuristic methods do not guarantee that an optimal solution can be obtained. On the other hand however, heuristics do provide a solution, while exact methods might run into problems for large problems as we will see in this thesis. Barry McCollum stressed the fact that there is still a gap between practice and theory in automated timetabling in his invited talk of Patat 2006 [113]. Some of the theoretical applications lack robust and flexible techniques handling the real-world complexity of the timetabling problems. The aim of this thesis is to give an onset to close this gap.

Essentially, the same procedure is followed throughout this thesis. All discussed cases are built starting from a domain model, that at an abstract level consists

of the resources and the smallest discrete time unit. All other domain concepts inherit from these abstract classes. The timetabling object, which is the central object of the domain, is the event that needs to be scheduled into a resource-time unit combination. Once the domain model is established, one has to decide on how the solution space will be represented. The solution space is the set of all possible solutions. Choosing a good solution space representation is important, since this can influence the quality of the algorithm. Some of the hard constraints can be included in the solution space representation, such that they are always satisfied, and thus need not to be checked. Dependent on which hard constraints already are satisfied in the solution space representation, the *cost function* is formulated. It is typically formulated as the weighted sum of the violations of the constraints and is consequently a measure for the quality of a solution. A solution in the solution space is transformed into another solution by altering parts of the original solution. This alterations are called *moves*. Whether a proposed solution is accepted as a new solution depends on the chosen acceptance criterion of the heuristic method. *Steepest descent* for example, will only accept solutions that are better than or equal to the current solution. In other words: only those solutions of which the value of the cost function is less than or equal to the cost function's value of the current solution are accepted. However, to avoid getting stuck in local optima, this can be combined with other acceptance criteria, such as *simulated annealing* or *tabu search* that add some memory to the search. In simulated annealing, the search starts at a (high) initial temperature, which is lowered (and reheated when the search gets stuck in a local optimum) during the execution of the algorithm. Dependent on the value of a probabilistic function that depends, among other things, on this temperature, a move is accepted or not. Tabu search tries to escape from local optima by keeping track of the characteristics of previous accepted moves. To avoid that the search returns to a previous visited local optimum, the moves that lead to this optimum are forbidden ('tabu'). These and other heuristics will be applied and further explained in the thesis. The domain model, the solution space representation and corresponding moves, and the cost function are problem dependent. However, for the problems studied in this thesis, many problem dependent classes could be reused without any problem.

To further prove the flexibility of the approach, we reworked the existing meta-heuristic framework to a hyper-heuristic framework, and applied the latter to solve the patient admission scheduling problem as well as the KaHo Sint-Lieven exam timetabling problem. Once this was established, only minor modifications to the approach were required to tackle the examination timetabling benchmarks.

1.2 Contributions

The main contributions of this thesis are:

- a detailed description and a model for several real-world timetabling applications;
- a ***flexible*** heuristic approach. It can be applied to solve a wide range of timetabling problems. To substantiate this, we solve three complicated real-world university timetabling problems [66, 70, 72] with constraints different from the ones that appeared in the literature. Besides the real-world university timetabling problems, also timetabling problems from health care are tackled with a similar heuristic approach [21, 67, 73];
- the introduction of a ***new type of problem decomposition*** [66]. Due to the complexity of the real-world university course timetabling problem, we are compelled to apply pillars to reduce the problem size. The pillar concept, which is a generalization of Kingston's [102] tile concept, is established to represent weekly and non-weekly repeating lectures;
- a ***multiple stages approach*** in which extra constraints are gradually added. The best solution obtained in a previous stage is applied as the starting solution for the next phase in which an extra constraint is added. In less or equal time, better solutions are obtained [66] than when all the constraints are solved in one go;
- the ***robustness*** of the approach. The performance of the heuristics are tested on benchmark instances. The hyper-heuristic approach for examination timetabling improves the best results in the literature for four out of the 13 exam timetabling benchmark instances and finds for one benchmark instance an equally good solution as the best solution in the literature (Toronto version I data set) [70]. Since all 13 benchmark data instances are solved with exactly the same parameters and low-level heuristics, this demonstrates the robustness of the approach;
- the introduction of a ***new scheduling problem*** to the operational research community: Patient Admission Scheduling [21, 73]. Besides a detailed description of the problem, test instances and a validation tool are foreseen. Although it is called a scheduling problem, it resembles timetabling very much, as we will show in Chapter 7.

1.3 Structure of the Thesis

We start by sketching the problem domain background by reviewing the literature on university timetabling, nurse rostering, and patient admission scheduling in Chapter 2. The different real-world cases are described in full detail in the subsequent chapters. In Chapter 3 the university course timetabling problem

at KaHo Sint-Lieven is described. A decomposition approach and a multi-neighbourhood tabu search algorithm that gradually adds extra constraints to the cost function are introduced. Besides the course timetabling problem at KaHo, which is essentially curriculum-based, a solution to the post-enrolment based course timetabling problem of K.U.Leuven is presented in Chapter 4. To this purpose, we employ a hybrid guided local search algorithm in which the weights of the cost function are dynamically adapted during the search. The last university timetabling problem is described in Chapter 5, in which we show that on the one hand the examination timetabling problem at KaHo Sint-Lieven and on the other hand the Toronto examination benchmarks and the instances of the examination timetabling track of the Second International Timetabling Competition are solvable with almost the same hyper-heuristic method. These benchmarks are addressed in order to evaluate the hyper-heuristic's performance. In the following two chapters, the focus is on timetabling problems from health care. In Chapter 6, the nurse rostering problem of the internal medicine ward of Maria Middelares in Gent is described and tackled with a multi-neighbourhood tabu search algorithm. The patient admission scheduling problem is described in Chapter 7. We solve the patient admission scheduling problem with integer programming, tabu search hybridized with token ring, variable neighbourhood descent, and hyper-heuristics respectively. In Chapter 8 we conclude and give pointers for future work.

Chapter 2

Literature Overview

2.1 University Timetabling: Course and Examination Timetabling

University timetabling is an umbrella term which can be thought of as comprising two disjunct categories, i.e. course and examination timetabling. One of the major difference between both is that in course timetabling only one lecture per room - time slot combination is allowed, while in exam timetabling several exams can be assigned to the same room in the same time slot. Although this sounds as a universal rule, exceptions do exist. We have knowledge of a real-world situation in which two different lab sessions can take place in one particular lab room. Another point of difference is that in examination timetabling students often want a timetable that is spread out as much as possible, so that they have sufficient study time, while in course timetabling students prefer more condensed timetables. Course timetabling can further be divided in two separate groups [1]: post enrolment and curriculum based course timetabling. The former corresponds to the Anglo-American education system, while the latter is for example more common in Europe, and Belgium in particular. In the Anglo-American system, students can select from a whole range of courses when they register, while in the Belgian system all students taking the same curriculum follow the same courses. This leads to particular timetables for every student in the post enrolment based timetabling case, and to common timetables for students taking the same curriculum in the curriculum based course timetabling case. Although not very common, the same division can be made for examination timetabling (see [114, 115]).

2.1.1 University Course Timetabling

In the literature, different university course timetabling problems are described in the sense that various (hard and soft) constraints and solution approaches are considered. Probably, the best known and studied university course timetabling problem was the subject of the first International Timetabling Competition (ITC) of the Metaheuristics Network [2] in 2002. The problem consists of scheduling (weekly recurring) events into 45 (non-overlapping) time slots (this is: 5 days multiplied by 9 lecture hours per day) and in appropriate and large enough rooms taking into account the following hard constraints:

- a student cannot attend more than one lecture in the same time slot;
- a room cannot contain more than one lecture at the same time;

and the following soft constraints:

- a lecture is not scheduled in the last time slot of the day;
- a student cannot take more than two lectures in a row;
- a student cannot be assigned to a single lecture in a day.

Several timetablers accepted the challenge and submitted their solutions. The best solutions are based on local search algorithms, such as hill climbing, tabu search, simulated annealing, great deluge, and combinations of these algorithms. Di Gaspero and Schaerf [76] combine hill climbing and tabu search, while Cordeau et al. [98] execute a pre-processing step to obtain feasible solutions. With the feasible solution as a starting point, a tabu search algorithm is applied to find better quality solutions (satisfying the soft constraints to the best possible extent). Burke et al. [27] applied the great deluge algorithm to the problem. This latter algorithm can be compared to simulated annealing, but instead of relying on a probability measure to decide whether to accept a move or not, a move is accepted when the fitness of the candidate solution is below a certain boundary condition, which gradually lowers during the search. The authors claim that the advantage of their approach is that only two parameters have to be fine-tuned, namely the search time and an estimate of the quality of the solution. P. Kostuch, the winner of the contest [103], also includes a pre-processing step to find a feasible solution, after which the two following stages start. Both are based on a simulated annealing search algorithm. The first algorithm is applied to find better quality solutions that satisfy the two soft constraints involving students, by optimally sequencing the time slots. To improve the overall solution quality, the second simulated annealing algorithm swaps pairs of events between time slots, provided that the swaps do not lead to infeasible solutions.

Other researchers used the instances of the first International timetabling Competition as a test bench for their algorithms. Abdullah et al. [7, 8] compare a randomized iterative improvement algorithm and a variable neighbourhood local search with the results from the competitors. For small instances the VNS algorithm generates results that are comparable to the best results of the competition, while the randomized iterative improvement algorithm generates for seven out of the eleven datasets equal or better results. Socha et al. [134] describe two ant algorithms, of which the Max-Min Ant System leads to the best solutions for large instances.

In 2007, the second International Timetabling Competition was launched [1]. In contrast to the first competition, three tracks are considered:

- an extension of the post-enrolment university course timetabling problem that was the subject of the first timetabling competition;
- a new curriculum based university course timetabling problem; and
- a post-enrolment based university examination timetabling problem (see also Section 2.1.2).

For the post-enrolment university course timetabling track [107], the following two hard constraints are added to the hard constraints of the first competition:

- some of the lectures can only be scheduled in specific time slots;
- some of the lectures can only be scheduled in a specific order.

The soft constraints for both competitions are the same.

Regarding the curriculum-based competition track [74] of the Second International Timetabling Competition, the considered problem is to schedule lectures within a number of rooms and time slots. The hard constraints are:

- all lectures of a course should be scheduled;
- a room cannot contain more than one lecture at the same time;
- a student following a curriculum cannot attend more than one lecture in the same time slot and a lecturer cannot teach more than one lecture in the same time slot;
- some of the lecturers have periods in which they are not available to lecture, meaning that none of their lectures should be assigned in these periods.

The soft constraints of the curriculum-based competition track are;

- the capacity of the assigned room should be taken into account, meaning that no more students can be assigned to a class room than there are seats available;
- two lectures of the same course in the same week should be scheduled with a minimum number of days between them;
- the timetable of the same curriculum should be as tight as possible;
- all lectures of the same course should be assigned to the same class room.

The best performing algorithms for the university course timetabling tracks are all based on meta-heuristics and combinations and hybridizations thereof. More information on the Second International Timetabling Competition can be found in McCollum et al.'s [115] overview paper.

Probably, both these university course timetabling benchmarks will become more and more important, taking over the leading role of the benchmarks of the First International Timetabling Competition, since they can be considered more 'realistic' because of their extra hard and soft constraints.

Apart from the above mentioned metaheuristic approaches, which we will describe in depth in the following chapters, that were used to solve the university timetabling problems as they were defined by the two international timetabling competitions, other publications exist on other university timetabling problems with similar or different constraints for which other approaches have been applied, such as:

- integer programming [64, 132, 140]. Integer programming is a special case of linear programming, but with the additional constraint that only integer values are allowed. In linear programming [35], an optimization problem is modelled as a set of linear expressions in the decision variables representing the objective function and the constraints.
- constraint logic programming [54, 89, 128]. Constraint logic programming is actually an extension of logic programming with constraint programming. The constraints are incorporated in the clauses of the logic program.
- reduction of the problem to graph colouring [13, 34, 69]. The idea is to translate the timetabling problem to a graph, in which the nodes (or vertices) correspond to the lectures and the edges between the nodes correspond to the constraints. An edge between two nodes is drawn when both lectures have students or lecturers in common, meaning that both lectures cannot occur in the same time slot. The graph colouring problem is the problem of assigning a limited number of colours to the nodes of the graph in such a way that no two nodes connected by an edge have the same colour. Rephrased,

this corresponds to deciding on the minimum number of time slots needed to obtain a solution in which there are no clashes. Graph colouring is explained in more detail in Section 2.1.2.

- case-based reasoning [36, 38, 45]. Case-based reasoning (CBR) is an AI method that tries to solve new problems by applying knowledge gained from previous experiences. In contrast to other methods that claim to be problem-independent but contain lots of problem specific information, Burke et al. [38] contend that CBR does not contain this kind of problem-specific information, which makes it suitable for solving different kinds of timetabling problems.

- hyper-heuristics [25, 37, 39]. In contrast to meta-heuristics, in which the search is executed on the space of solutions, hyper-heuristics operate on a search space of heuristics. The idea behind hyper-heuristics is to automate the selection of low-level heuristics, usually by applying machine learning techniques. In general, low-level heuristics can be built so that each of them individually can solve one specific part of the problem. Hyper-heuristics can exploit the particular properties of each low-level heuristic by combining them in a specific order. A typical hyper-heuristics framework consists of some low-level heuristics selection mechanism and a set of low-level heuristics. These low-level heuristics can be either perturbative (changing only small parts of the solution) or constructive (constructing a solution). The motivation for using hyper-heuristics is that they can be applied to a broad range of problems. In [25], Burke et al. apply their method to a nurse rostering and a university course timetabling problem. In chapters 5 and 7 hyper-heuristics is employed, and an in depth description is provided.

- neural networks [48, 49]. Carrasco et al. base their work on the findings of Hopfield and Tank [95], who showed that artificial neural networks can be used to solve combinatorial optimization problems. They proved that if the neural network satisfies specific constraints, it converges to stable states which can be shown to correspond to the local optima of a so-called neural energy function. Carrasco et al. formulate such a neural network energy function that contains a weighted sum of the violations of the problem's hard and soft constraints. Minimizing this sum provides neural network states which correspond to feasible or almost feasible solutions of the timetabling problem satisfying most of the soft constraints.

Several literature overviews on course timetabling have appeared: [5, 17, 24, 44, 123, 130]. Lewis [106] in particular classifies the metaheuristic techniques for solving university timetabling problems into three categories:

- one-stage optimization algorithms in which the hard as well as the soft constraints are satisfied simultaneously;

- two-stage optimization algorithms where one first tries to obtain a feasible solution, prior to starting the second phase in which one tries to satisfy as many soft constraints as possible;
- algorithms in which the hard constraints are temporarily relaxed.

We apply one-stage optimization algorithms to solve the post-enrolment university course timetabling problem of K.U.Leuven in Chapter 4, the nurse rostering problem in Chapter 6 and the patient admission scheduling problem in Chapter 7. A two-stage optimization algorithm is applied to solve the examination timetabling problem in Chapter 5. The curriculum based university course timetabling problem of KaHo Sint-Lieven is tackled in multiple stages in Chapter 3. Since we gradually add more constraints to the cost function and thus temporarily relax the rest of the constraints, this type of algorithm fits more in the third category of Lewis' metaheuristics classification.

2.1.2 Examination Timetabling

In general, the literature on examination timetabling can be divided into two categories: one part concentrates on solving real-world examination timetabling problems, while the other part focuses on improving benchmark results. Benchmarks give an objective comparison of the search algorithms' performance, on the condition that the same objective function is applied in all algorithms.

The most studied examination timetabling benchmark is probably the one that was introduced by Carter et al. [51]. This benchmark consists of 13 (simplified) real-world examination timetabling problems of different universities around the world. Unlike most real-world problems, the benchmark does not consider rooms and lecturers. It is assumed that rooms have infinite size. Until the introduction of the examination timetabling benchmarks of the Second International Timetabling Competition in 2007, the Toronto data sets were de facto standard test set for examination timetabling. The idea is to construct for every student clash-free examination timetables and to spread-out the exams as much as possible. If only the first constraint is taken into account, the problem reduces to a graph colouring problem [31]: exams correspond to nodes and edges correspond to exams that have at least one common student. Assigning exams to time slots corresponds to assigning colours to the nodes, such that connected nodes have different colours. Minimizing the number of time slots while satisfying the clash free constraint corresponds to minimizing the number of colours needed to colour all the nodes (exams). Carter et al. [51] introduced five basic sorting criteria for assigning exams to the time slots, namely: largest degree, saturation degree, largest weighted degree, largest enrolment, and random ordering. The specific properties of the 13 Toronto benchmark instances are summarized in Table 2.1. Since their

introduction different versions of the benchmark instances are in circulation. The version used in this thesis (see Section 5.1) is introduced as version I in [127] and is the most common version of the benchmark instances used in the literature. The values in the conflict density column (fifth column of Table 2.1) are the ratios of the total number of non-zero elements to the total number of elements of the 'conflict matrix'. The conflict matrix contains the number of common students for every exam. The conflict density value gives an indication of the number of conflicts between the individual exams. In the last column the minimum number of time slots needed to obtain a feasible solution is presented. The 'Benchmark Data Sets in Exam timetabling' website [3], provides a collection of several examination timetabling benchmark data instances and a validator to evaluate the solutions of the Toronto benchmark data sets. Initiatives like these are welcomed since they provide a means to check the correctness of the objective function (see also the discussion on measurability and reproducibility in [131]).

	# students	# exams	# enrolments	conflict density	# time slots
car91	16925	682	56877	0.13	35
car92	18419	543	55522	0.14	32
ear83	1125	190	8109	0.27	24
hec92	2823	81	10632	0.42	18
kfu93	5349	461	25113	0.06	20
lse91	2726	381	10918	0.06	18
pur93	30032	2419	120681	0.03	42
rye92	11483	486	45051	0.07	23
sta83	611	139	5751	0.14	13
tre92	4360	261	14901	0.18	23
uta92	21266	622	58979	0.13	35
ute92	2749	184	11793	0.08	10
yor83	941	181	6034	0.29	21

Table 2.1: Problem size of the 13 data sets of the Toronto benchmark (version I). The fifth column gives an indication of the number of conflicts between the exams. It is the ratio of the number of non-zero elements in the conflict matrix to the total number of conflict matrix elements. In the last column the minimum number of time slots needed to obtain a feasible solution is presented.

Several papers are published about solving the Toronto benchmarks (see [127] for an overview), but we choose to only discuss three of them. These three articles in particular are chosen, since they have common grounds with our approach of the Toronto examination timetabling problem (see Chapter 5).

Thompson and Dowsland [138] investigate the characteristics of a robust simulated annealing algorithm. It turns out that next to the cooling schedule, also the way the neighbourhoods are constructed and sampled is important. The Kempe chain

(see Section 5.3) based neighbourhoods outperform the other neighbourhoods. Thompson and Dowsland tackle the examination timetabling problem in two stages: first the problem is solved to feasibility while in the second stage the quality of the obtained solution is improved. Both stages apply different neighbourhoods. The second stage only employs neighbourhoods that do not violate any hard constraints. We will take these findings into account when we tackle the examination timetabling problem in Chapter 5.

In [100], Kendall and Hussin apply a tabu search based hyper-heuristic approach in which the low-level heuristics execute small changes on the current solution. The heuristic selection criterion uses a tabu list to select a non-tabu heuristic in the next iteration. The tabu list length is equal to the number of heuristics. Although they do not improve on the best results in the literature, Kendall and Hussin demonstrate that it is possible to find good enough solutions with a very general approach that can easily be applied to other problems. We will adopt a similar approach and solve different problems (two exam timetabling variants and a patient admission scheduling problem) with one general hyper-heuristic approach.

Burke and Bykov [26] introduce the late acceptance strategy and apply it to Carter et al. 's examination timetabling benchmarks. For some of the instances new best results are obtained. We will incorporate late acceptance among other acceptance criteria in our hyper-heuristic approach.

Since their introduction, the instances of the examination timetabling track of the Second International Timetabling Competition [1] gains more and more popularity, and probably will become the de facto standard. In contrast to Carter's Toronto benchmarks, these benchmarks enclose more types of constraints, which makes them actually more interesting and realistic. For example, the room capacity constraint is now taken into account. The hard constraints are:

- a student cannot attend more than one exam per time slot;
- an exam has to be organized in one single room;
- the capacity of the room cannot be exceeded;
- some exams require rooms with special properties;
- since every exam has a duration, its duration should be less than or equal to the duration of the time slot to which it is assigned;
- some exams should be scheduled before, after, together or absolutely not together with other exams.

The following soft constraints should be taken into account:

- two exams taken by the same student should not be assigned to the same day or to two consecutive time slots;
- exams should be spread out as much as possible;
- exams with different durations should not be assigned to the same room;
- larger exams should be scheduled earlier in the timetable;
- some of the time slots in the examination timetable should be avoided;
- some of the rooms should be avoided for examination.

The problem size of the 12 data sets is presented in Table 2.2.

	# students	# exams	# rooms	conflict density	# time slots
Instance 1	7891	607	7	0.0505	54
Instance 2	12743	870	49	0.0117	40
Instance 3	16439	934	48	0.0262	36
Instance 4	5045	273	1	0.1500	21
Instance 5	9253	1018	3	0.0087	42
Instance 6	7909	242	8	0.0616	16
Instance 7	14676	1096	15	0.0193	80
Instance 8	7718	598	8	0.0455	80
Instance 9	655	169	3	0.0784	25
Instance 10	1577	214	48	0.0497	32
Instance 11	16439	934	40	0.0262	26
Instance 12	1653	78	50	0.1845	12

Table 2.2: Problem size of the 12 data sets of the examination timetabling track of the Second International Timetabling Competition. The conflict density column indicates the number of conflicts between the exams. It is the ratio of the number of non-zero elements in the conflict matrix to the total number of conflict matrix elements.

The five finalists of the examination track of the competition have applied different approaches, such as:

- a constraint solver, in which in the first stage a complete feasible solution is constructed, afterwards hill climbing as well as great deluge are applied to improve the solution [119];
- a Greedy Randomized Adaptive Search Procedure, followed by simulated annealing, and integer programming [58];
- a constraint satisfaction solver combined with tabu search and iterated local search [14];

- Drools Solver, which is a combination of the JBoss Drools rules engine, that is used for the evaluation, and tabu search [68] and
- a heuristic method that is inspired by cell biology [125].

After the end of the competition, the organizers [114] of the examination timetabling track published a paper in which they show that for all 12 instances feasible solutions can be obtained. Also, compared to the five finalists of the examination timetabling track, they improve on 6 out of the 12 benchmarks. Their approach consists of two parts: first a feasible solution is constructed in the constructive phase and afterwards this feasible solution is further improved with an extended great deluge algorithm, to which a reheating mechanism is added to escape from local optima.

Besides the benchmarks, considerable attention has been paid to real-world examination timetabling problems [15, 52, 79, 91, 101, 109, 137].

For a complete review on examination timetabling we refer to [127].

2.2 Health Care: Personnel Planning and Patient Admission Scheduling

In the first part of this literature review we briefly mention some of the best studied health care examples that are solved using artificial intelligence and operations research techniques. In the two following sections 2.2.2 and 2.2.3, the relevant literature concerning personnel planning and patient admission scheduling is reviewed.

2.2.1 Artificial Intelligence and Operations Research in Health Care

The application of operations research and artificial intelligence in health care is widespread. Well-known applications are for example nurse rostering (see [31] for a review), scheduling operating rooms [19], emergency room physicians [50], and outpatient's appointments [99]. Spyropoulos [135] provides an overview of planning and scheduling approaches in the hospital management and therapy planning domain.

2.2.2 Personnel Planning

Bradley and Martin [22] distinguish three stages in the personnel planning process:

- staffing, which is the long-term process of determining how many personnel must be employed by the organization to provide a predetermined level of service;
- personnel scheduling, which is the process of converting expected daily work force requirements into precise scheduling assignments; and
- allocation, which is the actual assignment of scheduled personnel to work sites.

The planning horizon for the three stages ranges from years and months (in the case of staffing), over weeks (in the personnel scheduling case) to days and hours (in the allocation phase). The latter phase is the most relevant for our research, since it corresponds to the operational level in which people are assigned to tasks.

Ernst et al. [82] present an detailed review of applications, methods, and models in personnel scheduling. Ten different personnel scheduling domains are covered in the overview, of which three are considered economically much more relevant than the rest: crew scheduling, call center scheduling, and health care personnel scheduling. Crew scheduling differs from the other personnel scheduling problems in that locations play a significant role in the problem description. All aspects of transportation related problems have been widely examined over the years [23, 83]. Also the call centre sector has been the subject of many research papers. A review by Grossman et al. [88] describes the problems and solutions in call centers for outbound and inbound systems. Personnel planning in the health care sector is also well analyzed throughout the literature. Nurse rostering in particular is a complex problem for which automated decision support is still not common. Burke et al. [31] describe various aspects of the nurse rostering problem in their 2004 review paper. They classify models and approaches and formulate challenging directions for future research. A simplified case on police services, with specified shift sequences, is studied by Taylor and Huxley [136]. It considers service types with 4 days of 10 hours or 5 days of 8 hours each followed by 3 consecutive days off. However, real-world problems often are much more complex. Laporte [105] for example describes constraints concerning rotating schedules. Tien and Kamiyama [139] present a common framework with which they classify approaches to manpower scheduling. The framework consists of five stages: temporal manpower requirements, total manpower requirement, recreation blocks, recreation/work schedule, and shift schedule.

2.2.3 Patient Admission Scheduling

Hospital admission scheduling is well reported on in the research literature. This is attributed to the fact that governments want to reduce spending in health care, so that hospitals are compelled to use their resources more efficiently.

Hospital admission scheduling is the process of assigning patients to beds in such a way that the medical concerns and personal wishes are fulfilled as much as possible. This process can be carried out at three different levels: strategic, tactical or operational. According to [144], the operational level applies concrete rules for the admission of patients, which follow the strategic and tactical decisions that are made by the hospital management. A strategic decision could involve maximizing the usage of the resources (MRI scanner, operating theatres, nurses, beds, etc.) in the hospital or minimizing the waiting time of the patients [84, 144].

Numerous articles can be found that address tactical and strategic levels, but to the author's knowledge the admission of patients on the operational level is not widely reported on. In the two subsequent sections we discuss the literature on the tactical and strategic levels of admission scheduling and also on hospital capacity planning, since they influence the bed occupancy.

Admission Scheduling

Gemmel and Van Dierdonck [84] review the available literature on admission scheduling and conclude that not only the bed utilization should be taken into account, but also information on the availability and use of resources such as operating rooms, nurses, etc. From the reviewed literature they gather two conditions that have to be fulfilled before one can really talk about scheduling admissions, namely

- the scheduler should have access to valid information about the availability of resources;
- the scheduler should have (limited) authority to change the admission dates.

When these conditions are not fulfilled the admission service of the hospital administers patients instead of scheduling them. In the same article, Gemmel and Van Dierdonck survey how 83 Belgian hospitals perceive admission scheduling in practice. It shows a large gap between theory and practice. If the two above conditions are taken into consideration, the survey reveals that only one third of the surveyed Belgian hospitals schedule their admissions. The admission services of the remaining two thirds of the hospitals actually carry out only an administrative function. From the survey it is also clear that most hospitals have no information on the expected length-of-stay or on the expected workload, which is - of course - crucial when scheduling admissions.

In the extensive literature review of Smith-Daniels et al. [133], inpatient admission scheduling is classified as a facility allocation decision, in which it is assumed that the medical services' size already have been defined, and the resources are

constrained by the available beds. The maximization of bed occupancy levels depends on three variability sources:

- the admission of emergency patients: in general some slack capacity is available to admit emergency patients.
- the length-of-stay (LOS) of patients: the expected LOS of a patient is an estimate - based on empirical data, or on a probability distribution - that can differ from the actual duration.
- the requirements of the patient service-mix: dependent on the clinical picture of the patient, he/she may require, besides a bed, different other resources, such as telemetry, specific nursing care, surgical rooms, etc. Focusing solely on bed occupancy can have as a consequence that the other resources are over or under used. Ideally, the work load of the nurses should also be taken into consideration when admitting patients.

Although not directly related, the problem that is discussed and solved by Hans et al. [90] bears some resemblance with the patient admission scheduling problem. An algorithm is described that at the same time assigns elective patients to operating rooms, optimizes the operating theatre utilization and minimizes the total overtime of the surgeries. Both constructive and local search heuristics are applied to solve the problem. All the proposed optimization techniques lead to a better utilization of the operating theatres, compared to the schedule that was originally made by the (human) specialists.

Another, somewhat similar problem, is described by Ogulata et al. [121]. It involves generating a weekly work schedule for physiotherapists. Patients are selected from a list and scheduled on a day of the week, according to the priority and the duration of their treatment. They are scheduled in such a manner that the workload of the physiotherapists is equally balanced. Patients' preferences concerning the day of the week for treatment are not taken into consideration. Ogulata et al. solve the problem using the mathematical programming tools GAMS and MPL.

Marinagi et al. [111] describe the patient examination scheduling problem which precedes the patient admission scheduling problem. Before a physician can diagnose a patient, several examinations have to be performed, such as X-rays, ultrasound images, blood samples, etc. The goal is to minimize the patient's examination time in the hospital and to maximize the utilization of resources. The problem is solved by a combination of agent technology, a hierarchical planner which supports the decomposition of complex tests into smaller parts and a scheduler. Subject to the different actions that need to be executed, which is the result obtained by the planner, the scheduler tries to assign the actions to appropriate time slots.

Vermeulen et al. [142] describe a CT-scan scheduling problem, which will - when solved to optimality - contribute to efficient patient scheduling. Often central diagnostic resources such as CT-scans are bottlenecks in a hospital environment and the goal is to minimize the patients' waiting time and to maximize the resources' utilization. Patients are scheduled according to their arrival order, and for every patient a random free time slot is selected, that fits into the patient's time window.

The goal in the rehabilitation patient scheduling paper of Chien et al. [57] is similar to the one in [142]: reduce the patients' waiting times and increase the equipment usage. Since some of the rehabilitation therapies require a specific order of execution, this problem can be seen as a hybrid shop scheduling problem in which the medical resources correspond to the machines and the patients to the jobs. Chien et al. solve the problem with a genetic algorithm.

Vermeulen et al. [143] apply agent technology to solve the patient appointment exchanging problem. Patients are represented by agents that try to decrease the patients' waiting time. Initially, patients are assigned to a schedule that consists of several small appointments which are attributed a time slot. Each agent will try to exchange appointments with other agents in order to obtain a better schedule. An agent only accepts when it is not-worse off than before.

Vermeulen et al. [141] describe the on-line scheduling of outpatients that have a combination of appointments in different departments. The goal is on the one hand to optimize the appointments of the outpatients and on the other hand to optimize the departments' efficiency. This is a complex problem since it needs a lot of collaboration between the departments of the hospital. Similar to [143] agent technology is applied to model the departments and the outpatients. The agents communicate with each other to find the best time slot for each party involved.

The scheduling problems dealt with in the above papers consist of assigning patients to time slots in which they can be examined or diagnosed, whereas in the patient admission scheduling problem we are interested in assigning patients to beds. The size of the time slots differs too. In the patient admission scheduling case, the smallest time slot is 24 hours, whereas in the other papers the smallest time slot is 15 minutes. However, the major difference between the patient admission scheduling problem and the other problems found in the literature, is that we are also confronted with the problem of consecutiveness. This means that patients admitted for more than one night to the hospital are preferably assigned during their stay to the same bed. This extra constraint, which is not an issue in the other problems from the literature, considerably complicates the problem.

Hutzschenreuter et al. [96] describe an agent-based patient admission scheduling application. They opt to employ agent technology since the problem they consider is highly decentralized. To analyze and evaluate different policies concerning bed scheduling, what-if scenario's are applied. Patients are automatically assigned to

beds with a brute-force optimizer. The time needed for this optimizer to assign patients to a small number of beds in a planning horizon of one year was 12.8 hours. No further details are given about the brute-force algorithm.

Creemers et al. [62, 63] develop an appointment driven model in which on a strategic level the performance measures can be calculated. The most important performance measures in appointment driven systems are: the time a patient spent in the waiting list, the waiting time at the hospital and the probability and amount of working overtime for the physicians at the hospital. The authors combine a vacation queueing model and an appointment system.

Hospital Capacity Planning

Green [87] discusses both the problem of delays in emergency departments, and the problem of deciding on the available number of beds in the hospital. She argues that using bed occupancy levels as a base for determining the number of beds in the hospital is not accurate. Most reported occupancy levels in the literature are taken at midnight, when the hospital usually has a lower occupancy than during the day. This can lead to a discrepancy of 20% between the reported and the real occupancy levels. Another problem that arises is that the occupancy levels are yearly averages, and as such do not take into account periods with higher occupancy (e.g. due to flu epidemics, disasters, etc.) or lower occupancy (due to holiday periods, etc.). The occupancy level is even not uniform over a week: in general there are less patients admitted to the hospital in the weekend than on weekdays. Green [87] suggests to use - instead of occupancy levels - the average delay before a patient can be admitted. One of the solutions that is proposed in the article is to increase the bed flexibility. This means for example that more beds should be equipped with extra facilities.

Harper and Shahani [92] describe a simulation model in which bed occupancy and patients' refusals can be calculated, taking into account different what-if scenarios. Akcali et al. [10] describe a network flow approach that assists in determining the optimal bed capacity in hospitals. It takes into account the hospital budget and the maximum number of days a patient is on the waiting list before being admitted to the hospital. It is however assumed that all the beds in the hospital are identical.

Educational Timetabling

Chapter 3

Curriculum Based University Course Timetabling

3.1 Introduction

The university course timetabling problem is the process of assigning lectures into 'room-time slots' combinations, taking into account a set of hard and soft constraints. A solution is feasible when none of the hard constraints are violated. Neither students nor lecturers, for example, can physically be in two different places at the same time. This is what Lewis [106] calls the event-clash constraint. In contrast to hard constraints, violations of soft constraints are allowed but if they are satisfied as much as possible the solution's quality is experienced as 'better'. An example of a soft constraint can be to avoid time gaps between lectures on the same day.

The main contribution of this work consists of a multi-stage local search approach to solve a non-weekly recurring real-world timetabling problem with overlapping time slots. In every stage of the method an extra constraint is added, and the constraints are solved in a progressive manner:

- in the first stage the local search algorithm tries to find a solution satisfying the first constraint;
- the solution found in the first stage is used as the initial solution for the second stage in which an extra constraint is added;
- this process is repeated until there are no constraints left to evaluate.

We describe the university course timetabling problem as it is perceived at the KaHo Sint-Lieven Industrial Engineering Department in Section 3.2. The timetabling problem is quite different from most problems encountered in literature, since it deals with overlapping time slots and subjects that are not necessarily covered on a weekly basis. We are not aware of any similar problems in the literature. The different steps of the local search approach are discussed in Sections 3.3, 3.4 and 3.5. In Section 3.6 we discuss the obtained solutions and we conclude in Section 3.7.

The research reported on in this chapter appeared in an adapted version as *P. De Causmaecker, P. Demeester, and G. Vanden Berghe. A decomposed metaheuristic approach for a real-world university timetabling problem. European Journal of Operational Research, 195(1):307-318, May 2009, .*

3.2 The KaHo Sint-Lieven Course Timetabling Problem

3.2.1 Problem Description

In this section we describe the KaHo Sint-Lieven School of Engineering university course timetabling problem in detail. The KaHo problem is both described in words and mathematically. However, note that we do not employ the mathematical formulation to tackle the problem. It is included for clarity. An academic year consists of 2 semesters of 12 weeks each. A week consists of 5 lecture days. A typical lecture day starts after 8 AM in the morning and finishes before 7 PM in the evening. A day is divided into different time slots, which do not necessarily have the same duration. Lectures, for example, take 90 minutes and the duration of lab sessions ranges from 190 to 290 minutes. The time slots have a fixed begin and end time and cannot be changed during the search for a timetable. Table 3.1 presents the different time slots of a lecture day. Note that some time slots overlap. Suppose that, for example, a lecture day starts with a lab session (with a duration of 190 minutes) at 8:10 AM. The next lecture for one of the resources involved (student group, lecturer or room) cannot be scheduled before 11:20 AM. The overlapping time slots will introduce specific constraints to the timetabling problem. Several lectures for the same group of students are generally organised either in the morning or in the afternoon, while only one lab session is organised in the morning or afternoon. Possibly, this lab session can be preceded or followed by a theoretical lecture. No lectures or lab sessions can be scheduled between 1 and 2 PM.

The first 3 semesters are common for all engineering students. At the start of the fourth semester, students have to choose between different disciplines such

Time slots			
Begin	**End**	**Begin**	**End**
08:10	09:10	14:00	15:30
08:10	09:20	14:00	17:10
08:10	09:40	14:00	17:40
08:10	11:20	14:00	18:20
08:10	12:30	14:00	18:40
08:10	13:00	15:40	17:10
09:20	13:00	15:40	17:40
09:50	11:20	15:40	18:40
09:50	13:00	17:10	18:40
11:30	13:00		

Table 3.1: Overview of the 19 fixed time slots within a lecture day.

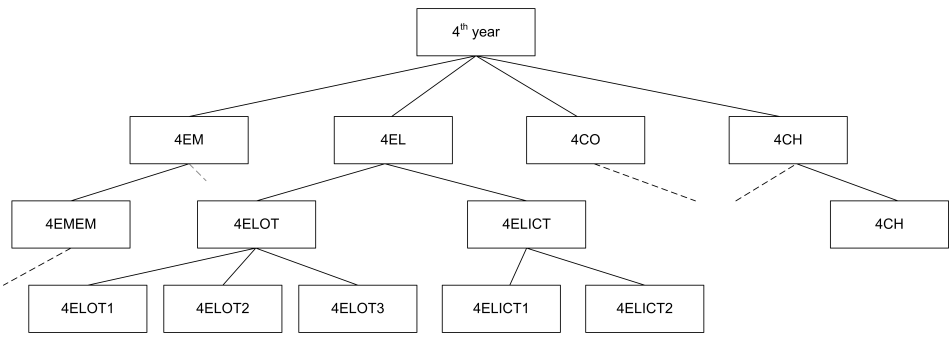

Figure 3.1: Graphical representation of the student group hierarchy of the Engineering Master at KaHo Sint-Lieven.

as electro mechanics, construction, chemistry, and electronics-ICT. Students are grouped according to the compulsory courses that they take. A course typically consists of several lectures and lab sessions. Lectures usually occur 12 times during a semester, but there exist lectures that deviate from this. Lab sessions have an occurrence that can range from 2 to 10 times during a semester. Each group of students is further divided into smaller lab session groups, mainly because students need more individual coaching for lab sessions. This leads to a hierarchy among the different student groups (see Fig. 3.1 for an example of the master's hierarchy). This hierarchy of student groups has to be taken into account during the evaluation process since we will consider student groups and not individual students and this can cause a violation of the above introduced event-clash constraint. The assignment of students to lab groups is given and is not part of the automated timetabling application. Most lab sessions can only take place in special purpose

rooms. This involves an important timetabling constraint.

In contrast to some other institutes [11], the assignment of the modules to the lecturers is not part of the timetabling problem. Lecturers know in advance which lectures and lab sessions they have to cover. In the first 3 semesters, most lab sessions (e.g. programming, technical drawing, physics or chemistry) are covered by 2 or more lecturers, depending on the number of students.

Generally speaking, when automating the process of scheduling courses, the following model is considered: lectures covered by lecturers and taken by student groups have to be scheduled into rooms and time slots, taking into account that:

- **C1** A lecturer or a student group cannot be assigned to more than one lecture in the same time slot.
- **C2** A room cannot hold more than one lecture at the same time.
- **C3** Student groups have to fit into the assigned class room.
- **C4** Some lab sessions can only take place in especially equipped rooms.

The above constraints resembles well the constraints described in general university course timetabling papers [12, 59, 60, 120, 129]. However, we are not aware of any problems similar to that of the KaHo Sint-Lieven Industrial Engineering Department. It differs from general descriptions in the literature, since it also deals with the following constraints:

- **C5** A resource (room, lecturer or student group) cannot be assigned to more than one lecture in overlapping time slots.
- **C6** Every course needs to be organized as many times as demanded, not less, not more.

A result of constraint **C6** will be that, since not every course is organized every week of the semester, timetables will differ from week to week.

When studying different manually constructed timetables we observed that not all of the above constraints were satisfied. A few lecturers for instance had to cover in the same time slot two different lab sessions in the same room. Upon inquiry it appeared that this was done on purpose in special occasions. All students of the third bachelor year have to execute a final bachelor project. Some of the lecturers supervising a bachelor project prefer that the fortnightly meetings are scheduled at the same moment with the other lab sessions they lecture. When the students of the first two bachelor years execute their lab session assignment, the lecturers meet with the third year bachelor students. In the beginning of the academic year it

happens that constraint **C3** is violated. Sometimes this is solved by moving chairs from another room to the assigned class room. After a few weeks this problem is automatically solved, when some of the students start skipping classes.

3.2.2 Formal Problem Description

In university course timetabling, C courses covered by L lecturers and taken by S student groups (grouped according their curriculum) have to be scheduled into R class rooms and T time slots taking into account different constraints. Each course c consists of E_c individual events (e.g. lectures, lab sessions). The number of students taking a course c is denoted by S_c. The capacity of a room r is denoted by CAP_r.

The decision variables are

$$x_{rtc} = 0 \; or \; 1 \tag{3.1}$$

with $(r = 1, .., R; t = 1, .., T; c = 1, .., C)$ and where $x_{rtc} = 1$ if a lecture of course c is assigned to room r in time slot t.

All E_c events of a course c need to be scheduled. This is expressed in Equation 3.2.

$\forall$ c:

$$\sum_{t=1}^{T} \sum_{r=1}^{R} x_{rtc} = E_c \tag{3.2}$$

Events of courses that cannot be scheduled in the same time slot since they have student groups and/or lecturers in common, are represented in a conflict matrix A. A is a symmetric $C \times C$ matrix, of which the elements on the diagonal are 0, and the non-diagonal elements are equal to the number of student groups and/or lecturers that the two courses have in common.

We use the conflict matrix A to express that student groups and lecturers can be assigned to a time slot at most once (see Eq. 3.3).

$\forall$ rooms r, time slots t and courses c:

$$M(1 - x_{rtc}) \geq \sum_{n=1}^{C} A_{cn} x_{rtn} \tag{3.3}$$

with M an arbitrary large number.
$\forall$ rooms r and time slots t:

$$\sum_{c=1}^{C} x_{rtc} \leq 1 \tag{3.4}$$

Equation 3.4 expresses that a room can contain at most one lecture per time slot.

Let O_t be the set of all time slots that overlap with time slot t.
$\forall$ rooms r, time slots t and courses c:

$$M(1 - x_{rtc}) \geq \sum_{n=1}^{C} \sum_{d \in O_t} A_{cn} x_{rdn} \tag{3.5}$$

Equation 3.5 expresses that student groups and lecturers can be assigned at most once to overlapping time slots.

We express that the capacity CAP_r of the assigned room r should be large enough to accommodate all students taking the course as Eq. 3.6.

$\forall$ courses c, time slots t, and rooms r:

$$S_c x_{rtc} \leq CAP_r \tag{3.6}$$

Let CP denote the set of all courses that require special room properties. We assume that for every course $c \in CP$ there exists a corresponding set of rooms R_c with the appropriate room properties.
$\forall c \in CP$ and time slots t: if room $r \notin R_c$:

$$x_{rtc} = 0 \tag{3.7}$$

Equation 3.7 expresses that lectures that require special room properties can only be assigned to rooms that are appropriately equipped.

3.2.3 Problem Size

The problem size of a semester's timetable is on average:

- 133 lecturer groups.

- 212 student groups, each containing on average 20 students.
- 67 class rooms of different sizes.
- 19 sometimes overlapping, fixed time slots per day. These time slots are defined in such a way that there is no lecturing between 1 and 2 PM.
- 1215 lectures (with a pre-assigned duration) that need to be scheduled. Each course consists of a set of lectures. Some of these courses have a non-weekly recurring frequency.

In the next section the local search algorithm applied to solve the university course timetabling problem is described.

3.3 Constructing a Local Search Algorithm for the KaHo Sint-Lieven Course Timetabling Problem

Since our main concern was solving a real-world university course timetabling problem with not so common constraints, we needed a means to easily model and express the problem. Moreover, due to the fact that the best solutions in the two International Timetabling Competitions are produced with local search algorithms, we have opted to employ a local search algorithm to tackle the problem.

In local search small adjustments are carried out to a solution s. These 'moves' transform a solution s into a new solution s'. The set of solutions that can be created from a solution s, by executing one move, is called the neighbourhood N of the solution s. Whether a new solution is regarded better than a previous solution depends on the value of the cost function. Different forms of local search exist, of which the most common are hill climbing, simulated annealing, and tabu search. We apply tabu search [86] which has been proven to be very successful for a variety of timetabling problems [93, 94, 98]. Extra memory is added to the local search algorithm so that recently executed moves are not executed again. The properties of these moves are put in a tabu list. It avoids that the algorithm gets stuck in local minima.

In the next sections we described the solution space, the neighbourhoods, and the cost function in detail.

3.3.1 Solution Space

The set of all candidate (feasible and infeasible) solutions is called the solution space. For the university course timetabling problem described in Section 3.2, we

represent the solution space as a collection of two dimensional arrays (matrices) with the class rooms in the rows and the time slots (over a period of a week) in the columns. Each individual matrix will only contain lectures and lab sessions of equal duration.

Thus, there will be as many matrices as there are lectures and lab sessions with different durations. With this representation we will avoid that lectures or lab sessions with a particular duration would end up in a room-time slot combination of which the time slot has a different duration. A graphical representation of the search space is shown in Fig. 3.2. The **R**s denote the rooms and the **T**s denote the time slots. Note that every individual matrix contains all the rooms, but that each time slot is only present once in all matrices.

Time slots are sorted lexicographically on the (start time, end time) combination. The start and end times of time slots are predefined, and do not change in the optimization process. The matrix that contains the lectures with a duration of 90 minutes contains no overlapping time slots (see also Table 3.1), the other matrices however can contain overlapping time slots within the same matrix. Apart from overlapping time slots in the same matrix, there are of course overlapping time slots between different matrices. This has to be taken into consideration for lecturers, student groups and rooms. We did not provide any special ordering for the rooms in the matrices.

A lecture or lab session allocated to a time slot-room combination is represented by its ID in the matrix. By choosing this representation, the constraint **C2** is automatically satisfied: no two lectures or lab sessions will ever be allocated to the same time slot-room combination. It does however not avoid that a class room will be occupied with lectures or lab sessions in overlapping time slots (constraint **C5**). A matrix element will contain 0 when no lecture or lab session is assigned to the corresponding time slot - room combination. As an initial solution, we randomly (without taking into account any of the other constraints of Section 3.2) assign lectures and lab sessions to the room-time slot combinations.

3.3.2 Neighbourhoods

To explore the solution space we implemented moves that define different neighbourhoods. The **Swap Move** is the basic move. This move swaps the contents of two slots within a two dimensional matrix (see Fig. 3.3). Restricting the directions for this basic Swap Move guides the exploration of the solution space. This leads to a number of neighbourhood flavours, of which the most important are:

- **Time Swap Neighbourhood**: In the exploration of this neighbourhood,

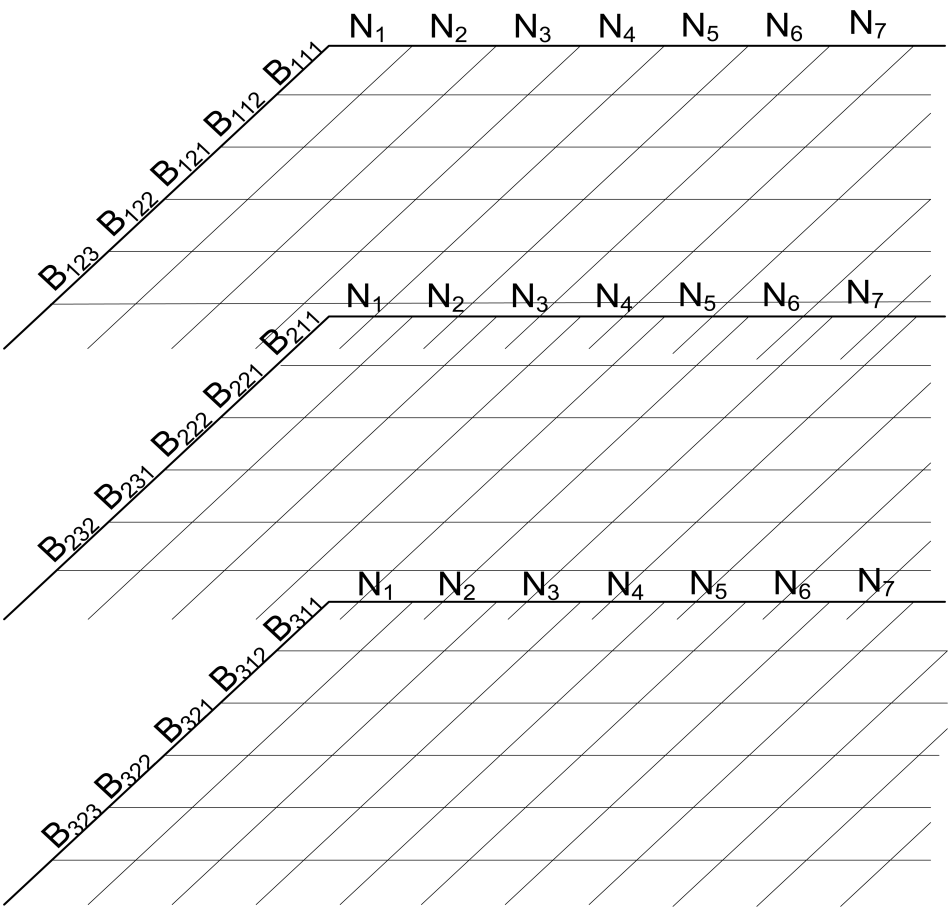

Figure 3.2: Graphical representation of the solution space. Each matrix contains subjects of equal length.

the basic move is limited to shifting a lecture or lab session within the same row. This neighbourhood originates from swapping a specific lecture or lab session with all elements in the same row of its current matrix.

- The **Room Swap Neighbourhood** explores the moves that shift a randomly chosen lecture or lab session into an appropriate class room in the same column. For lectures, it implies that the size of the class room should be large enough to accommodate the students taking the lecture, while for lab sessions the lecture should take place in a large enough room that is equipped for the course.

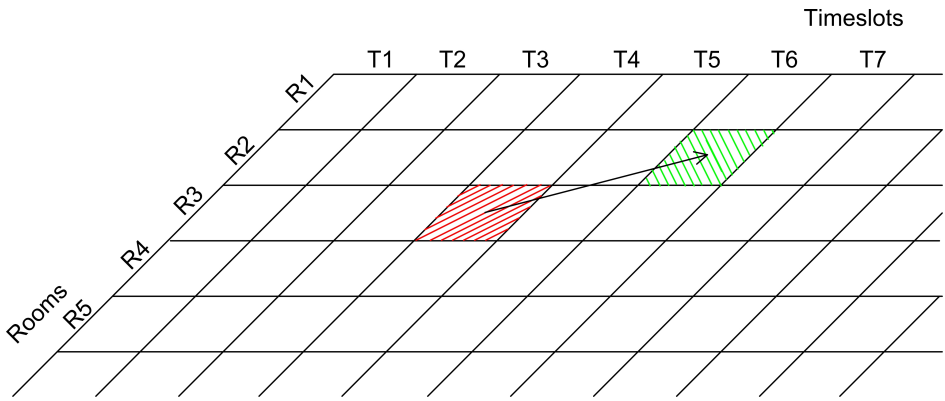

Figure 3.3: Graphical representation of a swap move: if no lecture or lab session is assigned to the destination room-time slot combination, the original lecture or lab session will be moved to the empty spot, in the other case, the two sessions will be swapped.

- In the **Time-Room Swap Neighbourhood**, the randomly chosen lecture or lab session will be swapped with every (zero or non-zero) lecture or lab session in the row and column where this lecture or lab session is situated. This neighbourhood is a combination of the first two neighbourhoods.

All the neighbourhoods discussed above only swap lectures or lab sessions within one matrix. Swapping lectures or lab sessions between two of the matrices would make no sense, since the time slots have a different duration. We call these neighbourhoods 'horizontal' neighbourhoods, to emphasise the fact that we swap lectures in the same matrix and to contrast with the 'vertical' neighbourhoods we will introduce in Section 3.4.1.

3.3.3 Cost Function

The cost function evaluates a solution. It is determined by the weighted sum of the violations of the constraints. Note that not all the constraints need to be evaluated. This can be accomplished either by

1. choosing a good representation of the solution space,
2. limiting the exploration of the neighbourhoods in such a way that moves leading to non-feasible solutions are avoided or
3. constructing the initial solution in such a way that these constraints are always satisfied.

In the problem at hand, constraint **C2** is always satisfied, because the representation of the solution space avoids any violation of the constraint. The constraints we take into account are **C1**, **C3**, **C4**, and **C5**. These constraints are modeled as soft constraints and can thus be violated. The **C6** constraint will be discussed in the Section 3.4. The only valid values for the weights are 0 or 1 (see Section 3.5).

To limit the computation effort, the solution is only partially reevaluated, i.e. in the rows and columns that have changed.

3.3.4 The Local Search Algorithm and its Parameters

We experimented with different parameter settings for the tabu search algorithm: a dynamically changing (prime) tabu list length and the number of non-improving moves before calling another neighbourhood.

- The idea is that the size of the tabu list increases when a move does not lead to a better solution, which forces the algorithm to explore other regions of the search space. The size decreases each time a better solution is found, which forces the algorithm to intensify the search in the current region of the search space. Before the search starts, an ordered list of primes starting at 7, with upper limit the tabu list length is generated.
- The tabu search approach is hybridized with token-ring search [77, 78]. This is a multi-neighbourhood method, in which a different neighbourhood is taken after a pre-defined number of non-improving iterations. The idea behind this method is that the search algorithm takes another neighbourhood when the original neighbourhood does not generate better solutions. By applying to another neighbourhood, one hopes to find better solutions because the solution space is searched in a different manner, which may lead to better solutions. When the maximum number of non-improving iterations is reached in the final neighbourhood, the algorithm applies the first neighbourhood again (e.g. $N_1 \rightarrow N_2 \rightarrow N_3 \rightarrow ... \rightarrow N_n \rightarrow N_1 \rightarrow N_2 ...$). Every time the algorithm switches neighbourhoods, the length of the tabu list is decreased to 7.

The algorithm is schematically presented as Algorithm 1. We made abstraction of the dynamic tabu list changes in the pseudo code of Algorithm 1.

As explained in Section 3.3.3 the cost function is the weighted sum of the violations of the constraints. Since we consider all constraints equally important, all the weights of the cost function are assigned the value 1.

Algorithm 1 Local search algorithm

```
CN: threshold parameter to change neighbourhood
Stop condition: maximum number of iterations reached or costfunction(solution)=0
best solution = initial solution
while stop condition is not met do
  apply tabu search: new solution = S*
  if costfunction(S*) < costfunction(best solution) then
    best solution=S*
  else
    if number of unimproving iterations > CN then
      change neighbourhood
      set number of improving iterations = 0
    else
      increment(number of unimproving iterations)
    end if
  end if
end while
```

3.3.5 Real-World Issues

The data needed to automate the timetabling process, is available in an administrative university database. This database contains information about the courses that they cover, the duration of the lectures and lab sessions, the student groups and their size, the rooms and their size. Currently no information is available about the room types. To avoid that lab sessions would end up in rooms that, even though they are large enough, are not equipped for the session, we assign rooms to lab sessions beforehand. In a pre-processing step an existing manually constructed solution that is present in the database is evaluated and the rooms to which the lab sessions are assigned are extracted. Each lab session gets an extra constraint enforcing in which lab room it should take place. By working in this manner we probably exclude some lab rooms that are equally equipped, and thus exclude possible better solutions to the problem.

3.4 Decomposition Approach

3.4.1 From Tiles to Pillars

The search algorithm generates a weekly recurring schedule with Algorithm 1. For a high school or a university timetabling problem with weekly recurring lectures,

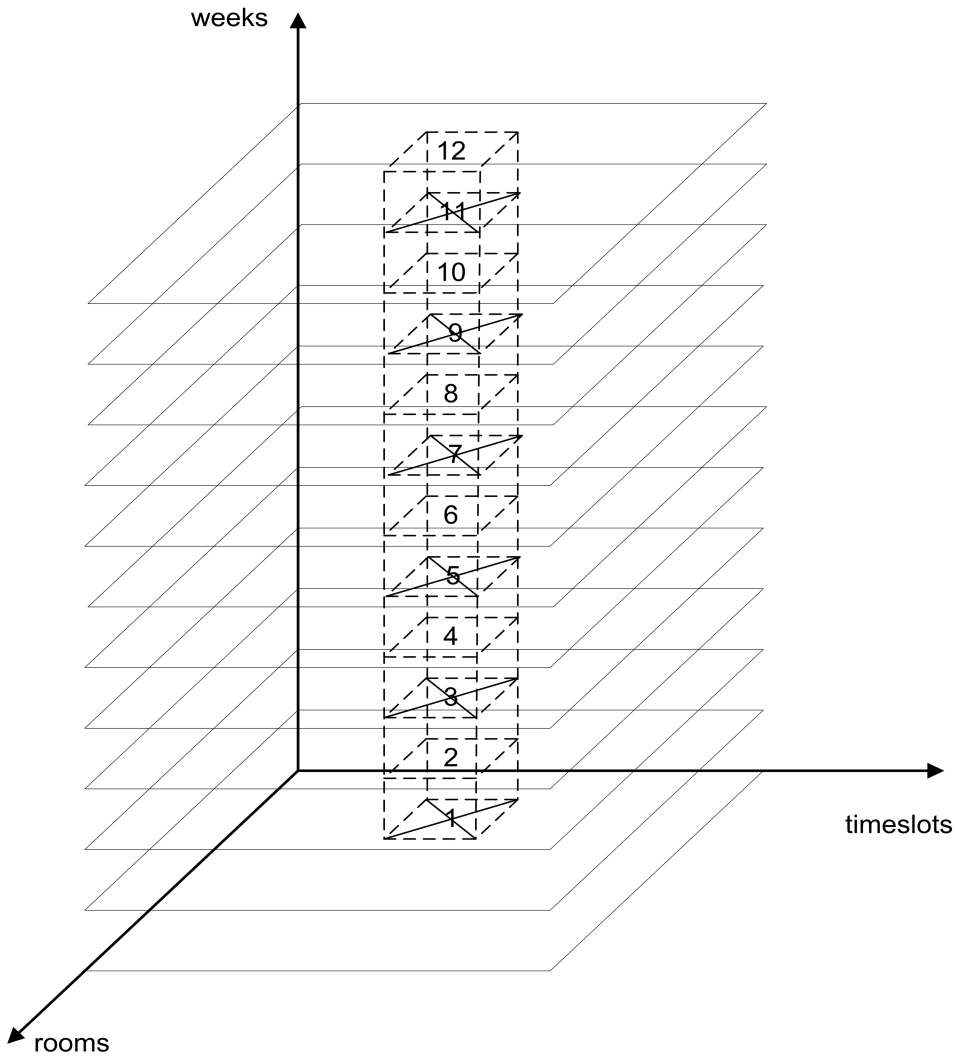

Figure 3.4: Graphical representation of pillars. Lectures in the pillar are scheduled on a fortnightly basis.

the method described in Section 3.3 will suffice. As explained in Section 3.2, the KaHo Sint-Lieven Industrial Engineering Department deals with courses that are often not organised more than a few times per semester, which involves 12 lecture weeks. The number can range from every fortnight, the first 6 weeks or the last 6 weeks of a semester, or three times evenly spread during the semester,... Since we want to avoid scheduling numerous independent weekly timetables, which would

make the timetabling problem larger and hence more complex, we need a different approach to tackle this issue.

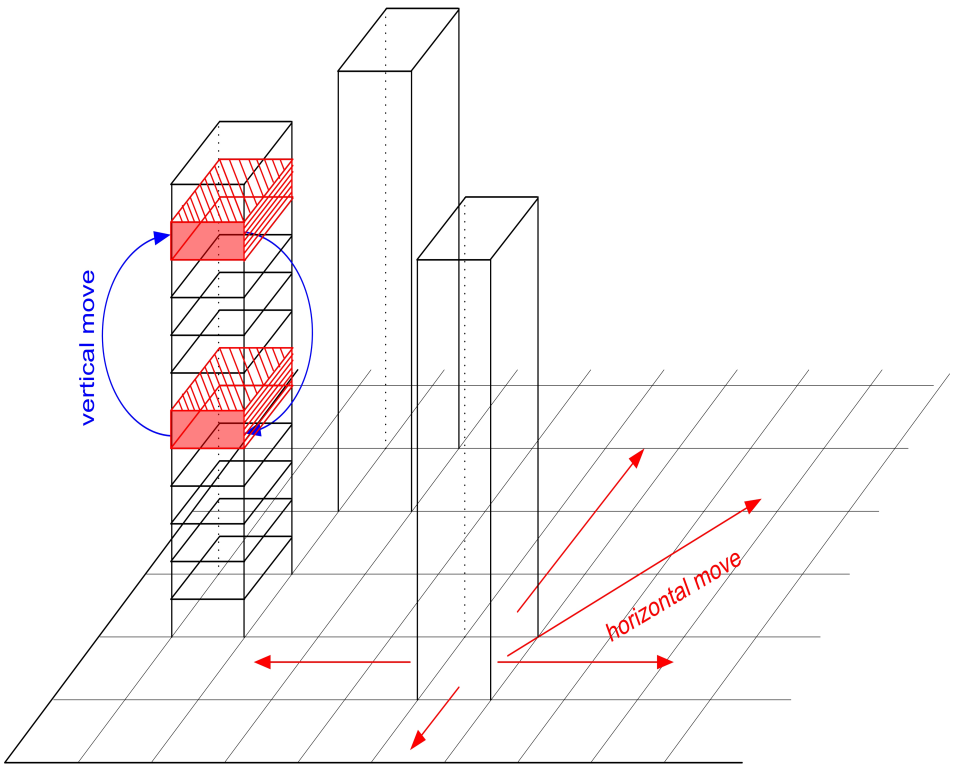

Figure 3.5: Graphical representation of the 'vertical' and 'horizontal' moves in a matrix containing only lectures of the same duration.

Constructing 12 entirely independent weekly timetables is not an option since that would not be appreciated by either students or lecturers, as their timetables would differ too much from week to week. To overcome that discomfort we want the weekly timetables to be as similar as possible for everyone involved. Therefore, a pre-processing step is carried out. The applied method, is inspired by the tiling approach that Kingston [102] successfully applied to Australian high school timetabling in which similar subjects are grouped and treated as one. The idea in the KaHo Sint-Lieven case is to sort lectures and lab sessions which are covered by the same lecturer. We start from the first lecturer and select the most frequent covered session (whether a lecture or a lab session) with a frequency less than 12. Dependent on whether the selected session is a lecture or a lab session, the continuation of the applied approach differs.

- If it is a lecture, then the algorithm searches for another lecture from the same lecturer, of which the frequency during the semester is added to the frequency of the first lecture. The lecture that first adds up to 12 is chosen to be grouped with the original lecture. If no such lecture is found, the lecture with a frequency that adds up closest to 12 is selected.
- If, however, the selected session is a lab session, the same procedure as described for a lecture is valid, except that the lab room of the original lab session is taken into account. This means that the lab session that the algorithm searches for, has to meet 2 conditions: apart from being the lab session with a sufficiently large frequency that adds up to 12 or closest to 12, it should also have the lab room in common with the original lab session. In the more general case in which we would have information about the different room types (see Section 3.3.5), the grouping would be based on equal room types instead of on common lab rooms.

This greedy procedure is repeated until all lectures and lab sessions of the same lecturer are evaluated. The first pre-processing step ends when there are no lecturers left to evaluate. After the initial grouping of lecturers is performed, a second pre-processing step is carried out. The groups of lectures and lab sessions that were formed in the first pre-processing step and which sum of frequencies is less than 12 are evaluated. Instead of grouping the lectures and lab sessions by the same lecturer, we group all the lab sessions that take place in the same room. These lab sessions are grouped in the same greedy manner as described in the first pre-processing step.

Fig. 3.4 is a graphical representation of a semester consisting of twelve weeks. The 'pillar' in the middle of the figure is composed of two interleaved lectures (marked with and without a cross) that are each organised six times in the same room during a semester. Grouping the lectures in this way reduces the 1215 lectures to be scheduled to 437 pillars, which reduces the combinatorial complexity of the timetabling problem. In the case described in Fig. 3.4, the pillars are constructed for lectures covered in the same location and time slot. Students or lecturers are not necessarily the same in one pillar. Note that by representation constraint **C6** is always fulfilled.

Apart from the moves described above, which we call 'horizontal', we also implemented a 'vertical' move that allows swapping lectures and lab sessions within the same pillar (see Fig. 3.5). This move is useful when one pillar combines lectures or lab sessions that have different student groups. Problems with overlapping time slots and student groups being assigned to two places at the same time can be solved by shifting lectures or lab sessions in such a pillar. We call this the **Pillar Move**. We explore this 'vertical' solution space with the **Pillar Plane Neighbourhood**. In one iteration we select a random matrix, in which we select

a random part of the list of pillars contained in this matrix. In each pillar of this list we try to execute a **Pillar Move** and finally the best **Pillar Move** is selected.

Another decomposition approach that is applied to university timetabling is described in Qu et al. [126]. The subject of the decomposition are the Toronto examination benchmark data sets. The exams are split into two distinct sets: a *difficult* and an *easy* set. The decomposition of the two sets depends on the ordering of the exams based on the number of remaining feasible time slots during the construction of a solution. The algorithm consists of two stages. In the first stage, the ordering of the exams in the difficult set is adjusted, aiming at finding in this manner an improved or feasible solution. If no feasible solution can be found with the current ordering, the exam responsible for the infeasibility is moved forward in the difficult set, while at the same time the difficult set is restricted to the exams that are ordered before the involved exam. If, however, a feasible solution can be obtained, the difficult set is extended with the first exam of the easy set. In the second stage, the exams in the difficult set are fixed, while the exams in the easy set are reordered. From the experiments it is clear that the difficult set, although only a small subset of the exams, contributes to a large amount to the cost. The obtained results are competitive with the best reported solutions in the literature.

3.5 Solving the Problem in Different Stages

Instead of trying to find a solution that satisfies all the constraints at once, we opted to work in different stages (see Algorithm 2). In the first stage, a solution that satisfies one constraint is constructed. This corresponds to assigning the value zero to all weights of the cost function (see Section 3.3.3) except for one of the weights, that is assigned the value 1. The stop criterion is met when either the solution does not violate that constraint anymore or when a maximum number of iterations is exceeded. Then, an extra constraint is added, or formulated differently: another weight (next to the weight that already was assigned the value 1) is assigned the value 1. By taking the best solution of the first stage as the initial solution of the second stage, the tabu search algorithm tries to find a second solution satisfying both constraints to the best possible extent. This process is repeated until no constraints remain to be evaluated, or if all weights of the cost function are assigned the value 1.

The approach leads to solutions of similar or better quality (see Section 3.6) compared to the approach in which all the constraints are taken into consideration in one go (see Algorithm 1).

Solving timetabling problems in different stages is a common technique reported on in literature. Lewis [106] discusses it in his survey paper on metaheuristics for

Algorithm 2 Local search algorithm with sequential evaluation of the constraints

```
CN: threshold parameter to change neighbourhood
Stop condition: maximum number of iterations reached or costfunction(solution)=0
best solution = initial solution
n = number of constraints
for i = 1 to n do
    add constraint i
    while stop condition is not met do
        apply tabu search: new solution = S*
        if costfunction(S*) < costfunction(bestsolution) then
            bestsolution = S*
        else
            if number of unimproving iterations > CN then
                change neighbourhood
            else
                increment(number of unimproving iterations)
            end if
        end if
    end while
end for
```

university timetabling problems. In the articles discussed, the first stage is used to find feasible solutions, while in the second stage, the feasible solution is applied as a starting point, to find feasible solutions that satisfy the soft constraints. The approach described in this chapter is different in that it applies several stages to find a solution satisfying the constraints defined in Section 3.2.

	Tabu list length	**Number of unimproving iterations**
Set 1	100	500
Set 2	100	1000
Set 3	200	800
Set 4	300	1500
Set 5	300	3000

Table 3.2: Overview of the 6 sets of parameters

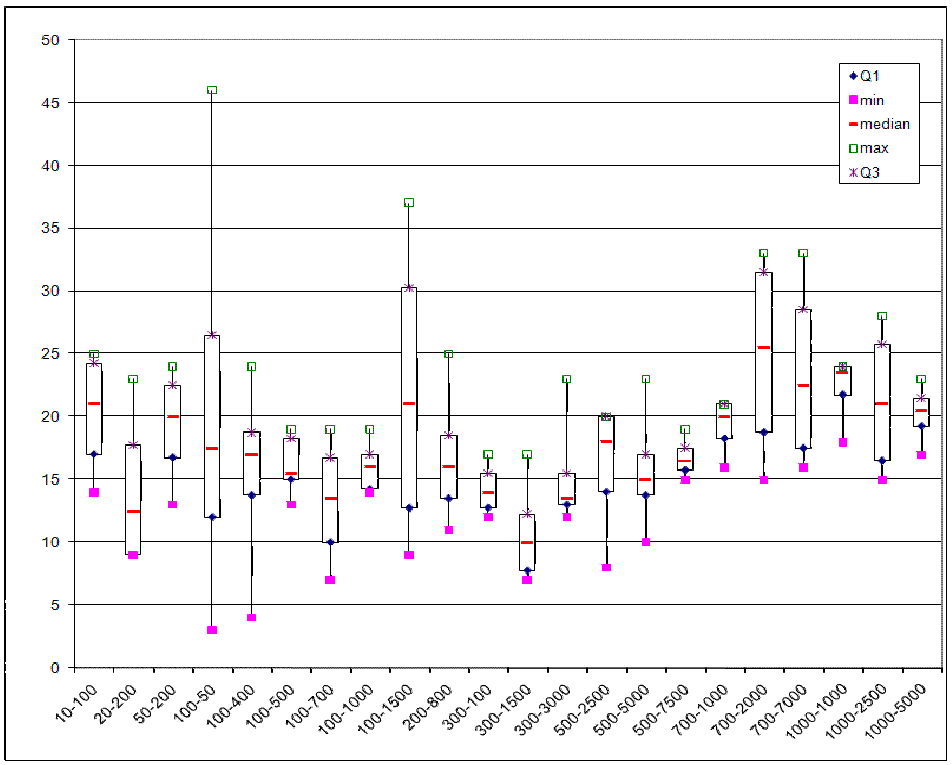

Figure 3.6: Box plots of results obtained with 22 parameter settings. Each algorithm is run for 50000 iterations. The first number of each set is the tabu list length and the second number is the number of non-improving iterations before changing neighbourhoods.

3.6 Test Results

As explained in Section 3.3.1, another neighbourhood is chosen after a certain number of unimproving iterations. The number of unimproving iterations is one of the parameters of the search algorithm. Another parameter is the upper bound of the dynamic tabu list length. We conducted several experiments for 5 sets of parameter settings and describe them in this section. Table 3.2 contains the 5 sets of parameter settings. We will discuss the first set of Table 3.2, the other sets are analogous. The upper limit of the tabu list length of the first parameter setting is 100 and the number of unimproving iterations before changing neighbourhoods is 500.

The 5 sets of parameters are chosen from a larger collection of 22 sets of parameters.

Every parameter setting is run for 50000 iterations. Box plots of the results obtained with 22 parameters settings are presented in Fig. 3.6. From this collection of parameter settings we make a selection of 5 sets with the smallest dispersion and the lowest median.

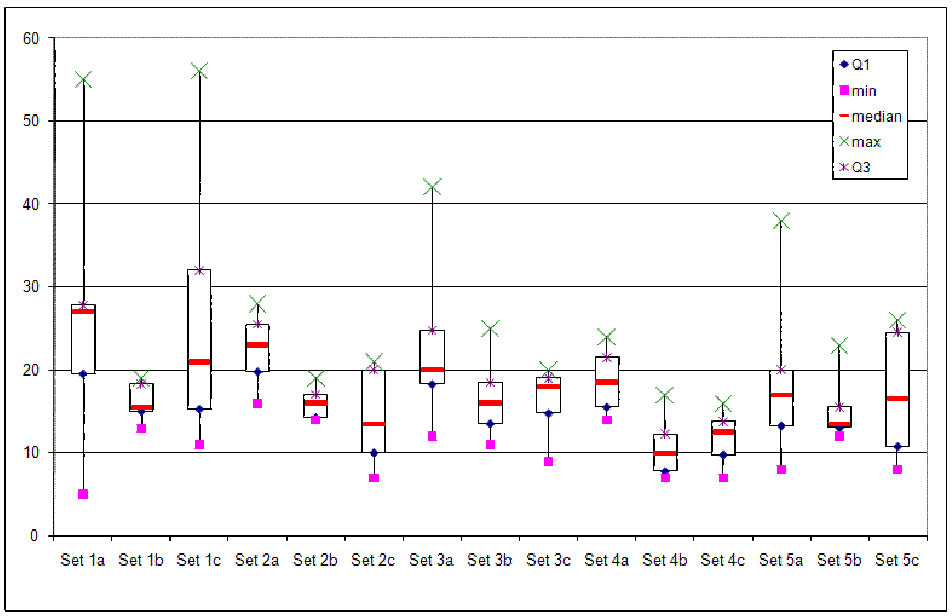

Figure 3.7: Comparison between starting from the best solution found in the previous neighbourhood ('a' sets), starting from the last solution obtained in the previous neighbourhood ('b' sets) and selecting the best or the last solution obtained with the previous neighbourhood at random ('c' sets).

In [77] Di Gaspero and Schaerf describe a similar token-ring search in which every new neighbourhood starts from the best solution found by the previous neighbourhood. In this experiment we examine whether it is useful to follow the approach of [77]. We executed for each of the 5 previous discussed parameter settings 6 runs of 50000 iterations per stage each (see Fig. 3.7). Per run, we execute experiments in which different solutions (best, current, or the solution that is randomly chosen between the best or current solution) are passed on to the next neighbourhood. In Fig. 3.7 the result of the tests that take the best solution of the previous neighbourhood are marked Set ia, with i ranging between 1 and 5. The result of the tests that take the last solution obtained with the previous neighbourhood are indicated by Set ib, with i ranging from 1 to 5. A combination of both previous methods is presented by the Set ic, with i ranging from 1 to 5, where we make a random choice whether we begin from the best or from the

last solution obtained in the previous neighbourhood. From Fig. 3.7 it is clear

S1:100-500	S2:100-1000	S3:200-800	S4:300-1500	S5:300-3000
12	12	5	13	8
13	12	19	10	15
11	12	6	11	18
13	12	9	18	6
11	10	8	15	7
7	7	10	11	15
16	6	7	9	6
15	9	5	7	11
14	27	12	10	18

Table 3.3: For each of the six parameter settings, the cost of 9 runs is presented, when using the multi-stages technique. The maximum number of iterations is 75000 for every stage.

that the result of the b test run sets (where the last solution obtained with the previous neighbourhood is used as an initial solution for the next neighbourhood) are among the best. The results are less dispersed than for the other methods, and the median of each b test run is less than or equal to the median of the other test runs (except for the second set of test runs). Based on these experiences we decided to conduct the rest of the experiments with the last solution obtained with the previous neighbourhood as the initial solution for the next neighbourhood.

In the next experiment we investigate the efficiency of the multi-stages approach of Section 3.5 compared to solving all the constraints at once. For each of the 5 sets of parameters, we executed 9 runs. Table 3.3 presents the results obtained with each of the parameter settings. For the same 5 sets of parameters, we conducted an

S1:100-500	S2:100-1000	S3:200-800	S4:300-1500	S5:300-3000
40	19	11	20	22
20	15	27	16	13
11	16	14	35	13
17	13	4	14	9
11	9	21	6	19
19	10	21	11	13
12	5	11	19	20
16	6	31	14	16
26	8	31	21	17

Table 3.4: Cost for 5 different parameter setting during 9 runs, when solving all-constraints-at-once. Maximum number of iterations is 100000.

analogous experiment, in which all the constraints are evaluated together. Every

parameter setting is executed 9 times. The results are presented in Table 3.4. Both

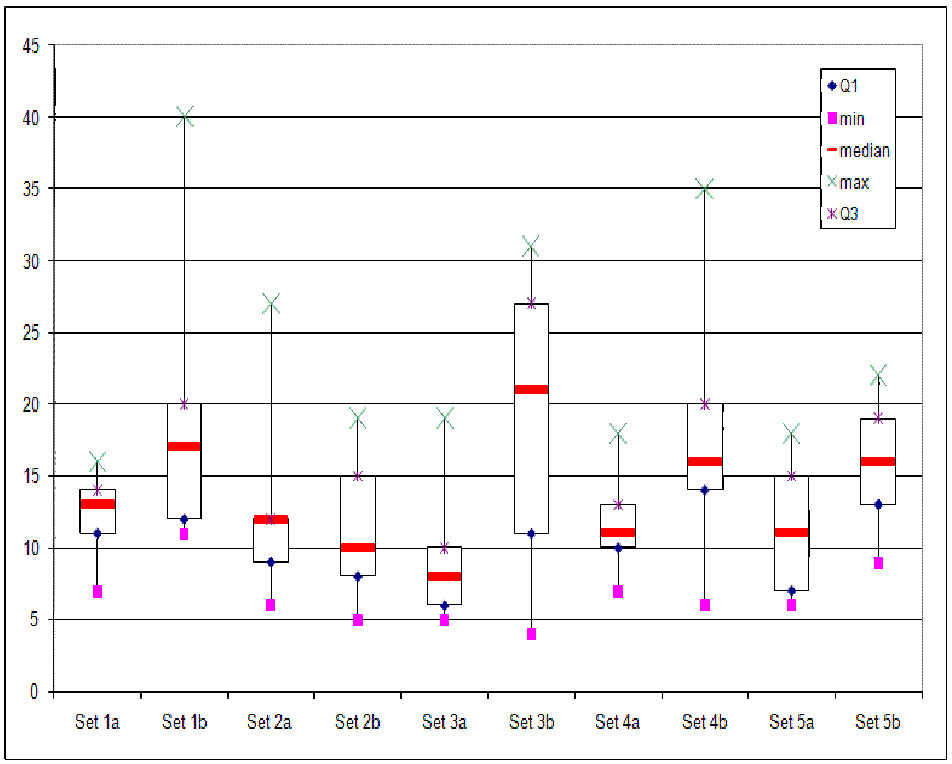

Figure 3.8: Box plots of the costs when solving the constraints in different stages ('a' sets) and solving all-constraints-at-once ('b' sets).

approaches' results are presented in Fig. 3.8.

	Set1		**Set2**		**Set3**		**Set4**		**Set5**	
	a	**b**	**a**	**b**	**a**	**b**	**a**	**b**	**a**	**b**
avg	**12.44**	19.11	11.89	11.22	**9**	19	11.56	17.33	11.56	15.78
P-value	**1**	0.05	0.8	1	**1**	0.01	1	0.07	1	0.07

Table 3.5: P-values for the 5 parameter settings.

Except for the second parameter set, the average of the results of each parameter set obtained via the multi-stages approach, is smaller than the corresponding average of the results obtained through the all-at-once approach (see Table 3.5).

The multi-stages approach is not outperformed by the all-at-once approach for any of the parameter settings in a statistically significant way.

cost	elapsed time	number of iterations
6	104292	180638
6	117956	197273
10	90118	143741
4	109627	183879
5	94467	167373
3	87920	151041

Table 3.6: Cost and elapsed time in seconds for 6 runs with the same parameter setting (300-1500). Maximum number of iterations is 200000 per stage.

In the last experiment we apply the multi-stage approach to the fourth parameter set, with a maximum number of iterations equal to 200000 per stage. From Table 3.6 it can be deducted that the longer a tabu search algorithm is executed the better the results become. With these settings the algorithm takes quite a long time to come up with a solution. For practical applications, long computation times are acceptable since the timetabling process is carried out only twice a year, at the beginning of each semester. The presented results correspond to the sum of

constraints				total
C1: C3	C2: C1	C3: C5	C4: C4	C1+C2+C3+C4: total
0	3	0	7	10
0	3	0	3	6
0	0	0	4	4
0	0	0	5	5
0	0	0	3	3
0	3	0	3	6

Table 3.7: The total cost is split up into the contribution per constraint. The maximum number of iterations per stage is 200000.

the violations of the constraints. As described in Section 3.3.3 the constraints that are evaluated by the cost function are **C1**, **C3**, **C4**, and **C5**. In Table 3.7 we split the total cost of Table 3.6 up in separate parts, which correspond to the violations of the 4 constraints. From this table, one can see that the **C4** constraint is hard to satisfy. As mentioned in Section 3.3.5, this is due to the fact that we assigned rooms to the lab sessions based on the current manually constructed timetable. By doing so, we probably exclude other appropriate rooms from the solution.

3.7 Conclusion

In this chapter we presented a new decomposition heuristic for solving a large, complex real-world timetabling problem at KaHo Sint-Lieven. By grouping similar lectures into pillars first, the remaining timetabling problem becomes less complex: the search space for the reduced problem is considerably smaller than the search space for the original problem. The reduced timetabling problem is then solved by sequentially evaluating the constraints one by one, in such a way that the obtained solution of the first stage is applied as the starting solution of the second stage and so on. We experienced that for this problem this procedure leads to similar or better solutions, compared to trying to solve all the constraints at once. The KaHo Sint-Lieven university course timetabling problem differs from other problems, due to the fact that time slots can be overlapping. This is an extra hard constraint that should be taken into account.

Chapter 4

Post Enrolment Based University Course Timetabling

4.1 Introduction

In this chapter we introduce a hybrid guided local search (GLS) algorithm that incorporates tabu search techniques for the university course timetabling problem at the K.U.Leuven. The problem resembles the post enrolment based course timetabling problem of the second International Timetabling Competition [1] to a certain degree. Post enrolment university course timetabling implies that students create their programme by selecting courses from a list. Timetables are generated in a way that enables students to attend all the courses they selected. This means that simultaneous events should be avoided in each student's timetable. Lewis et al. [108] describe the post enrolment based course timetabling problem of the International Timetabling Competition in full detail. They provide no information concerning constraints involving the lecturers. A lecturer could be considered as an extra student attending a course[1]. This limits expressiveness since it does not allow stating particular constraints regarding the lecturer.
The problem tackled in this chapter is on the other hand also somewhat different from the post enrolment based course timetabling of the International Timetabling Competition in that it is in essence a curriculum based course timetabling problem. A curriculum groups the courses that have students in common [130]. Students have to successfully finish the curriculum before they are allowed to enter the next year's curriculum. However, a large percentage of the students do not pass all the courses from the curriculum during one academic year. These students get another

[1] private communication with Rhydian Lewis

chance to pass the failed courses in the next academic year. Meanwhile they are allowed to enroll for a limited number of courses of the follow-up curriculum (as long as these courses do not build upon the courses for which they did not pass). For a large percentage of the students, the flexibility results in an individual timetable with numerous potentially conflicting events, since they are enrolled for courses from different curricula. It might be interesting to investigate whether the K.U.Leuven problem can be tackled as a post enrolment based course timetabling, in order to reduce the number of conflicting individual timetables.
In contrast to other course timetabling problems, the time slots are not fixed in advance in the considered problem. They are generated in a pre-processing step. For every lecture, the total number of contact hours during the semester and the number of lectures per week are given. From this information and taking into account that a semester includes 13 weeks, the duration per lecture can be deducted. In practice, the durations of the lectures range from 60 to 300 minutes. This also means that some of the time slots will overlap. Necessary precautions should be taken to avoid that lecturers or students are assigned to overlapping time slots, or that a room is scheduled for two overlapping events. Also, measures should be taken such that every lecturer and student has at least one hour of lunch time between noon and 2 PM.

In Table 4.1 we compare the post-enrolment track of the International Timetabling Competition with the K.U.Leuven course timetabling problem.

	ITC2007	**K.U.Leuven**
event-clash constraint	only for students	students (soft) and lecturers (hard)
room capacity constraint	hard	soft
room property constraint	hard	soft
one event per room per time slot	hard	hard
availability constraint	hard	hard
specific order of events	hard	no
no lecture on last slot of day (student)	soft	no
no more than 2 consecutive lectures (student)	soft	no
no single lecture on a day (student)	soft	no
at least 1 hour lunch time	no	soft
location preference (lecturer)	no	soft
room preference (lecturer)	no	soft
overlapping time slots constraints	no	students (soft) and lecturers (hard)

Table 4.1: Comparison between ITC2007 post-enrolment track and K.U.Leuven course timetabling problem

The K.U.Leuven course timetabling problem is also of interest from an software developer's point of view. The goal is to reuse (most of) the model and algorithm that were developed for the curriculum based course timetabling problem at KaHo

Sint-Lieven (see Chapter 3). According to their curriculum, students are divided into groups. As McCollum [113] notices, student groups are further divided into smaller groups for lab sessions. This gives rise to a hierarchy of student groups, which needs to be taken into account when scheduling courses. The K.U.Leuven timetabling problem is in this sense simpler, since it lacks such a hierarchy of student groups. Individual students are the smallest resource unit to schedule (next to lecturers and rooms). On the other hand, the problem can be perceived as more complex. Instead of only a few attendees participating in the course (a lecturer and a few student groups), we now have a lecturer and students (ranging from one to several hundreds). The K.U.Leuven problem is a good test case to investigate the performance of the algorithm that was originally implemented for curriculum based course timetabling.

We apply a hybrid GLS algorithm in which the weights of the objectives are dynamically adapted based on information gathered during the search. Guided Local Search [145] is a method that avoids local optima by selecting and penalizing certain features of the proposed solution. The objective function is augmented with extra terms that incorporate the penalties of violated features. The feature to be penalized, is selected based upon its effect on the overall cost. By attributing a higher value to a bad feature, the search algorithm is guided towards exploration of more promising regions of the search space. The proposed solutions are evaluated through a single cost function that consists of a linear combination of the different objectives. When the search algorithm fails to find an improvement over a number of iterations, the current best solution is evaluated and the constraint that is least satisfied is temporarily attributed a higher weight than the other constraints. Afterwards, the search continues with this new cost function.

The local search algorithm on which our hybrid GLS is based, is basically a tabu search algorithm with variable tabu list length, combined with a token-ring search [77, 78] in which another neighbourhood is selected whenever the algorithm fails in generating any improving moves for a pre-defined number of iterations.
The chapter is structured as follows: in Section 4.2 the literature in which optimization problems are solved by applying dynamic cost functions is reviewed. The K.U.Leuven problem is described formally in Section 4.3, while in Section 4.4 the hybrid guided local search algorithm is discussed. The hybrid GLS algorithm is compared to other algorithms in Section 4.5. We conclude in Section 4.6.

This chapter is a slighty adapted version of *P. Demeester, P. De Causmaecker, and G. Vanden Berghe. A hybrid guided local search approach to a university course timetabling problem. Technical report, KaHo Sint-Lieven, 2009.*

4.2 Dynamic Cost Function: Related Work

In the operations research literature several methods that address dynamic cost functions are presented. In this section we discuss a representative subset. All these approaches use a single cost function that is determined by a linear combination of different objectives.

Petrovic and Bykov [124] discuss a great deluge heuristic combined with a trajectories approach that they apply to exam timetabling. A candidate solution is accepted if its objective function value is better than or equal to that of the current solution or if it is within a certain threshold value B. During the search the value of B is decreased stepwise by a value ΔB. The initial value of the parameter B equals the objective value of the initial solution. The main idea of the trajectories approach of Petrovic and Bykov is that the search algorithm tries to direct the search along a virtual line in the criteria space (trajectory) from a start solution to the optimal solution. In fact, an extra weight Δw is added to the weight of the constraint that attributes most to the cost function. The authors apply their approach to the university examination data sets of the University of Waterloo and the University of Nottingham. Experiments show that this approach dominates the techniques published in [29, 43].

Di Gaspero and Schaerf apply tabu search to examination timetabling [75]. Next to a particular case of token-ring search, they introduce a cost function consisting of the weighted sum of the different constraints, in which the weight of each component of the function can be adapted according to the shifting penalty mechanism of Gendreau et al. [85]. If the objective is not violated during K consecutive iterations, the corresponding weight is divided by a random factor between 1.5 and 2. If an objective is violated in H consecutive iterations, the weight of that objective is multiplied by a random factor between 1.5 and 2. Experiments were conducted on the above mentioned Toronto and Nottingham examination data sets. For some of the data instances, new improved results were obtained [42, 52].

The SAW-ing technique (Stepwise Adaption of Weights) was first introduced by Eiben and van Hemert [80]. A small value is added to the weights of the constraints that are violated in the best solution after a specific number of fitness evaluations. Charon and Hurdy [53] introduce the noising method. 'Noise' is added to data in order to escape from local minima. The value of the added noise ranges between $-r$ and $+r$. During the search, the value of r is progressively decreased. The algorithm is constructed in such a way that 'bad transformations' (moves leading to a worse value) are also accepted. The noising method was successfully applied to graph partitioning, the linear ordering problem, the traveling salesman problem and coding theory [53].

Unlike the above approaches in which the weight adaptation happens dynamically,

we applied in Chapter 3 a stages approach in which all the weights but one are equal to 0 upon starting the search. When a pre-defined number of iterations is exceeded or when a solution is found that satisfies all the constraints under investigation, an extra constraint is added. Gradually more weights of constraints in the cost function become equal to 1. This process is repeated until all the constraints are being evaluated. We experienced that this approach leads faster to better solutions. A disadvantage of this approach compared to the above described methods is that the order in which the constraints are evaluated matters.

4.3 Problem Description

We describe the K.U.Leuven university course timetabling problem in detail:

- A course c (with C the total number of courses) consists of E_c individual events (lectures as well as lab sessions). The number of lectures usually is a multiple of 13, which is the number of weeks in a semester. The number of lab sessions can range between 1 and 13. The lecturers assigned to the different activities that constitute the course (lecture and lab session) do not need to be the same. The assignment of lecturers to lectures is not part of the problem.
- We consider the following resources:
  - students s ($s = 1, ..., S$, with S the number of students),
  - lecturers l ($l = 1, ..., L$, with L the number of lecturers), and
  - rooms r (with $r = 1, ..., R$, with R the number of rooms).

  The number of students taking a course c is represented by S_c. A room r can have several properties, such as the availability of a beamer, an overhead projector, computers, and so on. Each room r has a capacity CAP_r and a location Loc_r. For some courses the class room is fixed.
- Time slots t ($t = 1, ..., T$, with T the number of time slots) are not fixed in advance. Depending on the number of contact hours per course and the number of lectures per week, the duration of a time slot is calculated. The durations of time slots vary from 60 to 300 minutes. Activities in which the same student, lecturer or room is involved may overlap. Precautions should be taken to make sure that everybody involved (lecturers as well as students) has a break of at least one hour between noon and 2 PM.
- Some lecturers have preferences concerning the particular campus for their lectures. Also some of the lecturers are only available in particular time slots.

4.3.1 Constraints

The constraints are split into hard and soft constraints.

Hard Constraints

- **HC1**: all events (lectures and labs) of the courses c must be scheduled.
- **HC2**: a class room r can only hold one lecture at a time.
- **HC3**: a lecturer l can be assigned at most once in a time slot t.
- **HC4**: a student s can be assigned at most once in a time slot t.
- **HC5**: if two events overlap in time, they cannot have a room, a lecturer or a student in common.
- **HC6**: some lecturers are only available for a limited period.

Soft Constraints

- **SC1**: students as well as lecturers should have a break of at least one hour between 12 AM and 2 PM.
- **SC2**: some lecturers prefer to teach on the campus where their office is situated.
- **SC3**: the number of students assigned to a class room should not exceed its capacity.
- **SC4**: for some lectures special equipment is needed.
- **SC5**: some lectures can only take place in a specific class room.

4.3.2 Mathematical Formulation

Although we solve the K.U. Leuven course timetabling problem with a local search approach, we include a mathematical formulation for clarity.

Generally speaking in university course timetabling events (lectures, lab sessions, ...) covered by L lecturers and taken by S students have to be scheduled into R class rooms and T time slots taking into account different constraints.

The decision variables are:

$$x_{rtc} = 0 \; or \; 1 \tag{4.1}$$

with $(r = 1, .., R; t = 1, .., T; c = 1, .., C)$ and where $x_{rtc} = 1$ if a lecture of course c is assigned to room r in time slot t.

The constraints can be formulated as:

$\forall$ courses c:

$$\sum_{t=1}^{T} \sum_{r=1}^{R} x_{rtc} = E_c \tag{4.2}$$

Equation 4.2 expresses that all E_c lectures of every course c are scheduled.

We introduce the conflict matrix A: it represents the courses that cannot be scheduled in the same time slot, due to courses that have students and/or lecturers in common. It is a symmetric matrix containing C rows and columns. The elements on the diagonal are 0, and the other elements are equal to the number of students and/or lecturers that the two courses involved have in common.

$\forall$ rooms r, time slots t, and courses c:

$$M(1 - x_{rtc}) \geq \sum_{n=1}^{C} A_{cn} x_{rtn} \tag{4.3}$$

with M an arbitrary large number.

Equation 4.3 expresses that students and lecturers can be assigned at most once to a time slot.

$\forall$ rooms r, time slots t, and courses c:

$$\sum_{c=1}^{C} x_{rtc} \leq 1 \tag{4.4}$$

Equation 4.4 expresses that a room can at most contain one lecture per time slot.

Let O_t be the set of all time slots that overlap with t.
$\forall$ rooms r, time slots t, and courses c:

$$M(1 - x_{rtc}) \geq \sum_{n=1}^{C} \sum_{d \in O_t} A_{cn} x_{rdn} \tag{4.5}$$

Equation 4.5 expresses that students and lecturers can be assigned at most once to overlapping time slots.

Some of the lecturers are not available for lecturing at particular time slots. If lecturer l is unavailable at the time slots $T_l = \{t_1, .., t_n\}$ then:

$\forall$ courses c that lecturer l teaches:

$$\sum_{r=1}^{R} \sum_{t \in T_l} x_{rtc} = 0. \tag{4.6}$$

A solution that satisfies constraints 4.2 till 4.6 is feasible. However, a solution that minimizes the following objective function (Eq. 4.7) will be perceived as better.

$$\sum_{c=1}^{C} \sum_{r=1}^{R} \sum_{t=1}^{T} P_{cap,rtc} + \sum_{t=1}^{T} P_{prop,t} \tag{4.7}$$

A penalty of soft constraint **SC3** can be expressed as:

$\forall$ courses c, time slots t, and rooms r:

$$P_{cap,rtc} = max(S_c x_{rtc} - CAP_r, 0) \tag{4.8}$$

Equation 4.8 expresses that the capacity CAP_r of the assigned room r should be large enough to accommodate all students taking the course.

The penalty induced by violating soft constraints **SC2**, **SC4** and **SC5** can be expressed as follows: let D be the $R \times C$ matrix that indicates how well a room is fit to accommodate a course. A zero matrix element indicates that the room is fit for the course, while on the other hand a non-zero value corresponds to the penalty induced assigning the course to that room. The corresponding penalty for these three soft constraints can then be expressed as:

$\forall$ time slots t:

$$P_{prop,t} = \sum_{r=1}^{R} \sum_{c=1}^{C} D_{rc} x_{rtc} \tag{4.9}$$

4.4 A Hybrid GLS Algorithm

We apply a local search approach to the K.U.Leuven university course timetabling problem. Local search eases modelling the problem on the one hand and is flexible to include domain knowledge in the algorithm on the other hand. Several local search based algorithms obtained the best results [46, 56, 112, 119] in the post enrolment track of the second International Timetabling Competition [4]. For the university course timetabling problem we employ in particular a tabu search algorithm, with a dynamic tabu list lenght and multiple neighbourhoods that are evaluated in a token-ring fashion (see also Chapter 3).

4.4.1 Cost Function

The quality of candidate solutions is evaluated by means of the cost function. It is a linear combination of the violations of the hard and soft constraints:

$$c(S) = \sum_{i=1}^{6} v_i HC_i(S) + \sum_{i=1}^{5} w_i SC_i(S)$$

with HC_i the hard constraints' violations ($i = 1, .., 6$) and SC_i the soft constraints' violations ($i = 1, .., 5$) of the problem. The coefficients v_i and w_i correspond to the weights of the hard and soft constraints respectively. A higher value is attributed to the values of the weights v_i than to those of w_i. We attribute to the weights v_i the value 100, while the value 1 is assigned to the weights w_i.

Remark that this cost function has nothing to do with the objective function that we formulated in Section 4.3.2. The cost function discussed in this section consists of the violations of both hard and soft constraints, while the objective function (Eq. 4.7) only takes into account the violations of the soft constraints.

4.4.2 Dynamic Cost Function

During the search, the algorithm dynamically modifies the weights of the cost function. Each time the algorithm switches neighbourhoods, the best solution

found so far is evaluated. The violation of the constraint that attributes most to the overall cost, is penalised more than the other parts of the cost function. An extra small value Δx is added to the weight of that particular constraint. Suppose that constraint HC_i has the highest individual cost, then the altered cost function becomes:

$$c(S) = v_1 HC_1(S) + .. + (1 + \Delta x) v_i HC_i(S) + .. + v_6 HC_6(S) + \sum_{i=1}^{5} w_i SC_i(S)$$

If one of the soft constraints has the highest individual cost, the same procedure applies. Since the weights of the cost function have changed, the search landscape changes and the idea is that the algorithm will be directed to non visited regions of the search space. This hybrid method resembles the guided local search in [145], except that in the original approach the number of times a feature (or a constraint) is penalised is also taken into account. The more a feature is penalised in guided local search, the less probable that its weight will increase.

In parallel, every proposed solution is evaluated with a static cost function. This function is required to maintain an estimation of the real quality of the solution.

4.4.3 Representation of a Solution

In Figure 4.1 the solution space of the problem is graphically represented. The solution space is constructed so that events with equal durations are represented in the same matrix. Consequently, there are as many different matrices in the search space as there exist time slots with a different duration. In the case of Figure 4.1, there are three different time slot durations. Since it would not make sense to move an event with duration x to a time slot of duration y (with $x \neq y$), the local search moves will be constructed so that swaps between events positioned in two different matrices are not possible. The rows of a matrix represent the class rooms, whereas the columns represent the time slots with equal duration. Time slots are sorted lexicographically on the (start time, end time) combination. Although more efficient representations of the search space are likely to exist (e.g. with sparse matrices or linked lists), the advantage of the current representation is that the moves reflect the reality better: events can be either moved to another time slot or another room. The content of a matrix element is a lecture. Since all the lectures are repeated weekly, the solution for one week can be copied to each of the 13 weeks of the semester.

As already noted in the previous chapter (see Section 3.3.3), not all constraints have to be part of the cost function. In the K.U.Leuven case, hard constraint **HC1** (see Section 4.3.1), for example, will always be satisfied by putting all the courses

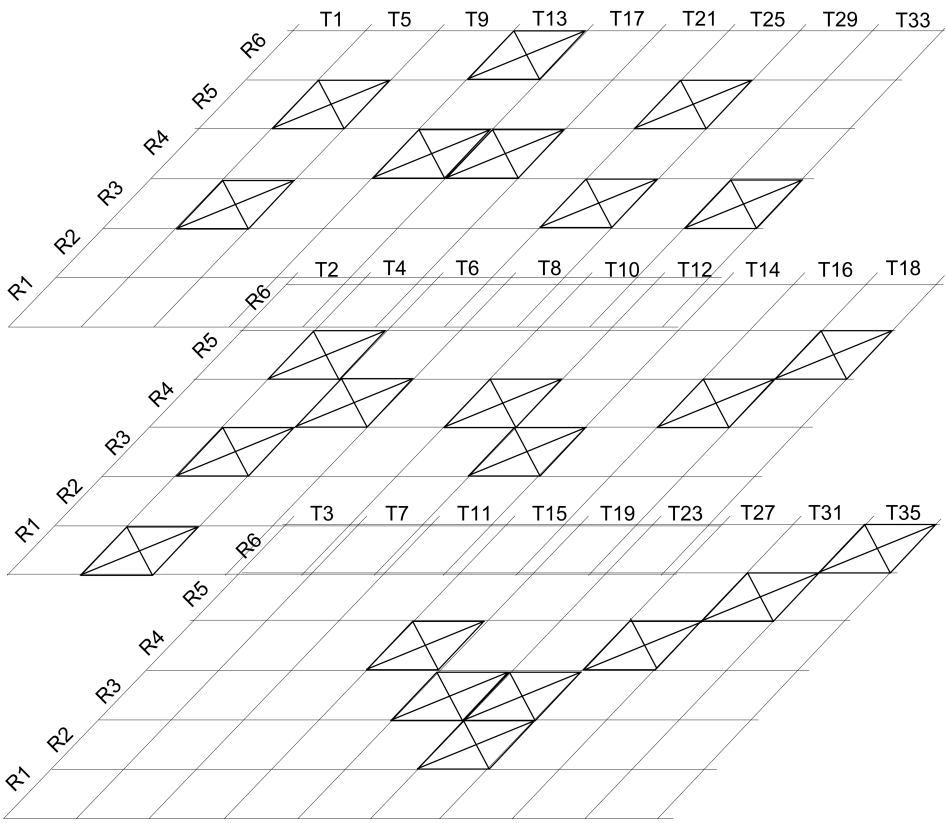

Figure 4.1: Graphical representation of the solution space. Each matrix contains events of equal length (in this case there are three different time slot durations).

into the initial solution and avoiding moves that delete courses from the schedule. Hard constraint **HC2** is always satisfied, since only one lecture is allowed in a room-time slot combination. The cost function of Section 4.4.1 reduces to:

$$c(S) = \sum_{i=3}^{6} v_i HC_i(S) + \sum_{i=1}^{5} w_i SC_i(S)$$

4.4.4 Neighbourhoods

We consider only one basic move that we call the **Swap Move**. It swaps the content of two slots within a two dimensional matrix (see Fig. 4.2). If both slots

are occupied, the lectures that reside in both slots are swapped. If one of the slots is empty, then the lecture in the occupied slot is moved to the empty time slot-room combination, whereas the original non-empty slot becomes empty.

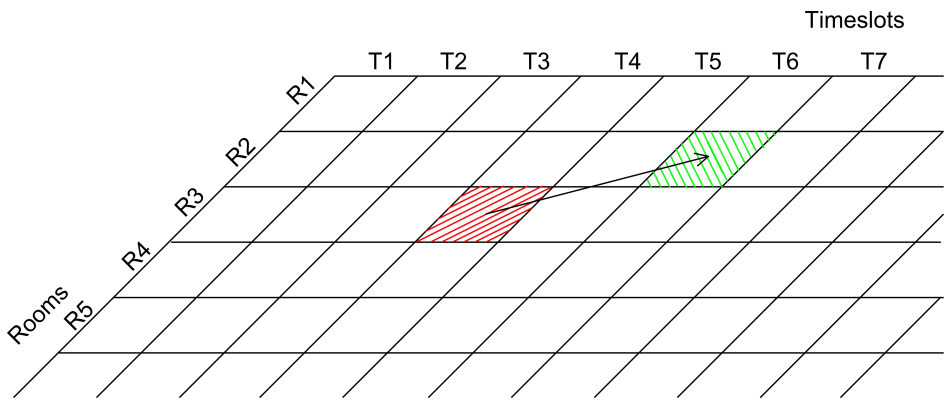

Figure 4.2: Presentation of a swap move.

By restricting the directions in which the basic move can explore the solution space several neighbourhoods can be created.

Swap-Events-In-The-Same-Row Neighbourhood: A random event with a particular duration (a lecture in one of matrices) is selected and swapped to a place in the same row. Only the time slot assigned to the event is changed, whereas the room stays the same. By executing this move, violations of the hard constraints **HC3**, **HC4**, **HC5**, and **HC6** can be solved. In every iteration, the number of moves that are evaluated equals the number of time slots with the same duration, which corresponds to the number of columns in the selected matrix.

Swap-Events-In-The-Same-Column Neighbourhood: This neighbourhood resembles the previous neighbourhood, except that the event is swapped within the same column. This corresponds to moving a lecture to another room, while keeping the same time slot. This neighbourhood assists in addressing violations of the soft constraints **SC2**, **SC3**, **SC4**, and **SC5**. The number of moves that are evaluated during one iteration equals the number of class rooms. Since this can be quite large, we can limit the number of class rooms to those of the preferred location of the event.

Swap-Events-In-The-Same-Row-And-Column Neighbourhood: The Swap-Events-In-The-Same-Row-And-Column Neighbourhood is the union of the two previous neighbourhoods. In practice, we employ this union neighbourhood instead of a separate execution of the first two neighbourhoods (see Section 4.5).

Swap-Events-To-Class-Room-With-The-Right-Properties Neighbourhood:

The fourth neighbourhood can be considered as a special case of the second neighbourhood. Instead of moving a randomly selected event anywhere in the same column, we only allow moves to class rooms that are large enough to accommodate all the students, and that moreover have the appropriate room properties to hold the event.

Swap-Overlapping-Events-In-The-Same-Row Neighbourhood: Select a random room and get the associated list of time slots that indicate when it is in use. Move the event so that the room is not occupied in overlapping time slots.

4.4.5 Initial Solution

An initial solution is constructed so that all lectures are assigned, without taking into account any of the constraints. Some approaches use a straightforward method (greedy method, exact method, ...) to find an initial solution that satisfies all the hard constraints, and apply a metaheuristic to find a feasible solution that satisfies as many soft constraints as possible (see [106] for an overview). For real-world university course timetabling problems however, it is not always easy to find a solution that satisfies all the hard constraints. Therefore, we accept any solution as a starting point for the search.

4.5 Experiments

In order to speed up the evaluation of the candidate solutions we employ delta evaluation. We only re-evaluate the parts of the solution that have changed with respect to the previous iteration.

By introducing extra data structures that contain redundant information, the local search's speed is increased further (see also [55]). A list that contains the occupancy for every resource (students, lecturers as well as rooms) is kept up to date. This list is employed in the cost function to quickly detect violations of the hard constraints **HC3**, **HC4**, and **HC5**.

We consider three K.U.Leuven data sets:

- bachelor Chemistry
- bachelor Physics
- bachelor Commercial Engineering

The first two sets are rather small data sets, since only a limited number of students (respectively 70 and 42) take these courses (respectively 67 and 46). In the third

data set, the number of students involved is larger, but the number of courses is rather small. Table 4.2 summarises the size of the data sets.

	#students	#lecturers	#courses	#lectures
Chemistry	42	38	46	758
Physics	70	55	67	1044
Commercial Engineering	367	19	13	144

Table 4.2: Problem size of the three data sets

In a first phase we solve each of the data sets independently, which is a simplification of the real-world situation. Some of the lecturers have duties in several departments and some of the rooms can be occupied by courses from different departments. Actually, the process that is followed in reality is mimicked: the human planner of each department constructs a feasible timetable. Afterwards mutual conflicts between the departments are solved in a meeting. In a second phase, we solve the problem taking into account the mutual relations between the three departments.

Since we in essence solve a curriculum based course timetabling problem as a post-enrolment based course timetabling problem, we cannot compare our results with the existing results. In order to have some means of comparison, we implemented three extra algorithms to which we will compare our hybrid GLS approach:

- a tabu search algorithm hybridized with token ring search and with a static cost function. We will refer to this algorithm as the tabu search token-ring (TS-TR) algorithm.
- a variable neighbourhood descent (VND) algorithm. Actually, this algorithm is a combination of a steepest descent local search algorithm and a slightly altered version of the token-ring search, in which the algorithm switches back to the first neighbourhood whenever a better solution is found in another than the first neighbourhood.
- an adapted variable neighbourhood descent (VND*) algorithm in which we only switch to another neighbourhood after a pre-defined number of non-improving iterations.

Every data set is solved using these four algorithms. In each algorithm the same neighbourhoods and the same stopping criterion is applied. All algorithms either end when the number of iterations exceeds 10000 or when the value of the cost function becomes zero, i.e. when a solution is found that satisfies all the hard and soft constraints.

In our implementation of the token-ring tabu search algorithm, two parameters are important:

- the maximum size of the tabu list length: we use an ordered list of primes as possible tabu list lengths and select the next prime whenever one iteration fails to produce a better solution. This process is repeated until the maximum tabu list length is reached. The tabu list length becomes the previous prime of the ordered prime list if a better solution is found. The upper limit is the largest prime that is smaller than the maximum tabu list length parameter, whereas the lower limit is 7. The idea behind this method is that by increasing the tabu list length, more moves become tabu and the algorithm is forced to explore new regions in the search space. A decreasing tabu list length compels the algorithm to investigate the current region in detail.
- the number of non-improving iterations before the algorithm switches to another neighbourhood.

In the following tests we search for the best combination of these two parameters for the four data sets.

Since the variable neighbourhood descent algorithms (VND and VND*) do not contain a tabu list, the only important parameter determines the number of non-improving iterations.

The two tabu search algorithms are executed with the ten parameter settings presented in the first two columns of Table 4.3, while the parameter settings for the variable neighbourhood descent algorithms are shown in the last column of Table 4.3. We choose 25 and 50 as the maximum tabu list lengths. This is partly based on the experiments in [77], and also on preliminary tests in which we deducted that these values lead to the best solutions.

4.5.1 First Phase: Independent Data Sets

In the first phase, we solve the three data sets independently. We assume that they do not share common resources. Since all three independent data sets can be solved to feasibility while satisfying all the soft constraints (in other words, solutions with cost 0), we decided to plot the number of iterations needed to obtain these solutions against the different parameter settings. The idea is to find the parameters settings that, on a statistically significant manner, lead to solutions with cost 0 in the shortest number of iterations.

tabu search parameter settings		**VND & VND* parameter settings**
number of non-improving iterations	max. tabu list length	number of non-improving iterations
5	25	5
10	25	10
10	50	15
20	25	20
20	50	25
25	50	50
50	25	100
50	50	200
75	25	500
100	25	1000

Table 4.3: Parameter settings for the tabu search and the variable neighbourhood descent algorithms

Data Set 1: Bachelor Chemistry

Each of the four algorithms is applied to solve data set 1. Figure 4.3 presents a graph of box plots for each algorithm. We have chosen not to rescale the y-axis, in order not to loose any information about the outliers. The top left diagram is the box plot representation of the hybrid GLS method. For every set of parameters the algorithm is executed ten times. This process is repeated for the ten different parameter settings (see Table 4.3). The values on the x-axis corresponds to the ten parameter settings. The first parameter is the value of the number of non-improving iterations before neighbourhoods are switched, and the second parameter represents the maximum value of the tabu list length. The values plotted along the y-axis correspond to the number of iterations needed to find feasible solutions satisfying all the soft constraints (we will use the more general term solution for this kind of solutions when no confusion is possible). Two parameter sets lead on average to the smallest number of iterations to find a solution: these are the parameter sets (20,25) and (20,50). The average number of iterations needed to find a solution in both cases is 270.5. Over all 100 test runs, the minimum number of iterations needed to find a solution is 76.

Applying the TS-TR tabu search algorithm to data set 1 results in the top right diagram of Figure 4.3. We use the same parameter settings as in the previous case, for which we repeat each algorithm ten times. The parameter setting (20,25)

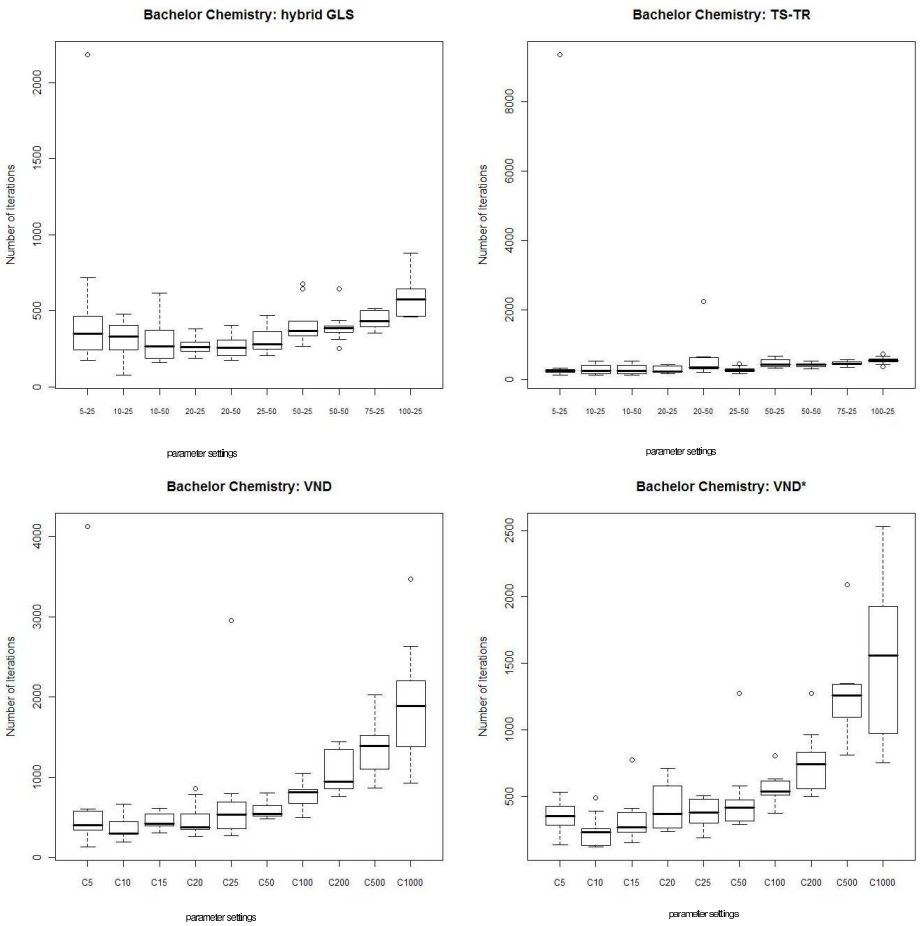

Figure 4.3: Experimental results for the bachelor Chemistry data set with each of the four algorithms.

leads to a feasible solution satisfying all the constraints in, on average, 272.6 iterations. Over the 100 runs of the TS-TR tabu search algorithm the overall minimum number of iterations needed to find a solution is 117.

In the variable neighbourhood descent algorithms VND and VND* (represented in the bottom left, respectively right diagram of Figure 4.3) only the number of non-improving iterations before the algorithm switches to another neighbourhood needs to be set. We choose for both variable neighbourhood algorithms 10 parameter settings (see Table 4.3) and each set is executed ten times. On average the VND* algorithm needs fewer iterations to find a solution. The VND* algorithm with

parameter setting 10 for the number of non-improving iterations outperforms the three other algorithms with an average value of 242.6 iterations to find a solution.

In Table 4.4 we present the p-values (with 95% confidence) for all the algorithms. The fastest algorithms - actually, the algorithms that find solutions in fewer iterations - are indicated in bold. The best performing algorithm (with respect to the average number of iterations) is VND* with maximum number of non-improving iterations equal to 10.

In Table 4.5 the performance of the different algorithms is compared with respect to the maximum number of non-improving iterations. None of the 4 algorithms performs better that the others in a statistically significant manner.

Data Set 2: Bachelor Physics

We repeat the same procedure as in Section 4.5.1 on data set 2. The upper part of Figure 4.4 presents the box plots that correspond to executing each tabu search approach ten times for the ten parameter settings of Table 4.3. On average the hybrid GLS method needs fewer iterations compared to the TS-TR algorithm to find solutions satisfying all the constraints. In both cases the parameter setting (10,25) performs best.

With respect to the VND and VND* algorithms, the lack of memory has an effect on the number of executed iterations before a solution is found (see bottom part of Figure 4.4). Both algorithms execute more iterations than their tabu search based counterparts to find solutions that satisfy all the constraints. Note that the VND method in one experiment cannot find a solution that satisfies all the constraints within 10000 iterations.

Applying a t-test with a 95 % confidence interval to all algorithms, shows that the hybrid guided local search method with parameters (25,10) is the performs best (see Table 4.6).

Table 4.7 shows that the hybrid local search algorithms, on 4 out of the 6 tests outperform the other algorithms, however not in a statistically significant way.

Data Set 3: Bachelor Commercial Engineering

The Commercial Engineering data set differs from the two previous data sets, in the sense that the number of students is five times larger compared to the number of students in the bachelor Physics data set. The tabu search based algorithms (hybrid GLS and TS-TR) outperform the variable neighbourhood descent algorithms (see Figure 4.5). The parameter set (25,50) applied on the hybrid GLS algorithm finds a solution in on average 144.4 iterations, whereas the

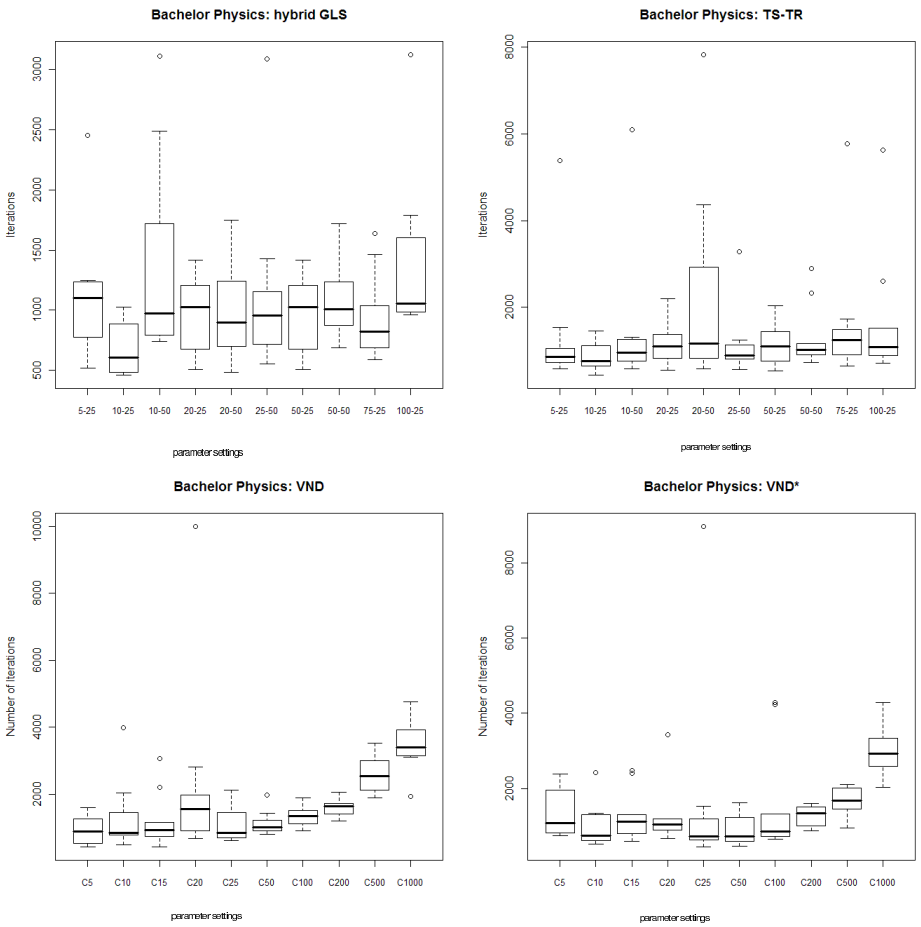

Figure 4.4: Experimental results for the bachelor Physics data set with each of the four algorithms.

same parameter set applied to the TS-TR algorithm finds a solution in on average 148.2 iterations. However if we compare this with the fastest parameter setting of the variable neighbourhood descent algorithms, we find that it takes VND and VND* on average 347.3 respectively 228.6 iterations to find a solution. This is also reflected in Table 4.8, in which the tabu search based algorithms are among the best performers.

Table 4.9 shows that the tabu search based algorithms outperform the variable neighbourhood search methods. Only for the last parameter setting, the difference between the two approaches (tabu search based versus variable neighbourhood

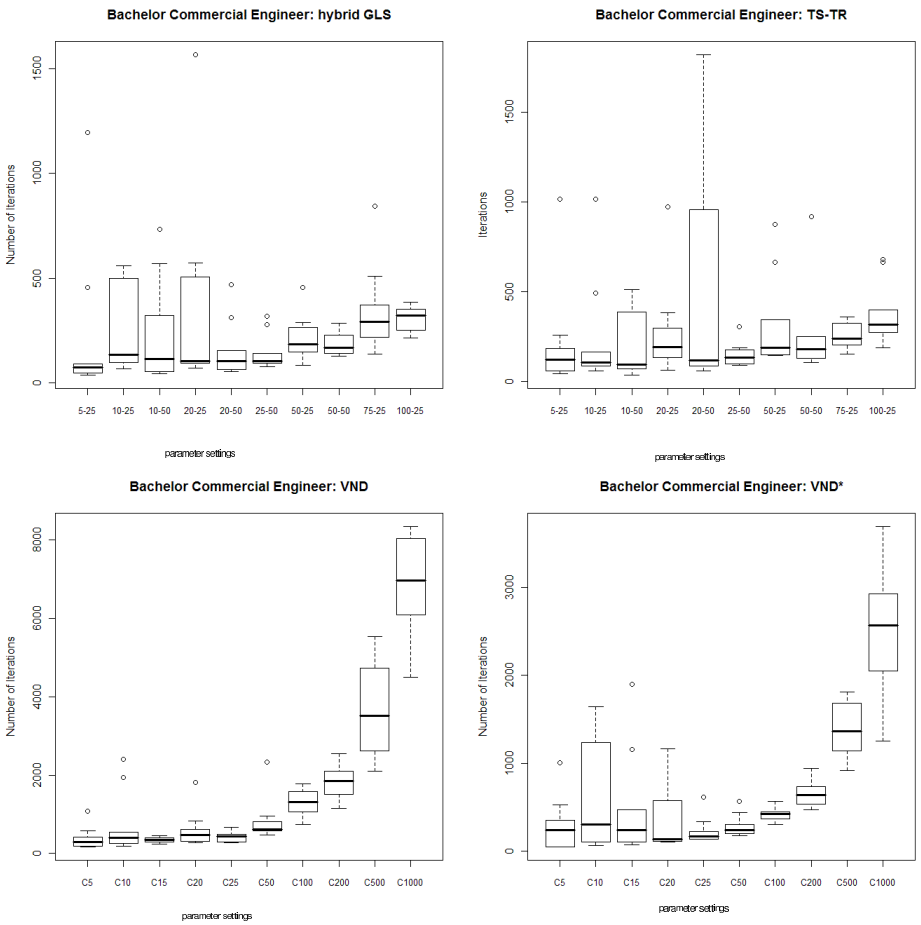

Figure 4.5: Experimental results for the bachelor Commercial Engineering data set with each of the four algorithms.

search based) is statistically significant.

4.5.2 Phase 2: Joint Data Sets

In the second phase we consider the three independent data sets as a joint data set. In this way the mutual relations between the different data sets are taken into account. The four above described algorithms are employed on this joint data set.

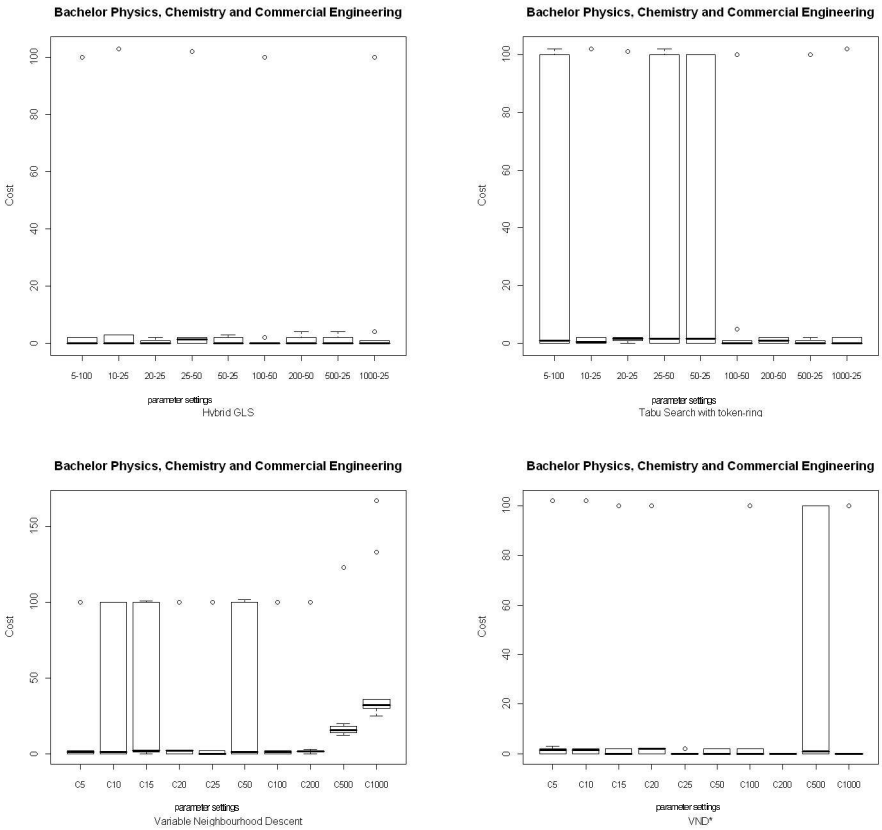

Figure 4.6: Experimental results for the joint data set for each of the four algorithms.

For this joint problem we assign the same values for the number of non-improving iterations parameter for all algorithms (see Table 4.10).

Figure 4.6 and 4.7 represent for the joint problem boxplots for the costs and the number of iterations needed to obtain the best solutions for each parameter setting. It is striking that the VND* algorithm with parameter setting 200 finds solutions with cost 0 for all ten runs. Compared with the hybrid guided local search the VND* algorithm does not find feasible solutions for all parameter settings in all ten runs. The VND and the tabu search hybridised with token-ring do not find feasible results for three of the parameter settings in any of the ten runs. The minimum number of iterations needed to find a feasible solution satisfying all the soft constraint is 1823, and this is obtained in the VND algorithm with parameter

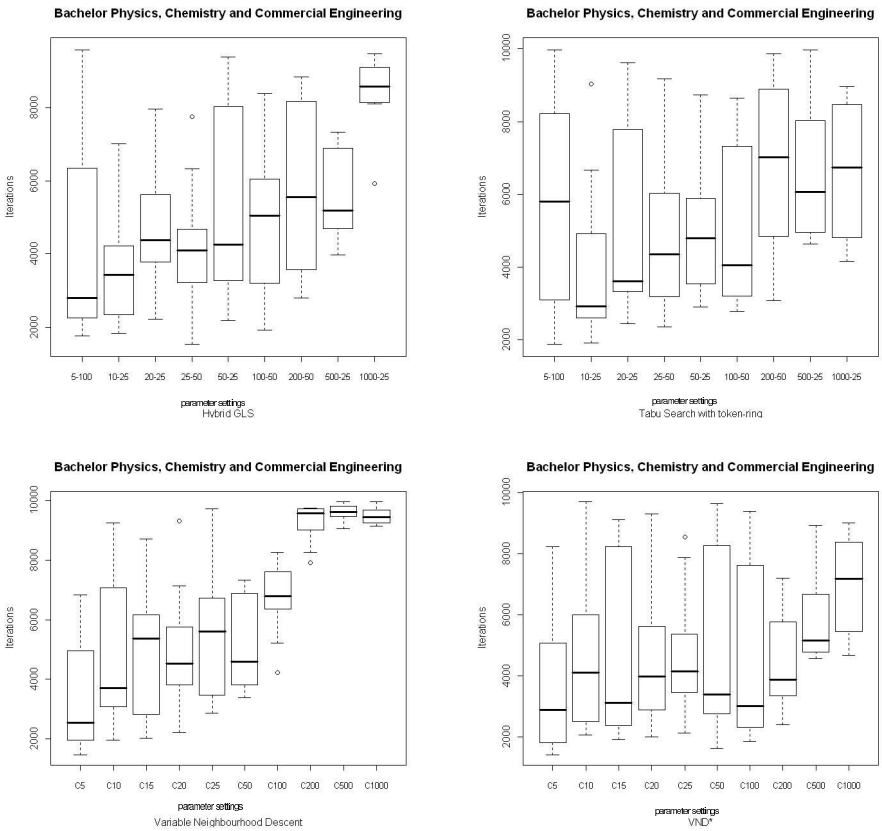

Figure 4.7: Experimental results for the joint data set for each of the four algorithms. The number of iterations needed to find the best solution are presented.

setting 5. Comparing the number of infeasible solutions among the four algorithms, the hybrid GLS only in 5 cases out of 90 runs cannot generate feasible solutions, while the other algorithms find at least twice as many infeasible solutions. Table 4.11 shows that there is no algorithm that performs significantly better than the hybrid guided local search algorithm. VND* performs almost equally well, except that for parameter setting 500, it is significantly outperformed by the hybrid guided local search.

4.5.3 Summary

Since less bookkeeping is involved than in the tabu search case (updating the tabu list, checking whether a move is on the tabu list or not,...), the variable neighbourhood descent algorithms can execute more iterations in less time, but as can be seen from the results it takes more iterations to find a feasible solution that satisfies the soft constraints as much as possible. The variable neighbourhood descent algorithm could be applied as a pre-processing step to generate a good initial solution from which the more complex tabu search based algorithms can be started.

From the experiments it can be deducted that the variable neighbourhood algorithms find equally good solutions as the hybrid GLS method in comparable time for the independent data sets.

However, for the joint data set, the variable neighbourhood descent as well the TS-TR algorithm have problems to provide feasible solutions in each of the ten runs for each of the parameter settings. On the other hand, the variable neighbourhood descent algorithms require only one parameter, whereas the tabu search based algorithms requires two.

The experiments show that for the first three data sets the less complicated VND* algorithm suffices to find feasible solutions that satisfy all the constraints. However for the joint data set the hybrid GLS algorithm prove useful. The hybrid GLS algorithm in general needs fewer iterations to find good solutions than the other three algorithms.

4.6 Conclusion

We report on a post-enrolment based university course timetabling problem at K.U.Leuven that differs from the cases described in the literature. Apart from general timetabling constraints such as event-clashes and unavailability of lecturers, it also considers overlapping time slots. The university course timetabling problem is solved with a hybrid GLS algorithm in which tabu search techniques are incorporated. We compared this approach with three other algorithms, and found that none of the other algorithms perform significantly better. Note however, that a variable neighbourhood descent algorithm hybridized with a token-ring search performs very well on the independent data sets.

We also showed that it is possible to solve an in essence curriculum-based course timetabling problem as a post-enrolment course timetabling problem. In this way, the individual timetables of students taking courses from different curricula are no longer in conflict.

Chemistry	
Method	**p-value w.r.t. method with the best method**
VND*-C10	**1**
GLS-20-25	**0.51**
GLS-20-50	**0.54**
TS-TR-20-25	**0.54**
TS-TR-25-50	**0.51**
TS-TR-10-50	**0.48**
GLS-25-50	**0.19**
GLS-10-50	**0.31**
GLS-10-25	**0.23**
TS-TR-10-25	**0.23**
VND*-C15	**0.26**
VND*-C5	**0.06**
VND-C10	**0.08**
VND*-C25	0.03
GLS-50-50	0.01
GLS-50-25	0.01
TS-TR-50-50	0
VND*-C20	0.02
GLS-75-25	0
VND-C15	0
TS-TR-75-25	0
TS-TR-50-25	0
VND-C20	0.01
VND*-C50	0.02
GLS-5-25	**0.14**
TS-TR-100-25	0
VND*-C100	0
TS-TR-20-50	**0.11**
VND-C50	0
GLS-100-25	0
VND-C25	**0.06**
VND*-C200	0
VND-C5	**0.18**
VND-C100	0
VND-C200	0
TS-TR-5-25	**0.34**
VND*-C500	0
VND-C500	0
VND*-C1000	0
VND-C1000	0

Table 4.4: All results of the Chemistry data sets are ordered according to the average number of iterations needed to obtain a solution with 0. The algorithm performing the best, is compared against the remaining algorithms with a t-test with 95% confidence. The algorithms that need, in a statistically significant manner, fewer iterations to find solutions are indicated in bold.

Chemistry				
Max. # of non-improving iterations	**GLS**	**TS-TR**	**VND**	**VND***
5	*0.34*	*0.4*	*0.29*	*1*
10	*0.23*	*0.23*	*0.08*	*1*
20	*1*	*0.95*	0.01	0.03
25	*0.46*	*1*	*0.08*	*0.06*
50	*1*	*0.32*	0.01	*0.42*
100	*0.44*	*1*	0	*0.88*

Table 4.5: Comparison between the algorithms that have the number of non-improving iterations in common for the chemistry case. In every row, the results of the 4 algorithms are ordered according to the average number of iterations needed to obtain a solution with cost 0. The three remaining algorithms are compared to the algorithm with the lowest average number of iterations with a t-test with a 95% confidence interval. Values in italics indicate that there is no statistical significance with the values obtained by the other algorithms.

Physics	
Method	**P-value w.r.t. best method**
GLS-25-10	**1**
VND*-C50	**0.18**
TS-TR-25-10	**0.11**
GLS-25-20	**0.05**
VND-C5	**0.11**
GLS-25-75	**0.06**
GLS-25-50	0.02
VND*-C10	**0.14**
GLS-50-20	**0.05**
GLS-50-50	0
VND-C25	0.03
GLS-50-25	**0.08**
GLS-25-5	0.02
VND-C50	0
TS-TR-50-25	**0.09**
TS-TR-25-50	0.01
VND-C15	**0.08**
TS-TR-25-20	0.01
VND*-C20	0.04
VND*-C15	0.01
VND*-C200	0
VND-C10	**0.08**
TS-TR-50-50	0.02
VND*-C5	0.01
TS-TR-C25-5	**0.18**
VND-C100	0
GLS-C25-100	0
GLS-C50-10	0.01
TS-TR-50-10	**0.16**
VND*-C100	**0.07**
VND-C200	0
TS-TR-25-75	**0.06**
VND*-C25	**0.26**
VND*-C500	0
TS-TR-25-100	**0.05**
TS-TR-50-20	**0.05**
VND-C20	**0.08**
VND-C500	0
VND*-C1000	0
VND-C1000	0

Table 4.6: All results of the Physics data sets are ordered with respect to the average number of iterations needed to obtain a solution with 0. The best performing algorithm is compared against the remaining algorithms with a t-test with 95% confidence. The algorithms that need significantly fewer iterations to find solutions are indicated in bold.

Physics				
Max. # of non-improving iterations	**GLS**	**TS-TR**	**VND**	**VND***
5	*0.39*	*0.43*	*1*	*0.13*
10	*1*	*0.11*	*0.08*	*0.14*
20	*1*	*0.18*	*0.13*	*0.28*
25	*1*	*0.18*	*0.13*	*0.28*
50	*0.9*	*0.88*	*1*	*0.52*
100	*1*	*0.58*	*0.88*	*0.75*

Table 4.7: Comparison between the algorithms that have the number of non-improving iterations in common for the physics case. In every row, the results of the 4 algorithms are ordered according to the average number of iterations needed to obtain a solution with cost 0. The three remaining algorithms are compared to the algorithm with the lowest average number of iterations with a t-test with a 95% confidence interval.

Commercial Engineering	
Method	**P-value w.r.t. best method**
GLS-25-50	**1**
TS-TR-25-50	**0.91**
GLS-20-50	**0.84**
GLS-50-50	**0.22**
TS-TR-10-50	**0.51**
GLS-50-25	**0.15**
TS-TR-5-25	**0.49**
GLS-5-25	**0.56**
VND*-C25	**0.14**
GLS-10-50	**0.3**
TS-TR-10-25	**0.4**
GLS-10-25	**0.2**
TS-TR-50-50	**0.21**
TS-TR-75-25	0.01
TS-TR-20-25	**0.15**
VND*-50	0.01
VND*-C5	**0.11**
GLS-100-25	0
TS-TR-50-25	**0.06**
GLS-75-25	0.01
GLS-20-25	**0.2**
VND-C15	0
VND*-C20	**0.11**
VND-C5	0.02
TS-TR-100-25	0
VND*-C100	0
VND-C25	0
VND*-C15	**0.1**
TS-TR-20-50	**0.08**
VND*-C10	0.04
VND-C20	0.01
VND*-C200	0
VND-C10	0.04
VND-C50	0
VND-C100	0
VND*-C500	0
VND-C200	0
VND*-C1000	0
VND-C500	0
VND-C1000	0

Table 4.8: All results of the Commercial Engineering data sets are ordered with respect to the average number of iterations needed to obtain a solution with 0. The best performing algorithm is compared against the remaining algorithms with a t-test with 95% confidence. The algorithms that need significantly fewer iterations to find solutions are indicated in bold.

Commercial Engineering				
Max. # of non-improving iterations	**GLS**	**TS-TR**	**VND**	**VND***
5	*0.99*	*1*	*0.23*	*0.48*
10	*0.96*	*1*	*0.09*	*0.13*
20	*0.69*	*1*	*0.06*	*0.6*
25	*1*	*0.91*	0	*0.14*
50	*1*	*0.25*	0	*0.17*
100	*1*	*0.2*	0	0

Table 4.9: Comparison between the algorithms that have the number of non-improving iterations in common for the commercial engineering case. In every row, the results of the 4 algorithms are ordered according to the average number of iterations needed to obtain a solution with cost 0. The three remaining algorithms are compared to the algorithm with the lowest average number of iterations with a t-test with a 95% confidence interval.

tabu search parameter settings		**VND & VND* parameter settings**
max. # of non-improving iterations	max. tabu list length	max. # of non-improving iterations
5	100	5
10	25	10
20	25	20
25	50	25
50	25	50
100	25	100
200	50	200
500	25	500
1000	25	1000

Table 4.10: Parameter settings for the two tabu search algorithms and the two variable neighbourhood descent algorithms

Joint Data Sets (cost)				
Max. # of non-improving iterations	**GLS**	**TS-TR**	**VND**	**VND***
5	*1*	*0.14*	*0.99*	*0.97*
10	*0.99*	*1*	*0.3*	*0.99*
20	*1*	*0.3*	*0.14*	*0.14*
25	*0.3*	0.02	*0.32*	*1*
50	*0.84*	*0.06*	0.03	*1*
100	*1*	*0.97*	*0.54*	*0.99*
200	*0.05*	0.01	*0.27*	*1*
500	*1*	*0.35*	0.03	0.03
1000	*0.97*	*0.97*	0.03	*1*

Table 4.11: Comparison between the algorithms that have the number of non-improving iterations in common for the joint data set case. In every row, the results of the 4 algorithms are ordered according to the average number of iterations needed to obtain a solution with cost 0. The three remaining algorithms are compared to the algorithm with the lowest average number of iterations with a t-test with a 95% confidence interval.

Joint Data Sets (Cost)	
Method	**P-value w.r.t. best method**
VND*-C200	**1**
VND*-C25	**0.15**
GLS-25-20	**0.08**
VND*-C50	**0.07**
GLS-25-50	**0.07**
GLS-25-500	**0.09**
GLS-50-200	**0.05**
TS-TR-50-200	0.01
VND*-C1000	**0.33**
GLS-50-100	**0.32**
TS-TR-25-500	**0.32**
VND*-C15	**0.31**
VND*-C100	**0.31**
GLS-25-1000	**0.31**
VND-C25	**0.31**
GLS-100-5	**0.3**
TS-TR-25-1000	**0.31**
TS-TR-50-100	**0.3**
VNDâĂŤC5	**0.29**
TS-TR-25-10	**0.3**
GLS-25-10	**0.29**
GLS-50-25	**0.29**
VND*-C10	**0.29**
TS-TR-25-20	**0.28**
VND*-C5	**0.28**
VND-C200	**0.27**
VND-C100	**0.14**
VND-C20	**0.13**
VND*-C20	**0.13**
VND-C500	0.03
VND-C10	**0.06**
TS-TR-25-50	**0.06**
TS-TR-25-5	**0.06**
VND*-C500	0.02
VND-C50	0.02
TS-TR-50-25	0.02
VND-C15	0.02
VND-C1000	0

Table 4.12: All results of the joint data sets are ordered with respect to the average number of iterations needed to obtain a solution with 0. The best performing algorithm is compared against the remaining algorithms with a t-test with 95% confidence. The algorithms that need significantly fewer iterations to find solutions are indicated in bold.

Chapter 5

Examination Timetabling

5.1 Introduction

In this last chapter on university timetabling cases, three examination timetabling problems are tackled. Examination timetabling is the problem in which exams are assigned to a limited number of time slots and rooms, taking into consideration several constraints. The most widespread constraints are:

- no students take two exams at the same time,
- the study intervals between exams are sufficiently long for each student, and
- the capacity of rooms is respected.

For this problem, competition instances [1] as well as benchmark problems are available. The benchmark problems have been derived from several real-world problems in universities such as the University of Toronto, the University of Nottingham, the University of Melbourne (see [127] for a comprehensive overview).

The real-world problem we introduce in this chapter is the examination timetabling problem at the Engineering Department of KaHo Sint-Lieven. We refer to it as the KaHo examination problem in the remainder of the chapter. The problem under investigation is new. It differs from known examination timetabling problems in that it addresses both oral and written exams. We address the problem with a hyper-heuristic approach and compare the timetable with the manual results that were generated in parallel. For the first semester of the academic year 2005-2006, we managed to arrange the exams on weekdays only, whereas the manual planner could not avoid scheduling some exams on Saturdays. The performance of the

hyper-heuristic algorithm is also tested on both the Toronto benchmarks and the 2007 International Timetabling Competition [1] data sets. For five of the Toronto benchmark instances, we obtain better results than the current best reported in the literature.

The contribution of this chapter is threefold:

1. the introduction of a complex real-world problem in examination timetabling,
2. a hyper-heuristic approach to successfully tackle the problem, and
3. a performance analysis of the solution approach, including comparison with the best approaches for two sets of benchmark problems in the recent exam timetabling literature.

The claim of this chapter is that the hyper-heuristic approach can successfully address complex real-world problems as well as benchmark problems from the scientific literature. There is no one to one match between the problem descriptions, and the difference between the models is taken care of by the low-level heuristics.

A detailed problem description and a model is presented in Section 5.2. We elaborate on the difference between the problem and two best known benchmark problems for examination timetabling. Section 5.3 introduces the components of the hyper-heuristic approach. In Section 5.4, we discuss the experimental results for the KaHo examination timetabling problem, the Toronto and the International Timetabling Competition 2007 instances. We conclude the chapter and point out some directions for future research in Section 5.5.

This chapter is based on *P. Demeester, B. Bilgin, P. De Causmaecker, and G. Vanden Berghe. A hyperheuristic approach for real world and benchmark examination timetabling problems. Technical report, KaHo Sint-Lieven, 2009. submitted to Journal of Scheduling.*

5.2 Examination Timetabling Problem Description

5.2.1 The KaHo Examination Problem

Unlike the examples discussed in the literature, the KaHo Sint-Lieven examination timetabling problem combines both written and oral exams. Written exams of the same subject should be scheduled in the same time slot for all the students involved, whereas a lecturer can take at most 20 students in the same time slot for oral exams. An oral and a written exam should not be assigned to the same

room at the same time. Different written exams can be organized in the same room simultaneously, provided that the room capacity is not exceeded. At KaHo Sint-Lieven the examination capacity of a room ranges from one third to half of its capacity for lectures. The actual ratio depends on the physical properties of the room. Students are automatically enrolled for the exams that correspond to their curriculum at the beginning of the academic year. This causes - especially in the first year - a high percentage of no shows and a bad usage of the available rooms. Exams are assigned to rooms larger than really needed. An exam takes at most half a day (from 8h30 till 12h30 or from 14h00 till 18h00) and should only be scheduled on workdays. Students should have at least 3 half days of study time between 2 consecutive exams. A typical examination period takes 4 weeks of 5 days each, resulting in 40 available time slots.

We solve the data set for the first semester of the academic year 2005-2006. This was the last semester in which exams were scheduled on Saturdays. The manual planner actually needed 24 days (or 48 time slots) to schedule the exams. Apart from satisfying the above constraints, the aim is to eliminate the extra Saturdays as well.

The problem size includes:

- 336 exams;
- 135 student groups;
- 71 rooms of different size and
- 40 time slots of 4 hours each.

Note that we do not take the assignment of invigilators into account. In practice, for oral exams, the examiner is at the same time invigilator, while for written exams, teaching assistants act as invigilators. We do plan to incorporate the assignment of invigilators in future work.

5.2.2 Constraints and Objective

We identify in this section the different constraints of the KaHo Sint-Lieven examination timetabling problem and split them up into hard and soft constraints.

The **hard constraints** include that:

- Students cannot take two exams at the same time. Lewis [106] calls this the *event-clash constraint*.

- The number of students assigned to a room cannot exceed its capacity. In [106] this is called the *capacity constraint*.
- All exams should be scheduled within the planning horizon of four weeks.

Although not all the following rules are considered to be **soft constraints** at KaHo Sint-Lieven, we rank them as soft since they do not make the solution 'physically infeasible' when violated. The soft constraints are presented in decreasing order of importance, meaning that violations of the first constraints will be attributed a higher weight than the latter.

- All written exams of the same subject should be scheduled in the same time slot.
- Oral and written exams should not be merged into the same room.
- All oral exams should be scheduled such that the maximum number of examinees per time slot is at most 20. Lecturers who are responsible for oral exams can only examine one student group at the same time.
- Students should have at least three time slots of study time between two consecutive exams. This corresponds to the *spreading-out constraint* [106].

The objective of the problem is to find a feasible solution that minimizes the weighted sum of the violations of the soft constraints.

5.2.3 Mathematical Formulation

In examination timetabling, E exams need to be scheduled into a limited number of time slots T and rooms R, taking into account the capacity of the rooms, and the fact that students cannot take more than one exam at the same time, among other constraints. Every student group takes a number of exams. In the examination timetabling problem at hand exams depends on the student group's curriculum, which is similar to the curriculum based university course timetabling case.

The decision variables are

$$x_{rte} = 0 \; or \; 1 \tag{5.1}$$

with $(r = 1, .., R; t = 1, .., T; e = 1, .., E)$, and where $x_{rte} = 1$ if exam e is assigned to room r in time slot t. An exam e is characterized by a subset of the set of all student groups, taking the same course and a lecturer l (with $l = 1, ..., L$, with L the total number of lecturers). A student group is characterized by s (with $s = 1, ..., S$, with S the total number of student groups).

- Let S_{rt} be the total number of students assigned to room r at time slot t.
- Let W denote the set of all written exams and O the set of all oral exams. Since no exam can be both written or oral, this means that: $W \cap O = \emptyset$.
- Let N_e denote the number of student groups taking an exam e of the same course and the same lecturer.
- Let CAP_r be the capacity of room r.
- Let A denote the conflict matrix. This matrix represents the exams that cannot be scheduled in the same time slot (these are the so-called 'conflicting exams'), due to exams that have common students. This is a symmetric $E \times E$ matrix. The matrix elements on the diagonal are 0, while the values of the non-diagonal matrix elements correspond to number of students that the exams involved have in common.

$\forall$ rooms r, time slots t, and exams e:

$$\sum_{m=1}^{E} A_{me} x_{rtm} \leq M(1 - x_{rte}) \tag{5.2}$$

with M an arbitrary large number.

Equation 5.2 expresses that no student group can take more than one exam in the same time slot.

$\forall$ exams e, rooms r, and time slots t:

$$S_{rt} x_{rte} \leq CAP_r \tag{5.3}$$

Equation 5.3 expresses that the total number of students assigned to a room cannot exceed the room's capacity.

$\forall$ exams e, and rooms r:

$$\sum_{t=1}^{T} x_{rte} = E \tag{5.4}$$

Equation 5.4 expresses that all exams are scheduled.

Solutions satisfying the equations 5.2 till 5.4 are feasible. In order to find feasible solutions that do not violate the soft constraints, the following objective function (Eq. 5.5) is introduced.

$$Min(\sum_{t=1}^{T}\sum_{e\in W} P_{1,te} + \sum_{e\in O} P_{2,e} + \sum_{r=1}^{R}\sum_{e=1}^{E}\sum_{t=1}^{T} P_{3,rte} + \sum_{e=1}^{E}\sum_{r=1}^{R} P_{4,er}), \tag{5.5}$$

with P_1, P_2, P_3, and P_4 the result of the violations of the soft constraints. The penalties corresponding to the violations of the soft constraints are expressed below.

$\forall$ exam $e \in W$, $\forall$ time slots t:

$$P_{1,te} = |\sum_{r=1}^{R} x_{rte} - N_e| \tag{5.6}$$

Equation 5.6 generates a penalty when not all of the written exams of the same subject are assigned to the same time slot.

$\forall$ exam $e \in O$:

$$P_{2,e} = |\sum_{r=1}^{R}\sum_{t=1}^{T} x_{rte} - N_e| \tag{5.7}$$

Equation 5.7 corresponds to the soft constraint which requires that the number of oral exams of the same course should equal the number of student groups taking the course.

$\forall e_1 \in W$ and $e_2 \in O$, and $\forall$ rooms r, and time slots t:

$$P_{3,rte} = max((x_{rte_1} + x_{rte_2}) - 1, 0) \tag{5.8}$$

Equation 5.8 corresponds to the soft constraint which requires that an oral and a written exam cannot be assigned to the same room in the same time slot.

$\forall$ exams e, and rooms r:

$$P_{4,er} = \sum_{t=1}^{T-1} x_{rte}x_{rt+1e} + \sum_{t=1}^{T-2} x_{rte}x_{rt+2e} + \sum_{t=1}^{T-3} x_{rte}x_{rt+3e} \tag{5.9}$$

This last equation 5.9 corresponds to the soft constraints which requires that for each student group there should be at least 3 time slots of study time between two exams.

5.2.4 Benchmarks versus KaHo Examination Problem

In Table 5.1 we summarize the differences between the KaHo Sint-Lieven problem and the two examination timetabling benchmarks against we compare: the Toronto and the International Timetabling Competition 2007 [1] benchmark data sets. Note that the KaHo examination problem differs from the others in the considered examination types and in the smallest examinee unit. The KaHo examination problem is clearly a curriculum-based examination timetabling problem, whereas the two benchmark problems are post-enrolment based examination timetabling problems. As a consequence, the exams at KaHo Sint-Lieven are organized for student groups taking the same curriculum, while in the Toronto and ITC 2007 benchmark instances, every student has an individual examination timetable.

	KaHo Sint-Lieven	**ITC 2007**	**Toronto**
exam types	oral AND written	written	written
smallest unit	student group	student	student
exam durations	equal	different	equal
exam order	not important	important	not important
room size	important	important	not important

Table 5.1: Differences between the Toronto and ITC 2007 benchmark and the KaHo Sint-Lieven examination timetabling problem.

5.3 Solution Methods

In this section we introduce the hyper-heuristic approach and its components. We start in Section 5.3.1 with the four acceptance criteria and the heuristic selection method that we employ. Section 5.3.2 introduces the low-level heuristics.

5.3.1 Hyper-heuristics Framework: Acceptance Criteria and Heuristic Selection Method

Hyper-heuristics are higher level search and optimization methods compared to (meta-)heuristics [32]. Instead of operating directly on a set of candidate solutions like it is the case in meta-heuristics, they operate on a set of (meta-)heuristics. These low-level heuristics can be either perturbative (changing only small parts of the solution) or constructive (constructing a solution). Another objective of the hyper-heuristics research is to generate synergy between heuristics involved in the search, making use of their strengths, avoiding their weaknesses, and taking

advantage of their combined capabilities. A typical hyper-heuristics framework consists of some mechanism to select a heuristic, apply it to a single candidate solution, and either accept or decline the resulting candidate solution at the end of the iteration. This means that typically an iteration of a hyper-heuristic consists of two parts: the heuristic selection method and the acceptance criterion. Several well-known meta-heuristic methods are deployed within hyper-heuristics as selection methods as well as acceptance criteria [122]. As a **heuristic selection method** we apply 'simple random', which randomly chooses a heuristic from a fixed list of heuristics. At each iteration, the selected heuristic creates a predefined number of random moves. This number is called the *tournament factor*. The move with the lowest cost among the random moves is selected. Evaluation of a solution by the cost function (Section 5.2.2) results in a cost value that reflects the solution quality: the lower the cost of a solution, the better the solution. If the best move is accepted by the acceptance criterion, it modifies the current solution.

Algorithm 3 Pseudo code of the simulated annealing acceptance criterion

T = total execution time
R = remaining execution time
C_i = cost value of candidate solution at i^{th} iteration
C^i_{best} = cost value of the best solution at i^{th} iteration
P_i = random number between [0,1[at i^{th} iteration
$\delta = C_{i+1} - C_i$
if $\delta \leq 0$ **then**
    Accept
else
    if $P_i < exp^{\frac{-\delta T}{(1.002T-R)\sqrt{C^i_{best}}}}$ **then**
        Accept
    else
        Reject
    end if
end if

We consider the following four **acceptance criteria** for the experiments:

- improving or equal, abbreviated as **IE**, which only accepts moves that do not worsen the solution;
- simulated annealing, abbreviated as **SA**. The algorithm is represented in pseudo code in Algorithm 3. Note that the acceptance criterion does not contain any parameters that need to be tuned. It is in contrast to [24], for example, in which the authors report spending considerable time tuning the problem specific parameters in their simulated annealing algorithm;

- great deluge, abbreviated as **GD**. The great deluge algorithm is presented in pseudo code in Algorithm 4;
- an adapted version of the late acceptance strategy [26], abbreviated as **LA**. Similar to tabu search, the late acceptance strategy makes use of a list of a given length l. However, the acceptance list does not contain properties of tabu moves but the cost values of the previous l candidate solutions. Rather than comparing a candidate solution with the current solution, it is compared with the oldest value in the list. When the candidate solution's cost is less than the oldest value in the list, the candidate solution is accepted, and its cost replaces the oldest value in the list. The late acceptance variant that we introduce in this paper first compares the candidate solution's cost with the current solution's cost, which is actually a (limited) classic steepest descent. It is no steepest descent in the strict sense, since instead of exploring the complete neighbourhood, only a small part is explored. In case the candidate solution is worse than the current solution, the original late acceptance strategy is applied. Algorithm 5 summarizes the steepest descent late acceptance strategy algorithm in pseudo code.

Algorithm 4 Pseudocode of the great deluge acceptance criterion

T = Total Execution Time
R = Remaining Execution Time
C_i = Cost value at i^{th} iteration
C_0 = Initial Cost
D = R / T
if $C_{i+1} \leq C_i$ **then**
    Accept
else
    if $C_{i+1} < C_0 * D * \sqrt{D}$ **then**
        Accept
    else
        Reject
    end if
end if

5.3.2 Low-Level Heuristics

The hyper-heuristics framework is built on top of a local search framework, for which an appropriate solution representation is generated. We mimic the timetabling process of a human planner as closely as possible using a two dimensional matrix. The rows of the matrix correspond to the rooms and the columns to the time slots. The total number of time slots corresponds to the

Algorithm 5 Pseudo code of the *steepest descent late acceptance strategy* acceptance criterion

initialize list C_L^*: fill complete list of fixed length L with the cost of initial solution
C_v^* = cost value in the list C_L^* at the ith iteration with $v = i(\text{mod})L$
C_i = cost value of candidate solution at i^{th} iteration
$\delta = C_{i+1} - C_i$
if $\delta \leq 0$ **then**
    add C_{i+1} to beginning of list
    Accept
else
    if $C_{i+1} \leq C_v^*$ **then**
        add C_{i+1} to beginning of list
        Accept
    else
        add C_i to beginning of list
        Reject
    end if
end if

examination period. A room-time slot combination can hold several exams, which can be individually moved to another room-time slot combination. The total number of exams in the same room-time slot combination is limited by the room's capacity.

To overcome the under-utilization of the rooms (see Section 5.2.1), we introduce a show up factor: this factor represents the number of students that will probably attend the exam. Throughout this paper we assume that 95% of the students show up. That is a safe estimation for the first year's students. Anyway, rooms are in general sufficiently flexible to accommodate a few extra students.

We distinguish between the heuristics that were developed for solving the capacitated KaHo and ITC 2007 examination problem and the heuristics that lead to good results according to the literature for the uncapacitated Toronto examination timetabling problem. We call the first set of heuristics the capacitated heuristics and the latter the uncapacitated heuristics:

- All the capacitated heuristics rely on one basic move: relocating an exam to another room-time slot combination. In case the capacity of the destination position's room is insufficient, the content of both matrix elements is swapped; otherwise, the exam is simply moved from the original to the destination position. The heuristics restrict the directions in which the content can be moved. The four heuristics that both capacitated examination

timetabling problems have in common are:

- **Move-An-Exam-To-A-Random-Room-TimeSlot-Combination-Heuristic** – K_1: This heuristic restricts the allowed moves to random room-time slot combinations. The destination room and time slot need to be different from the original room and original time slot.
- **Move-An-Exam-To-A-Large-Enough-Room-Heuristic** – K_2: This heuristic only allows moves in the same time slot to rooms that have enough excess capacity to accommodate the students taking the exam.
- **Move-An-Exam-To-A-Random-Room-In-The-Same-TimeslotHeuristic** – K_3: An exam is moved to a randomly chosen room, within the original time slot.
- **Move-An-Exam-To-A-Random-Timeslot-In-The-Same-RoomHeuristic** – K_4: An exam is moved to a randomly chosen time slot, while maintaining the original room.

Since the ITC 2007 examination timetabling problem has some extra soft constraints, compared to the KaHo problem, we constructed an additional set of low-level heuristics that address these constraints in particular:

- **Move-An-Exam-To-Another-TimeSlot-In-The-Same-Room-Taking-Into-Account-Period-Penalty** – C_1: A randomly chosen exam is moved to another time slot in the same room, avoiding these time slots that introduce an extra period penalty.
- **Move-An-Exam-To-Another-Room-In-The-Same-TimeSlot-Taking-Into-Account-Room-Penalty** – C_2: A randomly chosen exam is moved to another room in the same time slot, avoiding rooms that introduce an extra room penalty.
- **Move-Large-Exams-To-The-Beginning-Of-The-Exam-Period** – C_3: A randomly chosen exam is first analyzed with respect to its size. If it is recognized as a large exam, then it is moved to a time slot in the beginning of the examination period.

None of the above heuristics takes special precautions to maintain the feasibility of a solution: executing one of these heuristics on a feasible solution does not guarantee that the resulting new solution will also be feasible.

- *Kempe chain* based heuristics allow the perturbation of feasible solutions for the uncapacitated examination timetabling problem without making them infeasible. Suppose a partial solution that corresponds to the left hand side of Figure 5.1. If we want to move exam e_1 to time slot t_j, the solution becomes infeasible, since exam e_1 has students in common with exams e_6, e_7, and e_8 that are assigned to time slot t_j. To overcome this, exams e_6, e_7,

and e_8 should be moved to time slot t_i. This process is repeated until all the exams that have students in common are assigned to different time slots. The result is depicted at the right hand side of Figure 5.1. The corresponding Kempe Chain heuristic L_1 can be seen as moving an exam to another time slot while maintaining feasibility by repairing any infeasibilities that may have been introduced.

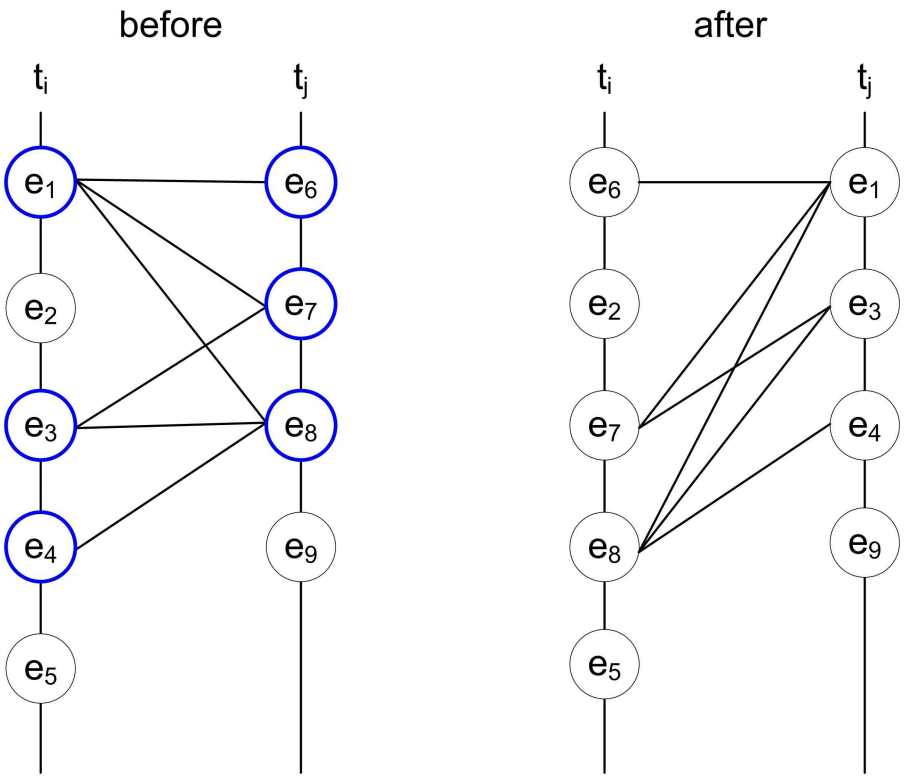

Figure 5.1: An example of the Kempe chain heuristic. A move of exam e_1 in time slot t_i to another time slot t_j requires repair moves to maintain feasibility. Exams e_3 and e_4 have to be moved to time slot t_j and exams e_6, e_7, and e_8 have to be moved to time slot t_i.

The swap-two-time slots heuristic L_2, introduced in [26], swaps all exams of two randomly chosen time slots. This heuristic maintains feasibility, since the swapping of exams includes all exams assigned to a time slot.

5.3.3 Three Low-Level Heuristics Sets for three Examination Timetabling Problems

All three examination timetabling problems are tackled in the same way. The approaches only differ in the applied heuristics and in the way the initial solution is constructed.

- For the KaHo examination problem, we do not spend much effort in constructing a good quality initial solution. All the exams are randomly assigned to free room-time slot combinations. The hyper-heuristic adapts the solution with perturbative heuristics until a feasible solution is obtained. By attributing a higher weight to the violations of the hard constraints in the cost function, the search is biased towards finding a feasible solution while trying to improve the quality. Since none of the low-level heuristics remove exams from the timetable, the third hard constraint of the KaHo examination problem (Section 5.2.2) is always satisfied and will not be a part of the cost function.
- The same random initialization procedure was also tried for the Toronto instances, but too many iterations were required to reach a feasible solution from a randomly generated one for most of the problem instances. For this reason, we decided to apply the conflict matrix (Section 2.1.2) in order to construct a possible feasible solution. Starting from the conflict matrix, we assign exams to time slots, such that no student takes two exams at the same time. The procedure assigns exams in order of decreasing number of conflicts. This is comparable to what many human planners do. They start planning with the difficult exams that have a lot of constraints, so that the exams with a lower number of constraints are planned at the end. Carter et al. [51] call this the largest degree node colouring heuristic. Some noise is added to the initialization phase through randomly assigning exams that cause no conflicts with exams that were assigned before to one of the non-conflicting time slots. If no feasible solution can be found after assigning all exams, the conflicting exams are removed from the solution together with the randomly assigned non-conflicting exams. After that, the removed exams are reassigned. This process is repeated until a feasible solution is found or until the total number of attempts exceeds a certain parameter. In the latter case, the solution with the lowest number of unassigned exams is taken as the new solution, and the unassigned exams are randomly assigned to time slots. The KaHo capacitated heuristics are then applied to obtain a feasible solution through perturbative changes. Once a feasible solution is found, the set of allowed low-level heuristics are limited to the literature heuristics L_1 and L_2. This corresponds to the findings of Thompson and Dowsland [138] who also solve the examination timetabling problem in two distinct phases. They use different neighbourhoods for every phase. The

capacitated heuristics are not used in the second phase since they do not maintain feasibility. Preliminary tests also showed that turning an infeasible solution into a feasible one was really hard, so we opt to construct a feasible solution that is locally perturbated to improve its quality.

- In the ITC 2007 examination timetabling problem, the exams are ranked in decreasing order according to the number of students taking the exam. In a first phase, exams are assigned to a room such that both the capacity and duration constraints are satisfied. This process is repeated for the first T (=number of time slots) exams. The remaining exams are assigned to room-time slot combinations as follows:
  - First check whether the exam can be assigned to one of the occupied room-time slot combinations. If this is the case, assign the exam to the first appropriate room-time slot combination that is encountered;
  - If there is no sufficient capacity left in any of the occupied room-time slot combinations, randomly assign the exam to an empty one, taking into account that the size of the room should be larger than or equal to the number of students taking the exam. In addition, the duration of the exam should be less than or equal to the duration of the time slot.

  The construction of the initial solution does not guarantee the feasibility of the solution, since we only take two of the five hard constraints into account. The remainder of the search algorithm consists of two large parts:
  - in the first part the goal is to find a feasible solution;
  - in the second part, starting from the feasible solution, the goal is to find a feasible solution that satisfies as much as possible all the soft constraints.

  The first part of the search is further divided in three stages:
  - In a first stage, starting from the constructed initial solution, the algorithm tries to obtain a solution that satisfies the hard constraints concerning the room occupancy, the period utilization and room related issues (some of the exams can only be assigned to rooms that do not hold any other exams). The period utilization and room occupancy constraints are already satisfied due to the construction of the initial solution.
  - In a second stage, which starts from a solution that satisfies the three hard constraints discussed above, the evaluation of the period related constraint is added to the cost function. This stage stops when a solution is found that satisfies all four constraints.
  - In the final stage, the last hard constraint (two conflicting exams cannot be assigned to the same time slot) is added to the cost function. The search finishes when a feasible solution is found.

Once a feasible solution is found, the second part of the search begins: the soft constraints are added to the cost function and the search starts from the feasible solution obtained in the first part of the search. To avoid as much as possible any violation of the hard constraints, the weights of the violations of the hard constraints in the cost function are increased to 1000, while the weights of the soft constraints get the value 1.

5.4 Experiments

Every hyper-heuristic variant is executed ten times on each problem instance. Throughout the remaining of this chapter, all the quantitative comparisons are assessed using the Student t-test method with a 95% confidence level. The stopping criteria differ from problem to problem:

- for the KaHo problem it is 1 hour;
- for the Toronto benchmarks the stopping criteria range from 1 hour to 48 hours. This is due to the fact that since the larger instances (e.g. `car91`, `car92`, `rye92` and `pur93`) contain more students and exams, the number of iterations that can be executed in the same period of time is rather low compared to the smaller data instances;
- for the ITC 2007 data sets, the stopping criterion depends on the performance of the computer on which the data sets are executed. To allow a fair comparison between the different solution approaches, the organizers released a benchmark programme that, dependent on the contestant's computer performance, gives the available calculation time to find a solution. In our case the stopping criterion was fixed at 5 minutes.

We also compare the effect of the original late acceptance strategy [26] and the late acceptance strategy combined with the steepest descent algorithm. From the experiments it is clear that the combination of the late acceptance strategy and steepest descent leads to better results for the KaHo examination problem. However, the experiments show that the length of the acceptance list plays a crucial part in the success of the algorithm. On the other hand, the performance difference between the two late acceptance strategies for the Toronto and the ITC 2007 data set benchmarks is not that clear. For some of the instances, the original late acceptance strategy outperforms the adapted version and vice versa.

5.4.1 The KaHo Examination Timetabling Problem

Table 5.2 presents the experimental results for the KaHo case. For the hyper-heuristic with the simulated annealing acceptance criterion with tournament factor 4 (abbreviated as SA-4), we experimented with different types of low-level heuristics. In the first experiment, we applied only the four KaHo heuristics, while in the second experiment, all heuristics (four KaHo and two literature heuristics) were applied. From Table 5.2 it is clear that applying only the KaHo heuristics leads to better results. Based on these findings, we decided to only employ the KaHo heuristics for the further experiments. The steepest descent variant of the late acceptance strategy outperforms the original late acceptance strategy for each of the tournament factors in a statistically significant way. The difference between the two best performers, i.e. the simulated annealing acceptance criterion with tournament factor 4 and the combination of the late acceptance strategy and steepest descent with tournament factor 8 (abbreviated as LA-SD-8) is not statistically significant.

Several tests were performed to obtain the ideal list length for the late acceptance criterion. In Figure 5.2, the average cost over 10 runs versus the list length is plotted for the two late acceptance strategies. The best list length for both strategies is 10000. Compared to the other acceptance criteria (i.e. simulated annealing, improving equal, and great deluge), the dependence of both late acceptance criteria's performance on one parameter is a disadvantage. The length of the acceptance list is an important factor of the algorithm's success for the KaHo examination problem. From the experiments it is also clear that the late acceptance strategy combined with steepest descent is less dependent on the list length than the original late acceptance strategy method.

The experiments show that all algorithms, except the late acceptance strategy, generate examination timetables satisfying all the hard and soft constraints. The timetable is organised in 40 time slots, instead of the 48 time slots that were needed by the manual planner.

5.4.2 Toronto Benchmark Problems

The Toronto benchmark data sets differ considerably from the KaHo Sint-Lieven examination timetabling problem. For example, they do not take classrooms into account, and they do not distinguish between oral and written exams. The only soft constraint considers spreading out the exams for the individual students to the highest extent possible. This corresponds more or less to one of the KaHo soft constraints, in which there should be at least three time slots of study time between two consecutive exams for the same student group. The two literature heuristics L_1 and L_2 are ideal candidates for spreading out the exams as much as

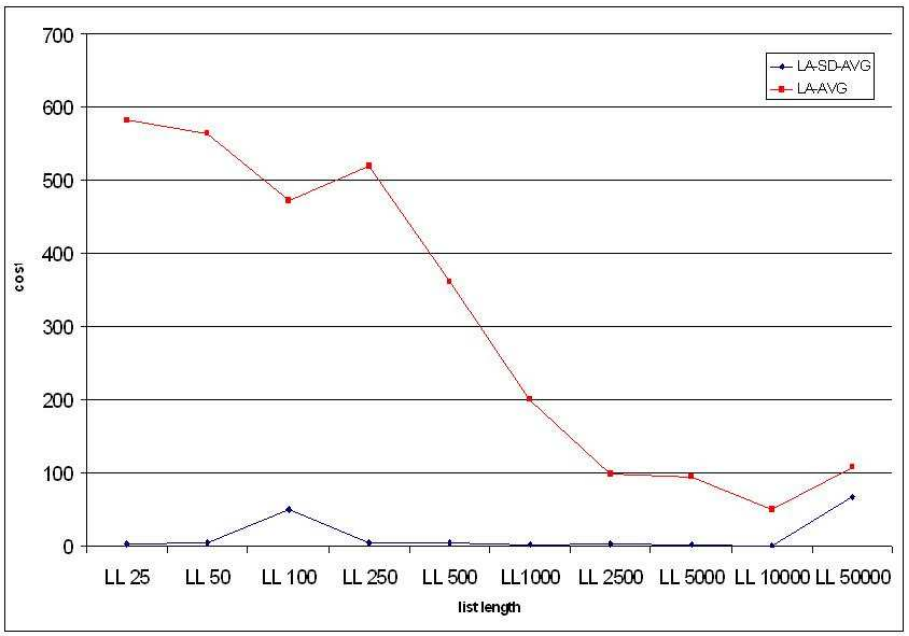

Figure 5.2: Average result after ten runs for different values of the list length (tournament factor 16).

possible. On the one hand they maintain feasibility while on the other hand they manage to spread out the exams at the same time. Although these two heuristics cannot make a feasible solution infeasible (as explained in Section 5.3.2), they are not able to turn an infeasible solution into a feasible one. Therefore, we need to construct a feasible solution in a preceding phase. After this the improvement phase is started (see Section 5.3.3). In the first phase, we apply a variant of the largest degree node colouring heuristic to obtain a feasible solution. In case the heuristic is not able to solve the problem to feasibility in reasonable time, we apply the KaHo heuristics afterwards.

In Tables 5.3 and 5.4, the results for the different acceptance criteria are presented for each of the 13 instances of the Toronto benchmark. Each algorithm finishes after one hour of calculation and is executed 10 times. For every algorithm, which is identified by the acceptance criterion and a tournament factor, we present the average, the minimum cost, and the standard deviation. The best cost for every instance is indicated in italics.

It is important to mention that the parameter setting of the hyper-heuristic approach is the same for all the data instances. For example, the same SA-4 algorithm is applied to every data instance.

10 runs and 1 hour of calculation time each					
SA with KaHo heuristics					
tournament factor	**4**	**8**	**16**	**32**	**64**
AVG	*1*	1.1	1.1	2.9	6.1
MIN	*0*	*0*	*0*	*0*	3
STDEV	1.15	1.6	*1.1*	1.79	2.85
SA with KaHo and literature heuristics					
tournament factor	**4**	**8**	**16**	**32**	**64**
AVG	21.9	15.5	4.4	4.1	*3.9*
MIN	14	11	1	1	*0*
STDEV	4.72	3.5	2.84	*2.33*	2.38
IE with KaHo heuristics					
tournament factor	**4**	**8**	**16**	**32**	**64**
AVG	*3.7*	4	5.1	4	5.4
MIN	*0*	*0*	1	1	2
STDEV	*1.89*	2	2.64	2.83	2.27
LA-SD with KaHo heuristics with list length 10000					
tournament factor	**4**	**8**	**16**	**32**	**64**
AVG	1.6	*0.3*	1.7	3.8	4.1
MIN	*0*	*0*	*0*	1	*0*
STDEV	1.37	*0.45*	1.44	2.11	2.46
LA with KaHo heuristics with list length 10000					
tournament factor	**4**	**8**	**16**	**32**	**64**
AVG	6.4	5.5	*4.7*	8.4	10.2
MIN	3	*1*	2	3	4
STDEV	3.31	3.63	2	2.88	3.82
GD with KaHo heuristics					
tournament factor	**4**	**8**	**16**	**32**	**64**
AVG	*1.2*	3.6	2.7	3.1	4.2
MIN	*0*	1	1	*0*	1
STDEV	1.32	1.58	*1.16*	3	2.35

Table 5.2: Experimental results of the five algorithms applied to the KaHo examination problem. Numbers in italic correspond to the best value obtained with the acceptance criterion.

The algorithms that performed best in a statistically significant manner are indicated in Table 5.5. In general, the best performing acceptance criterion is SA-4. Only for data instances `pur93` and `sta83`, other acceptance criteria perform better.

tournament	4														
acceptance	SA			LA			LA-SD			GD			IE		
	AVG	MIN	STD	AVG	MIN	STD	AVG	MIN	STD	AVG	MIN	STD	AVG	MIN	STD
car91	5.13	5.07	0.05	5.69	5.62	0.05	*5.12*	*5.04*	0.05	5.22	5.1	0.08	5.56	5.47	0.07
car92	*4.06*	*3.99*	0.05	4.39	4.35	0.03	4.17	4.04	0.06	4.16	4.09	0.05	4.6	4.45	0.11
ear83	*33.1*	*32.76*	0.2	33.61	32.99	0.41	35.05	34.39	0.64	33.55	33.06	0.32	38.25	35.44	1.31
hec92	*10.2*	10.09	0.13	10.49	10.27	0.16	10.79	10.45	0.2	10.28	*10.06*	0.14	11.91	11.46	0.34
kfu93	13.51	*13.27*	0.17	*13.47*	13.29	0.14	13.47	13.28	0.08	13.54	13.38	0.13	14.67	14.1	0.34
lse91	*10.49*	*10.21*	0.19	10.75	10.43	0.25	10.85	10.58	0.17	10.59	10.45	0.16	12.11	11.79	0.25
pur93	7.22	7.05	0.1	8.33	8.21	0.06	8.19	8.15	0.03	6.96	*6.79*	0.09	*6.9*	6.8	0.06
rye92	8.61	8.5	0.09	9.27	9.16	0.09	*8.56*	*8.43*	0.07	8.73	8.5	0.1	9.28	8.89	0.19
sta83	*157.05*	*157.03*	0.01	157.1	*157.03*	0.06	157.12	157.05	0.05	157.19	*157.03*	0.15	157.3	157.05	0.25
tre92	*7.95*	*7.82*	0.09	8.24	8.16	0.07	8.45	8.33	0.07	8.22	8.04	0.11	9.14	9.02	0.11
uta92	*3.46*	3.4	0.04	3.72	3.64	0.04	3.49	3.42	0.05	3.48	*3.38*	0.05	3.78	3.7	0.05
ute92	*24.89*	*24.82*	0.07	25.04	24.82	0.31	25.15	24.88	0.23	25.07	24.92	0.13	26.37	25.82	0.5
yor83	*35.21*	*34.7*	0.27	36.23	35.4	0.62	36.96	36.02	0.51	36.33	35.86	0.31	39.58	37.69	1.2

Table 5.3: Average and minimum cost and standard deviation for each of the Toronto instances after 10 runs. The acceptance list length for both late acceptance strategies is 500 for all experiments. The stopping criterion is one hour.

In Table 5.6, we present the results of experiments with longer execution times of the SA-4 hyper-heuristic. In all cases longer running times lead to better results.

In Table 5.7, we compare the validated hyper-heuristics results with the Toronto benchmark results in the literature. For five out of the 13 data instances, we obtain better or equal results.

From Table 5.7, we can conclude that for the smaller instances, longer running times lead to better or equal results compared to the results from the literature. However, for the `rye92` and `pur93` data instances, these results are considerably worse than the best results from literature.

Compared to Kendall et al. [100], who deploy a more elaborate heuristics selection method and use running times up to four hours, we obtain better results in general, even after one hour of calculation. Possibly, this is partly attributed to the use of other heuristics and different acceptance criteria.

In Table 5.8 both late acceptance strategies are compared with Student t-tests. For the larger data instances (`car91`, `car92`, `pur93`, `rye92`, and `uta92`), the adapted version of the late acceptance strategy method performs better than the original late acceptance strategy in a statistically significant way. On the other hand, the original late acceptance strategy performs significantly better for the following data instances: `ear83` `hec92`, `tre92`, `ute92`, and `yor83`. For the three remaining data instances `kfu93`, `lse91`, and `sta83`, there is no statistical significance in the difference between the performance of the two late acceptance strategies.

5.4.3 The ITC 2007 Examination Timetabling Benchmark Problems

As it was the case for the Toronto data sets, the ITC 2007 data sets are also solved in two phases. First, we try to find a feasible solution, and in a second phase, this feasible solution is further improved. In both stages (the feasibility and the improvement stage) we have experimented with hyper-heuristics with different acceptance criteria (see Tables 5.11, 5.12, and 5.13). For all eleven hyper-heuristics, the same heuristic selection method is applied, i.e. simple random. The tournament factor is 4 for all algorithms. An overview of the eleven hyper-heuristics is presented in Table 5.10. Some of the hyper-heuristics only differ in the amount of calculation time that is spent in finding a feasible solution. Hyper-heuristics **HH2**, **HH4**, **HH6**, **HH7**, **HH9** can spend at most 150 seconds in the first (feasibility) stage. If no feasible solution was found in this period, the algorithm automatically goes to the improvement phase, while at the same time trying to make the solution feasible. From the obtained results and the t-tests (see Table 5.14), there is no hyper-heuristic that significantly performs better than the others. It is however noticeable that **HH11** finds the most feasible solutions: 95

on a total of 120 runs (see Table 5.13). **HH2** finds 3 of the best solutions. For two of the data sets (instance 4 and instance 11) we do not find any feasible solution. Possibly, for obtaining feasible solutions for these two data instances, it requires refactoring the initial solution's construction. Instead of only taking the room occupancy and the duration of the exams into account, the event-clash constraint (exams that cannot be scheduled together) should be taken into account. Although we do not improve on the best solutions of the ITC 2007 literature, our solutions are competitive with the finalists' results (see Table 5.9).

5.5 Conclusion

We propose a hyper-heuristic approach for examination timetabling problems that successfully tackles a curriculum-based real-world problem as well as two post-enrolment benchmark instances from the literature.

For the real-world examination timetabling problem, the hyper-heuristic generates better solutions, in terms of the number of time slots, than those obtained by the manual planner. Moreover, due to the problem size and its complexity, it takes the manual planner a few weeks to generate a timetable. On the other hand, the hyper-heuristic succeeds to strongly improve on the manual schedule in not more than one hour on an average desktop computer.

Equal or better results for 5 out of 13 data instances from the Toronto benchmarks are obtained. Although the improvement of the benchmark instances was not our primary concern, it shows that algorithms that were initially developed for real-world problems, can result in competitive algorithms. The same hyper-heuristic with other low-level heuristics was also applied to the ITC2007 benchmark instances. Instead of using a computation time of 1 hour, like in the other cases, we followed the rules of the competition. According to the competition rules, the computation time was limited to only 5 minutes. Although we could not beat the best solutions of the competition, they are still competitive with the results of the other contestants. Of course, in a real-world situation, the computation time for an examination timetabling application to find a solution is usually more than 5 minutes. As we have shown in the Toronto case, longer calculation times will lead to better solutions.

Since all three examination timetabling problems are not completely equivalent, different low-level heuristics had to be applied. Note that these low-level heuristics are rather specific: spreading out exams with the uncapacitated heuristics or moving exams into the appropriate room with the capacitated heuristics. The pure literature heuristics could not beat the KaHo heuristics for the real-world problem, neither were the capacitated heuristics able to solve the Toronto benchmarks data instances.

tournament	8														
acceptance	SA			LA			LA-SD			GD			IE		
	AVG	MIN	STD	AVG	MIN	STD	AVG	MIN	STD	AVG	MIN	STD	AVG	MIN	STD
car91	*5.19*	5.13	0.05	5.59	5.52	0.04	5.27	5.19	0.05	5.28	*5.03*	0.1	5.6	5.45	0.09
car92	4.2	4.09	0.08	4.44	4.33	0.07	*4.16*	*4.06*	0.07	4.32	4.23	0.05	4.71	4.55	0.12
ear83	*33.12*	*32.75*	0.45	34.68	33.26	1.2	34.87	33.4	1.04	33.52	33.02	0.41	37.49	35.3	1.16
hec92	*10.2*	*10.03*	0.16	10.53	10.23	0.16	10.72	10.55	0.11	10.38	10.14	0.21	11.63	11.17	0.32
kfu93	13.84	13.42	0.36	13.7	13.33	0.17	13.68	13.43	0.19	*13.6*	*13.31*	0.22	14.41	13.86	0.38
lse91	10.87	10.63	0.25	10.89	10.32	0.31	10.9	10.57	0.19	10.65	10.44	0.18	12.2	11.61	0.3
pur93	7.2	7	0.09	8.12	8.07	0.04	8.02	7.96	0.04	7.04	6.94	0.06	*6.93*	*6.8*	0.08
rye92	8.62	8.47	0.09	9.11	8.9	0.13	*8.61*	8.5	0.08	8.72	*8.46*	0.13	9.33	9.06	0.2
sta83	*157.04*	**157.03**	0	157.1	157.05	0.06	157.16	157.05	0.11	157.14	**157.03**	0.1	157.26	**157.03**	0.2
tre92	*8.04*	*7.93*	0.08	8.24	8.13	0.1	8.47	8.25	0.14	8.26	8.08	0.14	9.09	8.81	0.24
uta92	3.51	3.46	0.04	3.7	3.64	0.05	*3.5*	*3.45*	0.05	3.5	3.46	0.03	3.84	3.73	0.07
ute92	25.07	24.87	0.24	*25.04*	*24.83*	0.16	25.23	25.04	0.18	25.18	25.02	0.15	26.72	25.95	0.53
yor83	*35.48*	*35.04*	0.52	35.62	35.31	0.36	36.7	35.97	0.52	36.28	35.38	0.53	39.44	38.38	0.71

Table 5.4: Average and minimum cost and standard deviation for each of the Toronto instances. The acceptance list length is for both late acceptance strategies 500 for all experiments. The stopping criterion is 1 hour.

	statistically significant best performing algorithm	**p-value**
car91	*LA-SD-4*, SA-4	0.46
car92	*SA-4*	
ear83	*SA-4*, SA-8	0.88
hec92	*SA-4*, SA-8, GD-4	0.92, 0.18
kfu93	*LA-SD-4*, LA-4, SA-4, GD-4, GD-8	0.95, 0.54, 0.16, 0.12
lse91	*SA-4*, GD-4, GD-8	0.24, 0.08
pur93	*IE-4*, IE-8, GD-4	0.36, 0.11
rye92	*LA-SD-4*, SA-4, LA-SD-8, SA-8	0.15, 0.14, 0.16
sta83	*SA-8*	
tre92	*SA-4*	
uta92	*SA-4*, GD-4, LA-SD-4	0.32, 0.23
ute92	*SA-4*, LA-4	0.16
yor83	*SA-4*, SA-8	0.17

Table 5.5: Statistically significant best performing algorithms and corresponding p-values for Table 5.3 and 5.4. Each i-th p-value corresponds to the p-value of the (i+1)-th algorithm in the second column compared to the algorithm in italics.

tournament	**4**			
acceptance	**SA**			
	stopping criterion	**average**	**minimum**	**standard deviation**
car91	4 h	4.75	4.68	0.05
car91	12 h	4.64	4.52	0.05
car92	4 h	3.94	3.84	0.05
car92	12 h	3.86	3.78	0.06
ear83	2 h	33.02	32.82	0.16
ear83	12 h	32.69	32.49	0.13
hec92	1 h	10.2	10.09	0.13
hec92	12 h	10.06	10.03	0.03
kfu93	2 h	13.45	13.06	0.31
kfu93	12 h	13.24	12.90	0.2
lse91	2 h	10.38	10.06	0.19
lse91	12 h	10.21	10.04	0.13
pur93	4 h	6.57	6.45	0.07
pur93	12 h	5.75	5.67	0.05
pur93	48 h	5.23	5.13	0.06
rye92	4 h	8.31	8.18	0.1
rye92	12 h	8.2	8.05	0.12
rye92	48 h	8.01	7.91	0.07
sta83	1 h	157.05	157.03	0.01
tre92	2 h	7.91	7.73	0.06
tre92	12 h	7.79	7.69	0.07
uta92	2 h	3.37	3.32	0.03
uta92	12 h	3.17	3.13	0.03
ute92	2 h	24.99	24.83	0.24
ute92	12 h	24.88	24.77	0.17
yor83	2 h	35.06	34.79	0.25
yor83	12 h	34.83	34.64	0.14

Table 5.6: Average and minimum cost and standard deviation for the SA-4 algorithm.

instance	HH	[51]	[47]	[40]	[116]	[41]	[28]	[101]	[147]	[6]	[33]	[81]	[26]
car91	4.52	7.10	*6.60*	4.65	5.10	5.00	4.80	5.37	**4.50**	5.20	4.60	5.20	4.58
car92	**3.78**	6.20	*6.00*	4.10	4.30	4.30	4.20	4.67	3.93	4.40	4.00	4.30	3.81
ear83	32.49	36.40	***29.30***	37.05	35.10	36.20	35.40	40.18	33.70	34.90	32.80	36.80	32.65
hec92	10.03	10.80	***9.20***	11.54	10.60	11.60	10.80	11.86	10.83	10.30	10.00	11.10	10.06
kfu93	12.90	14.00	*13.20*	13.90	13.50	15.00	13.70	15.84	13.82	13.5	13.00	14.50	**12.81**
lse91	10.04	10.50	***9.60***	10.82	10.50	11.00	10.40	-	10.35	10.20	10.00	11.30	9.86
rye92	7.91	7.30	***6.80***	-	8.40	-	8.90	-	8.53	8.70	-	9.80	7.93
sta83	**157.03**	161.50	*158.20*	168.73	157.30	161.90	159.10	157.38	158.35	159.20	159.90	157.30	**157.03**
tre92	**7.69**	9.60	*9.20*	8.35	8.40	8.40	8.30	8.39	7.92	8.40	7.90	8.60	7.72
uta92	**3.13**	3.50	*3.20*	3.20	3.50	3.40	3.40	-	3.14	3.60	3.20	3.50	3.16
ute92	24.77	25.80	***24.40***	25.83	25.10	27.40	25.70	27.60	25.39	26.00	24.80	26.40	27.79
yor83	**34.64**	41.70	*36.20*	37.28	37.40	40.80	36.70	-	36.35	36.20	37.28	39.40	34.78

Table 5.7: Selection of the best results from the literature, compared with the best values obtained with the hyper-heuristics (HH) approach. We adopt the notation of Qu et al. [127], in which values in italics indicate that there is some confusion about the version of the data sets used. The values indicated in bold, are the best among all values.

	4		8	
	LA-SD	**LA**	**LA-SD**	**LA**
car91	X		X	
car92	X		X	
ear83		X		
hec92		X		X
kfu93				
lse91				
pur93	X		X	
rye92	X		X	
sta83				
tre92		X		X
uta92	X		X	
ute92				X
yor83		X		

Table 5.8: An X indicates which method performs better in a statistically significant manner. The comparisons are conducted on the results of Table 5.3 and 5.4.

Instance	**[119]**	**[58]**	**[14]**	**[68]**	**[125]**	**[114]**	**HH**
1	***4370***	5905	8006	6670	12035	4633	5965
2	***400***	1008	3470	623	3074	405	498
3	10049	13862	18622	-	15917	***9064***	18455
4	18141	18674	22559	-	23582	***15663***	-
5	***2988***	4139	4714	3847	6860	3042	4820
6	26950	27640	29155	27815	32250	***25880***	27445
7	4213	6683	10473	5420	17666	***4037***	5905
8	7861	10521	14317	-	16184	***7461***	8949
9	***1047***	1159	1737	1288	2055	1071	1170
10	16682	-	15085	14778	17724	***14374***	15023
11	34129	43888	-	-	40535	***29180***	-
12	5535	-	***5264***	-	6310	5693	5409

Table 5.9: Comparison of our solutions with the best solutions from the literature. The best solutions are indicated in bold and italics.

Abbrevations	Description
HH1	feasible stage : IE, improvement stage: IE
HH2	feasible stage : SA (until 50% calculation time), improvement+feasible stage: IE
HH3	feasible stage : SA, improvement stage: IE
HH4	feasible stage : LA (until 50% calculation time), improvement+feasible stage: SA
HH5	feasible stage : LA, improvement stage: SA
HH6	feasible stage : LA-SA (until 50% calculation time), improvement+feasible stage: SA
HH7	feasible stage : LA-SD (until 50% calculation time), improvement+feasible stage: LA-SD
HH8	feasible stage : LA-SD, improvement stage: LA-SD
HH9	feasible stage : LA (until 50% calculation time), improvement+feasible stage: GD
HH10	feasible stage : LA, improvement stage: GD
HH11	feasible stage : IE, improvement stage: SA

Table 5.10: Description of the 11 acceptance criteria. The hyper-heuristics differ in the acceptance criteria used in the feasible and the improvement stage.

	SR-4-HH1				SR-4-HH2				SR-4-HH3				SR-4-HH4			
	AVG	MIN	STD	#F	AVG	MIN	STD	#F	AVG	MIN	STD	#F	AVG	MIN	STD	#F
1	6471.6	6262	171.57	10	6314.8	***5965***	166.03	10	6451.1	6321	112.64	10	6330.2	6060	162.69	10
2	550	510	39.21	8	596.3	510	65.69	10	569.67	505	44.43	10	612.5	515	99.65	10
3	26358.67	19104	6285.19	3	-	-	-	0	-	-	-	0	23580	23580	0	1
4	-	-	-	0	-	-	-	0	-	-	-	0	-	-	-	0
5	5446.14	5006	322.02	7	5165.63	***4820***	157.92	8	5360.6	5021	194.89	10	5323	4855	350.5	8
6	29388.33	27490	2628.45	6	28641.43	27880	475.62	7	28279.29	***27445***	512.38	7	28578.13	27605	698.92	8
7	6276.8	6033	170.82	10	6185.89	***5905***	179.15	9	6315	6079	291.4	10	6250	6065	168.38	10
8	9379.9	9199	155.51	10	9362.8	9089	173.78	10	9507.7	9220	231.61	10	9260.9	9038	184.16	10
9	1292.8	1211	65.15	10	1315.2	1213	57.01	10	1335.7	1217	72.45	10	1255.9	1184	63.63	10
10	15945.33	15587	305.64	6	15565.3	15204	244.04	10	15638	15425	157.9	9	16113.33	15561	464.63	9
11	-	-	-	0	-	-	-	0	-	-	-	0	-	-	-	0
12	5876.29	5527	257.81	7	5974.22	5679	267.64	9	5839.29	5580	147.29	7	5829.14	5483	176.21	7

Table 5.11: Average, minimum, standard deviation and the number of feasible solutions for the 12 data sets obtained by four hyper-heuristics.

	SR-4-HH5				SR-4-HH6				SR-4-HH7				SR-4-HH8			
	AVG	MIN	STD	#F	AVG	MIN	STD	#F	AVG	MIN	STD	#F	AVG	MIN	STD	#F
1	6490.1	6338	125.31	10	6466.6	6146	162.76	10	6286.1	6170	102.69	10	6591.3	6243	338.88	10
2	630.11	***498***	128.64	9	638.88	549	81.94	8	580.1	520	45.37	10	590.8	508	57.44	10
3	-	-	-	0	-	-	-	0	22966.5	20806	1866.77	4	24206.33	21449	2183.45	6
4	-	-	-	0	-	-	-	0	-	-	-	0	-	-	-	0
5	5397	4986	250.75	6	5485.56	5054	283.47	9	6168.2	5391	470.57	5	6839	6480	370.24	4
6	29146	28590	413.12	5	28922.86	28435	809.18	7	28468.89	27890	525.91	9	28567.5	27665	660.17	6
7	6296.8	6039	200.91	10	6350.9	6133	173.56	10	6458	5980	377.79	8	6665.6	6223	475.44	10
8	9408.5	9195	169.31	10	9439.1	9146	182.59	10	9422.22	9073	193.39	9	9396.3	9011	280.3	10
9	1291.7	1236	34.33	10	1312.5	1203	45.1	10	1384.4	1272	52.74	9	1320.2	1224	55.95	10
10	15843.4	15594	187.58	5	17072.38	15201	3792.23	8	15486.9	***15023***	325.5	10	15811.7	15360	304.88	10
11	-	-	-	0	-	-	-	0	-	-	-	0	-	-	-	0
12	5865.38	5525	236.87	8	5738.56	***5409***	219.77	9	5896.63	5449	266.28	8	5944.14	5630	184.59	7

Table 5.12: Average, minimum, standard deviation and the number of feasible solutions for the 12 data sets obtained by hyper-heuristics SR-4-HH5 to SR-4-HH8.

	SR-4-HH9				SR-4-HH10				SR-4-HH11			
	AVG	**MIN**	**STD**	**#F**	**AVG**	**MIN**	**STD**	**#F**	**AVG**	**MIN**	**STD**	**#F**
1	6667.6	6437	178.55	10	6809.6	6428	199.65	10	6404.2	6260	126.18	10
2	773.3	603	77.57	10	873.22	689	278.45	9	576.1	526	57.04	10
3	21437.33	20485	1207.16	3	-	-	-	0	20810.4	***18455***	1596.6	10
4	-	-	-	0	-	-	-	0	-	-	-	0
5	7790	7639	133.63	3	6439	6439	-	1	8354.86	5378	5369.49	7
6	28000	27530	556.84	3	28893.75	27955	1028.27	4	28801.5	27655	926.55	10
7	7028.7	6658	222.86	10	7004.3	6776	175.23	10	6390.9	5995	229.43	10
8	10220.11	9902	178.35	9	10211.8	9790	229.39	10	9257.4	***8949***	149.52	10
9	1244	1196	32.84	10	1281.1	***1170***	62	10	1301.6	1202	65.28	10
10	15614.78	15079	421.74	9	-	-	-	0	15648.9	15145	329.27	10
11	-	-	-	0	-	-	-	0	-	-	-	0
12	5733	5444	254.88	5	5534	5469	66.89	4	5875.13	5411	372.27	8

Table 5.13: Average, minimum, standard deviation and the number of feasible solutions for the 12 data sets obtained by hyper-heuristics SR-4-HH9 to SR-4-HH11.

instances	HH1	HH2	HH3	HH4	HH5	HH6	HH7	HH8	HH9	HH10	HH11
1	0.01	**0.63**	0	**0.48**	0	0.01	**1**	0.01	0	0	0.03
2	**1**	**0.1**	**0.35**	**0.12**	**0.11**	0.02	**0.16**	**0.11**	0	0.01	**0.29**
3	**0.18**	-	-	-	-	-	**0.09**	**0.08**	**0.32**	-	**1**
5	**0.05**	**1**	0.04	**0.27**	**0.06**	0.01	0	0	0	-	**0.12**
6	**0.41**	**0.1**	**0.46**	**0.23**	0.02	**0.11**	**0.22**	**0.24**	**1**	**0.24**	**0.19**
7	**0.27**	**1**	**0.27**	**0.43**	**0.22**	**0.06**	**0.07**	0.01	0	0	**0.05**
8	**0.09**	**0.16**	0.01	**0.96**	**0.05**	0.03	**0.05**	**0.18**	0	0	**1**
9	**0.05**	0	0	**0.61**	0.01	0	0	0	**1**	**0.11**	0.02
10	0.01	**0.55**	**0.22**	0	0.04	**0.2**	**1**	0.03	**0.47**	-	**0.28**
12	0.03	0.01	0	0.01	0.02	0.1	0.03	0	**0.18**	**1**	**0.11**

Table 5.14: The Student t-test p-values for the different hyper-heuristics. The algorithms are compared to the best performing algorithm (indicated with value 1). Algorithms that are not significantly better are indicated in bold.

Health Care

Chapter 6

Personnel Planning

6.1 Introduction

In this chapter we focus on real-world timetabling problems in health care. The considered problem is the nurse rostering problem of the internal medicine ward at Maria Middelares in Gent.

The work presented in this chapter is actually an extension of the nurse rostering model described in Burke et al. [30], in which the authors present a systematic representation of nurse scheduling constraints allowing for a fast evaluation in a local search procedure. The main advantage of the representation is the reduction of development effort, of errors, of code complexity, and of execution time. When working with employee rostering problems other than nurse rostering, a number of constraints appeared that either could not or were very difficult to express in the original formalism. The presented extension of this model does allow the expression and evaluation of these constraints.

The employee rostering terminology and model used throughout the chapter is based on Burke et al.'s [31] nurse rostering review. We tackle an employee rostering problem in which employees with different skill categories need to be assigned to shifts in order to meet constraints. The quality of a roster is measured in terms of the number of violated constraints. We consider 2 types of constraints:

- the coverage constraints express how many employees are required per skill category in every shift type during the planning period and
- the time related constraints are personal and legal constraints that take into account the preferences of the employees and the statutory rules. The

constraints that are not personal are grouped in a contract.

A working day can be divided in different overlapping or non-overlapping shift types. Common shift types are early, late, and night shift.

The employee rostering model in this chapter does not consider any hard constraints. Several models described in the literature treat either the coverage or the time related constraints (e.g. [20, 118]) as hard constraints. There exist examples, however, in which both the coverage and the time-related constraints determine the schedule quality (e.g. [97, 117]). When looking at real-world practice in employee rostering, we found that the latter models compare best to the actual situation.

Bard and Purnomo [16] model coverage as a constraint that should be satisfied all the time. Any coverage shortage will be solved by assigning outside personnel (from an outside ward with excess coverage or even from outside the hospital). Violations of the coverage are multiplied with a high number and added to the value of the cost function. Beddoe and Petrovic [18] present a case based reasoning approach to personnel rostering. There are no hard and soft constraints in the model. Both coverage and time constraints are implicitly captured in the case base.

We present a flexible approach to solve employee rostering, i.e. there are no cyclic shift patterns to assign. Other papers [9, 110], conversely, present a particular problem in which shift patterns are placed in employees' schedules in order to meet the requirements.

In Section 2 we describe the problem that lead to the new constraints. We shortly discuss the search algorithm, which is in this case rather straightforward, in Section 3. Section 4 presents the new model and the XML implementation. The results are discussed in Section 5. We conclude in Section 6 and point out directions for future research.

This chapter appeared in slightly altered form in *P. De Causmaecker, P. Demeester, G. Vanden Berghe, and B. Verbeke:. Evaluation of employee rosters with the extended linear numberings method. In 24th Annual Workshop of the UK Planning and Scheduling Special Interest Group (UK PlanSIG 2005), pages 125–133, London, December 2005.*

6.2 Problem Description

Both the coverage and the time related constraints can be violated and the cost function combines all these (soft) constraints. Different weights can be applied to the constraints in order to set their relative importance. Any under or over

coverage will incur a cost. The computations are straightforward and systematic by nature. Burke et al. [30] developed a systematic evaluation procedure - called linear numberings - for the remaining, less structured, constraints.

In real-world problems, we expect a large diversity of constraint types. In order to minimise development efforts and maintenance costs; and at the same time ascertain fast evaluation of constraints, we developed the evaluation functions along the lines of the general model and associated evaluation method described in [30]. The model is applicable to a large class of employee rostering constraints. The present chapter describes an extension to that evaluation method. It allows for the evaluation of several new constraints. These constraints are expressed in XML.

6.3 Search Algorithm

In the employee rostering literature [31, 82], methods from both operations research as artificial intelligence are used. We use the tabu search metaheuristic with a dynamic tabu list and multiple neighbourhoods that are evaluated according to the token-ring approach (see chapters 3 and 4). The search is guided by a cost function, in which our evaluation method is embedded. Since the cost function evaluation is at the heart of the heuristic, it is evaluated extensively and should be implemented as efficiently as possible.

The solution space is represented as a two dimensional matrix with the employees in the rows and the shifts in the columns. An entry in the matrix is the smallest scheduling object carrying two attributes:

- a `value` attribute indicating whether an employee is working (value is 1) or not (value is 0) in a particular shift,
- a `fixed` attribute indicating whether the value of the entry can be changed or not during the search.

To search the solution space we experimented with a small variety of moves defining different neighbourhoods. We implemented different moves, of which we discuss two:

- the flip move simply toggles the value of an entry between 0 and 1. This move allows a temporary violation of the coverage constraint.
- the swap in the same column move swaps the values of two entries in the same column (shift).

6.4 Linear Numberings Evaluation Method

6.4.1 Description of the Linear Numberings Method

A formal description of the linear numberings evaluation mechanism is presented in [30]. In the present chapter we will briefly review the method informally by means of an example. The local search algorithm moves through a state space of possible solutions (briefly referred to as solutions when no confusion is possible). After each step in this space, the degree of constraint satisfaction is evaluated. A solution is represented in the format of Table 6.1. In the example, a day consists of 3 non-overlapping shifts: E(arly), L(ate) and N(ight). In order to be perfectly clear, shifts can overlap in reality but they will always be represented in separate columns in the model.

	E	**L**	**N**
Day 1	*	-	-
Day 2	-	*	-
Day 3	*	-	-
Day 4	-	-	*
Day 5	-	-	*
Day 6	-	-	-
Day 7	-	-	-

Table 6.1: Graphical representation of the proposed solution (E: early shift, L: late shift and N: night shift)

An asterisk indicates that the employee is working in a specific shift. This is referred to as an 'event'. In the example, an employee works in the early shift of day 1, the late shift of day 2, the early shift of day 3, the night shift of day 4 and day 5, and is free on days 6 and 7. A constraint in the model is expressed as a 'numbering' (this is a template of numbers), a set of numbering constraints and their corresponding values. We will introduce numberings and numbering constraints by means of an example. Let us express the constraint 'can work at most 3 night shifts in a week of which at least two should be consecutive'. A numbering that can be used for this constraint is represented in Table 6.2:

Since all other shifts are marked U(ndefined), only night shifts are considered. By overlaying the numbering with Table 6.1, the asterisks matching a number represent an event for which the constraint is relevant. In the example these are the two events on the night shifts of days 4 and 5.

A numbering constraint can now be formulated as an inequality. The left hand side is the condition that is checked against the value on the right hand side. If the

	E	L	N
Day 1	U	U	0
Day 2	U	U	1
Day 3	U	U	2
Day 4	U	U	3
Day 5	U	U	4
Day 6	U	U	5
Day 7	U	U	6

Table 6.2: Numbering to express constraints on night shifts (U: undefined)

condition is not met, the constraint is violated and a cost will be generated. In the above example we encounter two numbering constraints: the maximum number of night events (value 3) and the minimum number of consecutive night events (value 2). The model allows for eight different numbering constraints, of which two are active in our example. The numbering constraints can be grouped into 4 types.

- The first type checks the total (maximum or minimum) number of events:
  - `max_total`
  - `min_total`
- The second set checks for every number in the numbering the (maximum or minimum, max_pert and min_pert) number of events that are mapped onto it. In the numbering of Table 6.3 we can express that an employee should work at most 3 early shifts, 3 late shifts and 2 night shifts per week as follows by

$$max_pert\{0\} = 3, max_pert\{1\} = 3, max_pert\{2\} = 2 \tag{6.1}$$

	E	L	N
Day 1	0	1	2
Day 2	0	1	2
Day 3	0	1	2
Day 4	0	1	2
Day 5	0	1	2
Day 6	0	1	2
Day 7	0	1	2

Table 6.3: Numbering example that is used to explain the second numbering constraint

Overlaying this numbering with Table 6.1 and checking the events (the asterisks) against the numbering constraints leads to (pert(_) is the corresponding value)

$$pert\{0\} = 2, pert\{1\} = 1, pert\{2\} = 2 \tag{6.2}$$

Since every value of pert is less or equal to the value of the corresponding numbering constraint there is no violation.

- The third group of numbering constraints checks the (maximum or minimum) number of consecutive events with the value of the numbering constraint. These numbering constraints are symbolised by:
  - `max_consecutive`
  - `min_consecutive`
- The fourth group checks the (maximum or minimum) gap between two non-consecutive events. These are represented by:
  - `max_between`
  - `min_between`

  This numbering constraint can e.g. be used to express that an employee should have zero or at least two consecutive days off between working days. Table 6.4, with min_between = 2, expresses exactly that.

	E	**L**	**N**
Day 1	0	0	0
Day 2	1	1	1
Day 3	2	2	2
Day 4	3	3	3
Day 5	4	4	4
Day 6	5	5	5
Day 7	6	6	6

Table 6.4: Numbering for expressing constraints on entire days

When overlaying this numbering with Table 6.1, we note that this constraint is not violated, since this employee does not work on day 6 and day 7.

The above described numberings, numbering constraints and their values allow for expressing and evaluating constraints on full weekends, a maximum number of shifts of a certain type, at most x consecutive days,âĂę etc. We refer to [30] for details and a formal description of the method.

6.5 Extended Linear Numberings

When used in combination with a metaheuristic algorithm, numberings provide a fast way to evaluate the high number of solutions explored (its execution time per solution is linear in the number of days times the number of different shifts per day [30]). The method is modular and allows for easy expansion with new constraints. When using this model in the personnel scheduling project mentioned in the introduction however, we encountered some constraints that were not straightforward to express through the method. This section lists these constraints and presents an extension to the model.

The evaluation model consists of an initialization, an intermediate evaluation and a final evaluation module [30]. The extended method applies the same modules but it requires an evaluation function that is slightly more complex. We will explain how the extended method works, by introducing three types of real-world constraint that can be evaluated in an elegant way:

1. forward rotation (i.e. an assigned shift should start at the same time or later than the shift on the day before),
2. minimum/maximum number of days off for each employee,
3. minimum/maximum number of hours worked.

The *initial evaluation* is the same as in the original method. The working period that precedes the current planning period determines the initial values of the counters. In the *intermediate evaluation* algorithm, the counters are set to these initial values (`last_nr`, `index_last_nr`, `consec`). Starting from these values, the intermediate evaluation algorithm (pseudo code in Algorithm 6) then visits the numberings one by one, updating counters whenever an event corresponds to a number. From the pseudo code, it can be seen that violations of `consecutive` and `between` constraints immediately lead to an increase of the penalty value. For constraints of the `total` and `pert` category, penalties will be assigned in the *final evaluation* (Algorithm 7).

1. For the forward rotation constraint, which penalises insufficient rest time between consecutive assignments, we have defined a numbering with non-increasing numbers (examples in Tables 6.6 and 6.7). The first numbering (Table 6.6) is constructed such that the corresponding number of every night shift is greater than the numbers of the two following shifts. The numbering in Table 6.7 is constructed analogously for the late shifts. Mapping the numbering for the late shifts (Table 6.7) onto Table 6.1 learns that there is a potential problem on the second and third day, since the number of the late shift of the second day is 8, which is greater than the number

corresponding to the early shift of the third day. Indeed this is a problem, since there is only one shift period of rest between the 2 events. There are no problems concerning the night shifts in the example. The constraint can be evaluated by extending the original algorithm with a conditional statement that compares the distance between the numbers of two consecutive events with the value of `min_between`. It supposes that there should at least be 2 non working, non overlapping shifts between two night shifts (or two late shifts), which is expressed by `min_between = 2`. Negative distances obviously correspond to invalid sequences and should therefore induce a penalty.

2. Events that we call fixed correspond to holidays. They are treated differently than free days (days without shifts assigned) through two new counters `total_day_off` en `consec_day_off`. The evaluation algorithm treats days off as 'dummy' events that can be treated in such a way that they relax constraints. The dummy events are treated as events in cases where a free day would violate the constraint. They are interpreted as a free day or shift in cases where an assignment would cause a violation of the constraint.

3. In order to calculate the assigned number of hours, we added a flag that indicates whether the evaluated constraint considers hours or other events (first conditional statement in Algorithm 7). If hours are to be counted, the constraint's numbering will not contain small integer values but the duration of the shift corresponding to the position in the numbering. Suppose that in the previous example an early shift takes 6 hours, a late shift 8 hours and a night shift 10 hours, then the numbering will be (Fig. 6.5):

	E	**L**	**N**
Day 1	6	8	10
Day 2	6	8	10
Day 3	6	8	10
Day 4	6	8	10
Day 5	6	8	10
Day 6	6	8	10
Day 7	6	8	10

Table 6.5: Example numbering to count the total number of working hours

In the final evaluation (Algorithm 7), the total is counted and compared with the value of the numbering constraint.

	E	L	N
Day 1	0	1	5
Day 2	3	4	8
Day 3	6	7	11
Day 4	9	10	14
Day 5	12	13	17
Day 6	15	16	20
Day 7	18	19	23

Table 6.6: Example numbering to express forward rotation for the night shifts

	E	L	N
Day 1	1	5	3
Day 2	4	8	6
Day 3	7	11	9
Day 4	10	14	12
Day 5	13	17	15
Day 6	16	20	18
Day 7	19	23	21

Table 6.7: Example numbering to express forward rotation for the late shifts

6.5.1 Extended Linear Numberings Representation

In order to facilitate interfacing with other programs, an XML format is provided to formulate the constraints. All constraints formulated in the above described numberings method can be expressed. This allows the constraints to be constructed fastly in a human readable format. An example of constraints expressed in the XML format is available at `http://ingenieur.kahosl.be/projecten/dingo/XMLdata/h`

6.6 Results

6.6.1 A Sample Application

In the project mentioned in the introduction, we collected real-world data on complex personnel rostering problems. One of the cases, implementing the largest number of constraints was in a hospital. We used the numbering method in an application to schedule a ward of nurses as an example application. There are 4 different shift types and the ward consists of 30 nurses of which:

- 11 nurses have a full time contract (38 hours a week),
- 7 nurses work part time (30 hours a week),
- 5 nurses work half time (20 hours a week),
- 2 nurses have a night contract of 35 night hours a week,
- 4 nurses work 20 night hours a week and
- 1 nurse works 30 night hours a week.

Each nurse works according to a contract specifying the constraints, e.g.

- nurses working according to a half time contract are not allowed to work night shifts
- nurses working according to a âĂŸ35 night hours a week' contract work in cycles of 7 consecutive nights and one week rest
- etc.

In this ward nurses work in 4 different shift types: early, late, day and night. The first 3 shift types take 8 hours (nurses working in a day type have 2 hours lunch break) and the night shift takes 10 hours.

Figure 6.8 displays the cyclic roster for nurses working according to a full time contract. The roster recurs every 18 weeks. The symbol F marks an extra day of rest since full time contract nurses work on average 40 hours a week while they are supposed to work only 38 hours a week. Therefore, every 4 weeks, they are granted an extra day off. Note also that full time nurses work only 3 night shifts during 18 weeks.

Each contract has a cyclic roster. A typical characteristic of all these cyclic rosters is that there is one free weekend every two weeks. The departmental roster combines all the cyclic rosters such that the coverage constraints of the ward are met. As a result, every employee has one free weekend out of two (by construction) but the coverage constraints are often violated (Fig. 6.1).

From this picture it is clear that not all the constraints are satisfied. Violations of the coverage constraint are indicated in couloured boxes in the lower part of Fig. 6.1. Per day and per shift type, excess and shortage is reported (different colours). From the cyclic rosters we derived constraints such as the maximum number of events on shifts of a certain type, the maximum and minimum number of consecutive events, the maximum and minimum of rest days, etc. We attempted to approximate the original problem without enforcing the cyclic behaviour of the

WEEK	Mon	Tue	Wed	Thu	Fri	Sat	Sun
1		N	N	N			
2	L	L	L		E	E	E
3	E		E	E	E		
4	L	L		F	E	E	D
5	E	E		E	E		
6	E	E		D	L	L	L
7		F	L	L	L		
8	E	E	L	L		E	E
9	L	L		E	E		
10	E	E	E		F	L	L
11	L		L	L	L		
12	L	L	L		E	D	E
13	E		E	E	E		
14	E	E	E	E	L	L	L
15		F	E	E	E		
16	L	L	L	L		E	E
17	E		L	L	L		
18	L	L			L	L	L

Table 6.8: Cyclic roster for full time contracts

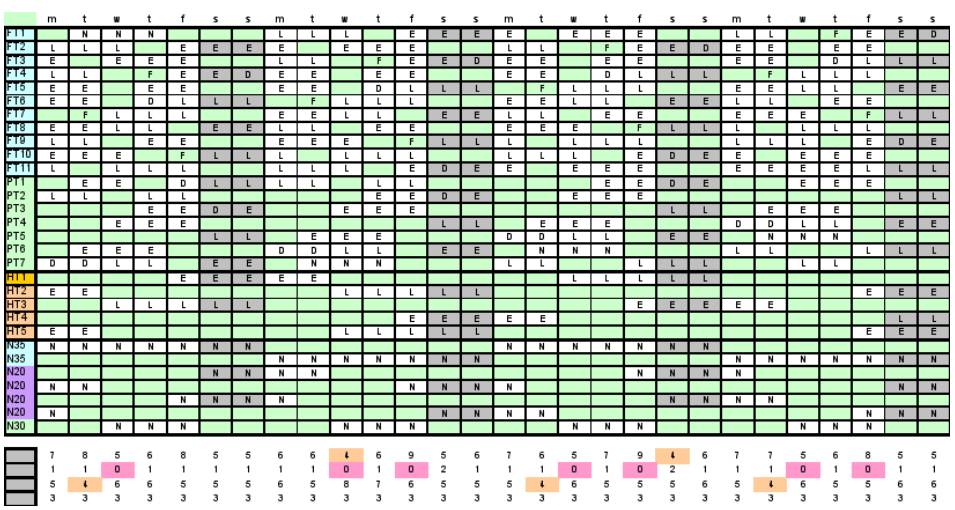

Figure 6.1: Manually produced roster over a period of 4 weeks.

different contracts. The constraints that we took into account are described in

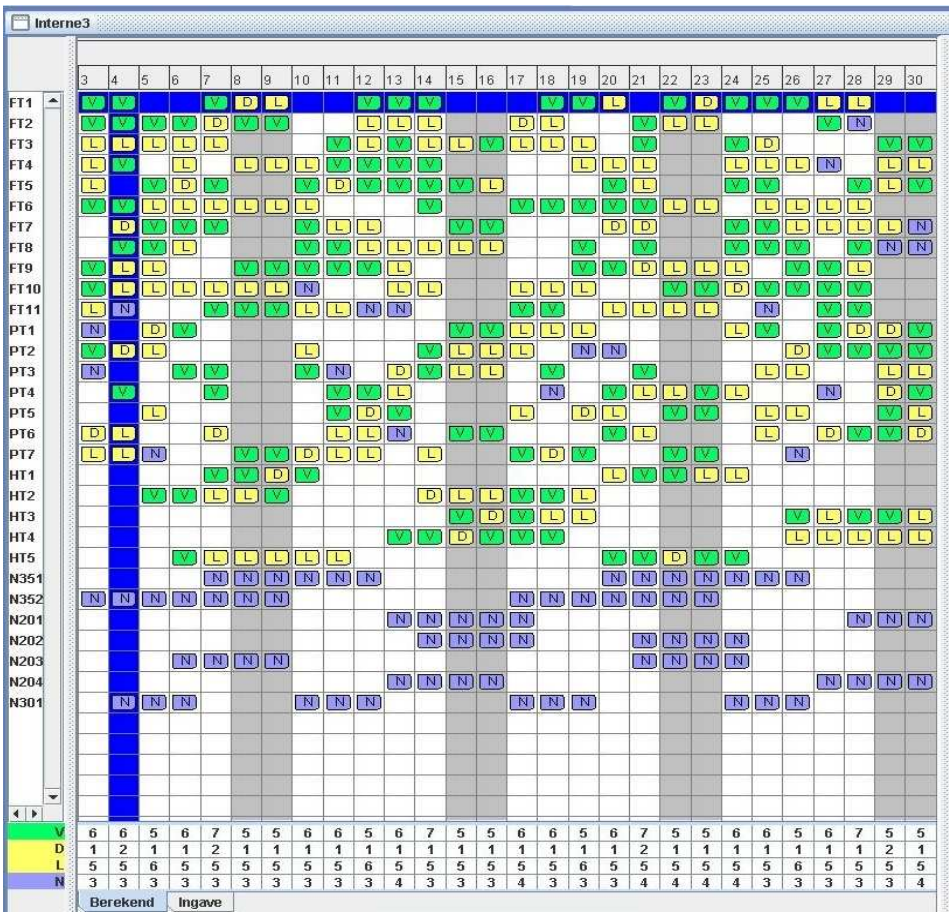

Figure 6.2: Automatically generated roster after 50000 iterations

`http://ingenieur.kahosl.be/vakgroep/it/plansig/appendix.htm`.

After 50000 iterations, the tabu search algorithm came up with the solution depicted in Fig 6.2. When comparing the results with the manually constructed roster, we can see that the automated approach has resolved the under coverage on the Day shifts. The value of the evaluation function for the manually generated roster is 861, while it is 436 for the automatically generated roster. This solution was found after 36651 iterations (1h07m45s of calculation time on a Pentium IV computer with 1 GB RAM). The first improvement (cost 856) on the manual solution was found after 5991 iterations (11 minutes and 36 seconds calculation time). An iteration in this case consists of evaluating 112 different moves (which

is the number of shifts per day multiplied by the number of days), of which the algorithm chooses the best, based on the evaluation function. Details on the relative importance of the different constraints (weights) can be found at `http://ingenieur.kahosl.be/vakgroep/it/plansig/appendix.htm`.

This experiment demonstrated the feasibility of tackling the rostering problem using tabu search with a multi neighbourhood approach and our numbering method as the central device in the evaluation function. In order to estimate the impact of the method on the speed of the algorithm, we tested it in isolation.

6.6.2 Efficiency

The aim of our isolated test was to compare the evaluation speed of the presented method with the speed of an equivalent ad hoc programmed function. Such functions were available from earlier developments. To perform this test we timed 10000 iterations of:

- the extended linear numberings method including the evaluation of the coverage constraints (which we call T3),
- of the ad hoc function including the evaluation of the coverage constraints (which is called T2) and
- the time needed to evaluate solely the coverage constraint (T1) which is not evaluated with the extended numberings method.

Using the following formula:

$$\frac{T_3 - T_1}{T_2 - T_1} \tag{6.3}$$

we found that the linear numberings outperform ad hoc programmed functions by reducing the execution time to about 58%. This adds to the ease of development and maintenance comfort experienced in the development process.

6.7 Conclusion

We presented an extension of the numberings method originally presented in Burke et al. [30]. The extension allows for a larger set of constraints to be expressed while preserving the simplicity and the efficiency of the original model. The extension allows expressing most of the constraints which could conceivably be evaluated in a one pass algorithm. The numberings method has become a powerful tool in

the development of personnel scheduling systems. It helps to easily and quickly incorporate new constraints as they come up. The fact that the evaluation is systematic and fast, removes the need to design a separate function for every new constraint. The linear numbering method could be an onset for a more general application, in which constraints are expressed in a separate language, using predefined keywords. In this way, the evaluation mechanism can be reused for different types of problems. The only thing that needs to be changed when tackling another problem are the representation of the constraints, as long of course these constraints can be expressed in terms of the predefined keywords.

Algorithm 6 Intermediate evaluation

```
for i = 0 to number of numberings do
    last_nr_i = last_nr_ini_i
    index_last_nr = 0
    consec_i = consec_ini_i
    nr = last = 0
    for each event e_j do
        {nr is the corresponding numbering for the event e_j}
        nr = N_i(e_j)
        if e_j is not fixed then
            if nr!=undefined then
                total_i = total_i + 1
                total_hours_i = total_hours_i + nr
                if last_nr_i!=undefined then
                    {are the events consecutive?}
                    if nr == last_nr_i + 1 then
                        consec_i = consec_i + 1
                    else
                        if nr > last_nr_i + 1 then
                            if consec_i + consec_day_off_i < min_consec_i then
                                penalty_min_consec_i = penalty_min_consec_i +
                                min_consec_i - consec_i - consec_day_off_i
                            end if
                            if consec_i > max_consec_i then
                                penalty_max_consec_i = penalty_max_consec_i + consec_i -
                                max_consec_i
                            end if
                            if nr - last_nr_i - 1 + consec_day_off_i < min_between_i then
                                penalty_min_between_i = penalty_min_between_i +
                                min_between_i - (nr - last_nr_i - 1)
                            end if
                            if nr - last_nr_i - 1 > max_between_i then
                                penalty_max_between_i = penalty_max_between_i + (nr -
                                last_nr_i - 1) - max_between_i
                            end if
                            consec_i = 1
                            consec_day_off_i = 0
                        else
                            {check forward rotation}
                            if nr - last_nr_i < 0 then
                                if nr - last_nr_i - 1 < min_between_i then
                                    penalty_min_between_i = penalty_min_between_i +
                                    min_between_i - (nr - last_nr_i - 1)
                                end if
                                consec_i = 1
                            end if
                        end if
                    end if
                end if
                pert_per_type_i,nr = pert_per_type_i,nr + 1
                last_nr_i = nr
            end if
            index_last_nr = index_e_j
            last = nr
        else
            {days off}
            if e_j is fixed then
                total_day_off_i = total_day_off_i + 1
                if nr == last + 1 then
                    consec_day_off_i = consec_day_off_i + 1
                else
                    if nr > last + 1 then
                        if consec_i + consec_day_off_i < min_consec_i then
                            penalty_max_consec_i = penalty_max_consec_i + min_consec_i -
                            consec_i - consec_day_off_i
                        end if
                        consec_day_off_i = 0
                    end if
                end if
                last = nr
                last_nri = nr
                index_last_nr = index_ej
            end if
        end if
    end for
    i = i + 1
end for
```

Algorithm 7 Final evaluation

for i = 0 to number of numberings **do**
  {count the events}
  if $count_events_i$ **then**
    if $total_i > max_total_i$ **then**
      $penalty_max_total_i = penalty_max_total_i + total_i - max_total_i$
    end if
    if $total_i + total_day_off_i < min_total_i$ **then**
      $penalty_min_total_i = penalty_min_total_i + min_total_i - total_i$
    end if
  else
    {count the hours}
    if $count_hours_i$ **then**
      if $total_hours_i > max_total_i$ **then**
        $penalty_max_total_i = penalty_max_total_i + total_hours_i - max_total_i$
      end if
      if $total_hours_i < min_total_i$ **then**
        $penalty_min_total_i = penalty_min_total_i + min_total - total_hours_i$
      end if
    end if
  end if
  if $consec_i > max_consec_i$ **then**
    $penalty_max_consec_i = penalty_max_consec_i + consec_i - max_consec_i$
  end if
  {take into account the days off}
  if $consec_i + consec_day_off_i < min_consec_i$ **then**
    $penalty_min_consec_i = penalty_min_consec_i + min_consec_i - consec_i - consec_day_off_i$
  end if
  {check for every number in the numbering the max. or min. number of events that can be mapped to it}
  ïĂć
  for t in N_i **do**
    if $pert_per_type_i[t] > max_pert_per_type_i[t]$ **then**
      $penalty_max_pert_i = penalty_max_pert_i + pert_per_type_i[t] - max_pert_per_type_i[t]$
    end if
    if $pert_per_type_i[t] < min_pert_per_type_i[t]$ **then**
      $penalty_min_pert_i = penalty_min_pert_i[t] + min_pert_per_type_i[t] - pert_per_type_i[t]$
    end if
  end for
  if $N_i(0) + N_i(T_n) - last_nr - i > max_between$ **then**
    $penalty_max_between_i = penalty_max_between - i + N_i(0) + N_i(T_n) - last_nr_i - max_between$
  end if
  $i = i + 1$
end for

Chapter 7

Patient Admission Scheduling

7.1 Introduction

A decision support approach to automatically assign patients to hospital beds is introduced. This application will assist admission schedulers to carry out the hospital management's decisions at the operational level. It enables scheduling patients more efficiently, meaning that they will be assigned to beds in the appropriate departments, satisfying as much as possible the patients' wishes and all the constraints on medical requirements. Meanwhile the algorithm will also try to balance the number of patients between the different departments.

Newly admitted patients need a free bed that satisfies both the personal preferences (single, twin room, or rooms with more beds) and the medical needs of the patient (availability of oxygen or telemetry in the room) located in the department that is specialized in treating the clinical picture. The assignment of patients to beds is often carried out by a central admission office that individually contacts every appropriate department a few days before the effective admission of the patient. Other hospitals organize the admission of patients without a central admission office, leaving admission responsibility with the departments. In the latter case, a lack of overview of the departments may result in under occupancy. Patients may be refused in one department while free suitable beds are available in another department.

Generally speaking patients can be divided in two groups: inpatients and outpatients. Inpatients spend several days or nights in a hospital, whereas the admission of outpatients is expressed in hours. We will concentrate in this chapter on inpatients only. Inpatients can further be divided in three groups: emergency, elective and admitted patients. Emergency patients are hard to schedule, since

by definition they have no appointment with the physician and arrive at random. Several papers are devoted to the arrival and assignment of emergency patients (see [133] for an overview). Most hospitals provide some slack beds that can be used for emergency patients. Elective inpatients are waiting for an admission date. This means that an admission office can determine when to admit them. Such patients allow the hospital to improve its occupancy rate as they can be assigned to the most appropriate period. In this chapter, however, we simplify the problem by assuming that the patients' admission dates are known beforehand. The physician who advised the patient to be admitted to the hospital, diagnosed the patient's disease which is associated with a default (average) length-of-stay. Since the aim of this study is to develop an automated solution method that calculates a patient admission schedule in less than one hour and since the number of patients and their medical needs do not fluctuate critically in one hour in a hospital, the number of patients can be assumed static in this time frame. If these figures change at the end of the day, the hospital administrator can easily execute the solution another time, and receive a schedule that satisfies the changed needs of the hospital.

The available literature on admission scheduling mainly concentrates on more abstract levels than what we consider in this chapter. Those papers focus on the decisions of the hospital management at the tactical and strategic level, in order to increase the hospital's efficiency. We will focus on the operational level, in which the decisions made by the hospital management are executed at the level of the assignment of patients to specific beds in given days. It is our experience, and this was pointed out by Kusters and Groot [104] as well, that a bed admission support application only stands a chance to get accepted if its decisions are supported by human experts and if interactions with the user are allowed. The admission scheduling algorithm described in this article suggests solutions to the manual admission officer that satisfy the decisions made by the hospital management. Since we focus solely on the operational level, we assume that all admission rules and issues have been resolved by the hospital management beforehand. It is aimed at automatizing the work of the admission scheduler, who allocates patients to the most appropriate beds/rooms/departments over their entire hospital stay.

In Section 7.2 we introduce some problem related terminology. The problem is described formally in Section 7.3. In Section 7.4 the mathematical model for the patient admission scheduling problem is formulated. Integer programming is applied to solve the model. However, it does not generate a feasible solution in reasonable time. To this end, we apply a metaheuristic as well as a hyper-heuristic approach to assign patients to beds, which is described in detail in Sections 7.5 and 7.6. In Section 7.8, we give future research directions and conclude.

This chapter is a summary of the two following papers:

- *P. Demeester, W. Souffriau, P. De Causmaecker, and G. Vanden Berghe. A hybrid tabu search algorithm for automatically assigning patients to beds. AI*

in Medicine, in Press. http://dx.doi.org/10.1016/j.artmed.2009.09.001.

- *B. Bilgin, P. Demeester, and G. Vanden Berghe. A hyperheuristic approach to the patient admission scheduling problem. submitted to Journal of Heuristics.*

7.2 Terminology

We first introduce some problem specific terminology that we will use throughout the rest of the chapter.

- **Admitted patients** are patients that are effectively admitted to the hospital and are assigned to a room and a bed.
- **Planned patients** have an admission date and an expected length-of-stay (LOS), which is determined by the government for every pathology. To reduce the expenses in the public health sector, the government demands the hospitals to treat every pathology within this pre-defined amount of time. Planned patients are not admitted to the hospital yet.
- **Transfer**: moving a patient from one room to another during his/her stay. We distinguish between planned and unplanned transfers. A planned transfer can, for example, concern a move to intensive care after surgery, and a move back to another department after recovery. An unplanned transfer could be caused by bed or room shortage. Unplanned transfers should be avoided as much as possible.
- **Room**: a room is characterized by a number of physical properties, e.g. the availability of oxygen or telemetry. Not every room is appropriate for every patient: different equipment is required for babies and elderly people, for example. A room may contain one or more beds. Hospitals (in Belgium) normally do not assign patients of different gender to the same room at the same time.
- **Specialism**: a hospital department is in general highly specialized in treating one kind of pathology (cardiovascular diseases, oncology, dermatology, etc.). We call this the major specialism of the department. Often the same department is also carrying out other treatments as a minor specialism. Some of the rooms are equipped to accommodate and some of the nurses are trained to treat patients with diseases that correspond to the department's minor specialism. Nurses belonging to a department are trained to care for patients with the specific pathology. This is similar to what the authors of [87] call nursing units. Rooms that belong to the department can be

especially equipped for the major specialism or one of the minor specialisms of the department.

- **Night**: we will consider a night as the smallest unit of time. Each length-of-stay is expressed in nights. During the night it is highly uncommon to admit planned patients.

7.3 Problem Description and Generation of Test Data

In this section we describe the admission scheduling problem formally and how the test data is generated.

7.3.1 Parameters and Variables

The following notation is used throughout the chapter.

- nights are denoted n, with $n = 1, ..., T$, with T the number of nights in the planning period of the time horizon;
- departments are denoted as d, with $d = 1, ..., D$, with D the total number of departments in the hospital. Departments can support one or more specialisms s, with $s = 1, \ldots, S$, with S the total number of specialisms in the hospital. A department d can enforce that assigned patients have a specific age;
- a room is denoted r, with $r = 1, ..., R$, with R the number of rooms in the hospital. A room r can have one or more room properties rp ($rp = 1, .., V$, with V the total number of room properties in the hospital) and a gender. According to the specialisms that are supported in the department, rooms can support in some degree different specialisms;
- a specialism s can enforce that rooms satisfy specific room properties;
- the bth bed of room r is denoted b_r, with $b_r = 1, ..., B$, with B the number of beds in room r. The capacity (number of occupied beds) of the room r at night n is denoted b_{rn};
- patients are denoted p, with $p = 1, ..., P$, with P the total number of patients. There are F female patients and M male patients, with $P = F + M$. Patients have the following properties:

- an admission date p_{AD}, with p_{AD} in 1, ..., $T-1$, and a discharge date p_{DD}, with p_{DD} in 2, ..., T and $p_{AD} < p_{DD}$. This means that we consider only patients that stay at least one night in the hospital,
- an age p_A and a gender p_G,
- a treatment, which corresponds to a specialism s, and
- a room preference $rpref$ ($rpref = 1, .., W$, with W the total number of room types (single, double, ...)).

7.3.2 Constraints

Hard Constraints

- **HC1**: During the considered planning period the room needs to be available. A blocked room (due to maintenance, refurbishment, etc.) cannot be used.
- **HC2**: The admission and discharge date and consequently the length-of-stay of a planned patient cannot be changed by the admission office. This can only be adapted by the responsible doctor.
- **HC3**: For each admission of a patient the length-of-stay is contiguous.
- **HC4**: Two patients p_i and p_j ($i \neq j$) cannot be assigned to the same bed b_r in the same time slot n.

Note that hard constraint **HC2** is in contradiction with Gemmel and Van Dierdonck's [84] definition of admission scheduling (see Section 2.2.3). We opt not to modify the admission dates of patients.

Soft Constraints

- **SC1**: The patient's room choice (single, twin, or a room with more beds) has to be respected if possible. A patient who asked for a single room, should preferably be assigned - in case of lack of single rooms - to a twin room. If it is not possible to assign a patient to his/her room of choice, the hospital may miss out on revenues. Physicians in some countries (such as Belgium) are allowed to charge a higher honorarium to patients who are nursed in a single room, if and only if these patients requested a single room during the admission process.
- **SC2**: A patient is preferably nursed in a department that has the right equipment and staff to treat the patient's disease.

- **SC3**: A patient is preferably assigned to a room that in some degree corresponds to the specialism that is needed to treat the patient's clinical picture.
- **SC4a**: The medical treatment of a patient may require that he/she is assigned to a room with special equipment. These room properties are mandatory for the treatment.
- **SC4b**: The medical treatment of a patient may require that he/she is assigned to a room with special equipment. These room properties are recommended to treat the patient.
- **SC5**: The number of unplanned transfers should be minimized.
- **SC6**: Male/female patients should be assigned to appropriate rooms as described in Section 7.2.
- **SC7**: Patients should be assigned to departments that are suited for their age. Elderly patients, for example, should not be assigned to a department that is specialized in paediatrics.
- **SC8**: Some patients have to be assigned to a single room for medical reasons (quarantine).

In some hospitals soft constraint **SC1** is treated as a hard constraint. Attributing a higher relative importance to constraint **SC1** than, for example, to constraint **SC5** may lead to a higher number of unplanned transfers. Suppose that an inpatient asked for a single room, which was not available at the time of admission. When a single room becomes available in a department with the right specialism, the admission officer may choose between moving the patient to that free single room or leaving the patient where she/he is. Although the former decision will lead to an unplanned transfer, it will probably please the patient and the physician in attendance more. It is up to the hospital management to balance the relative importance of each of the constraints.

7.3.3 Constraint Weights

Not all the constraints of Section 7.3.2 are equally important. The weights we attribute to the constraints determine their mutual relative importance (Table 7.1).

In this example we preferred to avoid unplanned transfers above fulfilling the room preferences of the patients. This explains the high value of **SC5**, compared to the low value of **SC1**.

Constraint	Corresponding weight
SC1	0.8
SC2	1.0
SC3	1.0
SC4a	5.0
SC4b	2.0
SC5	11.0
SC6	5.0
SC7	10.0
SC8	10.0

Table 7.1: Weights of the constraints.

7.3.4 Generation of the Data Set

We opted to generate data for the experiments, since obtaining real-world data was, due to privacy issues, hard. Although the data is generated, it is based on a realistic hospital situation, and on interviews with experienced 'human' patient admission schedulers.

We consider as a first problem a hospital that consists of 6 departments and a planning horizon of two weeks, each having one major specialism and two minor specialisms. The departments are configured in such a way that every specialism is assigned to three different departments, once as a major and twice as a minor. The number of rooms in every department ranges from 20 to 30. They are divided as follows:

- at most five single rooms,
- between five and ten twin rooms,
- and the rest of the rooms in the department contain four beds.

Each room can have 0, 1 or 2 room properties. Per specialism a random number (with a maximum of five) of subspecialisms is generated. With each subspecialism a length-of-stay is associated that is generated randomly based on a normal distribution with mean 5 and variance 3. Each planned patient is randomly associated with a subspecialism (and hence with a corresponding length-of-stay), and his/her room preference is generated. The specialism that is awarded to the patient also implies the requested/required room properties. For every day of the considered planning period 60 new patients are generated. The data set can be found at `http://allserv.kahosl.be/~peter/pas/`.

Of course, when this application will be used in daily practice, all the necessary patient data, including lenght-of-stay, assigned specialism, etc. will be provided by the hospital administration.

7.4 Integer Programming Approach

We introduce a mathematical model for the patient admission scheduling problem. This will be the starting point for solving the problem with integer programming. We start by introducing some extra concepts, that ease the mathematical expression of the problem. After that, we present the decision variables, the objective function and the constraints. Note that the mathematical model only takes into account the most common case in which the gender of the first patient determines the gender of the room.

7.4.1 Extra Constants

- P_{pr} is the penalty incurred when assigning patient p to room r. The penalty value P_{pr} will be the result of violations of the soft constraints **SC1**, **SC2**, **SC3**, **SC4a**, **SC4b**, **SC7**, and **SC8**.

7.4.2 Decision Variables

We consider the following decision variables:

- x_{prn}
  - is 1 if patient p is assigned to room r on night n,
  - 0 otherwise;
- t_{prn}
  - is 1 if patient p is transferred from room r to another room at night n,
  - 0 otherwise;
- y_{rn}
  - is 1 if room r at night n can accommodate only female patients,
  - 0 in case of only males.

7.4.3 Objective Function

The objective is to minimize the weighted sum of the total penalty incurred for assigning patients to unpreferred rooms and the number of transfers:

$$Min \sum_{prn} P_{pr}.x_{prn} + w.t_{prn}, \tag{7.1}$$

with w the weight for the violations of the transfers.

7.4.4 Constraints

Each patient p has to be assigned to a room during his/her stay:

$$\sum_{r} x_{prn} = 1, \forall n = p_{AD}, .., p_{DD}, \forall p = 1, .., P \tag{7.2}$$

At any time, a room should not contain more patients than the available number of beds in the room. Moreover, all patients in a room should have the same gender:

$$\sum_{p=1}^{F} x_{prn} \leq b_{rn} y_{rn}, \forall r = 1, .., R, \forall n = 1, .., T \tag{7.3}$$

$$\sum_{p=F+1}^{P} x_{prn} \leq b_{rn}(1 - y_{rn}), \forall r = 1, .., R, \forall n = 1, .., T \tag{7.4}$$

Equation 7.3 limits the number of female patients per room and per time slot, while Equation 7.4 limits the number of male patients per room and time slot. If a room accommodates only women, then the right hand side of Equation 7.4 becomes 0; and vice versa.

Decision variable t_{prn} is 1 if patient p is transferred from room i to room j at time n, 0 otherwise:

$$t_{prn} \geq x_{pr,n} - x_{pr,n+1}, \forall p = 1, .., P, \forall r = 1, .., R, \forall n = p_{AD}, .., p_{DD} - 1 \tag{7.5}$$

7.4.5 Results

The mathematical model was implemented in ILOG CPLEX. The program was run on a dedictated computer with an Intel quad core processor and 4 GB RAM. However, the integer program did not find a feasible solution within one hour of calculation. The first feasible solution was found after 13350 seconds of calculation. After one week of calculation the optimal solution was not obtained. Since the bed assignment application needs to respond promptly when new patients are admitted, the computation of a solution should take very little time.

7.5 A Tabu Search Algorithm for Assigning Patients to Beds

A good candidate search method that has - on condition a feasible solution exists - the property to always provide a solution (good or bad), even after a few moments of calculation, is local search. The solution is the result of iteratively changing small parts of the current solution into a new (or candidate) solution by applying neighbourhood moves. The number of moves that are evaluated in an iteration is called the *tournament* factor. From this set the move leading to the best solution is selected and executed, creating a new solution. The essential ingredients in any local search method are the solution representation, the neighbourhoods, and the cost function. These three ingredients are described in the next sections.

As in the KaHo and K.U. Leuven university course timetabling problems (see chapters 3 and 4) we also apply a tabu search algorithm with an intensification and diversification strategy, hybridised with a token-ring search.

7.5.1 The Representation of a Solution

We represent a solution as a set of two-dimensional matrices. Each row of a matrix represents a bed in a department. The columns represent the consecutive nights. The number of nights equals the period over which the patient to bed assignment is carried out. The number of individual matrices in a solution equals the number of different departments in the hospital. Figure 7.1 represents a rather small hospital that consists of three departments, each possessing several beds.

A patient in this model is represented as an ordered set of smaller objects. Each object corresponds to a night stay. We call them 'patient stay parts'. In Figure 7.2 patient P_1 has a LOS of four nights and is transferred to another bed at the second night. The solution presented in Figure 7.2 violates soft constraint **SC5**.

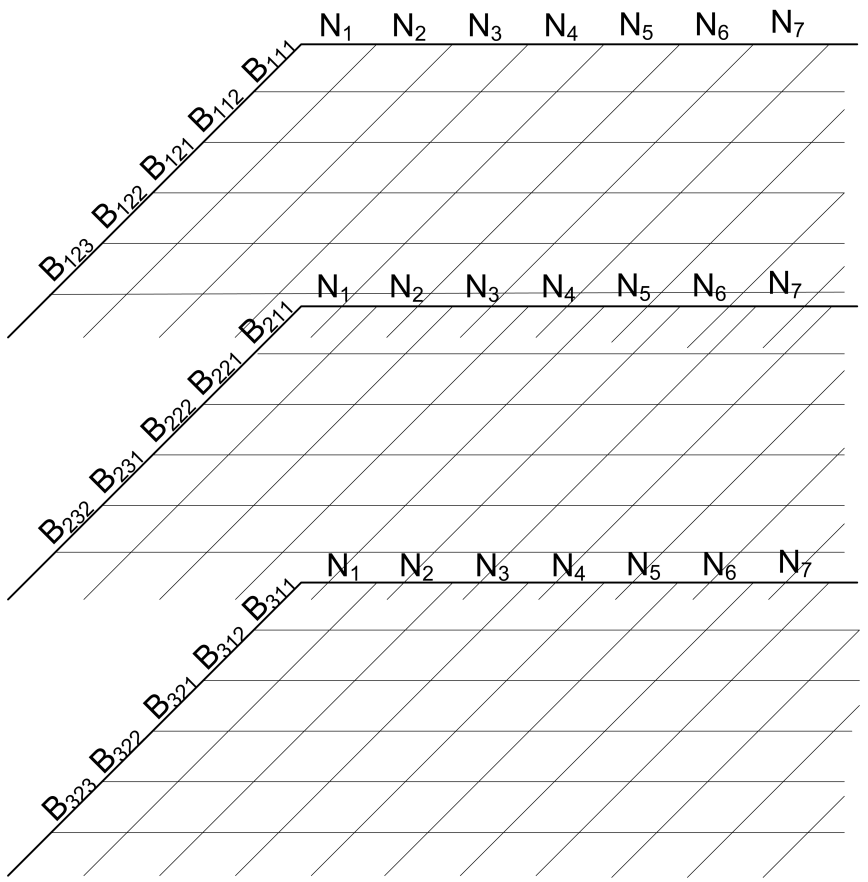

Figure 7.1: Graphical representation of a solution. Every department is represented by a matrix. The columns denote the nights N_j and the rows denote the beds B_{ktb} that are available in room R_{kt} in department D_k.

By choosing this representation the hard constraint **HC4** is always fulfilled.

7.5.2 Neighbourhoods

A neighbourhood defines the solutions that can be obtained from the current one by moving one or more patient stay parts to other positions in the matrix representation. Although we only have two basic moves, by adding restrictions on the direction, several neighbourhoods can be produced.

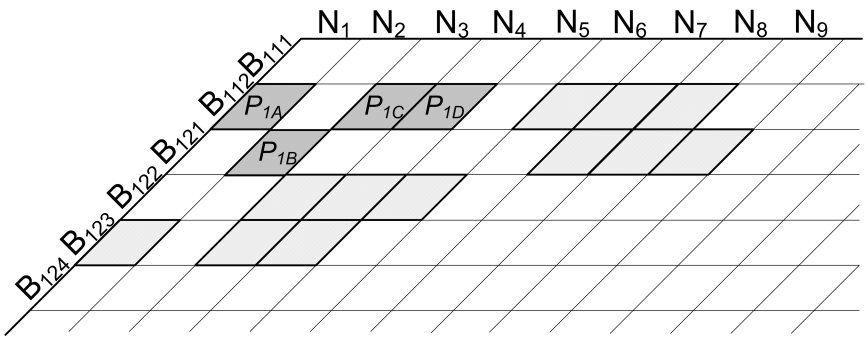

Figure 7.2: Graphical representation of patient P_1's stay. This patient has a LOS of four nights and he/she is transferred to another room during the second night of stay.

- The first basic move swaps the content of two matrix elements within the same department. If both matrix elements are non-empty, it means that the corresponding patient stay parts are swapped. If however one of the matrix elements is empty and the other is not, then the non-empty patient stay part is moved to an empty bed. This move will be the basis for the first neighbourhood (see Section 7.5.2).
- In the second basic move, all the patient stay parts of one patient are moved to an empty place either in another department or in the same department. This move is the basis for all the other neighbourhoods (see Sections 7.5.2, 7.5.2, 7.5.2).

Swap-Beds Neighbourhood

In the first neighbourhood we limit the moves to swapping patient stay parts in the same department. In order to avoid violations of the hard constraints **HC2** and **HC3**, only moves in the same column are allowed (see Figure 7.3). This means that patient stay parts can only be moved in the space dimension (to another bed) and not in the time dimension. If a bed is already taken by another patient stay part, the two patient stay parts are swapped.

In this neighbourhood we randomly select a row in which the algorithm searches the best matrix element to move to another row.

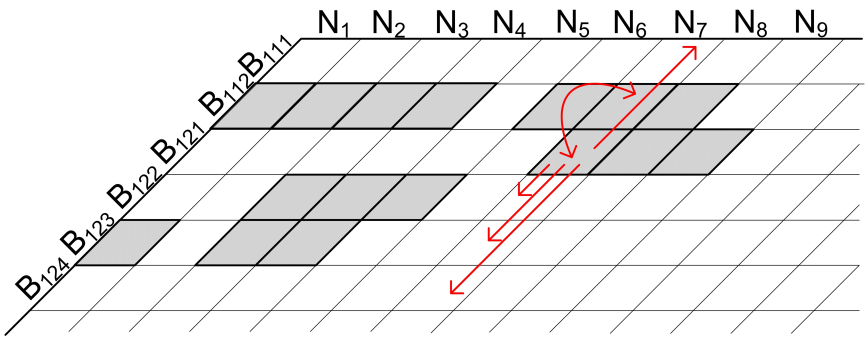

Figure 7.3: Graphical representation of the Swap-Beds Neighbourhood. The algorithm tries to move the patient stay part at night N_7 in bed B_{121} to other beds in the same column.

Move-Patient-To-Another-Department Neighbourhood

Since different departments with the same specialism exist, patients do not necessarily need to be nursed in the department supporting the specialism as a major. Due to over occupancy in one department, patients can be moved to another department with the same specialism. As the first neighbourhood (see Section 7.5.2) only allows swaps of beds within the same department, we constructed a second neighbourhood in which patients can be moved to another department with the same specialism in case of over occupancy.

This neighbourhood incorporates some load balancing of patients across departments. At the start of every iteration, the algorithm selects the department with the highest number of patient stay parts. From this department a patient is randomly chosen and moved to another department, which has a major or minor specialism that corresponds to the patient's medical requirements. The tabu search algorithm selects the best move from the proposed destination rows for each involved patient stay part. Similarly to the first neighbourhood, the patient stay parts are moved in such a way that constraints **HC2** and **HC3** are never violated.

Move-Patient-To-Same-Department Neighbourhood

The Move-Patient-To-Same-Department Neighbourhood is quite analogous to the previous neighbourhood, except that all the patient stay parts of one patient are moved to empty beds in the same department.

Move-Best-Patient-To-Another-Department Neighbourhood

This neighbourhood is similar to the second neighbourhood, except that the algorithm does not select a patient at random. Instead, it selects the best candidate patient to move to another department from a list of randomly chosen patients assigned to the same department.

7.5.3 Initial Solution

To start, an initial solution that only takes the hard constraints **HC1**, **HC2**, **HC3**, and **HC4** into account is generated. To assure that constraints **HC2** and **HC3** are satisfied, the patient stay parts of every patient are assigned in such a way that they are contiguous (**HC3**) and begin and end at the appropriate date (**HC2**). Constraint **HC1** is satisfied since we do not assign patient stay parts to blocked beds. The assignment of patients to departments in the initial solution is carried out in a greedy manner:

- The first patient from the patient list is selected.
- According to the patient's medical requirements, a list of departments that support the specialism is generated. The list is ordered as follows:
  - first the departments supporting the specialism as a major;
  - second the departments supporting the specialism as a minor and
  - last the departments not supporting the specialism.
- When for a specific night, no free room is left in the major specialism department, the initial algorithm searches an empty time slot in one of the departments that support the specialism as a minor.
- If also these departments are fully occupied for that particular night, the algorithm searches an empty time slot in another department.
- The above process is repeated until all the patients are assigned or until all the beds for a particular night are occupied and no more patients from the patient list can be admitted.

7.5.4 Cost Function

The cost function evaluates a solution and is determined by the weighted sum of the violations of the constraints. Four constraints do not need to be evaluated: the hard constraint **HC4** is always satisfied, due to the representation of a solution.

We do not assign patient stay parts to blocked beds (**HC1**). The neighbourhoods are constructed in such a way that hard constraints **HC2** and **HC3** cannot be violated. The constraints that we take into consideration in the cost function are **SC1**, **SC2**, **SC3**, **SC4a**, **SC4b**, **SC5**, **SC6**, **SC7**, and **SC8**.

Due to the fact that none of the hard constraints can be violated in this approach, the cost function does not correspond to the objective function of the integer programming formulation (see Section 7.4.3).

7.5.5 Experiments

We compare the tabu search algorithm hybridised with a token-ring approach with two implementations of variable neighbourhood descent [35]. From the tests we find that the hybridised tabu search produces better results than with a variable neighbourhood descent.

Variable Neighbourhood Descent Variants

We carry out experiments in which we hybridize the tabu search algorithm with

- a token-ring (abbreviated as TR) approach or
- a variable neighbourhood descent (abbreviated as VND).

The difference between the two methods lies in the point of time when the tabu search algorithm switches to another neighbourhood. In the token-ring search another neighbourhood is chosen each time a number of non-improving moves are exceeded, while the variable neighbourhood descent algorithm switches to another neighbourhood after one non-improving move. In contrast to the token-ring search, the variable neighbourhood descent switches to the first neighbourhood independently of the current neighbourhood, whenever a better solution than the current best solution is found. Intuitively, the first neighbourhood can be considered as a kind of repair neighbourhood that tries to fix some of the moves of the other neighbourhoods.

Since we have to take into account the randomness of the tabu search algorithm every experiment is carried out ten times. Randomness is intrinsic to the tabu search algorithm. Small parts of the solution, in this case individual patients are randomly selected in the neighbourhoods and moved. Randomness is furthermore inherently present in the algorithm itself. The quality of a solution may be equal for two possible moves in the same iteration. In this case the algorithm randomly selects one of the equal moves. The result of the statistical processing is denoted

in box plots, which graphically show the first and third quartiles, the median and the minimum and maximum values of each of the ten runs.

In the first experiment, we investigate two different approaches of the variable neighbourhood descent method. We make a distinction between returning to the first - repair - neighbourhood with or without any information about the department in which the best solution was found. In Table 7.2a we present four different settings of the algorithm (Test 1 to Test 4), which are each conducted ten times. The stop criterion of the tabu search algorithm was set to 50000 iterations. The corresponding box plots of the four algorithms are presented in Fig. 7.4.

- In Test 1 we add extra memory to the tabu search algorithm by registering the department in which a better solution than the current best solution was found. The registered department is then passed on to the first neighbourhood, and the search continues in this department starting from the best solution. We apply the four neighbourhoods discussed in Section 7.5.2.
- In Test 2, the algorithm jumps back to the first neighbourhood when finding a better solution than the current best in one of the other neighbourhoods, but we do not register the department in which this solution was found. We apply the four neighbourhoods described earlier.
- In Test 3, we repeat Test 1, except that we only take the first three neighbourhoods into account (Sections 7.5.2, 7.5.2 and 7.5.2).
- The second test is repeated in Test 4, except that we do not use the Move-Best-Patient-To-Another-Department neighbourhood.

Since the last neighbourhood is, compared to the other neighbourhoods, rather large to explore, we investigated whether it is favourable to exclude this neighbourhood or not. The tests in Table 7.2a demonstrate that the execution times for the third and last test are shorter than for the first two tests. However, there is a statistically significant difference between the results of the first two tests and the last two, in the sense that we obtain a better solution when applying the last neighbourhood.

We conducted experiments with two variable neighbourhood descent variants and we found that the differences between both approaches were not statistically significant.

Variable Neighbourhood Descent versus Token-Ring

In the next experiments we check whether there is a statistically significant difference between the variable neighbourhood descent and the token-ring

Test 1: VND			Test 2: VND			Test 3: VND			Test 4: VND		
cost	**iter.**	**time**	**cost**	**iter.**	**time**	**cost**	**iter.**	**time**	**cost**	**iter.**	**time**
2355,2	49377	3234	2600,2	49810	3183	2664,6	49957	1477	2764,4	**46687**	**1305**
2423,8	49131	3169	2381,8	48893	3223	2818,2	49246	1447	2799,2	48128	1343
2484,8	**48720**	3226	**2308**	49193	3175	2628,8	48343	**1419**	2655	49430	1385
2423,2	49900	3328	2389,8	49106	3119	2601,2	49238	1458	2650,8	48292	1366
2332,8	49720	3267	2472,2	49073	3176	2532	49955	1466	2511,2	49118	1385
2384,2	49730	3175	2616,6	49486	3174	2782,8	**48090**	1432	**2500,4**	49956	1391
2632,2	49860	3171	2500,2	**48553**	**3083**	2618,8	49622	1462	2740,6	48574	1361
2490	48840	**3078**	2425,2	49927	3151	**2426,2**	49531	1464	2583,4	49986	1389
2514,2	49832	3139	2457,2	49806	3124	2603,8	49679	1462	2673,2	47844	1325
2486,2	49216	3210	2346,8	49905	3318	2730,4	49405	1461	2620,6	49047	1364

Table 7.2a: The cost of the best solution; the number of iterations needed to find the best solution; and the elapsed time (in seconds) to obtain the best solution are presented for four tests (Test 1 to Test4). Each test is executed ten times. The best values for each test are presented in bold.

Test 5: VND			Test 6: VND			Test 7: TR(100-200)			Test 8: TR(300-1000)		
cost	**iter.**	**time**	**cost**	**iter.**	**time**	**cost**	**iter.**	**time**	**cost**	**iter.**	**time**
2232,6	96746	5908	2064,8	**95211**	5937	1792	99361	4844	1672,2	79923	3841
2329,2	98463	6031	**1917,6**	99911	6011	1832,4	**90586**	**4249**	2013	**36412**	**1865**
2311	97200	**5778**	2340,8	97656	6038	1805,8	99151	4759	1729,6	86997	3604
2131	99089	5913	2156,6	96389	**5694**	2035	91207	4370	1495,4	96159	4004
2376,8	97513	6102	2199,4	99824	6090	1661,8	91266	4393	1680,4	77239	3821
2054,4	**96700**	6202	2028,8	98540	6076	1678,8	99873	4937	1680,4	87349	3867
2127,4	99206	6403	2252,6	99578	6017	1737,2	98414	4782	1707,8	92963	4755
2209,6	96917	5856	2200	99680	6069	1901,2	92760	4576	**1465,6**	93594	4515
2060,6	99403	6179	2105,8	97748	5849	**1616,6**	98657	4997	1600,2	53695	2335
2118,2	98160	6169	2127	96645	5917	1861,6	97860	4612	1700,6	99917	4413

Table 7.2b: The cost of the best solution; the number of iterations needed to find the best solution; and the elapsed time (in seconds) to obtain the best solution are presented for four tests (Test 5 to Test 8). Each test is executed ten times. The best values for each test are presented in bold.

Test 9: TR+VND			Test 10: TR(300-1500)			Test 11: TR(120-180)			Test 12: TR(200-800)		
cost	**iter.**	**time**	**cost**	**iter.**	**time**	**cost**	**iter.**	**time**	**cost**	**iter.**	**time**
1906,2	97680	4460	1472,8	91463	5108	1701	96726	4796	**1607,6**	91197	4409
1806,4	98624	4425	1687	89927	4742	**1687,6**	91316	4512	1813,4	97210	3699
2030,2	99028	4351	1633,4	93201	4376	1740,2	98833	4758	1760,2	92586	4278
2137,6	99989	4410	**1446,4**	97193	4973	1787,2	96963	4806	1793,6	72221	3536
1983	96021	4326	1460,2	86285	4193	2039,8	99189	4744	1728,2	97709	3526
1911	99585	4510	1578,2	96073	4766	1883,6	**90767**	**4354**	2105,8	94513	4418
1837,6	96596	4454	1514,4	96794	4696	1978,2	99709	4717	1694,4	89477	4224
2044,2	**95179**	4276	1558,6	96073	4413	1821,2	99478	4857	1738,6	98581	4890
1838,8	99531	4381	1686,6	**57068**	**2470**	2037,2	92228	4394	1718,2	99888	4642
1836,2	96271	**4245**	1519,4	77205	4341	1745,4	97022	4780	1679,2	**69467**	**3293**

Table 7.2c: The cost of the best solution; the number of iterations needed to find the best solution; and the elapsed time (in seconds) to obtain the best solution are presented for four tests (Test 9 to Test 12). Each test is executed ten times. The best values for each test are presented in bold.

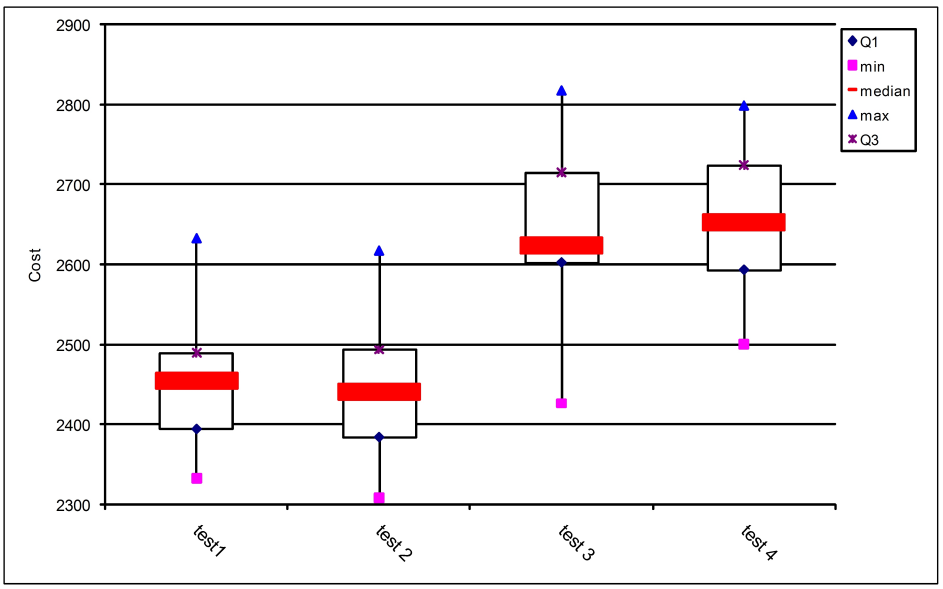

Figure 7.4: Box plot of the result of Test 1 up to Test 4. The maximum number of iterations is 50000.

approach. The stop criterion in all of the considered experiments is 100000 iterations and all four neighbourhoods are applied.

In Test 5 and 6 (see Table 7.2b), we repeat the experiments of Test 1 and Test 2, with a stop criterion that equals 100000 iterations. It is obvious from Tests 1, 2, 5 and 6 that the tabu search algorithm produces better results if it can search for a longer period of time.

In contrast to the variable neighbourhood descent approach, the token-ring search has two parameters to be set: the maximum length of the dynamic varying tabu list and the number of non-improving moves before the algorithm switches to another neighbourhood. We use primes as values for the tabu list length. Before the search starts, a list of primes smaller than the parameter value is generated. If in one iteration a better solution is found, the previous prime in the ordered list becomes the new tabu list length. The lower limit of the tabu list length is 7. If, however, an iteration generates no better solution, the next prime in the prime list becomes the new tabu list length. In the next tests we experiment with different parameter settings of the token-ring search.

- In Test 7 (see Table 7.2b) the maximum tabu list length is 100 (actually this means the largest prime smaller than 100), while the number of non-

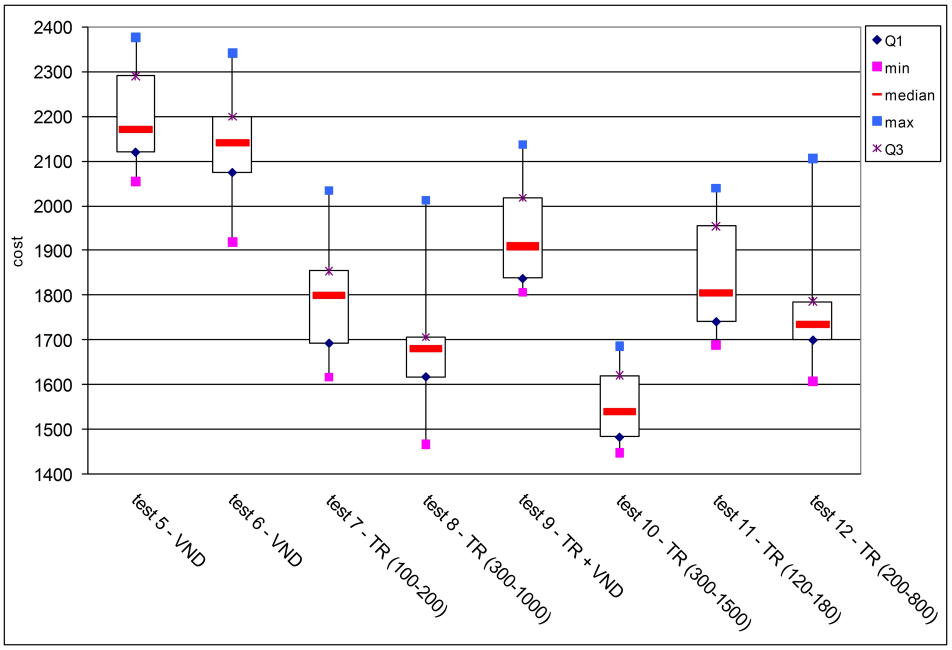

Figure 7.5: Box plot of the results of Test 5 up to Test 12. Maximum number of iterations is 100000.

improving moves is set to 200. Applying a t-test with a 95% confidence level to Test 5 and Test 7 reveals that the difference between the two tests is statistically significant, which means in this case that the token ring search of Test 7 finds better results than the variable neighbourhood descent of Test 5.

- The parameter settings in Test 8 for the maximum tabu list length are 300, while the algorithm switches neighbourhoods after 1000 non-improving moves. According to the applied t-test, Test 8 does not produce significantly better solutions than Test 7.
- In Test 9 (Table 7.2c) we combine both the token-ring search and the variable neighbourhood descent approach. The maximum tabu list length is 120 and the allowed number of non-improving moves is 10. When switching back to the first neighbourhood the algorithm remembers the department that was explored last.
- The next three tests (Test 10, Test 11, Test 12) are analogous to Test 8. The parameters for the maximum tabu list length are set to 300, 120 and 200

while the maximum number of non-improving moves are set to 1500, 180 and 800 respectively.

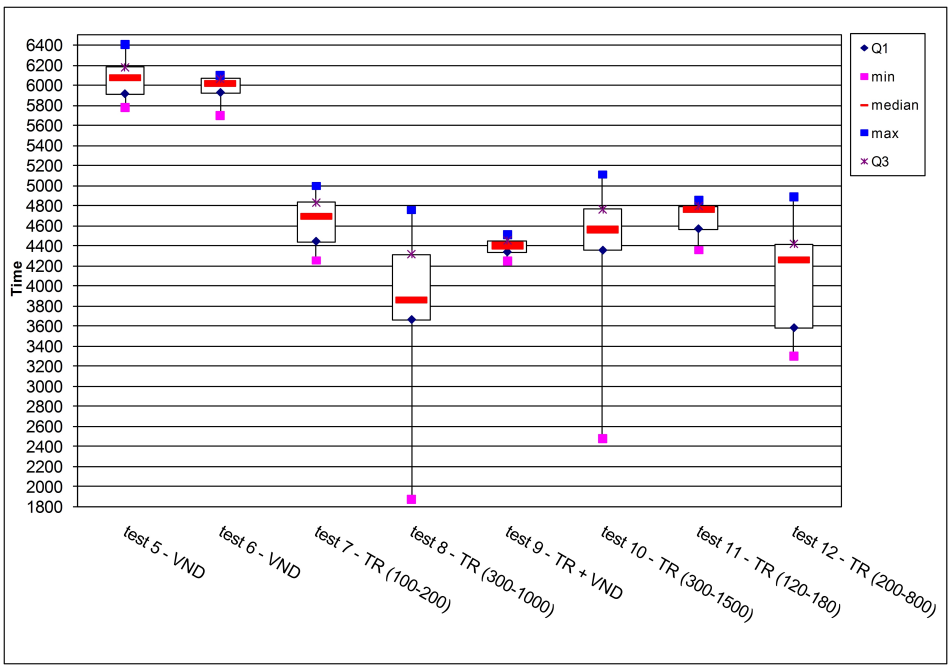

Figure 7.6: Elapsed time box plots of Test 5 up to Test 12

For Test 5 up to Test 12, we also statistically processed the elapsed times and present the results in box plots in Fig. 7.6. T-tests indicate that all token-ring search algorithms produce significantly better solutions in a smaller amount of time than the variable neighbourhood descent.

General Remarks

We end this section with a general note. The execution times seem rather long, since we start from a situation in which no patients are admitted or assigned to beds. In reality, several beds will be occupied and are - as long as these admitted patients are assigned to the right beds in the right departments - preferably not available for the planned patients. This means that only a percentage of the beds will be available for the planned patients, which reduces the search space considerably.

The data set considers the ideal case in which all patients leave the hospital at the foreseen dates. Of course, the discharge dates of the patients during their stay may change. The idea is to fix the current assignment of all patients that are in the hospital and to manually adapt the discharge dates.

7.6 A Hyper-heuristic Approach to Solve the Patient Admission Scheduling Problem

Besides the tabu search approach of Section 7.5, we also tackle the patient admission scheduling problem with the same hyper-heuristics approach that was introduced in Section 5.3.1. We reuse the solution space representation of Section 7.5.1.

7.6.1 Heuristics

The heuristics only consider moves that do not change the admission and the discharge dates of the patients, and respect the continuity of the patient stay.

The heuristics utilized by the hyper-heuristics are inspired by the mechanism of tournament selection in genetic algorithms. At each iteration the selected heuristic creates a number of moves and returns the move among them that results in the best cost value. The number of considered moves is a parameter of the search algorithm that we call the *tournament size*.

The type of moves considered by each heuristic is described below.

1. Swap the patient assignments of two beds in the same department. The department and the beds are selected randomly.
2. Swap the patient assignments of two beds in different departments. The departments and the beds are selected randomly.
3. Transfer all the stay parts of a patient to (an) empty bed(s) in another department. The source and destination departments, the patient, and the destination bed(s) are selected randomly.
4. Transfer all the stay parts of a patient to (an) empty bed(s) in the same department. The department, the patient, and the destination bed(s) are selected randomly.

The source and destination beds in the first and the second heuristic can either be empty or assigned to a patient.

7.6.2 Hyper-heuristics

Acceptance criteria

Four different hyper-heuristics, distinguished by their acceptance criteria are used. The acceptance criteria are *improving and equal*, *only improving*, *simulated annealing*, and *great deluge*.

Only the moves that lead to solutions that are better than the current solution are accepted by the *only improving* acceptance criterion. The *improving and equal* acceptance criterion works similarly, except that it accepts also the moves that result in solutions that are as good as the current solution.

The simulated annealing algorithm accepts all improving and equal moves. The moves that result in solutions that are not at least as good as the current solution are carried out with a probability, which is decreased throughout the execution. Further detail of this acceptance criterion at each iteration is presented in Algorithm 8.

Algorithm 8 Pseudocode of the *simulated annealing* acceptance criterion

C_i = Cost value at i^{th} iteration
$\delta = C_{i+1} - C_i$
T = Total Execution Time
R = Remaining Execution Time
P_i = Random Variable between [0,1[at i^{th} iteration
if $\delta \leq 0$ **then**
    Accept
else
    if $P_i < \exp(-\delta * T/R)$ **then**
        Accept
    else
        Reject
    end if
end if

The great deluge acceptance criterion maintains a deluge level throughout the execution. The cost value of the initial candidate is assigned as the initial value to the deluge level. Throughout the search this level is reduced at each iteration. The criterion accepts all improving and equal moves and the moves that result in a cost value below the deluge level. This acceptance criterion at each iteration is explained in further detail in Algorithm 9.

Algorithm 9 Pseudocode of the *great deluge* acceptance criterion

```
C_i = Cost value at i^th iteration
T = Total Execution Time
R = Remaining Execution Time
C_0 = Initial Cost
D = C_0 * R / T
if C_{i+1} ≤ C_i then
    Accept
else
    if C_{i+1} < D then
        Accept
    else
        Reject
    end if
end if
```

Heuristic selection method

The heuristic selection methods are *simple random* and *choice function*. Simple random randomly selects a heuristic from a list of heuristics at each iteration. On the other hand, a choice function [61] also considers, next to the performance of each heuristic, the performance of each pair of heuristics when called consecutively, and the elapsed time since the last call of each heuristic when selecting a low level heuristic.

7.7 Experiments

7.7.1 Settings

The experiments are carried out on six data instances. Each data instance has a planning horizon of two weeks. The properties of the benchmark instances are given in Table 7.3. The occupancy rate varies over the planning horizon. The rate is higher during the week than in the weekend. The occupancy rate is also higher in the first week than in the second week. That resembles the real world situation in which patients' medical treatments are not always scheduled two weeks in advance. In Figure 7.7, the occupancy rate of benchmark instance 3 is depicted over the planning horizon as an example. In Table 7.4 the distribution of the room types are depicted. There are, in terms of percentage, more quadruple rooms than double rooms, and more double rooms than single rooms. The benchmark

Property / Benchmark Instances	1	2	3	4	5	6
Number of Departments	4	6	5	6	4	4
Number of Rooms	98	151	131	155	102	104
Number of Beds	286	465	395	471	325	313
Maximum Occupancy	0.77	0.79	0.82	0.75	0.73	0.91
Average Occupancy	0.60	0.60	0.57	0.54	0.49	0.64

Table 7.3: The characteristics of the benchmark instances

instances and the weights of the objective function can be found on the Patient Admission Scheduling website [71].

	% of single rooms	% of double rooms	% of 4 person rooms
testdata1	16%	29%	54%
testdata2	12%	26%	60%
testdata3	13%	30%	57%
testdata4	12%	30%	58%
testdata5	8%	27%	64%
testdata6	10%	34%	56%

Table 7.4: Distribution of single, double and 4 person rooms

The four different acceptance criteria: *simulated annealing*, *great deluge*, *improving and equal*, and *only improving* are used within the hyperheuristic method for the experiments, next to the two heuristic selection mechanisms: simple random and choice function. We experiment with three ad hoc tournament sizes: 4, 16, 64. As a result, 24 hyperheuristic variations are tested on the benchmark instances.

Five parameter settings are tested for the tabu search method. The parameter setting consists of two parameters: *maximum tabu list length* and *the number of non-improving iterations* before the algorithm applies the next neighbourhood.

In total, 29 different solution methods are tested for each of the benchmark instances. Each solution method is run 10 times for each benchmark instance. The termination criterion is the execution time, which is set to 3000 seconds for each run. In order to have an objective means of comparison with other researchers we opt to apply the same benchmark method as in the Second International Timetabling Competition[1]. They provide a benchmark program that calculates the maximum allowed execution time based on the specifications of each individual PC . We multiply the resulting execution time by 10.

[1] `http://www.cs.qub.ac.uk/itc2007/index_files/benchmarking.htm`

The application is implemented in Java, and tested on Pentium D 3.4 GHz computers, with 2.0 GB of RAM, running Windows XP.

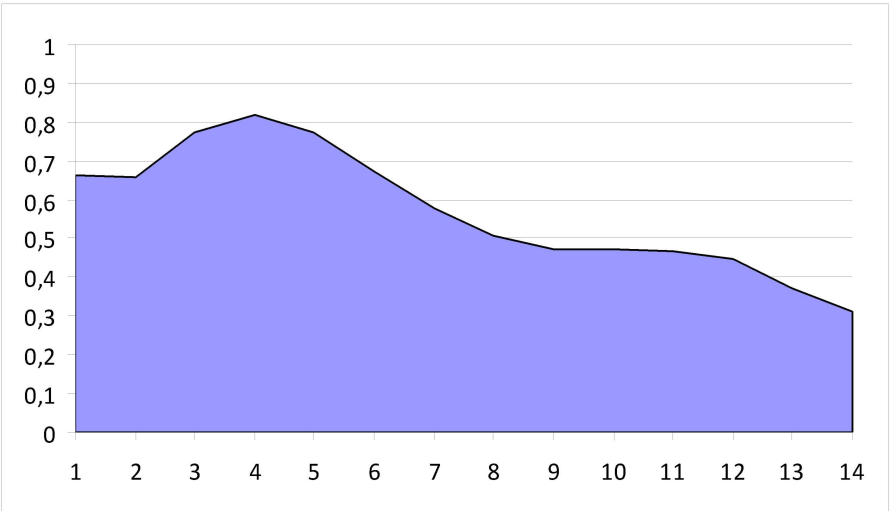

Figure 7.7: Example of the occupancy rate over the planning horizon (benchmark instance 3).

7.7.2 Results

The average objective function values over 10 runs for each benchmark instance are presented in Tables 7.5 - 7.10. The results of the experiments have been statistically processed for significant variances using the wilcoxon test. The best performing solution methods for each benchmark instance are indicated in bold. Since the objective function is the linear combination of the number of constraint violations, the goal is to find solutions which minimize the number of violations.

The hyper-heuristic method with the simulated annealing acceptance criterion and the simple random heuristic selection mechanism performs significantly better than the remaining solution methods. However, for benchmark instance 4, there is no statistically significant difference between the simple random simulated annealing and great deluge acceptance criteria for both the tournament factors 4 and 16. For this hyper-heuristic variant, tournament size 4 and 16 perform significantly better than 64 on five out of six benchmark instances. Only for benchmark instance 2, tournament size 4 outperforms 16 and 64.

The cost for the best solution obtained for benchmark instance 1 is 785.2. Analyzing the solution in more detail reveals that all hard constraints are satisfied, and that the following soft constraints are violated:

- there is one room-time slot combination where the gender constraint of a room is violated (soft constraint 6);
- there is one unplanned transfer (soft constraint 5);
- four of the patients are assigned to rooms that do not correspond to their preferred room properties (soft constraint 4b); and
- the highest contribution to the cost however, is caused by patients who are not assigned to the room type of their choice (soft constraint 1), which in fact is normal, since the demand of single and double rooms exceeds the supply.

Benchmark Instances	**Simple Random**					
	SA-4		**SA-16**		**SA-64**	
	Average	**St. Dev.**	**Average**	**St. Dev.**	**Average**	**St. Dev.**
1	**830.36**	**18.83**	**850.72**	**38.72**	896.42	41.97
2	**1382.28**	**14.21**	1432.64	51.1	1458.28	28.25
3	**923.5**	**15.3**	**923.16**	**20.74**	970.98	17.14
4	**1624.04**	**38.18**	**1608.68**	**29.24**	1710.98	74.31
5	**661.6**	**4.45**	**661.52**	**4.02**	680.08	4.55
6	**955.04**	**20.1**	**975.84**	**26.28**	992.12	26.94
	Choice Function					
	SA-4		**SA-16**		**SA-64**	
	Average	**St. Dev.**	**Average**	**St. Dev.**	**Average**	**St. Dev.**
1	851.94	20.28	871.42	41.82	935.5	47.1
2	1615.58	24.6	1599.06	39.06	1659.78	39.67
3	990.24	19.16	986.7	33.53	1027.08	40.46
4	1733.2	35.27	1748.9	25.56	1794.48	62.24
5	676.96	5.37	673.12	6.14	685.44	6.06
6	992.22	22.45	1040.08	32.14	1030.24	21.29

Table 7.5: *Simulated Annealing Hyperheuristic.* Results: Average objective function values over 10 runs and corresponding standard deviations

7.8 Conclusion

In this chapter, we have presented a hospital admission scheduling problem, which to our knowledge has not been reported on in the literature. The patient admission

Benchmark Instances	Simple Random					
	GD-4		GD-16		GD-64	
	Average	St. Dev.	Average	St. Dev.	Average	St. Dev.
1	876.98	20.59	865.3	17.26	891.42	35.31
2	1475.82	18.07	1483.96	30.11	1531.26	31.87
3	986.18	32.92	992.28	29.57	1015.16	15.26
4	**1631.26**	**34.17**	**1616.66**	**52.98**	1673.96	22.14
5	688.64	6.32	688	3.31	701.54	9.96
6	1034.9	30.92	1007.62	13.61	1040.64	23.05
	Choice Function					
	GD-4		GD-16		GD-64	
	Average	St. Dev.	Average	St. Dev.	Average	St. Dev.
1	905.5	15.22	936.06	27.61	878.64	17.27
2	1874.64	64.13	1785.48	26	1675.46	33.26
3	1128.94	34.38	1102.14	34.72	1047.62	26.97
4	1978.62	59.32	1879.08	45.81	1785.96	58.58
5	724.2	8.42	717.06	11.79	702.24	8.6
6	1130	21.9	1112.7	9.2	1078.04	27.4

Table 7.6: *Great Deluge Hyperheuristic*. Results: Average objective function values over 10 runs and corresponding standard deviations

Benchmark Instances	Simple Random					
	IE-4		IE-16		IE-64	
	Average	St. Dev.	Average	St. Dev.	Average	St. Dev.
1	893.96	46.1	933.38	45.7	947.7	64.85
2	1525.18	51.92	1562.44	65.51	1566.16	47.18
3	991.14	30.71	981.44	27.19	1031.4	36.52
4	1742.42	34.43	1665.72	48.07	1685.92	35.02
5	684.48	9.11	692.94	7.83	696.14	8.88
6	1001.8	32.99	1009.54	17.9	1037.7	16.44
	Choice Function					
	IE-4		IE-16		IE-64	
	Average	St. Dev.	Average	St. Dev.	Average	St. Dev.
1	968.84	34.35	972.56	62.76	1006.68	56.88
2	1948.34	88.32	1921.72	65.44	1867.3	45.02
3	1120.4	53.77	1103.68	33.43	1085.76	40.23
4	2105.1	45.84	2023.52	80.12	1891.62	50.56
5	708.92	6.93	707.32	11.06	702.94	9.41
6	1088.8	30.07	1093.16	27.28	1076.74	24.47

Table 7.7: *Improving Equal Hyperheuristic*. Results: Average objective function values over 10 runs and corresponding standard deviations

Benchmark Instances	Simple Random					
	OI-4		OI-16		OI-64	
	Average	St. Dev.	Average	St. Dev.	Average	St. Dev.
1	2368.96	130.27	2449.28	146.71	2420.98	62.73
2	3577.66	154.88	3467.64	161.94	3360.36	131.99
3	2236.52	116.69	2247.02	109.72	2348.74	147.4
4	3567.12	133.45	3483.8	201.61	3489.44	182.62
5	1347.56	50.63	1349.48	64.36	1299.44	89.47
6	2675.9	160.99	2627.38	115.99	2656.16	134.44
	Choice Function					
	OI-4		OI-16		OI-64	
	Average	St. Dev.	Average	St. Dev.	Average	St. Dev.
1	4092.2	191.07	3904.62	219.16	3963.66	215.03
2	7896.06	676.7	7700.48	654.66	7665.46	425.47
3	4602.36	333.03	4322.5	270.97	4446.78	356.77
4	7159.6	313.14	7005.9	280.32	6794.86	249.55
5	3421.24	217.75	3373.3	207.97	3081.38	231.82
6	4450.52	391.11	4376.86	391.67	4390.82	396.6

Table 7.8: *Only Improving Hyperheuristic*. Results: Average objective function values over 10 runs and corresponding standard deviations

Benchmark Instances	C30-P300		C80-P400		C400-P80	
	Average	St. Dev.	Average	St. Dev.	Average	St. Dev.
1	1051.02	48.17	1060.66	63.38	1072.96	61.28
2	2211.88	87.25	2258.56	97.24	2273.08	133.22
3	1184.24	60.31	1150.5	48.5	1177.5	52.38
4	3663.42	163.45	4083	280.47	4040.8	192.05
5	714.16	6.23	713.5	9.09	717.56	12.66
6	1188.1	58.1	1145.6	32.37	1204.6	70.66

Table 7.9: Tabu Search. Results : Average objective function values over 10 runs and corresponding standard deviations

Benchmark Instances	C800-P200		C1500-P300	
	Average	St. Dev.	Average	St. Dev.
1	1063.38	33.35	1038.04	55.11
2	2340.34	91.11	2241.32	87.07
3	1185.88	39.26	1146.86	25.17
4	3723.78	113.28	3620.9	186.09
5	712.48	10.45	714.84	9.23
6	1189.32	50.16	1184.42	23.7

Table 7.10: Tabu Search. Results : Average objective function values over 10 runs and corresponding standard deviations

problem is tackled with several solution methods. The problem was first solved with integer programming. However, it took a state-of-the-art commercial MIP solver more than 3 hours to find a feasible solution. To overcome this, we employed a tabu search algorithm hybridized with both a token-ring and a variable neighbourhood descent approach to tackle the problem in a first stage. Later on, we extended the approach with a novel hyper-heuristic algorithm, that outperformed both metaheuristics approaches.

Apart from the computational complexity exhibited by the integer programming attempt to solve the problem when compared to the tabu search, we feel that the latter approach in this case can profit from another essential difference. In general, heuristic and local search methods combine significant modelling power with superior flexibility in implementing domain dependent experience in the algorithm. In the present example this is apparent in the implementation of the various neighbourhoods. Our results demonstrate that the resulting transparency leads to fast algorithms producing satisfactory solutions. As our statistical analysis shows, the behaviour is sufficiently stable for use in an actual application.

We advocate for the application of existing local search neighbourhoods, as heuristics within a hyper-heuristic framework for general application. The implementation effort is limited compared to the experienced performance increase for the patient admission scheduling problem. Similar effects can be expected for different problems.

The developed application is meant to ease the admission schedulers task, since it automatically assigns patients to beds taking into account their wishes and medical nursing needs. Also the number of patients is balanced across the different departments. As a consequence of the latter, the workload and resource utilization will be spread more evenly over the entire hospital. The potential of the presented approach is to serve as an instrument for optimizing patient assignments each time new information becomes available (e.g. adjustment to the patient stay duration, additional patients on the list, cancelled treatments, etc.) in a real-world setting.

Initial test results of applying the presented approach to the patient assignment problem are promising and this research area is likely to become very challenging in the near future.

The benchmark problems that we developed for comparing algorithm performance, have been made available to the research community.

Chapter 8

Conclusions and Future Work

8.1 Conclusions

We have demonstrated in this thesis that our ***heuristic*** approaches are sufficiently flexible to solve a wide range of ***real-world timetabling*** problems in ***health care*** and ***education***. Problems from health care and education are chosen since these domains are responsible for a large part of a government's expenses. The total expenditure on health care in Belgium for example equals 10.2 % of the gross domestic product in 2007. The government in return demands that the available resources are used efficiently. Even if only a small percentage can be improved the nett effect will be huge. Hospitals are, for example, forced to make sure that patients recover faster, such that the average number of patients per bed per year increases. If a certain bed occupancy rate is not met, the hospital has to give up beds, and consequently lose income. Regarding the education domain, and especially higher education, the government demands that universities and polytechnics allow more student flexibility: students should have more courses to choose from, students should be able to start their studies in October or in February. Also, students are allowed to take courses from different curricula, which makes generating a clash-free timetable difficult. The complexity of formulating a clash-free timetable is nowadays so high, that it becomes almost impossible to do it manually. In this thesis we solve some of the most common problems encountered in the health care and education domain.

We have fully described, modelled and tackled these real-world timetabling problems using the same generic approach. The results of this thesis can contribute to further closing the gap between practice and theory in automated timetabling [113]. Some of the more theoretical applications lack robust and flexible techniques

handling the real-world complexity of the timetabling problems. By concentrating on real-world problems and their corresponding real-world constraints, one is compelled to come up with innovative ideas and methods. To measure the performance of the developed methods, they are applied to two benchmark data sets. The benchmark results can easily compete with the state-of-the-art algorithms from the literature. In the case of the Toronto examination benchmarks, for five out of the 13 benchmark instances better or equal results than the best results in the literature are obtained. Furthermore, we have successfully re-used a hyper-heuristic that was originally developed for tackling examination timetabling problems in a health care setting. The hyper-heuristic is the result of refactoring a local search algorithm with a multiple neighbourhood approach by downgrading the existing local search neighbourhoods into low-level heuristics in which extra randomness was added. Solving the patient admission scheduling problem with hyper-heuristics led to better solutions in less computation time than those obtained with the local search algorithm. Related to this, we also showed that hyper-heuristics, originally introduced as a general approach for solving a broad range of applications, can result in solutions that can contend with the best solutions from the literature.

We also applied an integer programming approach to solve the patient admission scheduling data instances. For the smaller data sets, optimal solutions could be obtained after several hours of computation. For the larger instances however, after three months of non-stop calculation no optimal solution was obtained. It seems that the complexity of the patient admission scheduling problem arises from the avoid-transfer constraint. If this constraint is not included in the problem, the integer programming approach can easily solve the problem. These long computation times prove why heuristic approaches are of interest for real-world problems: an optimal solution is not often required, but rather a good enough solution in a short amount of computation time. When a decision has to be made fast, one cannot afford to wait several months to obtain the optimal solution. It is better to have some good enough 'workable' solution than no solution at all.

We believe that much of the local search algorithm's performance depends on the way the solution space and the neighbourhoods are constructed, and on how these are applied, next to the skill of the search algorithm to avoid local optima. As a general advice we suggest to keep the representation of the solution space as close to reality and simple as possible. At the same time, try to satisfy most of the constraints. For example, in case of course timetabling, if multiple events per room - time slot combination are not allowed, a matrix representation is a good option. First, it is a one-to-one mapping of the manually constructed timetable, in which rooms and time slots are depicted. Secondly, and more importantly, every matrix element can only hold one event. As a consequence, the corresponding neighbourhoods will be easy to visualize. It is obvious that, in case we choose a matrix representation for the course timetabling problem, one of

the neighbourhoods will move an event to another room, and another will move an event to another time slot. Also, the neighbourhoods can be constructed such that some of the constraints are never violated. For example, in the uncapacitated examination timetabling case, the Kempe chain based neighbourhoods preserve feasibility (see Section 5.3.2). After the execution of this kind of neighbourhood, a solution remains feasible, since the structure does not allow to assign a student to more than one exam in the same time slot which corresponds to the only hard constraint. It is convenient that a partial evaluation suffices, since some of the constraints cannot be violated given the structure and the neighbourhoods.

8.2 Future Research

Some issues were not addressed in the research project. We list some of the challenging problems still left in the education and health care domain respectively.

8.2.1 Education

We have made abstraction of the fact that the campuses of KaHo Sint-Lieven as well as K.U. Leuven are geographically dispersed. Incorporating different campuses into the problem will lead to a new and compelling research opportunity. This will make the real-world problem even more complex, since the travelling times between the campuses should be incorporated, when assigning lecturers or students to activities at different campuses during the day. Also, in the future, universities and university colleges will be more and more compelled to cooperate more closely, resulting in exchanging lecturers between the different campuses. This will have consequences for the lecturers' timetables. University timetabling will more and more evolve from centralized to distributed timetabling over different campuses.

The best results for the education timetabling problems are obtained by hyper-heuristics. To solve the Toronto examination timetabling benchmarks a simple heuristic selection method is applied, i.e. one that randomly selects a heuristic from a fixed list of heuristics. It obtains in general better results than approaches that employ more 'intelligent' heuristic selection methods. This raises the research question whether further improvement can be obtained by merely looking at more sophisticated heuristic selection methods or by concentrating on smarter low level heuristics. There may be a trade-off in that smarter heuristics are required to reveal the power of a more intelligent heuristic selection mechanism. This may be the subject of further research on the interaction between the levels of a hyper-heuristic.

The work in this thesis is part of a more general effort to build a comprehensive volume of models and tools for the timetabling domain. Characteristic to this

domain is the combination and strong intertwining of both human and material resources. Traditionally these are different worlds, humans being subject to personnel and labour agreements while the material resources are managed by facility managers. Many timetabling applications nowadays take the resource setting for granted and try to meet the human constraints. Scientific publications tend to focus on either personnel or machinery. We think that both for practice and for science, significant progress can be realized by combining both. This raises the need for strong models and general techniques that allow investigating the mutual impact at the common frontier of these two worlds. The examples below merely illustrate how this could be realized.

In Chapter 6 we extended the linear numberings methods for solving additional nurse rostering constraints. This evaluation mechanism is now solely used for personnel rostering. We plan to investigate whether this method can be applied to construct university timetables. In that case we model students and teachers as the employees with their wishes as constraints that should preferably be solved. The advantage would be that additional constraints do not demand extra programming code, since they can be expressed in terms of the existing numberings. However, precautions have to be taken such that the difficulty is not shifted towards expressing the constraints in terms of linear numberings. This can be accomplished by means of a decision support system that assists the developer in expressing the constraints.

Due to the increased flexibility of the education system, the (human) timetabler's task becomes more and more complex. Students have more freedom in choosing their courses and in the order they want to take them. Also, in most disciplines the major courses can be combined with courses from completely different disciplines. Of course, this increased flexibility comes at a price. Because of all these extra constraints, it is almost impossible for a human timetabler to come up with a manually constructed clash-free timetable, simply because this problem is too hard to solve for a human, without any computer assistance. This shows that good timetabling software is now more than ever indispensable. Next to the obvious good search algorithms it should also provide a graphical user interface that does exactly what the human timetabler wants: blocking courses, moving courses to other periods, indicating where there is still a problem that needs to be solved, ... The human timetabler should have the feeling that he/she is still in command, otherwise the timetabling software will never be used. Here lays a task for GUI developers: the underlying search algorithm can be state-of-the-art, but if there is no user-friendly interface to employ it, the chances are low that anyone will ever use it.

8.2.2 Health Care

We have put the data sets that we generated for the patient admission scheduling problem on-line (Chapter 7). Also, a validation tool that evaluates the new best solutions against our objective function is provided. This is in line with the recommendations of Schaerf and Di Gaspero [131]. We also plan to make the real-world data of KaHo Sint-Lieven and K.U. Leuven publicly available.
Within the current problem settings of the patient admission scheduling research, it is not possible to significantly increase the bed occupancy rate in the hospital. That is due to the assumption of a fixed patient list and fixed patient stays over the considered time horizon. However, by taking all the partial objectives into account, the approach allows minimizing the financial loss that is related to ungranted room preferences, and to increase patients' satisfaction. Extending the problem definition with waiting lists will be one of our first future research concerns. Also, to make the approach more realistic, we will incorporate the fact that some patients recover better than others into our model, instead of assuming a fixed stay-of-length for all patients that undergo the same surgery. Concerning this last issue, we already came across a real-world case in which the length-of-stay of the patients in an intensive care unit depends on how good the patients recover. The problem confronted with here is that the number of intensive care beds is limited and the number of surgeries is high. It often happens that patients that were planned and admitted for surgery, are sent back home (without surgery), because there are no free beds in the intensive care unit due to admitted patients which do not recover as good as foreseen. An additional point of attention involves evenly spreading the nursing capacity across the entire hospital. Some patients demand more attention than others and that requires more detailled modelling. Another research direction involves addressing emergency admissions, considering the intensive care department, and operating theatre scheduling. More generally, the combination of both the patient admission scheduling, the operating theatre scheduling and the nurse rostering problem is an interesting research option. Combining them gives the scheduler a good idea of the current and future occupation of the hospital, which can be taken into account when one has to decide on the coverage of each department. As far as we know, the integration of these objectives is a new subject in the operational research field.

Although this thesis gives an onset to quickly developing timetabling applications, in future work we want to make the ideas of this thesis more general. The idea is that the application will assist the developer in mapping the existing data on the framework's domain model, while the constraints of the problem can be expressed in a uniform language. Dependent on the domain model, the application should suggest an associated solution space and low-level heuristics.

An intelligent heuristic selection mechanism should decide, based on a statistical analysis, whether to include extra low-level heuristics from the heuristics library.

Bibliography

[1] Second International Timetabling Competition, http://www.cs.qub.ac.uk/itc2007/.

[2] International Timetabling Competition, http://www.idsia.ch/Files/ttcomp2002/.

[3] Benchmark Data Sets in Exam Timetabling, http://www.cs.nott.ac.uk/∼rxq/data.htm.

[4] Post-enrolment based course timetabling track of the Second International Timetabling Competition, http://www.cs.qub.ac.uk/itc2007/winner/finalorder.htm.

[5] S. Abdullah. *Heuristic Approaches for University Timetabling Problems*. PhD thesis, School of Computer Science and Information Technology, University of Nottingham, June 2006.

[6] S. Abdullah, S. Ahmadi, E.K. Burke, and M. Dror. Investigating ahuja-orlins large neighbourhood search for examination timetabling. *OR Spectrum*, 29(2):351–372, 2007.

[7] S. Abdullah, E. K. Burke, and B. McCollum. An investigation of variable neighbourhood search for university course timetabling. In *Proceedings of the second Multi-disciplinary International Conference on Scheduling: Theory and Applications (MISTA)*, pages 413–427, July 2005.

[8] S. Abdullah, E.K. Burke, and B. McCollum. Using a randomised iterative improvement algorithm with composite neighbourhood structures for university course timetabling. In *Proceedings of the Metaheuristic International Conference 2005 (MIC'05), CD-Rom*, Austria, Augustus 2005.

[9] U. Aickelin and J. Li. *Scalable Optimization via Probabilistic Modeling*, volume 33/2006 of *Studies in Computational Intelligence*, chapter Bayesian Optimization for Nurse Scheduling, pages 315–332. Springer-Verlag Berlin Heidelberg, 2006.

[10] E. Akcali, M.J. Côté, and C. Lin. A network flow approach to optimizing hospital bed capacity decisions. *Health Care Management Science*, 9:391–404, 2006.

[11] S.M. Al-Yakoob and H.D. Sherali. Mathematical programming models and algorithms for a classfaculty assignment problem. *European Journal of Operational Research*, 173(2):488–507, September 2006.

[12] R. Alvarez-Valdes, E. Crespo, and J.M. Tamarit. Assigning students to course sections using tabu search. *Annals of Operations Research*, 96:1–16, 2000.

[13] A.S. Asratian and D. de Werra. A generalized classteacher model for some timetabling problems. *European Journal of Operational Research*, 143(3):531–542, December 2002.

[14] M. Atsuta, K. Nonobe, and T. Ibaraki. Itc2007 track 1: An approach using a general csp solver. Technical report, 2008.

[15] M. Ayob, A.M.A Malik, S. Abdullah, A.R. Hamdan, G. Kendall, and R. Qu. Solving a practical examination timetabling problem: A case study. In O. Gervasi and M. Gavrilova, editors, *Computational Science and Its Applications ICCSA 2007*, volume 4707 of *Lecture Notes in Computer Science*, pages 611–624. Springer-Verlag Berlin Heidelberg, 2007.

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

[17] V.A. Bardadym. Computer-aided school and university timetabling: The new wave. In *Selected and Revised Papers of the First International Conference on Practice and Theory of Automated Timetabling, (PATAT 1995)*, volume 1153 of *Lecture Notes in Computer Science*, pages 22–45, Edinburgh, 1996. Springer-Verlag Berlin Heidelberg.

[18] G.R. Beddoe and S. Petrovic. Selecting and weighting features using a genetic algorithm in a case-based reasoning approach to personnel rostering. *European Journal of Operational Research*, 175(2):649–671, 2006.

[19] J. Beliën and E. Demeulemeester. Building cyclic master surgery schedules with leveled resulting bed occupancy. *European Journal of Operational Research*, 176:1185–1204, 2007.

[20] F. Bellanti, G. Carello, F. Della Croce, and R. Tadei. A greedy-based neighborhood search approach to a nurse rostering problem. *European Journal of Operational Research*, 153(1):28–40, 2004.

[21] B. Bilgin, P. Demeester, and G. Vanden Berghe. A hyperheuristic approach to the patient admission scheduling problem. Technical report, KaHo Sint-Lieven, 2008. submitted to Journal of Heuristics.

[22] D.J. Bradley and J.B. Martin. Continuous personnel scheduling algorithms: a literature review. *Journal of the Society for Health Systems*, 2(2):8–23, 1990.

[23] H.-J. Bürckert, K. Fischer, and G. Vierke. Transportation scheduling with holonic mas - the teletruck approach,. In *Proceedings of the Third International Conference on Practical Applications of Intelligent Agents and Multiagents (PAAM'98)*, pages 577–590, 1998.

[24] E. K. Burke, D. de Werra, and J. H. Kingston. *Handbook of Graph Theory*, chapter Applications to Timetabling, pages 445–474. CRC, London, 2004.

[25] E. K. Burke, G. Kendall, and E. Soubeiga. A tabu-search hyperheuristic for timetabling and rostering. *Journal of Heuristics*, 9(6):451–470, December 2003.

[26] E.K. Burke and Y. Bykov. A late acceptance strategy in hill-climbing for exam timetabling problems. In E.K. Burke and M. Gendreau, editors, *Proceedings of the The 7th International Conference on the Practice and Theory of Automated Timetabling*, Montreal, Canada, August 2008.

[27] E.K. Burke, Y. Bykov, J. Newall, and S. Petrovic. A time-predefined approach to course timetabling. *Yugoslav Journal of Operations Research*, 13(2):139–151, 2003.

[28] E.K. Burke, Y. Bykov, J. Newall, and S. Petrovic. A time-predefined local search approach to exam timetabling problems. *IIE Transactions*, 36(6):509–528, June 2004.

[29] E.K. Burke, Y. Bykov, and S. Petrovic. A multicriteria approach to examination timetabling. In E.K. Burke and W. Erben, editors, *Selected revised papers of the third international conference on Practice and Theory of Automated Timetabling III (Patat 2000)*, volume 2079 of *Lecture Notes in Computer Science*, pages 118–131. Springer-Verlag Berlin Heidelberg, 2001.

[30] E.K. Burke, P. De Causmaecker, S. Petrovic, and G. Vanden Berghe. Fitness evaluation for nurse scheduling problems. In *Proceedings of Congress on Evolutionary Computation (CEC2001)*, pages 1139–1146, Seoul, 2001. IEEE Press.

[31] E.K. Burke, P. De Causmaecker, G. Vanden Berghe, and H. Van Landeghem. The state of the art of nurse rostering. *Journal of Scheduling*, 7(6):441–499, November 2004.

[32] E.K. Burke, E. Hart E., G. Kendall, J. Newall, P. Ross, and S. Schulenburg. *Handbook of Meta-Heuristics*, chapter Hyper-Heuristics: An Emerging Direction in Modern Search Technology, pages 457–474. Kluwer Academic Publishers, 2003.

[33] E.K. Burke, A. Eckersley, B. McCollum, S. Petrovic, and R. Qu. Hybrid variable neighbourhood approaches to university exam timetabling. Technical Report Technical Report NOTTCS-TR-2006-2, School of Computer Science, University of Nottingham, 2006.

[34] E.K. Burke, D.G. Elliman, and R.F. Weare. A university timetabling system based on graph colouring and constraint manipulation. *Journal of Research on Computing in Education*, 27(1):1–18, 1994.

[35] E.K. Burke and G. Kendall, editors. *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*. Springer-Verlag Berlin Heidelberg, New York, 2005.

[36] E.K. Burke, B.L. MacCarthy, S. Petrovic, and R. Qu. Case-based reasoning in course timetabling: An attribute graph approach. In *Proceedings of the fourth International Conference on Case-Based Reasoning: Case-Based Reasoning Research and Development*, volume 2080 of *Lecture Notes In Computer Science*, pages 90–104. Springer-Verlag Berlin Heidelberg, 2001.

[37] E.K. Burke, B.L. MacCarthy, S. Petrovic, and R. Qu. Knowledge discovery in hyper-heuristic using case-based reasoning on course timetabling. In *Selected revised papers of the fourth International Conference on the Practice and Theory of Automated Timetabling (PATAT 2002)*, volume 2740 of *Lecture Notes in Computer Science*, pages 276–286. Springer-Verlag Berlin Heidelberg, 2003.

[38] E.K. Burke, B.L. MacCarthy, S. Petrovic, and R. Qu. Multiple-retrieval case based reasoning for course timetabling problems. *Journal of the Operational Research Society*, 57(2):148–162, 2006.

[39] E.K. Burke, B. McCollum, A. Meisels, S. Petrovic, and R. Qu. A graph-based hyper-heuristic for educational timetabling problems. *European Journal of Operational Research*, 179:177–192, 2007.

[40] E.K. Burke and J. Newall. Enhancing timetable solutions with local search methods. In E.K. Burke and P. De Causmaecker, editors, *Selected revised papers of the fourth International Conference on Practice and Theory of Automated Timetabling (PATAT 2002)*, volume 2740 of *Lecture Notes in Computer Science*, pages 195–206. Springer-Verlag Berlin Heidelberg, 2003.

[41] E.K. Burke and J. Newall. Solving examination timetabling problems through adaptation of heuristic orderings. *Annals of operations Research*, 129:107–134, 2004.

[42] E.K. Burke, J. Newall, and R. Weare. A memetic algorithm for university exam timetabling. In E.K. Burke and P. Ross, editors, *Selected revised papers of the first International Conference on Practice and Theory of Automated*

Timetabling (PATAT 1995), volume 1153 of *Lecture Notes in Computer Science*, pages 241–250. Springer-Verlag Berlin Heidelberg, 1996.

[43] E.K. Burke and J.P. Newall. A multi-stage evolutionary algorithm for the timetable problem. *IEEE Transactions on Evolutionary Computation*, 13(1):63–74, April 1999.

[44] E.K. Burke and S. Petrovic. Recent research directions in automated timetabling. *European Journal of Operational Research*, 140:266–280, 2002.

[45] E.K. Burke, S. Petrovic, and R. Qu. Case based heuristic selection for timetabling problems. *Journal of Scheduling*, 9(2):115–132, 2006.

[46] H. Cambazard, E. Hebrard, B. O'Sullivan, and A. Papadopoulos. Local search and constraint programming for the post-enrolment-based course timetabling problem. In E.K. Burke and M. Gendreau, editors, *Proceedings of the seventh International Conference on the Practice and Theory of Automated Timetabling (PATAT 2008)*, Montreal, Canada, August 2008.

[47] M. Caramia, P. Dell'Olmo, and G.F. Italiano. Novel local-search-based approaches to university examination timetabling. *INFORMS Journal on Computing*, 20(1):86–99, Winter 2008.

[48] M.P. Carrasco and M.V. Pato. A comparison of discrete and continuous neural network approaches to solve the class/teacher timetabling problem. *European Journal of Operational Research*, 153:65–79, 2004.

[49] M.P. Carrasco and M.V. Pato. A potts neural network heuristic for the class/teacher timetabling problem. In *Applied Optimization, Metaheuristics: computer decision-making book*, pages 173–186. Kluwer Academic Publishers, 2004.

[50] M.W. Carter and S.D. Lapierre. Scheduling emergency room physicians. *Health Care Management Science*, 4:347–360, 2001.

[51] M.W. Carter and G. Laporte. Recent developments in practical examination timetabling. In E.K. Burke and P. Ross, editors, *Selected revised papers of the first international conference on Practice and Theory of Automated Timetabling (Patat 1995)*, volume 1153 of *Lecture notes in computer science*, pages 3–21. Springer-Verlag Berlin Heidelberg, 1996.

[52] M.W. Carter, G. Laporte, and S.Y. Lee. Examination timetabling: Algorithmic strategies and applications. *Journal of the Operation Research Society*, 74:373–383, 1996.

[53] I. Charon and O. Hudry. The noising methods: A generalization of some metaheuristics. *European Journal of Operational Research*, 135(1):86–101, November 2001.

[54] C. Cheng, L. Kang, N. Leung, and G.M. White. Investigations of a constraint logic programming approach to university timetabling. In *Selected papers of the first International Conference on Practice and Theory of Automated Timetabling (Patat 1995)*, volume 1153 of *Lecture Notes in Computer Science*, pages 112–129, Edinburgh, UK, August/September 1995.

[55] M. Chiarandini, M. Birattari, K. Socha, and O. Rossi-Doria. An effective hybrid algorithm for university course timetabling. *Journal of Scheduling*, 9(5):403–432, 2006.

[56] M. Chiarandini, C. Fawcett, and H. Hoos. A modular multiphase heuristic solver for post enrolment course timetabling. In E.K. Burke and M. Gendreau, editors, *Proceedings of the seventh International Conference on the Practice and Theory of Automated Timetabling (Patat 2008)*, Montreal, Canada, August 2008.

[57] C.F. Chien, F.P. Tseng, and C.H. Chen. An evolutionary approach to rehabilitation patient scheduling: A case study. *European Journal of Operational Research*, 189:1234–1253, 2008.

[58] C. Cogos, P. Alefragis, and E. Housos. A multi-staged algorithmic process for the solution of the examination timetabling problem. In E.K. Burke and M. Gendreau, editors, *Practice and Theory of Automated Timetabling (PATAT 2008)*, Montreal, Canada, 19-22 August 2008.

[59] A. Colorni, M. Dorigo, and V. Maniezzo. Metaheuristics for high school timetabling. *Computational Optimization and Applications*, 9(3):275–298, March 1998.

[60] D. Costa. A tabu search algorithm for computing an operational timetable. *European Journal of Operational Research*, 76(1):98–110, 1994.

[61] P. Cowling, G. Kendall, and E. Soubeiga. A hyperheuristic approach to scheduling a sales summit. In E.K. Burke and W. Erben, editors, *Selected Papers of the third International Conference on the Practice And Theory of Automated Timetabling (PATAT2000)*, volume 2079 of *Lecture Notes in Computer Science*, pages 176–190, 2001.

[62] S. Creemers and M. Lambrecht. An advanced queueing model to analyze appointment-driven service systems. *Computers and Operations Research*, 36(10):2773–2785, October 2009.

[63] S. Creemers and M. Lambrecht. Queueing models for appointment-driven systems. *Annals of Operations Research*, 2009.

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

[65] P. De Causmaecker. Towards a reference model for timetabling and rostering. In E.K. Burke and M. Gendreau, editors, *Proceedings of the seventh International Conference on the Practice and Theory of Automated Timetabling (PATAT 2008)*, Montreal, Canada, August 2008.

[66] P. De Causmaecker, P. Demeester, and G. Vanden Berghe. A decomposed metaheuristic approach for a real-world university timetabling problem. *European Journal of Operational Research*, 195(1):307–318, May 2009.

[67] P. De Causmaecker, P. Demeester, G. Vanden Berghe, and B. Verbeke. Evaluation of employee rosters with the extended linear numberings method. In *24th Annual Workshop of the UK Planning and Scheduling Special Interest Group (UK PlanSIG 2005)*, pages 125–133, London, December 2005.

[68] G. De Smet. Itc2007 - examination track: Practice and theory of automated timetabling. Technical report, 2008.

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

[70] P. Demeester, B. Bilgin, P. De Causmaecker, and G. Vanden Berghe. A hyperheuristic approach for real-world and benchmark examination timetabling problems. Technical report, KaHo Sint-Lieven, 2009. submitted to Journal of Scheduling.

[71] P. Demeester, B. Bilgin, and G. Vanden Berghe. Patient admission scheduling benchmark instances. http://allserv.kahosl.be/∼peter/pas/index.html, 2008.

[72] P. Demeester, P. De Causmaecker, and G. Vanden Berghe. A hybrid guided local search approach to a university course timetabling problem. Technical report, KaHo Sint-Lieven, 2009.

[73] P. Demeester, W. Souffriau, P. De Causmaecker, and G. Vanden Berghe. A hybrid tabu search algorithm for automatically assigning patients to beds. Technical report, KaHo Sint-Lieven, 2009. submitted to AI in Medicine.

[74] L. Di Gaspero, B. McCollum, and A. Schaerf. The second international timetabling competition (itc-2007):curriculum-based course timetabling (track 3),. In F. Pecora and N. Policella, editors, *International Workshop on Scheduling a Scheduling Competition, 17th International Conference on Automated Planning & Scheduling*, Providence, Rhode Island, USA., September 22nd-26th 2007 2007.

[75] L. Di Gaspero and A. Schaerf. Tabu search techniques for examination timetabling. In E.K. Burke and W. Erben, editors, *Selected revised papers of the third international conference on Practice and Theory of Automated Timetabling (Patat 2000)*, volume 2079 of *Lecture Notes in Computer Science*, pages 104–117. Springer-Verlag Berlin Heidelberg, 2001.

[76] L. Di Gaspero and A. Schaerf. A multineighbourhood local search solver for the timetabling competition ttcomp 2002. In *Proceedings of the fifth International Conference on Practice and Theory of Automated Timetabling (PATAT 2004)*, pages 475–478, Pittsburgh, USA, 2004.

[77] Luca Di Gaspero and Andrea Schaerf. Multi-neighbourhood local search with application to course timetabling. In E. Burke and P. De Causmaecker, editors, *Selected revised papers of the fourth international conference on Practice and Theory of Automated Timetabling (Patat 2002)*, number 2740 in Lecture Notes in Computer Science. Springer-Verlag Berlin Heidelberg, 2003.

[78] Luca Di Gaspero and Andrea Schaerf. Neighborhood portfolio approach for local search applied to timetabling problems. *Journal of Mathematical Modelling and Algorithms*, 5(1):65–89, April 2006.

[79] M. Dimopoulou and P. Miliotis. Implementation of a university course and examination timetabling system. *European Journal of Operational Research*, 130:202–213, 2001.

[80] A.E. Eiben and J.I. van Hemert. *New Ideas in Optimization*, chapter SAW-ing EAs: Adapting the Fitness Function for Solving Constrained Problems, pages 389–402. McGraw-Hill, 1999.

[81] Michael Eley. Ant algorithms for the exam timetabling problem. In E.K. Burke and H. Rudová, editors, *Selected revised papers of the sixth international conference on Practice and Theory of Automated Timetabling (Patat 2006)*, volume 3867 of *Lecture Notes in Computer Science*, pages 364–382. Springer-Verlag Berlin Heidelberg, 2007.

[82] A.T. Ernst, H. Jiang, M. Krishnamoorthy, and D. Sier. Staff scheduling and rostering: A review of applications, methods and models. *European Journal of Operational Research*, 153(1):3–27, 2004.

[83] M. Gamache and F. Soumis. *OR in the Airline Indystry*, chapter A method for optimally solving the rostering problem, pages 124–157. Kluwer, 1998.

[84] P. Gemmel and R. Van Dierdonck. Admission scheduling in acute care hospitals: does the practice fit with the theory? *International Journal of Operations and Production Management*, 19(9):863–878, 1999.

[85] M. Gendreau, A. Hertz, and G. Laporte. A tabu search heuristic for the vehicle routing problem. *Management Science*, 40:1276–1290, 1994.

[86] F. Glover and M. Laguna. *Tabu Search*. Kluwer Academic Publishers, 1997.

[87] L.V. Green. *Operations Research and Health Care: A Handbook of Methods and Applications*, volume 70 of *International Series in Operations Research & Management Science*, chapter Capacity Planning and Management in

Hospitals, pages 15–42. Kluwer Academic Publishers Group, Boston, Mass., 2004.

[88] T. Grossman, D. Samuelson, S. Oh, and T. Rohleder. *Encyclopedia of Operations Research and Management Science 2nd Edition*, chapter Call centres, pages 73–76. Springer-Verlag Berlin Heidelberg, 2001.

[89] C. Guéret, N. Jussien, P. Boizumault, and C. Prins. Building university timetables using constraint logic programming. In *Selected revised papers of the first International Conference on Practice and Theory of Automated Timetabling (PATAT 1995)*, volume 1153 of *Lecture Notes in Computer Science*, pages 130–145, Edinburgh, UK, 1996. Springer-Verlag Berlin Heidelberg.

[90] E. Hans, G. Wullink, M. van Houdenhoven, and G. Kazemier. Robust surgery loading. *European Journal of Operational Research*, 185:1038–1050, 2008.

[91] M.P. Hansen and R.V.V. Vidal. Planning of high school examinations in denmark. *European Journal of Operational Research*, 87:519–534, 1995.

[92] P.R. Harper and A.K. Shahani. Modelling for the planning and management of bed capacities in hospitals. *Journal of the Operational Research Society*, 53:11–18, 2002.

[93] A. Hertz. Tabu search for large scale timetabling problems. *European Journal of Operational Research*, 54:39–47, 1991.

[94] A. Hertz. Finding a feasible course schedule using tabu search. *Discrete Applied Mathematics*, 35:255–270, 1992.

[95] J.J. Hopfield and D.W. Tank. Neural computation of decisions in optimization problems. *Biological Cybernetics*, 52(3):141–152, July 1985.

[96] A.K. Hutzschenreuter, P.A.N. Bosman, I. Blonk-Altena, J. van Aarle, and J.A. La Poutré. Agent-based patient admission scheduling in hospitals. In J.P. Mueller L. Padgham, D.C. Parkes and S. Parsons, editors, *Proceedings of the Seventh International Conference on Autonomous Agents and Multiagent Systems - AAMAS-2008*, pages 45–54. ACM, 2008.

[97] A. Ikegami and A. Niwa. A subproblem-centric model and approach to the nurse scheduling problem. *Mathematical Programming*, 97(3):517–541, 2003.

[98] B. Jaumard, J. Cordeau, and R. Morales. Efficient timetabling solution with tabu search. Technical report, Metaheuristics Network, 2002.

[99] G.C. Kaandorp and G. Koole. Optimal outpatient appointment scheduling. *Health Care Management Science*, 10:217–229, 2007.

[100] G. Kendall and N.M. Hussin. An investigation of a tabu-search-based hyper-heuristic for examination timetabling. In G. Kendall, E.K. Burke, S. Petrovic, and M. Gendreau, editors, *Multidisciplinary Scheduling: Theory and Applications. Selected papers of the first International Conference (MISTA)*, pages 309–328. Springer-Verlag Berlin Heidelberg, 2005.

[101] G. Kendall and N.M. Hussin. A tabu search hyper-heuristic approach to the examination timetabling problem at the mara university of technology. In E. K. Burke and M. Trick, editors, *Selected revised papers of the fifth international conference on Practice and Theory of Automated Timetabling (Patat 2004)*, volume 3616 of *Lecture Notes in Computer Science*, pages 270–293. Springer-Verlag Berlin Heidelberg, 2005.

[102] J.H. Kingston. A tiling algorithm for high school timetabling. In E.K. Burke and M. Trick, editors, *Selected revised papers of the fifth international conference on Practice and Theory of Automated Timetabling (PATAT 2004)*, volume 3616 of *Lecture Notes in Computer Science*, pages 208–225, Pittsburgh, USA, August 2005. Springer-Verlag Berlin Heidelberg.

[103] P. Kostuch. The university course timetabling problem with a three-phase approach. In E.K. Burke and M. Trick, editors, *Selected revised papers of the fifth international conference on Practice and Theory of Automated Timetabling (PATAT 2004)*, volume 3616 of *Lecture Notes in Computer Science*, pages 109–125, Pittsburgh, USA, August 2005. Springer-Verlag Berlin Heidelberg.

[104] R.J. Kusters and P.M.A. Groot. Modelling resource availability in general hospitals. design and implementation of a decision support model. *European Journal of Operational Research*, 88:428–445, 1996.

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

[106] R. Lewis. A survey of metaheuristic-based techniques for university timetabling problems. *OR Spectrum*, 30(1):167–190, 2008.

[107] R. Lewis, B. Paechter, and B. McCollum. Post enrolment based course timetabling: A description of the problem model used for track two of the second international timetabling competition. Cardiff Working Papers in Accounting and Finance A2007-3, ISSN: 1750-6658, Cardiff Business School, Cardiff University, Wales, August 2007 2007.

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

[109] A. Lim, A. J. Chin, H.W. Kit, and W.C. Oon. A campuswide university examination timetabling application. In *Proceedings of the 17th national conference on artificial intelligence and 12th conference on innovative applications of artificial intelligence*, pages 1020–1025, 2000.

[110] B. Maenhout and M. Vanhoucke. An electromagnetic meta-heuristic for the nurse scheduling problem. *Journal of Heuristics*, 13(4):359–385, August 2007.

[111] C.C. Marinagi, C.D. Spyropoulos, C. Papatheodorou, and S. Kokkotos. Continual planning and scheduling for managing patient tests in hospital laboratories. *Artificial Intelligence in Medicine*, 20(2):139–154, October 2000.

[112] A. Mayer, C. Nothegger, A. Chwatal, and G. Raidl. Solving the post enrolment course timetabling problem by ant colony optimization. In E.K. Burke and M. Gendreau, editors, *Proceedings of the seventh International Conference on the Practice and Theory of Automated Timetabling (PATAT 2008)*, Montreal, Canada, August 2008.

[113] B. McCollum. A perspective on bridging the gap between theory and practice in university timetabling. In E.K. Burke and H. Rudová, editors, *Revised selected papers of the sixth International Conference on Practice and Theory of Automated Timetabling (PATAT 2006)*, volume 3867 of *Lecture Notes in Computer Science*, pages 3–23. Springer-Verlag Berlin Heidelberg, August/September 2007.

[114] B. McCollum, P.J. McMullan, A.J. Parkes, E.K. Burke, and S. Abdullah. An extended great deluge approach to the examination timetabling problem. In *The 4th Multidisciplinary International Conference on Scheduling: Theory and Applications (Mista 09)*, pages 424–434, Dublin, Ireland, August 2009.

[115] B. McCollum, A. Schaerf, B. Paechter, P. McMullan, R. Lewis, A.J. Parkes, L. Di Gaspero, R. Qu, and E.K. Burke. Setting the research agenda in automated timetabling: The second international timetabling competition. *INFORMS Journal on Computing*, Articles in Advance:1–11, 2009.

[116] L.T.G. Merlot, N. Boland, B.D. Hughes, and P.J. Stuckey. A hybrid algorithm for the examination timetabling problem. In E.K. Burke and P. De Causmaecker, editors, *Selected revised papers of the fourth international conference on Practice and Theory of Automated Timetabling (Patat 2002)*, volume 2740 of *Lecture Notes in Computer Science*, pages 207–231. Springer-Verlag Berlin Heidelberg, 2003.

[117] H. Meyer auf'm Hofe. Solving rostering tasks as constraint optimization. In E.K. Burke and W. Erben, editors, *Selected revised papers of the third International Conference on Practice and Theory of Automated Timetabling (PATAT 2000)*, volume 2079 of *Lecture Notes in Computer Science*, pages 191–212, Konstanz, 2001. Springer-Verlag Berlin Heidelberg.

[118] M. Moz and M. Pato. Solving the problem of rerostering nurse schedules with hard constraints: New multicommodity flow models. *Annals of Operations Research*, 128:179–197, 2004.

[119] T. Müller. ITC2007 solver description: A hybrid approach. In E.K. Burke and M. Gendreau, editors, *Proceedings of the seventh International Conference on the Practice and Theory of Automated Timetabling (Patat 2008)*, Montreal, Canada, August 2008.

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

[121] S.N. Ogulata, M. Koyuncu, and E. Karakas. Personnel and patient scheduling in the high demanded hospital services: A case study in the physiotherapy service. *Journal of Medical Systems*, 32(3):221–228, June 2008.

[122] E. Özcan, B. Bilgin, and E.E. Korkmaz. A comprehensive analysis of hyper-heuristics. *Intelligent Data Analysis*, 12(1):3–23, January 2008.

[123] S. Petrovic and E.K. Burke. *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*, chapter University Timetabling, pages 45–1 – 45–23. CRC Press, 2004.

[124] S. Petrovic and Y. Bykov. A multiobjective optimisation technique for exam timetabling based on trajectories. In E.K. Burke and P. De Causmaecker, editors, *Selected revised papers of the fourth International Conference on Practice and Theory of Automated Timetabling (PATAT 2002)*, volume 2740 of *Lecture Notes in Computer Science*, pages 179–192, 2002.

[125] N. Pillay. A developmental approach to the examination timetabling problem. Technical report, 2008.

[126] R. Qu and E.K. Burke. Adaptive decomposition and construction for examination timetabling problems. In P. Baptiste, G. Kendall, A. Munier-Kordon, and F. Sourd, editors, *Multidisciplinary International Scheduling: Theory and Applications (MISTA 2007)*, pages 418–425, Paris, France, August 2007.

[127] R. Qu, E.K. Burke, B. McCollum, L.T.G. Merlot, and S.Y. Lee. A survey of search methodologies and automated system development for examination timetabling. *Journal of Scheduling*, 12(1):55–89, February 2009.

[128] H. Rudova and K. Murray. University course timetabling with soft constraints. In *Selected revised papers of the fourth International Conference on Practice and Theory of Automated Timetabling (PATAT 2002)*, volume 2740 of *Lecture Notes in Computer Science*, pages 311–329, Gent, 2003. Springer-Verlag Berlin Heidelberg.

[129] A. Schaerf. Local search techniques for large high school timetabling problems. *IEEE Transactions on Systems, Man and Cybernetics Part A: Systems and Human*, 29(4):368–377, 1999.

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

[131] A. Schaerf and L. Di Gaspero. Measurability and reproducibility in university timetabling research: Discussion and proposals. In E.K. Burke and H. Rudová, editors, *Revised selected papers of the sixth International Conference on Practice and Theory of Automated Timetabling (Patat 2006)*, volume 3867 of *Lecture Notes in Computer Science*, pages 40–49, Brno, Czech Republic, August/September 2007. Springer-Verlag Berlin Heidelberg.

[132] K. Schimmelpfeng and S. Helber. Application of a real-world university-course timetabling model solved by integer programming. *OR Spectrum*, 29(4):783–803, October 2007.

[133] V.L. Smith-Daniels, S.B. Schweikhart, and D.E. Smith-Daniels. Capacity management in health care services: Review and future research directions. *Decision Sciences*, 19(4):889–919, Fall 1988.

[134] K. Socha, M. Sampels, and M. Manfrin. Ant algorithms for the university course timetabling problem with regard to the state-of-the-art. In *Proceedings of the third European Workshop on Evolutionary Computation in Combinatorial Optimization (EvoCOP 2003)*, volume 2611 of *Lecture Notes in Computer Science*, pages 334–345, 2003.

[135] C.D. Spyropoulos. Ai planning and scheduling in the medical hospital environment. *Artificial Intelligence in Medicine*, 20(2):101–111, October 2000.

[136] P. Taylor and S. Huxley. A break from tradition for the san francisco police: Patrol officer scheduling using an optimization-based decision support system. *Interfaces*, 19(1):4–24, 1989.

[137] J.M. Thompson and K.A. Dowsland. Variants of simulated annealing for the examination timetabling problem. *Annals of Operations Research*, 63:105–128, 1996.

[138] J.M. Thompson and K.A. Dowsland. A robust simulated annealing based examination timetabling system. *Computers and Operations Research*, 25:637–648, 1998.

[139] J.M. Tien and A. Kamiyama. On manpower scheduling algorithms. *SIAM Review*, 24(3):257–287, 1982.

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

[141] I.B. Vermeulen, S.M. Bohte, S.G. Elkhuizen, P.J.M. Bakker, and J.A. La Poutré. Decentralized online scheduling of combination-appointments in hospitals. In *Proceedings of the International Conference on Automated Planning and Scheduling*. ACM, 2008.

[142] I.B. Vermeulen, S.M. Bohte, S.G. Elkhuizen, H. Lameris, P.J.M. Bakker, and J.A. La Poutré. Adaptive resource allocation for efficient patient scheduling. *Artificial Intelligence in Medicine*, 46(1):67–80, May 2009.

[143] I.B. Vermeulen, S.M. Bohte, D.J.A. Somefun, and J.A. La Poutré. Multi-agent pareto appointment exchanging in hospital patient scheduling. *Service Oriented Computing and Applications*, 1(3):185–196, November 2007.

[144] J.M.H. Vissers, I.J.B.F. Adan, and N.P. Dellaert. Developing a platform for comparison of hospital admission systems: An illustration. *European Journal of Operational Research*, 180:1290–1301, 2007.

[145] C. Voudouris and E. Tsang. *Handbook of Metaheuristics*, chapter Guided Local Search, pages 185–218. International Series in Operations Research and Management Science. Kluwer Academic Publishers, 2002.

[146] A. Wren. Scheduling, timetabling and rostering - a special relationship? In E.K. Burke and P. Ross, editors, *Selected revised papers of the first International Conference on Practice and Theory of Automated Timetabling (Patat 1995)*, volume 1153 of *Lecture Notes in Computer Science*, pages 46–75, Edinburgh, U.K., August/September 1995 1996. Springer-Verlag Berlin Heidelberg.

[147] Y. Yang and S. Petrovic. A novel similarity measure for heuristic selection in examination timetabling. In E.K. Burke and M. Trick, editors, *Practice and theory of automated timetabling V: selected papers from the 5th international conference*, volume 3616 of *Lecture Notes in Computer Science*, pages 377–396. Springer-Verlag Berlin Heidelberg, 2005.

Publications

[1] M. Adriaen, P. De Causmaecker, P. Demeester, and G. Vanden Berghe. Tackling the university course timetabling problem with an aggregation approach. In *In Proceedings of the 6th International Conference on the Practice and Theory of Automated Timetabling (Patat 2006)*, pages 330–335, Brno, Czeck Republic, August 2006.

[2] B. Bilgin, P. Demeester, G. Vanden Berghe, and T. Wauters. A model and a hyperheuristic approach for automated assignment of patients to beds in a hospital. In *International Conference on Metaheuristics and Nature Inspired Computing (Meta 2008)*, Hammamet Tunisia, 2008.

[3] N. Custers, P. De Causmaecker, P. Demeester, and G. Vanden Berghe. Semantic components for timetabling. In *Selected Revised Papers of 5th International Conference on Practice and Theory of Automated Timetabling*, volume 2740 of *Lecture Notes in Computer Science*, pages 17–33. Springer-Verlag Berin Heidelberg, 2005.

[4] P. De Causmaecker, P. Demeester, Y. Lu, and G. Vanden Berghe. Using web standards for timetabling. In *Proceedings of 4th international conference on the Practice And Theory of Automated Timetabling (Patat 2002)*, pages 238–257, Gent, Belgium, August 2002.

[5] P. De Causmaecker, P. Demeester, and G. Vanden Berghe. Evaluation of the university course timetabling problem with the linear numberings method. In *25th Annual Workshop of the UK Planning and Scheduling Special Interest Group (UK PlanSIG 2006)*, pages 154–155, Nottingham, UK, December 2006.

[6] P. De Causmaecker, P. Demeester, and G. Vanden Berghe. A decomposed metaheuristic approach for a real-world university timetabling problem. *European Journal of Operational Research*, 195(1):307–318, May 2009.

[7] P. De Causmaecker, P. Demeester, G. Vanden Berghe, and B. Verbeke. Analysis of real-world personnel scheduling problems. In *Proceedings of*

the 5th International Conference on Practice and Theory of Automated Timetabling, pages 183–197, Pittsburgh, 2004.

[8] P. De Causmaecker, P. Demeester, G. Vanden Berghe, and B. Verbeke. Evaluation of employee rosters with the extended linear numberings method. In *24th Annual Workshop of the UK Planning and Scheduling Special Interest Group (UK PlanSIG 2005)*, pages 125–133, London, December 2005.

[9] P. Demeester, P. De Causmaecker, and G. Vanden Berghe. A tabu search algorithm for assigning patients to beds efficiently. In *Proceedings of 8th EU/MEeting on Metaheuristics in the Service Industry*, pages 86–89, Stuttgart, Germany, October 2007.

[10] P. Demeester, W. Souffriau, P. De Causmaecker, and G. Vanden Berghe. A hybrid tabu search algorithm for automatically assigning patients to beds. *Artificial Intelligence in Medicine*, 48(1):61–70, January 2010.

Submitted Papers

[1] B. Bilgin, P. Demeester, and G. Vanden Berghe. A hyperheuristic approach to the patient admission scheduling problem. Technical report, KaHo Sint-Lieven, 2008. submitted to Journal of Heuristics.

[2] P. Demeester, B. Bilgin, P. De Causmaecker, and G. Vanden Berghe. A hyperheuristic approach for real world and benchmark examination timetabling problems. Technical report, KaHo Sint-Lieven, 2009. submitted to Journal of Scheduling.

[3] P. Demeester, P. De Causmaecker, and G. Vanden Berghe. A hybrid guided local search approach to a university course timetabling problem. Technical report, KaHo Sint-Lieven, 2009.

Curriculum vitae

October 5, 1973	Born in Gent, Belgium
1991-1995	Master of Science in Mathematics, major Mathematical Physics and Astronomy at University Gent, Belgium
1995-1996	Gegradueerde in de Aanvullende Studies Informatica at University Gent, Belgium
1996-1998	Teaching Assistant at Department of Mathematical Physics and Astronomy at University Gent, Belgium
1998-2006	Researcher at Department of Information Technology, KaHo Sint-Lieven, Gent, Belgium
2006-present	Teaching/Research Assistant at Department of Information Technology, KaHo Sint-Lieven, Gent, Belgium

Arenberg Doctoral School of Science, Engineering & Technology
Faculty of Science
Computer Science Campus Kortrijk
Research group CODeS
Etienne Sabbelaan 53, 8500 Kortrijk

KATHOLIEKE UNIVERSITEIT LEUVEN

ASSOCIATIE K.U.LEUVEN