M.Sc. Thesis
Master of Science in Engineering

Group Project Exam Timetabling

at Roskilde University

Rasmus Ørnstrup Mikkelsen (s123915)
Kongens Lyngby 2018

DTU Management Engineering
Department of Management Engineering

DTU Management Engineering
Department of Management Engineering
Technical University of Denmark

Produktionstorvet Building 424
2800 Kongens Lyngby, Denmark
Phone +45 4525 4800
www.man.dtu.dk

Supervisor
Thomas Jacob Riis Stidsen

Summary

In this thesis, the Group Project Exam Timetabling problem at Roskilde University is introduced. The problem is very specific to Roskilde University and requires a lot of manual planning and resources every academic semester. The problem is previously unvisited and is therefore explained in depth. A Mixed Integer Programming (MIP) model is defined for the problem in two steps. First a model assuming all exams require the same amount of time is defined. Then the model is extended to allow for exams to have different time requirements. Additionally, multiple extensions are introduced to increase the model realism and to reduce complexity. However the model still proves too difficult to solve directly using standard solvers and instead the metaheuristic Adaptive Large Neighborhood Search (ALNS) is implemented to solve the problem.

A few extensions and adaptations compared to the original ALNS are implemented and tested on the problem. The implementation produces good results on the few available datasets. However, a problem with the acceptance criteria used in the heuristic is identified. Fixing this problem will allow for less "wasted" iterations and potentially lead to better performance. The best solutions identified using the metaheuristic are evaluated using a MIP defined to only look at single days, leading to slight improvements for two out of the five available datasets.

Finally some points of potential improvements are identified for both the defined model as well as the implemented solution method. Specifically the model can be improved to lessen the gap between model and reality, making it better suited for practical usage. Ideas for improving the performance of the implemented ALNS are also given as well as alternative solution methods using decomposition of the problem.

Preface

This master's thesis has been completed at the Department of Management Engineering at the Technical University of Denmark in fulfillment of the requirements for acquiring a master's degree in Industrial Engineering and Management. It accounts for a total of 30 ECTS points.

Kongens Lyngby, January 1, 2018

[illegible]

Rasmus Ørnstrup Mikkelsen (s123915)

Acknowledgments

I would like to thank my supervisor Thomas Jacob Riis Stidsen for his support and guidance throughout this thesis. I have enjoyed our regular meetings where he has provided me with fantastic sparring and invaluable feedback to my work. Thomas has shown great trust and interest in me and my work and I am very grateful for that. I would also like to thank Roskilde University and their staff for helping me to understand their problem and providing valuable data. A special thanks to Pernille Marie Storgaard and Hanne Tofteng for providing extra feedback and extremely helpful meetings. Furthermore I would like to thank Jørgen Lindgaard Pedersen for enlightening me on a censors' concerns regarding the exam planning at RUC. I am also very grateful to MaCom for allowing me to work in their offices and making me feel very welcome. Additionally, a special thank you goes to Kasper Torp Steffensen and Laura Hjerrild Andersen, whom both have given me great feedback on this report. Lastly thank you to Sara Jensen for her undying support. All have helped to enhance my work greatly.

Contents

Summary **i**

Preface **iii**

Acknowledgments **v**

Contents **vii**

1 Introduction **1**

2 Exam Timetabling **3**

2.1 Timetabling 3
2.2 General Exam Timetabling 4
2.3 Toronto Dataset 6
2.4 International Timetabling Competition 2007 7

3 Roskilde University **9**

3.1 Semester and Exams 10
3.2 Group Project Exam Planning 12

4 Group Project Exam Timetabling **15**

4.1 Mixed Integer Programming Model 15
4.2 Data 31
4.3 Model complexity 35
4.4 One Day Model 36

5 Solution Method **39**

5.1 Adaptive Large Neighborhood Search 39

6 Computational Tests **49**

6.1 Construction Algorithm 49
6.2 Parameter Tuning 50
6.3 Results 54

7 Discussion **67**

7.1 Modeling 67

7.2 Solution Method 70
7.3 Concluding Remarks 73

A The Linear Programming Relaxation **75**
A.1 The Linear Programming Relaxation 75

B ALNS Implementation **77**
B.1 Strict Greedy Schedule Repair Heuristic 78
B.2 Destroy and Repair Methods 79

C Parameter Tuning **81**
C.1 Parameter ξ and λ overview 81
C.2 Parameter Combination Performance Overviews 83
C.3 Reward Distribution 94

Bibliography **97**

CHAPTER 1 Introduction

Examination timetabling is an important and challenging administrative task which almost all academic institutes must go through on a regular basis. The planning problem can be very large and quit complex and is often performed manually by staff, which can be quite expensive and result in suboptimal exam timetables. Therefore methods that either help to or automatically finds "good" timetables are very beneficial. These can help to both save time and money but, perhaps more importantly, create exam timetables which work as well as possible for all parties involved.

Educational timetabling, such as class and exam timetabling, are very important optimization problems which have been the focus of a lot of research. This is evident by the number of publications in the area, as well as the establishment of conferences and international timetabling competitions. However, because many educational institutions have different wants and needs, it is difficult to make a one-size-fits-all solution. With regards to examination timetabling, some standardized problem definitions have been introduced. This allows for meaningful comparisons between solution methods. The two most important problem definitions are the Toronto Datasets and the International Timetabling Competition examination timetabling track. Both of these deal with exam timetabling, with the overarching goal being the creation of timetables such that all students can attend their exams.

In this thesis a specific examination timetabling problem is investigated, which is quit different from the aforementioned problem definitions. At Roskilde University (RUC) most students have a group project exam they must attend at the end of the semester. Each student is only part of one project group and the group project examination period is after the regular class examination period. Therefore the problem is not to construct feasible timetables that spread out students' exams, but instead timetables should be built such that they cater to the supervisors and co-examinators associated with the group project exams. Throughout this thesis, the Group Project Exam Timetabling problem at RUC is investigated and a mixed integer programming model developed. The problem is then solved using the metaheuristic Adaptive Large Neighborhood Search with some mathematical programming.

CHAPTER 2

Exam Timetabling

This chapter first briefly introduces the concept of timetabling in Section 2.1 before discussing general exam timetabling in Section 2.2. Exam timetabling is an area of much active research and the two most common problem definitions, the Toronto and ITC2007 datasets/formulations, are introduced in Section 2.3 and Section 2.4 respectively.

2.1 Timetabling

Within Operations Research timetabling problems have received a lot of attention and come in many varieties. A definition of *timetabling* is given in (Wren, 1995) as

"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" - Wren, 1995

Thereby many problems can be considered timetabling problems, for example public transportation timetables, some types of personnel allocation and educational timetabling such as class and exam allocation. In (Wren, 1995) an important distinction is made between timetabling and scheduling, where timetabling is considered as a special case of scheduling activities. A timetable shows when and/or where a specific event is to take place, but does not necessarily imply exactly what resources are allocated or a more specified plan. For example when making shift schedules for jobs without special competence requirements (all employees can complete the same work) a timetable may show how many employees are to be working at one time. But a schedule will show more details such as exactly which employees are assigned each shift and what work should be completed. In this thesis the verb *scheduling* is used synonymously with the act of *timetabling*.

Class and exam timetabling are very typical and important problems, as they need to be solved in almost all educational institutions around the world. With the increased availability of computing power at individual institutions, methods for solving these problems are becoming more and more applicable in the real world. However these problems are typically very complex and clever solution methods are required. A specific case of examination timetabling is the subject of this thesis and a general introduction is given in the following section.

2.2 General Exam Timetabling

Exam timetabling is the process of allocating a number of exams into a number of given timeslots while observing some constraints. This is a problem that needs to be solved at most academic institutions once per semester. The class structure and examination requirements can vary greatly for different institutions; resulting in examination timetabling problems of different complexity. In general, the more constraints are imposed on the problem, the more difficult it becomes to find not only a feasible solution but also a "good" solution.

Two different classes of constraints are typically defined for an exam timetabling problem; hard and soft constraints. Hard constraints are constraints which must be observed and are used to ensure that the timetable satisfies all physical and regulatory restrictions that may apply. A feasible solution is an examination assignment which does not violate any hard constraints. A typical hard constraint is that no students should attend exams overlapping in time. Soft constraints are used to measure the quality of a solution. Ergo a timetable is allowed to violate soft constraints but doing so incurs a cost. A smaller total cost indicates a better solution. A typical soft constraint is that students should have gaps of a certain size between their exams. Smaller gaps are allowable at a cost correlating to the size of the gap. The types of soft constraints vary greatly and sometimes even contradict each other.

An small timetabling example is shown where the objective is to find a feasible solution. There are seven students and six courses. Each student has between one and three courses as shown in Table 2.1. The goal is to find a timetable such that each student can take all of his or her exams. In this example there are three days and two separate rooms available. It is allowed for multiple exams to use the same room at a time. Room 1 and Room 2 have a capacity of three and four students respectively.

Course	Physics	Math	Chemistry	Biology	Statistics	Programming
Anders	x	x				
Birger	x	x				
Claus	x					
Dorthe		x	x			
Erik			x	x	x	
Freja				x		x
Gorm					x	x

Table 2.1: The students and their associated courses in the small examination timetabling example.

A feasible solution is shown in Table 2.2. If the statistics exam was to be held Thursday in Room 1, the timetable would be infeasible since Erik would have to be in Room 1 and Room 2 at the same time. This is a very small and simplistic example and, as stated before, many different constraints are usually enforced on the timetable.

	Room 1	Room 2	
Wedensday	Physics: A B C	Chemistry: D E	Programming: F G
Thursday	Math: A B D	Biology: E F	
Friday	Statistics: E G		

Table 2.2: A feasible timetable for the small examination timetabling example.

Many different approaches have been used for solving examination timetabling problems. Especially graph and network based solutions and heuristics are applied with great success. Graph based techniques view exams as vertices of a graph and hard constraints between exams are defined as edges between the vertices. Welsh and Powell (1967) showed the connection between the examination timetabling problem and the graph coloring problem; finding the minimum number of colors to cover all vertices of the graph such that no vertices of the same colors are adjacent. Many solution methods are based (at least partly) on this connection between the two problems. (Qu et al., 2009) is an excellent survey covering examination timetabling and provides a great overview of all important strides, sorted by solution method category, within examination timetabling research between 1996 and 2009.

Researches tackling exam timetabling sometimes focus on the practical problem at a specific institution; as for example done in (Kahar and Kendall, 2010). Since these problems be very different, and solution methods are tailored to specific problem variations, it can be difficult to compare performance of solution methods. Therefore some benchmarking data sets and problem definitions have been introduced and the most popular are the Toronto and the ITC2007 datasets. These are presented in the following sections.

2.3 Toronto Dataset

The Toronto datasets were introduced in (Carter et al., 1996) and have been greatly used in the examination timetabling research community ever since. The dataset consists of 13 real-world problems, where three are from Canadian high schools, five from Canadian universities, one from a British university, one from a Saudi Arabian university, and one from an American university. The only hard constraint of the problem, is that no student can be assigned multiple exams at the same time. This problem is uncapacitated meaning it does not take room capacity into account.

For each instance, a Conflict Matrix C is defined to indicate whether two exams i and j share students. Matrix element $c_{ij} = 1$ if exam i conflicts with exam j, and 0 otherwise. Note that this matrix is symmetrical. Also the Conflict Density is defined as the ratio between the number of 1-elements and the total number of elements in C.

Two problem objectives were defined for the original dataset: to minimize the number of timeslots needed or to minimize the average cost per student with a given number of timeslots. The latter variant has received the most attention by researchers (Qu et al., 2009, Table 6). This is likely because the problem is more difficult to solve and perhaps more useful in practice because the number of timeslots can be given as a parameter and institutions want to make fair exam timetables for students. Also for the first variant 12 out of 13 best known results were already obtained in (Carter et al., 1996), i.e. when the datasets were first introduced. This indicates that the problem already has been "solved" sufficiently well and therefore is not very interesting to examine further.

When minimizing the average cost per student the goal is to spread out conflicting exams as much as possible while staying within a certain number of timeslots. To capture this, the objective function is defined such that for each student sitting two exams s timeslots apart, a cost w_s is added depending on the size of the gap as follows: $w_1 = 16$, $w_2 = 8$, $w_3 = 4$, $w_4 = 2$ and $w_5 = 1$. For gaps of more than five timeslots there is no cost.

More variants have been introduced and researched, such as introducing capacities and changing the objective function to minimize the number of students sitting two exams in a row. However it is still today the latter of the two original problem definitions which is receiving the most research attention of all Toronto benchmark variants. Note that there are some problems associated these datasets which have been addressed and corrected in (Qu et al., 2009).

2.4 International Timetabling Competition 2007

In order to further research within timetabling the second International Timetabling Competition was held in 2007 (ITC2007). This competition was separated into three tracks where two focused on course timetabling and one on exam timetabling. The competition placed great emphasis on providing a 'real world' perspective to educational institution timetabling. Besides generating interest and comparing practical solutions, one of the greatest contributions of the ITC2007 was a set of benchmark datasets and problem definitions, which are more realistic than the Toronto datasets.

During the competition (McCollum et al., 2007) was used to communicate the rules of the Exam Timetabling track, and as such gives a very thorough description of both the problem and datasets.

The problem is made more realistic by including multiple aspects, conditions and quality measures compared to earlier models as well as allowing weighting of soft constraints. These weights are included as parameters in an attempt to make solutions more usable in practice since institutions may have different priorities. By including the weighting as parameters in the dataset, it is possible for an institution to 'play' with the settings until an timetable is found that best fits their specific needs.

In the ITC2007 each exam timetabling problem consists of defined examination periods, a set of exams, a set of students and their enrolled exams, a set of rooms and associated room capacities, hard constraints, soft constraints and weights for some of the soft constraints.

In the competition, constraints were separated into three categories; required, hard and soft constraints. Required hard constraints are constraints that "absolutely cannot be broken", hard constraint violations lead to non-zero "distance to feasibility" and soft constraints incur costs (as usual). The quality of the timetable is evaluated at two levels; the number of non-required hard constraints violated (total "distance to feasibility") and the weighted sum of soft constraint violations. This two level evaluation was used as a judging tool in the competition, but as the authors noted in (McCollum et al., 2007), "in general, gaining feasibility is not as important an issue as in some cases of course timetabling." If a solution violates a required hard constraint it does not get a score. The two required hard constraints are that every exam is allocated to at most one room and at most one period (exams cannot be split in time or space).

In order for a timetable to be considered feasible all hard constraints have to be satisfied. One of the hard constraints is the usual constraint of not having any students assigned to multiple exams which are held at the same time. Notice that it is explicitly allowed for multiple exams to use the same room at the same time, but it is not allowed for exams to be split into multiple rooms. Another hard constraint is that period lengths may not be violated. There are also a number any period or room related hard constraints must be observed, some examples are that a specific exam must follow another or that an exam must use a specific room. Lastly, room capacities must be observed for each room.

The soft constraints can be split into two groups; those related to either specific resources or global settings. The resource soft constraints are period and room related, where penalties can be associated with using specific periods or rooms. This allows for institutions to better control how their resources are used. Global soft constraints include penalizing whenever students have two exams in a row, on the same day, or their exams have a specific spread (as in the Toronto problem). Since exams are allowed to share rooms a penalty is applied when exams of different duration are assigned to the same room within the same period. The last soft constraint is to penalize the number of 'large' exams which appear 'later' in the timetable. What constitutes 'large' and 'late' is specified in the dataset as user defined parameters.

In (McCollum et al., 2007) the authors provide a non-linear mathematical formulation of the problem.

CHAPTER 3

Roskilde University

Roskilde University is a university located in Roskilde, Denmark. The university was founded in 1972 with a mission to “challenge academic traditions and to experiment with new ways to acquire knowledge” [1]. As such, RUC is very project and problem oriented in their teaching techniques and structure.

The university consists of four departments (see Table 3.1), with a total of around 8,800 students and 950 employees[2]. Both Bachelors and Masters programs are offered resulting in a total of 208 unique programs. A Bachelors program spans three years, with the first year consisting of a “basic section” where key interdisciplinary theories and methods are introduced. Then the last two years are spent on specializing within one or two subjects. A Masters program requires two years and also combines two main subjects. The first two semesters (first year) are focused on each individual subject. The third semester is an “interdisciplinary semester” where the two subjects are combined and lastly the fourth semester consists of writing a thesis.

On all semesters (except when writing a thesis) the aim is to distribute the work load such that 50% is courses related and 50% is spent on project work. In the project students from different disciplines work together to find a combined solution for a specific problem.

In this chapter the general semester and exam structure at RUC is described along with some of the associated planning activities.

English “Department of”	Danish “Institut for”	Abbreviation
Communication and Arts	Kommunikation og Humanistisk Videnskab	IKH
People and Technology	Mennesker og Teknologi	IMT
Science and Environment	Naturvidenskab og Miljø	INM
Social Sciences and Business	Samfundsvidenskab og Erhverv	ISE

Table 3.1: The names of the four departments at RUC.

[1]https://ruc.dk/en
[2]https://ruc.dk/om-roskilde-universitet

3.1 Semester and Exams

All programs follow the same general semester structure and since RUC places a lot of focus on project oriented knowledge creation, the semester consists of two concurrent elements; regular courses and group projects. The temporal division of each element is illustrated Figure 3.1. All non-thesis writing students therefor follow classes while simultaneously working on a group project with up to five other students. However, all courses and associated exams are finished before the group project is handed in and defended at the group project exam.

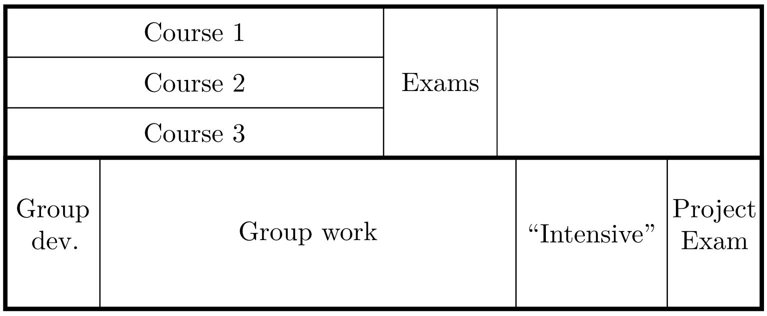

Figure 3.1: The general semester structure at RUC.

3.1.1 Courses

Each student will typically have two to three courses that often are held parallel to each other, but it is possible for courses to start or end at different times throughout the semester. The course element of the semester is always concluded with an examination period in which exams are held. Students may be evaluated entirely on projects and assignments during the semester, purely on an exam in the examination period or a mixture of both. Three exam types are used in the examination period: take-home, written and oral examinations. The take-home exams consists of an assignment which must be individually completed within one, two or up to 14 days. Oppositely both written and oral exams take place at the university. Typically written exams are held such that all students following the same course attend the course exam in the same room at the same time. Oral exams are also held at the university, but students take the exam individually. For oral exams the examinator (course lecturer) and censor (co-examinator with no relation to the course/project) have to be present and evaluate students immediately. This is not the case for written or take-home exams, where exams are sent to the examinator and censor, whom evaluate the exams separately before agreeing on a final grade for each student. This grading process may take weeks.

All course exams are fixed before the beginning of the semester. The exams are scheduled (manually) such that students following a recommended study plan are guaranteed to be able to take all their exams. If problems should arise for individual student following a recommended study plan, planners will ensure that they are given a replacement exam at some other time. Conflicts for students not following a recommended study plan are, in principal, the students responsibility; however planners will try to help, but no guarantees are given. Evidently the course examination timetabling at RUC is very similar to the standard exam timetabling problem definitions.

3.1.2 Group Project

The group project can be split into four phases; group development, normal group work during the semester, the "intensive" which is when all courses are finished and students work exclusively on the project, and finally a group project exam.

Each study line offers multiple relevant topics and groups of up to six students are assembled based on students personal preferences. On later semesters these projects become more interdisciplinary and students from different study lines need to work together on the project. A supervisor, whose responsibility is to provide assistance and guidance when necessary, is assigned to each group. A supervisor may be assigned to multiple groups working with different topics.

The group project ends with an exam at the very end of the semester. Since every student is in at most one group project, they will at most have one group project exam. The exam is very similar to typical oral exams with the exception that all group members are present throughout. First students together present their project, with everyone presents a small part individually for three to five minutes. Then the examinator and censor pose questions either to the group or to individual students. Afterwards the group exits the room while the supervisor and censor agree upon individual grades for each group member. The group either receives their grade separately or collectively; whichever the group prefers. Depending on the department, 20-30 minutes are given to each student, which specifies the amount of time required for the exam.

The examinator is the groups supervisor and the censor is either a colleague from RUC (internal) or from another university or from the industry (external). On earlier student semesters it is allowed to use internal censors, while on later semesters, especially for master students, external censors are required. When using internal censors the group supervisor usually is allowed to choose the censor. For groups requiring an external censor different rules apply as described in the next section.

All group projects are required to be handed in by a specific date at the end of the semester. In the months leading up to this date, RUC staff start timetabling project exams. Both supervisor and censor is likely to have multiple exams and it is therefore important to avoid conflicts. Since students are at most attending one group project exam, and should not have course exams during the same time period, it is generally not possible for exam conflicts with respect to students. However in rare cases there may be student related exam conflicts. These are described in Section 4.2.1.2.

It is possible that project groups split-up during the semester. Even though a project group separates, both parts usually continue working on the same project with the same supervisor. Therefore both groups can still use the same censor and even though they require two separate exams, they can be considered as one exam for planning purposes, since one exam can simply follow the other immediately. This is especially useful if the initial group exam has already been planned, as no major changes have to be made to the timetable as a whole.

In case a group does not turn in their project, planners will try to amend the timetable to avoid undesirable consequences. This is often very difficult as many of the exams are relatively fixed since they have already been agreed upon by supervisors and censors, requiring some effort to reschedule.

3.2 Group Project Exam Planning

In the months leading up to the final group project deadline, planners attempt to timetable the group project exams. Censors are secured through very different means depending on the program. All censors belong to a *censor-corps*, which is a collection of all authorized censors within a given field. Each censor-crops can have very different ways of distributing censors to exams. Usually planners make a request for censors for a number of exams. In some censor-corps individual censors are allowed to bid on exams and in others each censor has little influence and is assigned to exams by the censor-corps. With some corps the planner can make censor request and with others this is not allowed. Many censor-corps try to distribute censors evenly such that each censor within the corps has about the same amount of exams every year.

In general the university does not have the power to decide which censor is assigned to what exams; except when an internal censor is used. RUC is moving towards using internal censors more. This may be because these censors typically can be secured more quickly, communication is easier and they are less costly. However, this means that more RUC personal will be used for exams during the same time period, making the production of good examination timetables even more difficult.

Once the censor is secured for an exam, the planner communicates with the censor to find a number of dates on which the censor is available. Typically four to five days are given. Planners have access to the supervisors calendars and will schedule the exam on one of the proposed days where the supervisor is available.

Each planner is responsible for a set of group project exams and collect all needed data about each project group and associated exam in an Excel document. This document is made at a department level and as a consequence four separate documents are created. The contents of this document is described in greater detail in Section 4.2. Producing this Excel document poses a lot of challenges and frustrations for planners (whose details are beyond the scope of this project). Suffice it to say that a lot of man-hours are spent every semester producing an overview of all group project exams for each of the four departments. While gathering the information planners also book rooms for exams at times that fit both the supervisor and censor. However, they do not always have access to information about exams planned by other planners (within the same or a different department) and it can therefore be difficult to make good timetables; both because of the problem size and complexity but also because of lack of information and time. Also, planners make timetables for their department, and it is possible that supervisors and censors are involved with exams in other departments. Therefore it is highly likely that there are shared persons between department plans. RUC is also aiming for using more internal censors, which will increase the number of shared resources and the number of exam conflicts. This only increases the difficulty in making a “good” unified timetable.

The quality of group project examination timetable is evaluated mainly on the individual exam schedules for external censors. This is because these censors may live far from, and have little to no familiarity with, RUC campus. Thus planners aim for producing schedules such that any given censor has all of his/her exams in one day, with as little idle time and as few room changes as possible. It is better to have supervisors, whom are very familiar with the campus, move between rooms and buildings for other exams. Additionally RUC staff have offices at RUC and in case of long wait times likely have other work they can do. Although internal censors also work at RUC, planners also try to optimize the examination timetable for them.

CHAPTER 4

Group Project Exam Timetabling

The goal of group project exam timetabling problem at RUC is different from the typical examination timetabling problem definitions. The goal is not to make fair examination timetables for individual students; instead focus is placed on the censors.

This chapter further examines the group project exam timetabling problem at RUC. In Section 4.1 a Mixed Integer Programming (MIP) model is defined for the problem. Then in Section 4.2 all available and generated data is discussed. The model complexity is investigated in Section 4.3. Lastly, Section 4.4 introduces a version of the model concerning only a single day.

4.1 Mixed Integer Programming Model

Using fixed timeslots is very typical when timetabling exams; even when done manually. Often planners may only schedule exams such that they start every half hour or every quarter of an hour. Using fixed timeslots can become a problem if there is a mismatch in the timeslot and exam length. For example, if only allowed to assign exams to timeslots spanning two hours, and one exam requires just 30 minutes, then a big (and likely unwanted) gap is very probable in the schedule. However, when allowing for exams to span multiple timeslots, this problem can be circumvented by defining each timeslot sufficiently small. This can held make the timetable more "smooth" but decreasing timeslot length directly increases problem size.

Every exam is assumed to have exactly one supervisor (examinator) and one censor already determined. Since both supervisor and censor may have other exams they need to attend, it is not allowed for conflicting exams (exams sharing supervisor, censor or both) to be scheduled at the same time. Also room capacities must be observed such that there is enough room for both the project group as well as supervisor and censor.

This section defines the MIP model for the group project exam timetabling problem at RUC through a few steps. First a model is defined under the assumption that all exams require exactly one timeslot. This model is then extended to handle exams having varying timeslot requirements. Lastly the model is further extended to increase the applicability of produced timetables.

4.1.1 Single Timeslot Model

Here a MIP model is defined for the problem under the assumption that all exams require the same amount of time, i.e. one timeslot. No consideration is given as to how long each timeslot is but it is assumed that the same amount of timeslots is available on all days. The sets, parameters and variables defined for the Single Timeslot Model are shown in Table 4.1.

Sets	**Description**
$c \in C$	The set of censors
$s \in S$	The set of supervisors
$i \in E$	The set of exams
$i \in E_c$	The set of exams assigned to censor c
$c \in C^{>1}$	The set of censors with more than one exam (i.e. $C^{>1} = \{c \in C : \lvert E_c \rvert > 1\}$)
$r \in R$	The set of rooms
$d \in D$	The set of days
$t \in T$	The set of timeslots
Parameters	**Description**
N_c	The number of exams for censor c
S_i^E	The size of exam i (number of students)
S_r^R	The capacity of room r
ν_{ij}	1 if exam pair (i, j) are conflicting, 0 otherwise
t_{last}	The last timeslot available each day
p^{RS}	The penalty for violating room stability
p^{CG}	The penalty for having timeslot gaps for a censor
p^{CMD}	The penalty for assigning a censor on multiple days
Variables	**Description**
$x_{rdt}^i \in \mathbb{B}$	1 if exam i is assigned to room r on day d in timeslot t
$u_{cr} \in \mathbb{B}$	1 if censor c is assigned to room r
$v_{cd} \in \mathbb{B}$	1 if censor c is assigned to day d
$f_{cd} \in \mathbb{Z}_+$	The identifier of the first timeslot used by censor c on day d
$l_{cd} \in \mathbb{Z}_+$	The identifier of the last timeslot used by censor c on day d

Table 4.1: The sets, parameters and decision variables used in the Single Timeslot model.

All sets can be considered as sets of integers starting at zero, eg. $D = \{0, \dots, |D|-1\}$. With all relevant sets, parameters and decision variables defined in Table 4.1 the constraints of the model are written.

$$\sum_{r \in R} \sum_{d \in D} \sum_{t \in T} x^i_{rdt} = 1 \quad \forall i \in E \tag{4.1}$$

$$\sum_{i \in E} x^i_{rdt} \leq 1 \quad \forall r \in R, d \in D, t \in T \tag{4.2}$$

$$\sum_{r \in R} \left(x^i_{rdt} + x^j_{rdt} \right) \leq 1 \quad \forall i, j \in E : \nu_{ij} = 1, d \in D, t \in T \tag{4.3}$$

$$x^i_{rdt} = 0 \quad \forall i \in E, r \in R, d \in D, t \in T : S^E_i + 2 > S^R_r \tag{4.4}$$

$$f_{cd} \leq \sum_{r \in R} \sum_{t \in T} \left(t \cdot x^i_{rdt} \right) + t_{last} \left(1 - \sum_{r \in R} \sum_{t \in T} x^i_{rdt} \right) \quad \forall c \in C^{>1}, i \in E_c, d \in D \tag{4.5}$$

$$l_{cd} \geq \sum_{r \in R} \sum_{t \in T} t \cdot x^i_{rdt} \quad \forall c \in C^{>1}, i \in E_c, d \in D \tag{4.6}$$

$$l_{cd} \geq f_{cd} \quad \forall c \in C^{>1}, d \in D \tag{4.7}$$

$$\sum_{i \in E_c} \sum_{d \in D} \sum_{t \in T} x^i_{rdt} \leq N_c \cdot u_{cr} \quad \forall c \in C^{>1}, r \in R \tag{4.8}$$

$$\sum_{i \in E_c} \sum_{r \in R} \sum_{t \in T} x^i_{rdt} \leq N_c \cdot v_{cd} \quad \forall c \in C^{>1}, d \in D \tag{4.9}$$

$$\tag{4.10}$$

Constraint (4.1) ensures that every exam is held exactly once. Constraint (4.2) ensures that at most one exam can be held in a room in any given timeslot. Constraint (4.3) ensures that no conflicting exams (exams sharing censor, supervisor or both) are scheduled at the same time. Since the conflict matrix ν is symmetrical only the upper triangle is defined to avoid generating identical constraints. Constraint (4.4) enforces the observation of room capacities by fixing the exam allocation decision variable to zero for infeasible exam and room combinations. Note that a room has to have capacity for both the students of the group as well as both censor and examinator. Altogether constraints (4.1) - (4.4) ensure that a feasible timetable is found. Constraints (4.5) - (4.9) set the values of f_{cd}, l_{cd}, u_{cr} and v_{cd} respectively; which are used to evaluate the quality of the produced timetable.

Variables f_{cd} and l_{cd} are used to respectively capture the first and last timeslot in which a censor is scheduled on each day. If the censor is not scheduled on a given day, then the associated variables should attain the same value. Constraint (4.1) ensures that exactly one x^i_{rdt} variable is set to 1 for every exam. Therefore $\sum_{r \in R} \sum_{t \in T} x^i_{rdt}$ for any combination of exam i and day d will either be 0 or 1. Considering a censor with just one exam, constraint (4.5) enforces that f_{cd} must be less than or equal to the timeslot value of the last timeslot on all days where the censors' exam is not scheduled. On the day where the censor is scheduled, f_{cd} is forced to be less than or equal to the timeslot in which the censor has an exam. Now considering a censor with multiple exams, this constraint forces f_{cd} to be less than or equal to the first timeslot in which the censor c is scheduled on day d. The l_{cd} variable is set using constraints (4.6) and (4.7). The first constraint ensures, that if the censor is scheduled on a given day, l_{cd} must be equal to or greater than the value of the last timeslot in which the censor is assigned. The model rewards negative value for the expression $l_{cd} - f_{cd}$ (see (4.13)), which should not be possible as this means a censor on a given day has their last timeslot before their first. On days where censor c has no exams, $l_{cd} \geq 0$ and $f_{cd} \leq t_{last}$, which allows for negative values to the shown expression. By enforcing that $l_{cd} \geq f_{cd}$ (constraint (4.7)) this problem is negated. The model rewards (or penalizes less) f_{cd} and l_{cd} values that are close to each other, so on days where censor c has exams f_{cd} is pushed up and l_{cd} is pushed down. On days where the censor has no exams, $f_{cd} \leq t_{last}$ and $l_{cd} \geq f_{cd} \geq 0$ and therefore these variables are allowed to attain the same value and incur no penalty.

Variables u_{cr} and v_{cd} are set using constraints (4.8) and (4.9) respectively. In both constraints u_{cr} and v_{cd} are multiplied by N_c since this is the largest amount of exams that censor c can be assigned in total, and therefore also the limit for any given room or day.

Notice that the constraints for setting f_{cd}, l_{cd}, u_{cr} and v_{cd}, as well as the variables themselves, are only defined for censors whom have more than one exam, since no penalty can be applied if this is not the case.

In constraints (4.5) and (4.6) all parts are integer. Therefore the variables definitions for f_{cd} and l_{cd} can both be relaxed to be non-negative real numbers since they automatically attain integer values regardless.

This problem is a multi-objective optimization problem since the quality of a solution is evaluated on three separate objectives: room stability, time gaps and number of required days. As noted above, these are only defined for censors whom have more than one exam, since each penalty is an evaluation of the placement of multiple exams associated with the same censor. The objective function is composed of three separate terms, each capturing one aspect of the solution. There are different techniques for handling multi-objective optimization and here a weighted penalty function is used.

The u_{cr} variable is used to capture which rooms are used by each censor. For a given censor c the total number of room changes is given by $\sum_{r \in R} (u_{cr}) - 1$ and the total number of room changes (or inversely room stability **RS**) considering all potential penalty inducing censors is given by

$$RS = \sum_{c \in C^{>1}} \left(\sum_{r \in R} (u_{cr}) - 1 \right) \tag{4.11}$$

To register which days censor c is scheduled the variable v_{cd} is used. Since the goal is to avoid censors being assigned to multiple days, this is very similar to the room stability term of the objective function. The part of the objective function used to penalize when censors are scheduled on multiple days (**CMD**), i.e. the number of days more than one, is then

$$CMD = \sum_{c \in C^{>1}} \left(\sum_{d \in D} (v_{cd}) - 1 \right) \tag{4.12}$$

To count the number of censor timeslot gaps variables f_{cd} and l_{cd} are used. The number of timeslots where the censors would be present at RUC (including gaps) is given by $\sum_{d \in D} (l_{cd} - f_{cd} + v_{cd})$. Variable v_{cd} is added since on days where the censor is scheduled, $l_{cd} - f_{cd}$ is one less than the number of timeslots the censors is present (because the censor is present throughout the last timeslot). For days where the censor is not scheduled, $f_{cd} = l_{cd}$ and $l_{cd} - f_{cd} + v_{cd} = 0$. Since each exam requires exactly one timeslot, the total number of timeslot in which a censor c is at RUC, but not attending exams, is given by $\sum_{d \in D} (l_{cd} - f_{cd} + v_{cd}) - N_c$. When considering all censors, the total number of censor gaps (**CG**) is given by

$$CG = \sum_{c \in C^{>1}} \left(\sum_{d \in D} (l_{cd} - f_{cd} + v_{cd}) - N_c \right) \tag{4.13}$$

The objective function is then the weighted summation of (4.11), (4.12) and (4.13) with the corresponding penalty parameter.
The full model is shown below.

Single Timeslot Model

Minimize: $p^{RS}RS + p^{CMD}CMD + p^{CG}CG$

Subject to:

$$\begin{array}{ll}
\sum_{i \in E} x^i_{rdt} \leq 1 & \forall r \in R, d \in D, t \in T \\
\sum_{r \in R} \sum_{d \in D} \sum_{t \in T} x^i_{rdt} = 1 & \forall i \in E \\
\sum_{r \in R} \left(x^i_{rdt} + x^j_{rdt} \right) \leq 1 & \forall i, j \in E : \nu_{ij} = 1, d \in D, t \in T \\
x^i_{rdt} = 0 & \forall i \in E, r \in R, d \in D, \\
 & t \in T : S^E_i + 2 > S^R_r \\
f_{cd} \leq \sum_{r \in R} \sum_{t \in T} \left(t \cdot x^i_{rdt} \right) + t_{last} \left(1 - \sum_{r \in R} \sum_{t \in T} x^i_{rdt} \right) & \forall c \in C^{>1}, i \in E_c, d \in D \\
l_{cd} \geq \sum_{r \in R} \sum_{t \in T} t \cdot x^i_{rdt} & \forall c \in C^{>1}, i \in E_c, d \in D \\
l_{cd} \geq f_{cd} & \forall c \in C^{>1}, d \in D \\
\sum_{i \in E_c} \sum_{d \in D} \sum_{t \in T} x^i_{rdt} \leq N_c \cdot u_{cr} & \forall c \in C^{>1}, r \in R \\
\sum_{i \in E_c} \sum_{r \in R} \sum_{t \in T} x^i_{rdt} \leq N_c \cdot v_{cd} & \forall c \in C^{>1}, d \in D \\
x^i_{rdt} \in \mathbb{B} & \forall i \in E, r \in R, d \in D, t \in T \\
u_{cr} \in \mathbb{B} & \forall c \in C^{>1}, r \in R \\
v_{cd} \in \mathbb{B} & \forall c \in C^{>1}, d \in D \\
f_{cd}, l_{cd} \in \mathbb{R}_+ & \forall c \in C^{>1}, d \in D
\end{array}$$

4.1.2 Multiple Timeslots Model

In order to change the model to be able to handle exams that span multiple timeslots a few changes and additions need to be made. Furthermore, to make the model more inline with RUC's wishes, a few model extensions are introduced. The new set, parameters and decision variables needed are shown in their introduced order in Table 4.2.

Set	**Description**
E_i^{ν}	The set of exams conflicting with exam i
Parameters	**Description**
N_i^E	The number of required timeslots for exam i
M_c	The total number of timeslots censor c needs to be assigned; $M_c = \sum_{i \in E_c} N_i^E$
U_{id}^E	1 if exam i is unavailable on day d
U_{cd}^C	1 if censor c is unavailable on day d
U_{rdt}^R	1 if room r is unavailable on day d in timeslot t
p^D	The penalty for using a day
d_{last}	The last day on which timetabling is possible
p^L	The penalty for each late timeslot
t_{late}	The timeslot value of the first timeslot designated late
Υ_{rdt}^i	1 if exam i is unavailable in room r on day d in timeslot t
Variables	**Description**
$y_{rdt}^i \in \mathbb{B}$	1 if exam i begins in room r on day d in timeslot t
$b_d \in \mathbb{B}$	1 if day d is used
$w_i \in \mathbb{Z}_+$	The number of late timeslots used by exam i
$x_{rdt}^i \in \mathbb{B}$	1 if exam i is scheduled in room r on day d in timeslot t

Table 4.2: The new parameters, set and decision variables introduced in the Multiple Timeslots Model with extensions.

Every exam must be held for N_i^E coherent timeslots on the same day and only in one room. Therefore if exam i begins in room r on day d in timeslot t, this room will be occupied by exam i for the following $N_i^E - 1$ timeslots.

The model is changed such that the main decision variable y_{rdt}^i sets the start timeslot of each exam (decision variable x_{rdt}^i is reintroduced in Section 4.1.2.1). This requires all constraints to be updated; some with only minor changes. The updated constraints are:

$$\sum_{r\in R}\sum_{d\in D}\sum_{t\in T} y^i_{rdt} = 1 \quad \forall i \in E \tag{4.14}$$

$$\sum_{i\in E}\sum_{t'=t-N^E_i+1}^{t} y^i_{rdt'} \leq 1 \quad \forall r \in R, d \in D, t \in T \tag{4.15}$$

$$\sum_{r\in R}\left(y^i_{rdt} + \sum_{t'=t}^{t+N^E_i-1} y^j_{rdt'} \right) \leq 1 \quad \forall i,j \in E : \nu_{ij} = 1, d \in D, t \in T \tag{4.16}$$

$$f_{cd} \leq \sum_{r\in R}\sum_{t\in T}\left(t \cdot y^i_{rdt}\right) + t_{last}\left(1 - \sum_{r\in R}\sum_{t\in T} y^i_{rdt}\right) \quad \forall c \in C^{>1}, i \in E_c, d \in D \tag{4.17}$$

$$l_{cd} \geq \sum_{r\in R}\sum_{t\in T}(t + N^E_i - 1)\cdot y^i_{rdt} \quad \forall c \in C^{>1}, i \in E_c, d \in D \tag{4.18}$$

$$l_{cd} \geq f_{cd} \quad \forall c \in C^{>1}, d \in D \tag{4.19}$$

$$\sum_{i\in E_c}\sum_{d\in D}\sum_{t\in T} y^i_{rdt} \leq N_c \cdot u_{cr} \quad \forall c \in C^{>1}, r \in R \tag{4.20}$$

$$\sum_{i\in E_c}\sum_{r\in R}\sum_{t\in T} y^i_{rdt} \leq N_c \cdot v_{cd} \quad \forall c \in C^{>1}, d \in D \tag{4.21}$$

$$y^i_{rdt} = 0 \quad \forall i \in E, r \in R, d \in D, t \in T : S^E_i + 2 > S^R_r \tag{4.22}$$

Constraint (4.14) ensures that every exam begins and therefore is held. Constraint (4.15) ensures that at most one exam takes place in a room at any given time. Assume variable x^i_{rdt} still indicates that exam i is scheduled in the given room, day and timeslot combination. Then using the start of exam i variable y^i_{rdt} the x^i_{rdt} variables can be set by $\sum_{t'=t-N^E_i+1}^{t} y^i_{rdt'} = x^i_{rdt} \forall i \in E, r \in R, d \in D, t \in T$. This is then substituted into the old constraint (4.2) to get constraint (4.15). To ensure conflicting exams are not scheduled at the same time constraint (4.16) is used. Looking across all rooms, this constraint ensures that from the start of exam i, the conflicting exam j cannot begin until exam i if finished. The constraints for setting variables f_{cd} and l_{cd} ((4.17) - (4.18)) are very similar to the Single Timeslot model, with the only change being being in constraint (4.19) to reflect the value of the last timeslot used by exam i. The constraints for setting u_{cr}, v_{cd} and observing room capacity ((4.20), (4.21) and (4.22) respectively) are the same as before with variable x^i_{rdt} substituted by y^i_{rdt}.

Additionally the calculation of censor gaps needs to be updated to reflect the potentially increased number of required timeslots (due to multiple timeslots per exam) for each censor. This is done by exchanging the N_c parameter with M^C_c such that censor gaps are counted by

$$CG = \sum_{c \in C^{>1}} \left(\sum_{d \in D} (l_{cd} - f_{cd} + v_{cd}) - M_c \right) \tag{4.23}$$

Lastly to ensure feasibility, it must be enforced that an exam is only started if there is sufficient time for it to be completed before the end of the day. This is done by fixing y^i_{rdt} variables for exam and timeslot combinations where this is not the case, by the introduction of the following constraint.

$$y^i_{rdt} = 0 \qquad \forall i \in E, r \in R, d \in D, t \in T : t + N^E_i - 1 > t_{last} \tag{4.24}$$

4.1.2.1 Extensions

The described changes could define the Multiple Timeslots Model. However, a few extensions and improvements are introduced to both increase the realism and applicability of produced timetables as well as to decrease the size of the model.

Exam and room unavailability When timetabling exams it is very likely that rooms and persons involved are unavailable at certain times. These unavailabilities impose some extra constraints on when and where exams can be scheduled.

It is likely that supervisors and censors have certain days, that they are not able to attend exams. As they both need to be present for their designated exams, these exams cannot be held on days where either one is not available. Additionally, there may be other reasons for not allowing certain exams to be scheduled on specific days; for example in the case that students have made an extraordinary agreement with planners. A combination of all these aspects can be used to define exam unavailabilities, such that an exam can only be scheduled on days where censor, supervisor and students are available.

Rooms unavailabilities are considered in smaller time intervals where a room is allowed to be unavailable in specific timeslots. This type of unavailability is introduced since it is likely that some exam rooms may be used for other events, like meetings, throughout a day. As opposed to person related unavailabilities, it is more likely that a room is unavailable for only part of the day.

A more detailed description of exam and room unavailability is given in Section 4.2.1.

Given the parameter definitions seen in Table 4.2 the following constraints are introduced in order to fix y^i_{rdt} variables where necessary.

$$y^i_{rdt} = 0 \quad \forall i \in E, r \in R, d \in D, t \in T : U^E_{id} = 1 \tag{4.25}$$

$$y^i_{rdt} = 0 \quad \forall i \in E, r \in R, d \in D, t \in T : U^R_{rdt} = 1 \tag{4.26}$$

$$y^i_{rdt} = 0 \quad \forall i \in E, r \in R, d \in D, t \in T : U^R_{r,d,t+N^E_i-1} = 1 \tag{4.27}$$

Constraint (4.25) ensures that an exam cannot be scheduled on a day where it is not available. Constraints (4.26) and (4.27) enforce that an exam can only be assigned a room that is available and only if the exam can be completed before the room becomes unavailable.

Minimization of days Instead of having to administrate exams throughout the whole examination period, it may be desirable to minimize the number of days on which exams are held. However, these days should not be spread out sporadically but should be somewhat collected. Additionally, RUC would like the exams to be scheduled towards the end of the examination period in order to give students ample time to prepare for their exams.

The binary variable b_d is used to indicate which days are used in the the timetable. The constraint for setting this variable is as follows:

$$\sum_{i \in E} \sum_{r \in R} \sum_{t \in T} y^i_{rdt} \leq |E| b_d \quad \forall d \in D \tag{4.28}$$

Variable b_d is multiplied by $|E|$ since this is the largest amount of exams that can be scheduled on any single day. Practically it would rarely be possible to scheduled all exams in one day, but $|E|$ offers an guaranteed limit. The number of days (**ND**) on which exams are scheduled, is given by

$$ND = \sum_{d \in D} b_d$$

and the objective function is amended to include this extra goal of minimizing days by adding the following term

$$p^D ND$$

In order to ensure coherency of scheduled days as well as pushing these days towards the end of the examination period, the following constraint is introduced.

$$\sum_{i \in E} \sum_{r \in R} \sum_{t \in T} y^i_{rdt} \leq |E| \sum_{i \in E} \sum_{r \in R} \sum_{t \in T} y^i_{r,d+1,t} \quad \forall d \in D \setminus d_{last} \tag{4.29}$$

This constraint makes it so it is only possible to schedule exams on days where the following day also has one or more exams scheduled. The exception is of course the last available day. These constraints force the model to use the last day, only use coherent days and rewards using as few days as possible. The combined effect is pushing the timetable back while also compressing it.

Penalize late exams In general neither supervisors, censors or students are interested in having exams in the late afternoon or at night. Therefore the model is amended to penalize whenever timeslots are used that are defined as "late".

The w_i variable counts the number of late timeslots used by exam i and is set using the constraint

$$w_i \geq \sum_{r \in R} \sum_{d \in D} \sum_{t \in T} y^i_{rdt} \cdot \left(t + N^E_i - t_{late}\right) \qquad \forall i \in E \tag{4.30}$$

Notice that the w_i variables are non-negative and from this constraint definition can only attain integer values. The variable definition can therefore be relaxed to non-negative real numbers.
The number of late timeslots (**NLT**) is then calculated as

$$NLT = \sum_{i \in E} w_i \tag{4.31}$$

and the objective function is updated to include the following term

$$p^L NLT \tag{4.32}$$

In this project a timeslot is designated late if it start at 17:00 or later.

Notice that the minimization of days and late timeslots has a different focus than the previously defined objective terms. These two additions handle more general timetable characteristics while the others are censor specific.

Break when switching rooms In the current model it is allowed to have censors and supervisors assigned to exams directly following each other in time but in different rooms. In practice this is not feasible since some transportation time between rooms is to be expected. RUC would like to enforce a break whenever a censor or supervisor has to change rooms.

This is done by considering conflicting exams, since exams are conflicting if they share censor and/or supervisor. Thereby conflicting exams cannot be scheduled directly after each other, unless they are assigned to the same room. This is enforced by introducing the following constraint:

$$\sum_{j \in E^{\nu}_i} \sum_{r' \in R : r' \neq r} y^j_{r',d,t+N^E_i} \leq |E^{\nu}_i| \cdot (1 - y^i_{rdt}) \qquad \forall i \in E, r \in R, d \in D, t \in T \tag{4.33}$$

where $|E^{\nu}_i|$ is the total number of exams conflicting with exam i.

In the case that $y^i_{rdt} = 1$, this constraint enforces that the sum of exams conflicting with exam i assigned to any room besides room r in the timeslot just after exam i has ended must be zero. If $y^i_{rdt} = 0$ then no actual limit is imposed. This is because two exams conflicting with exam i are not necessarily conflicting with each other and could therefore potentially be allowed to be scheduled simultaneously. Setting the right side of the constraint to $|E^{\nu}_i|$ is simply the smallest upper bound available.

4.1.2.2 Tightening

Here some improvements are introduced to reduce model complexity by reformulation of the conflicting exams constraint, reducing unnecessary constraint generation, including additional variable fixing and finally reducing the total number of non-zeros of the model by reintroducing the x^i_{rdt} variable to the model.

Conflicting exams constraint The constraint included to ensure no conflicting exams are scheduled at the same time, is defined for each pair of conflicting exams for all day and timeslot combinations. Instead of considering each conflict pair individually, the constraint can be defined for each exam and consider all of its conflicting exams at the same time.

$$\sum_{r\in R}\sum_{j\in E^\nu_i}\sum_{t'=t}^{t+N^E_i-1} y^j_{rdt'} \leq |E^\nu_i| \cdot \left(1 - \sum_{r\in R} y^i_{rdt}\right) \quad \forall i \in E, d \in D, t \in T \tag{4.34}$$

This constraint functions in much the same way as (4.33). If $y^i_{rdt} = 1$ looking across all rooms, then the sum of exams conflicting with i starting in any room within the duration of i must be zero. However, if $y^i_{rdt} = 0$ then no limit is enforced with the same argumentation that exams conflicting with exam i are not guaranteed mutually conflicting.

As long as the number of exam conflicts is greater than the number of exams, this change will result in less constraints needing to be generated. This is the case for all available datasets (see Section 4.2).

In Section 7.1 it is described how this constraint could be tightened even further, although it is not done in this project.

Reducing constraint definitions and additional variable fixing Constraints should only be defined when they are necessary for the model. Using a priori knowledge it is possible to avoid generating constraints that have no effect.

In this model the main decision variable y^i_{rdt} is forced to be zero in order to observe room capacities, room unavailabilities, exam unavailabilities and to ensure exams are not assigned if it cannot be completed before a room becomes unavailable or before the end of the day.

Variables f_{cd}, l_{cd} and v_{cd} are used in different objective terms and using censor unavailabilities (U^C_{cd}) these variables can also be fixed when they are guaranteed to be zero. The variable fixing conditions are summarized below in the aforementioned order.

$$\begin{aligned}
y^i_{rdt} &= 0 \quad \forall i \in E, r \in R, d \in D, t \in T : S^E_i + 2 > S^R_r \\
y^i_{rdt} &= 0 \quad \forall i \in E, r \in R, d \in D, t \in T : U^R_{rdt} = 1 \\
y^i_{rdt} &= 0 \quad \forall i \in E, r \in R, d \in D, t \in T : U^E_{id} = 1 \\
y^i_{rdt} &= 0 \quad \forall i \in E, r \in R, d \in D, t \in T : U^R_{r,d,t+N^E_i-1} = 1 \\
y^i_{rdt} &= 0 \quad \forall i \in E, r \in R, d \in D, t \in T : t + N^E_i - 1 > t_{last} \\
f_{cd} &= 0 \quad \forall c \in C, d \in D : U^C_{cd} = 1 \\
l_{cd} &= 0 \quad \forall c \in C, d \in D : U^C_{cd} = 1 \\
v_{cd} &= 0 \quad \forall c \in C, d \in D : U^C_{cd} = 1
\end{aligned}$$

Thereby y^i_{rdt} should be fixed when any y^i_{rdt} fixing condition is met. For simplicity Υ is defined such that it denotes any combination of i, r, d and t where any of these conditions are met and y^i_{rdt} should be fixed.

$$\Upsilon^i_{rdt} = 1 \quad \forall i \in E, r \in R, d \in D, t \in T :$$
$$S^E_i + 2 > S^R_r \vee U^R_{rdt} = 1 \vee U^E_{id} = 1 \vee U^R_{r,d,t+N^E_i-1} = 1 \vee t + N^E_i - 1 > t_{last}$$

Constraint (4.33), which is defined for all i, r, d and t is only defined for combinations not found in Υ, as it has no effect when y^i_{rdt} cannot attain a value of 1.

In the same way, all constraints defined for exam i and day d or for censor c and day d need only be defined when $U^E_{id} = 0$ and $U^C_{cd} = 0$ respectively. The same goes for the one exam per room constraint, which only needs to be defined when the room is available, i.e. $U^R_{rdt} = 0$. Lastly, the new constraint ensuring that conflicting exams are not held simultaneously, and the constraint enforcing breaks, are only defined for exams that actually have conflicting exams, i.e. $|E^\nu_i| > 0$. Including these conditions in constraint generation helps to avoid generating unnecessary constraints. With restricted constraint definitions for constraints used for setting f_{cd}, l_{cd} and v_{cd} it is important that these variables are fixed for censor and day combinations where corresponding variable setting constraints are not defined; otherwise these variables are free to attain incorrect values.

Reducing number of non-zeros Because the model is defined such that the primary decision variable denotes the start of an exam, and not whether or not the exam is present, some of the constraints as currently defined have to sum over timeslots to detect the presence of an exam. One consequence is that these constraints become more complex and have many non-zero coefficients, which drastically increases the size of the model and causes problems when implemented in modeling software.

In an attempt to counter this problem, the variable x^i_{rdt} is reintroduced as a non-negative continuous variable with an upper bound of 1. As mentioned in the beginning of this section (page 22), the x^i_{rdt} variable can be set using the constraint

$$\sum_{t'=t-N_i^E+1}^{t} y^i_{rdt'} = x^i_{rdt} \qquad \forall i \in E, r \in R, d \in D, t \in T \tag{4.35}$$

Notice that the definition of x^i_{rdt} can be relaxed to non-negative real numbers if an upper bound of 1 is imposed.

Additionally the x^i_{rdt} variables should be fixed for when exams and rooms are unavailable as well as for exam and room combinations where room capacity is not observed.

The x^i_{rdt} variables can then be substituted into constraints to remove the summation over timeslots to detect y^i_{rdt} variables. These constraints are the one exam per room constraint (4.15) and the conflicting exams not simultaneously constraint (4.34). These are are now defined respectively

$$\sum_{i \in E} x^i_{rdt} \leq 1 \quad \forall r \in R, d \in D, t \in T : U^R_{rdt} = 0 \tag{4.36}$$

$$\sum_{j \in E_i^\nu} \sum_{r \in R} x^j_{rdt} \leq |E_i^\nu| \cdot \left(1 - \sum_{r \in R} x^i_{rdt}\right) \quad \begin{array}{l} \forall i \in E : |E_i^\nu| > 0, \\ d \in D : U^E_{id} = 0, t \in T \end{array} \tag{4.37}$$

The full model for multiple timeslots including all discussed extensions is shown below.

Tightened Multiple Timeslots

Minimize: $$p^{RS}RS + p^{CG}CG + p^{CMD}CMD + p^{D}ND + p^{L}NLT$$

Subject to:

$$\sum_{r\in R}\sum_{d\in D}\sum_{t\in T} y^i_{rdt} = 1 \qquad \forall i \in E$$

$$\sum_{t'=t-N_i^E+1}^{t} y^i_{rdt'} = x^i_{rdt} \qquad \forall i \in E, r \in R, d \in D, t \in T$$

$$\sum_{i\in E} x^i_{rdt} \le 1 \qquad \forall r \in R, d \in D, t \in T : U^R_{rdt} = 0$$

$$\sum_{j\in E_i^\nu}\sum_{r'\in R: r'\neq r} y^j_{r',d,t+N_i^E} \le |E_i^\nu| \cdot \left(1 - y^i_{rdt}\right) \qquad \forall i \in E : |E_i^\nu| > 0, r \in R, d \in D, t \in T : \Upsilon^i_{rdt} = 0$$

$$\sum_{j\in E_i^\nu}\sum_{r\in R} x^j_{rdt} \le |E_i^\nu| \cdot \left(1 - \sum_{r\in R} x^i_{rdt}\right) \qquad \forall i \in E : |E_i^\nu| > 0, d \in D : U^E_{id} = 0, t \in T$$

$$f_{cd} \le \sum_{r\in R}\sum_{t\in T}\left(t \cdot y^i_{rdt}\right) + t_{last} \cdot \left(1 - \sum_{r\in R}\sum_{t\in T} y^i_{rdt}\right) \qquad \forall c \in C^{>1}, i \in E_c, d \in D : U^E_{id} = 0$$

$$l_{cd} \ge \sum_{r\in R}\sum_{t\in T}\left(t + N_i^E - 1\right) \cdot y^i_{rdt} \qquad \forall c \in C^{>1}, i \in E_c, d \in D : U^E_{id} = 0$$

$$l_{cd} \ge f_{cd} \qquad \forall c \in C^{>1}, d \in D : U^C_{cd} = 0$$

$$\sum_{i\in E_c}\sum_{d\in D}\sum_{t\in T} y^i_{rdt} \le N_c \cdot u_{cr} \qquad \forall c \in C^{>1}, r \in R$$

$$\sum_{i\in E_c}\sum_{r\in R}\sum_{t\in T} y^i_{rdt} \le N_c \cdot v_{cd} \qquad \forall c \in C^{>1}, d \in D : U^C_{cd} = 0$$

$$w_i \ge \sum_{r\in R}\sum_{d\in D}\sum_{t\in T} y^i_{rdt} \cdot (t + N_i^E - t_{late}) \qquad \forall i \in E$$

$$\sum_{i\in E}\sum_{r\in R}\sum_{t\in T} y^i_{rdt} \le |E| b_d \qquad \forall d \in D$$

$$\sum_{i\in E}\sum_{r\in R}\sum_{t\in T} y^i_{rdt} \le |E| \sum_{i\in E}\sum_{r\in R}\sum_{t\in T} y^i_{r,d+1,t} \qquad \forall d \in D \setminus d_{last}$$

$$x^i_{rdt} \le 1 \qquad \forall i \in E, r \in R, d \in D, t \in T$$

Tightened Multiple Timeslots - Continued

$$
\begin{array}{ll}
y^i_{rdt} = 0 & \forall i \in E, r \in R, d \in D, t \in T : \Upsilon^i_{rdt} = 1 \\
x^i_{rdt} = 0 & \forall i \in E, r \in R, d \in D, t \in T : S^E_i + 2 > S^R_r \\
x^i_{rdt} = 0 & \forall i \in E, r \in R, d \in D, t \in T : U^R_{rdt} = 1 \\
x^i_{rdt} = 0 & \forall i \in E, r \in R, d \in D, t \in T : U^E_{id} = 1 \\
f_{cd} = 0 & \forall c \in C^{>1}, d \in D : U^C_{cd} = 1 \\
l_{cd} = 0 & \forall c \in C^{>1}, d \in D : U^C_{cd} = 1 \\
v_{cd} = 0 & \forall c \in C^{>1}, d \in D : U^C_{cd} = 1 \\
y^i_{rdt} \in \mathbb{B} & \forall i \in E, \forall r \in R, \forall d \in D, \forall t \in T \\
x^i_{rdt} \in \mathbb{R}_+ & \forall i \in E, \forall r \in R, \forall d \in D, \forall t \in T \\
u_{cr} \in \mathbb{B} & \forall c \in C^{>1}, \forall r \in R \\
v_{cd} \in \mathbb{B} & \forall c \in C^{>1}, \forall d \in D \\
f_{cd} \in \mathbb{R}_+ & \forall c \in C^{>1}, \forall d \in D \\
l_{cd} \in \mathbb{R}_+ & \forall c \in C^{>1}, \forall d \in D \\
b_d \in \mathbb{B} & \forall d \in D \\
w_i \in \mathbb{B} & \forall i \in E
\end{array}
$$

4.2 Data

Data has been made available by RUC in the form of the established group project exam plan for June of 2017. A separate plan is made in an Excel document for each of the four departments independently of each other. Each plan is formatted in the same manner with every row being associated with an individual exam and data for each exams arranged in columns. Each column retains data for a specific aspect of the exam, such as the responsible planner, names of students, title of the group project and so on. Much of the data is of a practical nature needed by planners to ensure all formalities are taken care of. Columns of especial importance for this project are described here.

Each group project is assigned an unique **group exam number**, which is used for reporting grades in the central study administrative system. Since this number is unique it is used as an identifier for each exam. The **number of students** in the group is also recorded and used to define the exam time requirement. As described in Section 3.1.2, between 20 and 30 minutes are given per student for the exam. Although it does not fit exactly for all exams in the established plans, 30 minutes for each student of the group seems to be a good estimate of the general examination duration. Therefore, in this project, an assumption of 30 minutes per student is used to calculate the total time required by each exam. The **supervisor** and **censor** for each exam is also noted and is used to generate an overview of which exams each censor must attend and which exams are in conflict with each other. Lastly, the room, day, start and end time for each exam is given. These data are used to determine how many days the timetable spanned, how many rooms were used and to get a better understanding of the time requirement of exams. The capacity of each room is unavailable, but is estimated by taking the largest group (number of students) using the room and adding two (supervisor and censor).

Staff at RUC would like timetabling to be done in 15 minute intervals and therefore timeslots are defined accordingly. Since each exam is assumed to require 30 minutes per student, the total number of timeslots for each exam is given as the number of students multiplied by two. If it would be beneficial to change the timeslots to span less time, this can easily be implemented by multiply the number of required timeslot for each exam with the ratio of 15 minutes to the new timeslot length. For example in the case that timeslots should span 5 minutes, the new number of required timeslots for each exam is calculated by multiplying the current number of timeslots by $15/5 = 3$. It is important to ensure that only integer amounts of timeslot are needed, either by the choice of timeslot length or by rounding. However, planning in intervals of less than 15 minutes seems superfluous.

As mentioned, data for each of the departments is available separately as a consequence of current planning practices. However, it would be interesting to be able to plan all exams across all departments at once. Therefore all data has been collected into one dataset designated “ALL”. While combining all data, special care is put into ensure that there are no inconsistencies regarding people names (for example a person missing a middle name in one dataset) such that exam conflicts are observed correctly.

An overview of each of the dataset sizes is shown in Table 4.3. For all datasets it is assumed that exams can be scheduled from 8:00 to 19:00. This results in 45 timeslots each day.

Attribute	IKH	IMT	INM	ISE	ALL
$\|C\|$	164	124	43	158	469
$\|C^{>1}\|$	128	79	20	120	339
$\|S\|$	133	107	45	108	259
$\|E\|$	510	313	78	515	1,416
$\sum_{i,j \in E} \nu_{ij}$	2,053	1,128	87	2,937	7,126
$\|R\|$	60	25	18	41	141
$\|D\|$	20	19	10	20	20
$\|T\|$	45	45	45	45	45

Table 4.3: Set sizes and total number of conflicts for each dataset.

4.2.1 Generated Data

Most data used in modeling can be extracting from the available plans (with minor assumptions). But no data is directly accessible on the unavailability of rooms or persons involved with the exams. This data is generated as discussed here.

4.2.1.1 Room Unavailability

Room unavailabilities are considered on a timeslot basis, but for simplicity three unavailability patterns are used. Rooms can either be unavailable for the first half, second half or for a whole day. Unavailability is assigned by going through all room and day combinations and assigning a pattern at random. For room-day each combination there is a 10% chance for each of the unavailability patterns to be assigned and thus 70% chance for no unavailabilities.

RUC usually have rooms reserved for group project exams within the examination period. It is confirmed by RUC, that the number of fully available days for these rooms is approximately 70%. An overview of room unavailabilities is shown in Table 4.4

Unavailability pattern	IKH	IMT	INM	ISE	All
First half	9.67%	9.05%	10.56%	10.24%	9.75%
Second half	8.75%	9.26%	8.33%	10.73%	10.39%
All day	10.00%	9.26%	11.67%	10.61%	9.75%
No unavailability	71.58%	72.42%	69.44%	68.41%	70.11%
Total % unavailable	19.22%	18.41%	21.14%	21.09%	19.82%

Table 4.4: Percent distribution of generated room unavailability patterns for all room/day combinations for each dataset. Total % unavailable denotes the total percent of timeslots made unavailable due to room unavailability.

4.2.1.2 Exam Unavailability

In practice, planners schedule exams by looking at the calendars of supervisors to determined when they are available and offer about three to five of these days to potential censors. Therefore, currently in practice, each exam has only very few days on which it can be held and days on which the supervisor is unavailable are not even considered.

To generate exam unavailabilities simulating the current practice, a random number between three and five (inclusive) is assigned to each censor. This is the number of days on which a censor can attend an exam. It is assumed that a supervisor is available on all days. Randomly chosen days are marked available for each censor observing the randomly assigned limit. Each exam is then only available on days where the assigned censor is available. Thereby exam unavailability U^E_{id} is directly decided by censor unavailability U^C_{cd}.

Some censors are also supervisors and are therefore also expected to be flexible regarding examination days. Just as all other supervisors, these censors are assumed to be available on all days which is observed when generating data. An overview of exam unavailabilities is shown in Table 4.5.

Exam unavailabilities can also be used to ensure practical feasibility in the rare case that students have other exams scheduled in or around the group project examination period. If students have other exams, then their group project exam should be marked unavailable on those days. In order to ensure students have some time to prepare for their exams, the group exam could also be set as unavailable in a couple of days around their other exam. It is rather rare that this happens and therefore the randomly generated exam unavailability data does not consider student unavailability.

Set	IKH	IMT	INM	ISE	All
Total exam/day combinations	10,200	5947	780	10,300	28,320
% unavailable	61.38%	65.33%	58.85%	59.15%	62.60%

Table 4.5: Percent of exam/day combinations which are unavailable for each dataset.

4.2.2 Dataset conflicts

If it is the case that each department uses no shared resources (rooms or persons), then the group project exam timetabling could be handled separately at each department without any consideration of the others. In Table 4.6 an overview of the number of rooms and persons used in each of the department plans is shown. This also includes the number of conflicts; ie. rooms or people whom are used in multiple department plans. A total of 144 rooms are used where 141 of them are only used in one department and three are used in multiple. The same numbers are presented for censors, supervisors and “unique persons” - people whom are both censor and supervisor and are only counted once. Therefore “unique persons” is not the sum of censors and supervisors. A total 751 unique people are included in the group project exam timetable across all departments, where 60 (7.99%) of them appear in multiple timetables. This is too many people conflicts to ignore and it is not possible to timetable exams for each department completely independently of the others.

A more detailed overview of person conflicts is shown in Table 4.7. This table shows how many persons are shared between individual department plans. Notice that this does not sum to 60 as some people are cause for multiple conflicts. The least conflicting department timetable is INM, which is not surprising considering it is by far the smallest dataset.

	IKH	IMT	INM	ISE	Unique	Conflicts
Rooms	60	25	18	41	141	3
Censors	164	124	43	158	469	20
Supervisors	133	102	45	108	352	36
Unique Persons	249	211	87	204	691	60

Table 4.6: Overview of number of rooms and individual people used in each department timetable and number of conflicts.

	IKH	IMT	INM	ISE
IKH	-	37	0	21
IMT	37	-	6	35
INM	0	6	-	0
ISE	21	35	0	-

Table 4.7: Overview of person related conflicts between each department timetable.

4.3 Model complexity

The model complexity is investigated by examining the number of constraints and variables defined in the model for each of the datasets. The results are shown in Table 4.8. The model is especially large for the ALL dataset with more than 38 million variables and 84 million constraints.

The model is implemented in the mathematical modeling software GAMS 24.4.2 and CPLEX is used to solve the model. The variables fixing constraints are not counted in the total number of constraint because in the GAMS the variables are fixed using the `.FX` attribute and the the CPLEX option `holdfixed = 1`. This makes CPLEX consider all fixed variables as constants and they are therefore not counted as variables either.
Since the models are so large they require a lot of memory to build. With 128GB available memory it is only possible to build the model on the IMT, INM and ISE datasets; the two remaining dataset require more memory to build.

Including the x^i_{rdt} variables in the model does increase the total number of variables by a substantial amount. However, it allows for reducing the total number of non-zeros in the model. Without the x^i_{rdt} variables, the model for the IMT dataset has 220,508,831 non-zeros and with x^i_{rdt} variables 169,377,710. The difference is especially apparent on the ISE dataset, which can only be built when the x^i_{rdt} variables are included; requiring more than 128GB memory when not. When the x^i_{rdt} variables are included, the model for the ISE dataset has a total of 973,493,849 non-zeros.

For models of this size, even solving the Linear Programming (LP) relaxation can be very difficult. This is investigated in Appendix A. However, since it is not possible to solve (or even build) the models on some datasets, solution methods relying on solving the whole model at once are not very promising.

Variables	IKH	IMT	INM	ISE	ALL
y^i_{rdt}	5,630,386	1,369,735	139,231	4,738,964	38,621,482
x^i_{rdt}	6,635,853	1,369,735	161,607	5,533,258	45,165,170
u_{cr}	7,680	1,975	360	4,920	47,799
v_{cd}	1,099	539	78	1,092	2,798
f_{cd}/l_{cd}	1,099	539	78	1,092	2,798
b_d	20	19	10	20	20
w_i	510	313	78	515	1,146
Total	12,277,746	2,743,394	301,520	10,280,953	38,678,841
Constraints	IKH	IMT	INM	ISE	ALL
setX	6,635,852	1,577,129	161,606	5,533,258	45,165,169
everyExamHeld	510	313	78	515	1,416
oneExamPerRoom	43,622	17,438	6,388	29,117	101,754
breakIfRoomChange	5,602,797	1,330,829	119,309	4,728,636	38,335,043
conflictingExams	176,489	90,157	12,239	189,000	472,769
setF / setL	3,638	1,824	222	3,679	9,449
LGreaterThanF	1,099	539	78	1,092	2,798
setU	7,680	1,975	360	4,920	47,799
setV	1,099	539	78	1,092	2,798
setW	510	313	78	515	1,416
setB	20	19	10	20	20
pushExamsBack	19	18	9	19	19
fixY	21,909,614	5,320,640	492,568	14,264,536	141,068,918
fixX	20,904,147	5,113,245	470,193	13,470,241	134,525,230
fixF /fixL / fixV	1,461	962	122	1,308	3,982
Total	12,476,973	3,022,917	300,432	10,495,542	84,149,899

Table 4.8: The number of constraints and non-fixed variables for each dataset. The "Total" row for constraints does not include fixing constraints.

4.4 One Day Model

In this section a MIP model is defined for considering only a single day. This has multiple potential uses and is used as part of the solution method as explained in Section 5.1.7.

The model is defined for a fixed day d, under the assumption that a non-empty set of exams $E_d \subseteq E$ is to be scheduled on day d; observing exam unavailability. Given the exams to be scheduled, the set of censors to be scheduled $C_d \subseteq C$ is defined as all censors associated with exams in E_d. The set of exams associated with censor c on day d is denoted E_{cd}. The set of censors whom have multiple exams on day d is denoted $C_d^{>1} \subseteq C_d$. The number of timeslots that censor c must be assigned on day d is then $M_{cd} = \sum_{i \in E_c \cap E_d} N_i^E$. The set of exams conflicting with exam i on day d is denoted E_{id}^{ν}.

When considering a single day only three variable objective terms should be included: censor gaps, room stability and number of late timeslots. Since $E_d \neq \emptyset$ then day d must have exams scheduled and thereby the cost p^D is guaranteed to incur and is therefore added to the objective function. Variables are no longer defined for d because d is fixed and the subscript is therefore removed. The daily objective function terms are calculated as:

$$\mathbf{RS}_d = \sum_{c \in C_d^{>1}} \left(\sum_{r \in R} (u_{cr}) - 1 \right)$$

$$\mathbf{CG}_d = \sum_{c \in C_d^{>1}} (l_c - f_c + 1 - M_{cd})$$

$$\mathbf{NLT}_d = \sum_{i \in E_d} w_i$$

It is assumed that all exams in E_d observe exam unavailability and as such U_{id}^E is unnecessary and therefore removed. Now Υ is defined as

$$\Upsilon_{rdt}^i = 1 \quad \forall i \in E_d, r \in R, t \in T :$$

$$S_i^E + 2 > S_r^R \vee U_{rdt}^R = 1 \vee U_{r,d,t+N_i^E-1}^R = 1 \vee t + N_i^E - 1 > t_{last}$$

Constraints for setting v_{cd} and b_d and for pushing exams back are removed as they are not necessary when only considering one day. All other constraints are amended where needed.

It is important to notice, that room stability (**RS**) is measured across days which imposes some cross-day dependencies and restrictions. It is chosen to avoid this problem by ensuring, that a solution from the One Day Model cannot worsen the room stability objective, by enforcing that censors can only use the rooms which they are already assigned that day. This restriction guarantees that a solution to the One Day Model cannot increase the total room stability cost. Parameter γ_{crd} is used to indicated that censor c uses room r on day d by setting it to 0, and 1 otherwise. This parameter is then used for fixing of u_{cr} variables.

The One Day Model is seen below.

One Day Multiple Timeslots

Minimize: $p^{RS}RS_d + p^{CG}CG_d + p^{L}NLT_d + p^{D}$

Subject to:

$$\sum_{r\in R}\sum_{t\in T} y^i_{rt} = 1 \qquad \forall i \in E_d$$

$$\sum_{t'=t-N^E_i+1}^{t} y^i_{rt'} = x^i_{rt} \qquad \forall i \in E_d, r \in R, t \in T$$

$$\sum_{i\in E_d} x^i_{rt} \le 1 \qquad \forall r \in R, t \in T : U^R_{rdt} = 0$$

$$\sum_{j\in E^\nu_i \cap E_d}\sum_{r'\in R: r'\neq r} y^j_{r',t+N^E_i} \le |E^\nu_i| \cdot \left(1 - y^i_{rt}\right) \qquad \forall i \in E_d : |E^\nu_{id}| > 0, r \in R, t \in T : \Upsilon^i_{rdt} = 0$$

$$\sum_{j\in E^\nu_i}\sum_{r\in R} x^j_{rt} \le |E^\nu_i| \left(1 - \sum_{r\in R} x^i_{rt}\right) \qquad \forall i \in E_d : |E^\nu_i| > 0$$

$$f_c \le \sum_{r\in R}\sum_{t\in T}\left(t \cdot y^i_{rt}\right) + t_{last} \cdot \left(1 - \sum_{r\in R}\sum_{t\in T} y^i_{rt}\right) \qquad \forall c \in C^{>1}_d, i \in E_{cd}$$

$$l_c \ge \sum_{r\in R}\sum_{t\in T}\left(t + N^E_i - 1\right) \cdot y^i_{rt} \qquad \forall c \in C^{>1}_d, i \in E_{cd}$$

$$l_c \ge f_c \qquad \forall c \in C^{>1}$$

$$\sum_{i\in E_{cd}}\sum_{t\in T} y^i_{rt} \le N_c \cdot u_{cr} \qquad \forall c \in C^{>1}_d, r \in R$$

$$w_i \ge \sum_{r\in R}\sum_{t\in T} y^i_{rt} \cdot (t + N^E_i - tl_{first}) \qquad \forall i \in E_d$$

$$x^i_{rt} \le 1 \qquad \forall i \in E_d, r \in R, t \in T$$

$$y^i_{rt} = 0 \qquad \forall i \in E_d, r \in R, t \in T : \Upsilon^i_{rdt} = 1$$

$$x^i_{rdt} = 0 \qquad \forall i \in E, r \in R, d \in D, t \in T : S^E_i + 2 > S^R_r$$

$$x^i_{rdt} = 0 \qquad \forall i \in E, r \in R, d \in D, t \in T : U^R_{rdt} = 1$$

$$x^i_{rdt} = 0 \qquad \forall i \in E, r \in R, d \in D, t \in T : U^E_{id} = 1$$

$$u_{cr} = 0 \qquad \forall c \in C^{>1}_d, r \in R : \gamma_{crd} = 1$$

$$y^i_{rt} \in \mathbb{B} \qquad \forall i \in E_d, r \in R, t \in T$$

$$x^i_{rt} \in \mathbb{R}_+ \qquad \forall i \in E_d, r \in R, t \in T$$

$$u_{cr} \in \mathbb{B} \qquad \forall c \in C^{>1}_d, r \in R$$

$$f_c \in \mathbb{R}_+ \qquad \forall c \in C^{>1}_d$$

$$l_c \in \mathbb{R}_+ \qquad \forall c \in C^{>1}_d$$

$$w_i \in \mathbb{B} \qquad \forall i \in E_d$$

CHAPTER 5

Solution Method

For very difficult problems it may not be possible to use standard solvers or current mathematical techniques to find optimal solutions. Because of the model complexity it is decided to focus on a programmatic solution approach using a metaheuristic. In this chapter the metaheuristic design and implementation is described.

5.1 Adaptive Large Neighborhood Search

In this project the Adaptive Large Neighborhood Search (ALNS) metaheuristic is applied to the problem. ALNS was first introduced in (Ropke and Pisinger, 2006) for solving the *pickup and delivery problem with time windows*; an NP-hard special case of the traveling salesman problem. This metaheuristic is very robust and good at navigating problems with a very large and constrained solution space. The general ALNS framework is described in section 5.1.1. Section 5.1.3 provides a description of the implemented destroy and repair methods. Then in sections 5.1.5 and 5.1.6 more details about the design and implementation of the ALNS heuristic is given.

5.1.1 General ALNS

The ALNS metaheuristic is an extension of the Large Neighborhood Search (LNS) introduced in (Shaw, 1998). LNS is a heuristic that iteratively creates new solutions through the use of a destroy and repair method. A destroy method removes part of the solutions and the repair method attempts to rebuild the solution. Destroy methods often include a stochastic element, to ensure the search is moved in new directions each iteration. Repair methods are often greedy heuristics, but it is possible to implement more complex methods. However, care should be taken to ensure that the repair methods often rebuild feasible solutions. The destroy method can remove a large part of the solution and thus, in combination with the repair method, defines a very large neighborhood to examine.

Instead of using the same destroy and repair method each iteration (as in LNS), the ALNS heuristic chooses in a probabilistic manner a destroy and repair method from a set of predefined methods. Once both methods have been applied, the resulting solution is evaluated and the probability of choosing the used methods again is adjusted based on performance. That is, methods that perform well become more likely to be chosen in subsequent iterations and oppositely poor performing methods become less likely to be chosen. The combination of multiple destroy and repair methods define a larger neighborhood than in the LNS by allowing for multiple solution neighborhoods. The probability system is then used to dynamically guide the heuristic towards exploring neighborhoods which have previously been rewarding and to avoid neighborhoods that have not.

Algorithm 1 General ALNS

```
1: Generate feasible solution $x$
2: Set weights $\rho^- = (1, 1, \dots, 1), \rho^+ = (1, 1, \dots, 1)$
3: Set $x^b = x$
4: while stop criterion not met do
5:     Select destroy method $d \in \Omega^-$ using $\rho^-$
6:     Select repair methods $r \in \Omega^+$ using $\rho^+$
7:     Apply destroy and repair methods $x^t = r(d(x))$
8:     if accept$(x^t, x)$ then
9:         $x = x^t$
10:    end if
11:    if $cost(x^t) < cost(x^b)$ then
12:        $x^b = x^t$
13:    end if
14:    Update $\rho^-$ and $\rho^+$
15: end while
16: return $x^b$
```

Pseudocode for the ALNS heuristic is shown in Algorithm 1. Three variables are used in the algorithm to store solutions; x^b, x and x^t. The best solution found is x^b, x is the current solution and x^t is the temporary solution generated from applying a destroy method followed by a repair method. The set of destroy and repair methods is denoted Ω^- and Ω^+ respectively. Destroy methods are denoted $d \in \Omega^-$ and each has an associated weight ρ^-. This is the same for each repair method $r \in \Omega^+$ with associated weight ρ^+.
The heuristic initially generates a feasible solution x and initializes the best seen solution $x^b = x$. Each destroy and repair method is given the same initial weight of 1. The algorithm runs for as long as a given stop criterion is not met; typically either time or number of iterations. In each iteration a destroy and repair method is picked using roulette wheel selection. The probability ϕ_i^- for selecting destroy method i is calculated as

$$\phi_i^- = \frac{\rho_i^-}{\sum_{k=1}^{|\Omega^-|} \rho_k^-}$$

and likewise the probabilities for the repair methods.
The chosen destroy and repair method is applied to the current solution to give the temporary solution x^t. This solution is then either kept or discarded pending an *accept* function. One possibility is to only accept improving solutions but another option is

to used the acceptance criterion from the metaheuristic *Simulated Annealing* as done in (Ropke and Pisinger, 2006). The latter option is chosen in this project (see Section 5.1.5). If the temporary solution is better than the previously best observed solution, then x^b is updated. Lastly, the weights of the used destroy and repair methods are updated using the different levels of rewards:

ω_1:	solution is new global best
ω_2:	solution is accepted and better than current solution
ω_3:	solution is accepted and worse than current solution
ω_4:	solution is rejected

The weights are then updated using the following formula

$$\rho^- = \lambda\rho^- + (1-\lambda)\omega$$
$$\rho^+ = \lambda\rho^+ + (1-\lambda)\omega$$

where ω is the reward given the quality of the temporary solution and $\lambda \in [0,1]$ is the *decay* parameter which controls how sensitive the weights are to changes in performance. If the decay parameter is large (close to 1), then more weight is given to the previous performance of the method than the reward of the current iteration. The aim of the adaptive weight adjustments is to make the heuristic robust by allowing it to select weights that work well for the given instance of the problem being solved. Methods that find new and improving solutions should of course be rewarded (ω_1 and ω_2). Reward ω_3 is given to methods that result in solutions that, even though they are not improving, help to bring the search forward. In the case of using the simulated annealing acceptance criterion these are solutions that are not “too bad” compared to the current solution. Other options are also possible, like in (Ropke and Pisinger, 2006) where non-improving but not previously unseen solutions are rewarded ω_3. Methods that result in rejected solutions are “punished” by receiving the lowest value ω_4.

5.1.2 Delta Evaluation

The most computationally expensive part of applying a metaheuristic to solve problems is evaluation of solutions (Stidsen and Reinhardt, 2016). Therefore delta evaluation is used to reduce the time consumption by considering changes instead of reevaluating whole solutions.

In the Group Project Exam Timetabling problem at RUC the quality of a solution is measured on five separate objectives. Three objectives are strictly censor related; censor room stability, censor wait time (gaps) and number of assigned days for each censor. Another objective is the number of late timeslots used, which is counted for each exam. The last objective is the total number of days used in the schedule. The total timetable score is a weighted summation of these separate objectives.
In this implementation, the total contribution of each censor (on the three separate

objectives), each exam and each day is stored. This is beneficial because it allows for easily identifying censors and exams whom incur costs and rank them based on cost. When destroying and repairing the solution, the contribution of affected censors, exams and days is adjust and changes are added to the total cost. An additional benefit of delta evaluation, is that changes can easily be undone since scheduling and unscheduling exams can be done in reverse to return to the previous solution.

5.1.3 Destroy and Repair Methods

In total 12 destroy and two repair methods are implemented. Some destroy methods are very similar to each other, with some only varying in their degree of destruction. In the following all implemented methods are described and method overviews are given in Table B.1 and B.2 i Appendix B.2.

5.1.3.1 Destroy Methods

The implemented destroy methods are designed such that some are meant for intensifying and other for diversifying the search through randomness. Additionally, care is put into ensuring a varying degree of destruction, such that the adaptive layer of the ALNS can adjust how aggressive the search is.

Unschedule Random Exams This is the most simple destroy method employed, which simply removes n randomly picked exams from the schedule. Three instances of the destroy method is included; to each a random number is passed in the interval $[3, 10]$, $[11, 25]$, and $[26, 40]$ respectively.

Unschedule Day This destroy method unschedules all exams scheduled on a given day. It is included in three different versions; one that unschedules a random day, one that unschedules the day with the least amount of scheduled exams and one that unschedules the first day used.

Unschedule Room used by Censor with most CMD This method targets the censor with the highest CMD cost by looking at the rooms and days on which this censor is scheduled. All exams scheduled in these room/day combinations are unscheduled.

Unschedule Censor with most CMD Unschedules all exams for the censors with the highest CMD cost. The number of censors to unschedule is picked at random in the interval $[1, 5]$.

Unschedule Costly Censors Unschedules all exams for the censors with the greatest contribution to the total cost. The number of censors to unschedule is picked at random in the interval $[1, 5]$.

Unschedule Late Exams Unschedules all exams that contribute to the NLT cost, i.e. all exams scheduled in timeslots defined as late.

Unschedule Random Censors Unschedules all exams associated with a number of censors picked at random. Two instances of this destroy method are included, where the number of censors is picked at random in the interval $[1, 3]$ and $[4, 10]$.

5.1.3.2 Repair Methods

Both repair methods are greedy heuristics that only differ in the size of the available neighborhood.

Greedy Schedule Takes all unscheduled exams and tries to schedule them one at a time in random order. When attempting to schedule an exam the heuristic considers days going backwards through days available for that exam. Only rooms with sufficient capacity are considered and in order of increasing capacity. Finally looping through timeslots from first to last to check if the exam can be scheduled such that it begins in that day, room and timeslot combination, and how much that scheduling would cost. If an exam can be scheduled without incurring any cost it is immediately scheduled and a new unscheduled exam is chosen. If the exam can be scheduled at some cost it is considered a possible candidate scheduling, and is saved if it is the cheapest candidate scheduling seen. If the heuristic makes it through all available days, rooms and timeslots without finding a zero cost scheduling, then the best candidate scheduling is chosen. In the case that the exam could not be scheduled anywhere (no candidate scheduling found), the exam is removed from the set of exams to schedule. Pseudocode for the Greedy Repair Method is seen in Algorithm 2.

Strict Greedy Schedule This repair method, as the name implies, is a more strict version of the Greedy Schedule heuristic. Instead of at most considering each exam once, the Strict Greedy Schedule heuristic goes through all exams until either a scheduling of zero cost is found or until all exams have been checked. In the latter case, the exam scheduling with the smallest cost is scheduled. If it is not possible to schedule any of the remaining unscheduled exams, the heuristic stops.

As described in the following section, the Strict Greedy Schedule heuristic is used for constructing an initial solution. When used as a repair method, the set of unscheduled exams is shuffled before being passed to the heuristic, such that the order of exams is random..

The pseudocode is very similar to the Greedy Schedule heuristic, with the greatest difference being that exams are not picked at random and the addition of a loop to consider all exams instead of just one. The pseudocode for the Strict Greedy Schedule heuristic is shown in Algorithm 4 in Appendix B.1.

Algorithm 2 Greedy Schedule

```
1:  Given set $V$ of exams to schedule
2:  while $|V| > 0$ do
3:      pick random exam $i \in V$
4:      candidateCost = $\infty$, candidateScheduling = $\emptyset$
5:      for each day $d$ going backwards through available days do
6:          for each room $r$ with sufficient capacity in order of increasing capacity do
7:              for $t = 0$ to $t_{last}$ do
8:                  if can schedule $i$ in $\{r, d, t\}$ then
9:                      if cost of scheduling = 0 then
10:                         schedule $i$ in $\{r, d, t\}$
11:                         $V = V \setminus i$
12:                         break to pick new unscheduled exam
13:                     else if cost of scheduling < candidateCost then
14:                         candidateScheduling = $\{i, r, d, t\}$
15:                         candidateCost = cost of scheduling
16:                     end if
17:                 else
18:                     if other exam scheduled at this time in this room then
19:                         $t$ = timeslot following end of other exam
20:                     end if
21:                 end if
22:             end for
23:         end for
24:     end for
25:     if $i$ has not been scheduled and candidateScheduling $\neq \emptyset$ then
26:         schedule $i$ using candidateScheduling
27:     else if no candidateScheduling found then
28:         $V = V \setminus i$
29:     end if
30: end while
```

5.1.4 Initial Solution Construction

The Strict Greedy Schedule heuristic is used to construct the initial solution used in the ALNS. However, if this heuristic is used alone, the constructed solution for some of the datasets are not feasible. This is chiefly due to the “push exams back” constraint, which does not allow for exams to be scheduled on a day unless there are exams scheduled on the following day. Initially the schedule is empty and as such, only exams that are available for scheduling on the last day can be scheduled. Furthermore it can be difficult to schedule exams that have very early last available days. For example consider a timetable spanning 20 days and an exam that is only available on days $0, 1, 2, 3, 4$. In order to be able to schedule this exam, the rest of the

timetable must at least cover days 5 to 19. This is not guaranteed when using Strict Greedy Schedule.
To circumvent this problem the empty timetable is first "seeded" using exams associated with censors whom only have one exam, i.e. censors who cannot incur censor related costs. The timetable is seeded such that only one of these exams is scheduled on each day. It is possible to initially fully seed an empty solution for all given datasets. By starting with seeded solutions, it is much easier for the Strict Greedy Schedule heuristic to schedule exams immediately, which leads to better initial solutions.

Exams are scheduled in a sorted order such that those assumed to be difficult and/or potentially costly are scheduled first. This is done by first sorting all censors by number of associated exams in decreasing order. Exams that are exceptionally long can be difficult to schedule in an already very filled timetable. Thus, censors with an exam that requires more than half a day (22 timeslots), are moved to the front of the scheduling queue; observing their initial sorting with regards to number of exams.

After filling a fully seeded initial solution, the resulting schedule is guaranteed to spread across all days and with the design of the Strict Greedy Schedule heuristic, the first days in the schedule are likely to have very few exams. Therefore, after the empty schedule is seeded and subsequently fully scheduled using the Strict Greedy Schedule heuristic, the schedule is compressed through a "push back" operation. This entails looping through all days from the beginning and for each day unscheduling all exams and attempting to reschedule these exams using the Strict Greedy Schedule heuristic. If this results in a better schedule, then the changes are kept and otherwise undone. Regardless the same operations are performed on the following day until attempted on all days.

Pseudocode for construction of the initial solution is seen in Algorithm 3.

Algorithm 3 Initial Solution Construction

```
1: Given all exams $E$, censors $C$ and an empty schedule
2: Seed solution using censors in $C \setminus C^{>1}$
3: Sort remaining censors by number of associated exams
4: Move censors with long exams to front (observing order)
5: Schedule all unscheduled exams using Strict Greedy Schedule in given censor
   order
6: Perform push back on resulting schedule
```

5.1.5 Acceptance and Stopping Criteria

To avoid getting stuck in a local minimum, it may be beneficial to allow the accept function to sometimes accept solutions worse than the current solution. One way to do so, is to use the acceptance criteria used in simulated annealing. In simulated annealing a new solution x^t found in the neighborhood of the current solution x, is accepted (for minimization problems) with the probability of $e^{(cost(x^t)-cost(x))/T}$

where T is the temperature. The temperature is initially set to T_{start} and is decreased in every iteration using $T = T \cdot \alpha$ where $\alpha \in [0, 1]$ is the *cooling rate* parameter. Using this acceptance criteria it is more likely to accept worse solutions for larger values of T; which is decreased during the search. Thus the heuristic is allowed more freedom to traverse the solution space through the acceptance of worse solutions in the beginning of the search.

Following the example set in (Ropke and Pisinger, 2006), the start temperature T_{start} is set using the initial solution such that a new solution that is $w\%$ worse than the current solution is 50% likely to be accepted. The parameter w needs to be set and is denoted the *start temperature control* parameter. Given the current solution x and a new solution x^t, the amount x^t is worse in percent is calculated by

$$w = \frac{cost(x^t) - cost(x)}{cost(x)} \Leftrightarrow cost(x^t) = (1 + w) \cdot cost(x)$$

The expression on the right is inserted in the simulated annealing acceptance probability formula with acceptance probability 0.5. Then T is isolating

$$0.5 = e^{(cost(x)-(1+w)\cdot cost(x))/T} = e^{-w \cdot cost(x)/T} \Leftrightarrow T = \frac{-w \cdot cost(x)}{ln(0.5)}$$

So the start temperature T_{start} is calculated using the initial solution x^i as

$$T_{start} = \frac{-w \cdot cost(x^i)}{ln(0.5)}$$

The ALNS heuristic is allowed to run for a predetermined amount of time. Thereby the stopping criteria is when time runs out.

5.1.6 Extensions

Two extensions to the original ALNS framework are implemented in the hope of achieving better results.

5.1.6.1 Pairwise

The weights of both the destroy and repair method are adjusted in each iteration based on the overall performance of both methods. In general it is possible that there are some destroy and repair methods that, due to their design, simply do not work well together. The performance of a single destroy or repair method depends heavily on the other method it is used in conjunction with. Thereby, otherwise well performing methods, may be "punished" simply because it was "unlucky" to be paired with a method that does not complement it well.

To avoid this problem, a variation of the weight system is introduced that considers all destroy and repair methods in pairs instead of separately. With 12 destroy methods and two repair methods this totals 24 destroy/repair pairs. As before rewards are given based on performance, but with weights now considered for pairs these are updated by

$$\rho^{\pm} = \lambda\rho^{\pm} + (1-\lambda)\omega$$

and the probability $\phi_i^{\pm}$ for selecting destroy/repair pair i is calculated by

$$\phi_i^{\pm} = \frac{\rho_i^{\pm}}{\sum_{k=1}^{|\Omega^{\pm}|} \rho_k^{\pm}}$$

where $\Omega^{\pm}$ is the set of destroy/repair pairs, i.e. the Cartesian product of Ω^{-} and Ω^{+}.

In this implementation the two available repair methods are very similar and as such it is unlikely that there are destroy and repair method combinations that are more incompatible than others.

An overview of all pairs is shown in Table B.3 in Appendix B.2.

5.1.6.2 Max Destroy Limit

Another initiative is to reduced time requirement by the introduction of a max destroy limit. Some of the destroy methods are (potentially) very aggressive. For example when working with the ALL dataset and unscheduling a random day, it is possible to remove more than 300 exams; approximately 20% of all exams. If followed by the Strict Greedy Schedule heuristic this could be very time consuming.

Therefore a *max destroy limit* parameter ξ is introduced to control the degree of destruction at an overall level. When this limit is imposed, a destroy method is not allowed to destroy more than ξ exams. All destroy methods are designed to remove exams in random order, such as to avoid unfortunate situations where the exact same exams are remove repeatably when the same destroy method is imposed.

Each destroy method is designed with the intention of allowing the search to go in some specific direction. Imposing a limit on the degree of destruction may effect some destroy methods more than others. However, even when they are limited, they still allow the search to move closer to their intended goal.

5.1.7 Using the One Day Model

Given a solution produced by the ALNS heuristic the One Day model, as introduced in Section 4.4, can be used to evaluate and improve the solution by considering each individual day separately. This model can be used to determine if the found solution is, in some sense, locally optimal. A day from the given solution is extracted and inserted into the One Day Model and solved. Notice that the daily schedule provides a feasible solution, which should make the MIP easier to solve. If the found single day schedule is better, then it is reinserted in the ALNS identified solution. Due to the censor-room restrictions imposed, the results returned from the One Day Model cannot result in a worse solution. So if a single day solution is better then the overall solution will also improve.

This has two potential uses in conjunction with the ALNS heuristic. The One Day Model could be used as part of the ALNS as a combined destroy and repair method, such that a whole day is unscheduled and rebuilt using the MIP solution. Another potential use is as a post-processing tool for ALNS found solutions. Each individual day from the ALNS found solution is completely unscheduled and repaired using the corresponding MIP solution. This may be able to produce a better solution than otherwise found through the ALNS. If no improvements can be found, then it can be concluded that the given solution is a type of local optimum with regard to daily censor gaps, censor room stability and number of late timeslots.

Because the One Day Model is potentially very time consuming, it is decided to not use as part of the ALNS heuristic, but purely as a post-processing tool.

CHAPTER 6

Computational Tests

In this chapter all tests related to the implementation of the ALNS heuristic are discussed. In Section 6.1 the consequences of different construction algorithm settings are examined. Then in Section 6.2 the parameters used in the ALNS are tuned. Finally in Section 6.3 the parameter settings found through tuning are used to test performance and examine some characteristic of the ALNS implementation. Additionally, the best ALNS solutions are put through post processing as discussed in Section 5.1.7.

The implementation is tested both with and without the pairwise extension discussed in Section 5.1.6.1. This gives both a pairwise ALNS and a non-pairwise ALNS.

The cost parameters are user defined in order to allow freedom for planners to weight the different objectives to their liking. The cost parameter settings used for testing here are shown in Table 6.1.

Parameter	p^{RS}	p^{CG}	p^{CMD}	p^{D}	p^{L}
Value	3	1	10	15	3

Table 6.1: The cost parameters used for testing.

6.1 Construction Algorithm

Construction of an initial solution is done as described in Section 5.1.4. Here the time requirement and quality of the constructed solution is examined depending on different seeds and whether or not the push back operation is applied. The construction heuristic is tested with no seed, a "semi" seed and a full seed; all with and without push back. The semi seed is where the initially empty schedule is seeded from the earliest, last censor day (i.e. the first day that is a censors' last available day) to the last timetable day. Thereby it does not open the initial schedule as much as the full seed, with the intention of ending with less used days or exams that should be pushed back.

The results are shown in Table 6.2. Although there is some stochasticity involved with the push back operation, the difference in initial constructed solution cost is generally small. Only when using a full seed does the construction algorithm produce feasible start solutions for all datasets. The table shows that the larger the seed, the longer it takes to construct the initial solution. This is because each exam has more feasible placements in the timetable, thus increasing the time it takes to find its' best scheduling. The construction of an initial solution for the ALL dataset is especially time consuming, requiring more than 15 minutes when using a full seed. The push back operation requires almost five additional minutes and only results in a cost reduction of 4. In all cases, the push back operation leads to better solutions, however, if time is limited it is not important to perform the push back as the improvements are minute.

		IKH			IMT			INM		
Seed	Pushback	Cost	Time (s)	Unscheduled	Cost	Time (s)	Unscheduled	Cost	Time (s)	Unscheduled
No	No	-	15	19	763	2	-	174	1	-
No	Yes	-	23	19	725	5	-	166	1	-
Semi	No	-	34	19	667	6	-	178	1	-
Semi	Yes	-	43	19	658	6	-	161	1	-
Full	No	1,324	52	-	733	7	-	219	1	-
Full	Yes	1,293	60	-	640	9	-	174	1	-

		ISE			ALL		
Seed	Pushback	Cost	Time (s)	Unscheduled	Cost	Time (s)	Unscheduled
No	No	-	10	7	3,589	216	-
No	Yes	-	21	7	3,306	502	-
Semi	No	-	19	7	3,330	615	-
Semi	Yes	-	28	7	3,312	845	-
Full	No	1,292	41	-	3,038	990	-
Full	Yes	1,251	51	-	3,034	1.278	-

Table 6.2: An overview of the initial solutions generated from the construction algorithm using different types of seed and with and without the push back operation.

6.2 Parameter Tuning

Parameter tuning is performed by evaluating the performance of the heuristic for every possible parameter combination. Every combination is allowed to run for 10 minutes and repeated 10 times where each time the best timetable cost is recorded. The runs are performed in parallel on the DTU High Performance Computing (HPC) clusters. The ALNS heuristic is implemented in Java and each parameter tuning run is done on one core of an Intel Xeon Processor E5-2660 v2 of 2.80GHz with a RAM limit of 2GB running the Scientific Linux 6.4 operating system.

To evaluate the performance of a parameter combination the average gap E (in percent) and spreading σ is calculated for every parameter combination as follows

$$E = \frac{1}{N}\sum_{i=1}^{N}\frac{z_i - z^*}{z_i}\cdot 100\%$$

$$\sigma = \sqrt{\frac{\sum_{i=1}^{N}(z_i-\mu)^2}{N}}$$

where N is the number of runs, z_i is the best objective value found in the i'th run, z^* is the best know solution value and $\mu = \frac{1}{N}\sum_{i=1}^{N} z_i$ is the average value for all 10 runs.

6.2.1 Tuning Dataset

Unfortunately the number of available datasets is very limited and the sets themselves very different. It is interesting to evaluate the performance of the solution method on the ALL dataset, as this dataset provides the most difficult version of the problem. However, since the ALL dataset is a combination of all others, there is no other dataset with similar characteristics to use as a training dataset. The ISE and IKH datasets are the most similar datasets, where the IKH set is only slightly larger in most aspects, but the ISE dataset is more constrained by exam conflicts.

Parameter tuning and testing is carried out in two separate tracks; one with the ALL dataset and one with all others. Thus, parameter tuning is performed on the ALL dataset and the best parameter combination found is then also used for testing on the ALL dataset. Although this carries the risk of overfitting, it is done because there are no other comparable datasets. Therefore parameter tuning is also done on the ISE set and the results then used for testing on the remaining datasets. Performing these separate tracks of tests should provide better insight than only doing one.

6.2.2 Overview of parameters

In order to make parameter tuning less cumbersome, it is decided to fix the reward weight parameters $\omega_1, \omega_2, \omega_3$ and ω_4 following the example set in (Ropke and Pisinger, 2006); that is $\omega_1 = 33$, $\omega_2 = 9$, $\omega_3 = 13$ and $\omega_4 = 1$. In (Ropke and Pisinger, 2006) awards are given to previously unseen solutions in the following manner; ω_1 for a new global best, ω_2 for a previously unseen and improving solution and ω_3 for a previously unseen and worsening but accepted solution. While it is surprising, that accepted worsening solutions are reward more than improving solutions, Stefan Røpke generously clarified through email, that this result is simply because there was no measurable difference between the parameter settings. His interpretation is, that it was randomness that determined $\omega_2 < \omega_3$. This supports the notion that perhaps this parameter setting is not the most important parameter to fine tune.
Weight ω_4 is set to 1 to ensure that there is always some probability of choosing a method, since the minimum weight possible is then 1. Also let it be noted, that in this implementation weights are updated every iteration, which is different from other implementations such as (Ropke and Pisinger, 2006) and (Ribeiro and Laporte, 2012), where weights are updated after a number of iterations.

Parameter tuning is done for both the pairwise and non-pairwise versions of the ALNS implementation. An overview of the parameter settings used for tuning is shown in Table 6.3.

Parameter	Variable	Values
Max Destroy Limit	ξ	10 25 50 ∞
Decay	λ	0.1 0.5 0.9
Cooling Rate	α	0.999 0.99975 0.9999
Start Temperature Control	w	0.01 0.05 0.10 0.15

Table 6.3: Parameter settings used in parameter tuning.

6.2.3 Parameter Tuning Results

In total, 288 parameter combinations are tested on each dataset and only the most important results are shown here.

The most influential parameter is by far the max destroy limit ξ. Looking at the ALL parameter tuning runs, measured on the average gap the 24 best performing parameter combinations (12 non-pairwise and 12 pairwise) have $\xi = 10$. Conversely, out of the 50 worst performing parameter combinations 37 have $\xi = \infty$ and 13 have $\xi = 50$, i.e. the two largest ξ parameter settings tested. The same trend is seen for parameter tuning on the ISE dataset.
Table 6.4 shows an overview of the average gaps and spreadings for different ξ and λ combinations for the ALL dataset parameter tuning. Each entry in the table is the average of all parameter tuning runs for the specified ξ and λ combinations, i.e. the average across all α and w parameter settings with fixed ξ and λ. This tables shows that both the average gap and spreading increases when ξ increases almost independently of the λ setting. The same is the case for the non-pairwise ALNS and the ISE pairwise and non-pairwise as shown in the tables found in Appendix C.1.

	E (%)			σ		
$\xi\backslash\lambda$	0.1	0.5	0.9	0.1	0.5	0.9
10	105.65	107.52	108.71	58.70	55.68	56.47
25	139.59	143.03	147.31	74.49	76.09	67.98
50	152.05	156.93	161.82	75.84	70.67	69.56
∞	168.57	174.02	178.67	76.76	92.00	98.33

Table 6.4: The mean average gap E (%) and spreading σ across all parameter setting for α and w with different ξ and λ combinations when parameter tuning the pairwise ALNS.

The average gap and spreading for all parameter settings with $\xi = 10$ for the pairwise ALNS is shown in Table 6.5 and 6.6 respectively. The parameter setting chosen for further testing with the pairwise ALNS is $\xi = 10$, $\lambda = 0.9$, $\alpha = 0.999$ and $w = 0.01$ because this combination has the second lowest average gap of 42.30% (1.43% from the best) while also having the lowest spreading of 26.30 (16.95 lower than the combination with the lowest average gap).

For the non-pairwise ALNS the parameter setting chosen for further testing is the same as for the pairwise. The reason being that this parameter combination results in the the smallest average gap (40.07%) and the second smallest spreading (42.05).

	$\xi = 10$											
	$\lambda = 0.1$				$\lambda = 0.5$				$\lambda = 0.9$			
$\alpha \backslash \omega$	0.01	0.05	0.10	0.15	0.01	0.05	0.10	0.15	0.01	0.05	0.10	0.15
0.999	47.50	49.46	53.80	56.31	40.87	49.06	51.97	55.79	42.30	52.68	51.63	54.73
0.99975	75.33	133.80	141.96	146.66	83.98	139.00	147.91	146.64	98.86	138.11	145.60	147.83
0.9999	123.54	146.65	148.66	144.14	124.81	150.13	149.19	150.93	131.08	145.35	147.51	148.88

Table 6.5: The average gap (%) for all parameter combinations with $\xi = 10$ on the ALL dataset with the pairwise ALNS.

	$\xi = 10$											
	$\lambda = 0.1$				$\lambda = 0.5$				$\lambda = 0.9$			
$\alpha \backslash \omega$	0.01	0.05	0.10	0.15	0.01	0.05	0.10	0.15	0.01	0.05	0.10	0.15
0.999	64.68	39.03	63.08	49.65	43.25	54.30	67.95	56.14	26.30	64.51	60.94	44.38
0.99975	75.83	75.37	81.58	47.16	89.66	58.48	54.58	45.57	88.49	62.79	67.63	37.60
0.9999	34.60	35.10	58.26	80.04	52.19	53.15	41.00	51.93	58.93	57.14	51.65	57.28

Table 6.6: The spreading (σ) for all parameter combinations with $\xi = 10$ on the ALL dataset with the pairwise ALNS.

The parameter tuning on the ISE dataset gives very similar results, and the parameter combinations chosen for further testing only vary with regard to the w parameter. For the pairwise ALNS the parameter combination $\xi = 10$, $\lambda = 0.9$, $\alpha = 0.999$ and $w = 0.10$ is chosen because this results in the lowest average gap (3.54%) and has a low spreading (8.06). For the non-pairwise the settings are the same except $w = 0.05$ because this gives the lowest average gap overall (3.01%) and a very small spreading (6.68).

Average gaps and spreading are shown for all parameter combinations on both tuning datasets in Appendix C.2. As could be expected, the parameter tuning runs get much better results (lower average gaps and spreading) on the ISE dataset compared to the ALL dataset.

The parameter settings used for testing for each dataset are shown in Table 6.7. The settings found through parameter tuning for the ALL dataset is used on that dataset and the settings found for the ISE dataset is used on all others.

Dataset	Pairwise	ξ	λ	α	w
IKH	No	10	0.9	0.999	0.05
IKH	Yes	10	0.9	0.999	0.10
IMT	No	10	0.9	0.999	0.05
IMT	Yes	10	0.9	0.999	0.10
INM	No	10	0.9	0.999	0.05
INM	Yes	10	0.9	0.999	0.10
ISE	No	10	0.9	0.999	0.05
ISE	Yes	10	0.9	0.999	0.10
ALL	No	10	0.9	0.999	0.01
ALL	Yes	10	0.9	0.999	0.01

Table 6.7: The parameter settings used for testing on each dataset.

6.3 Results

All computations are performed on the HPC cluster using the same setup as explained in Section 6.2. First overall performance is examined on all datasets. Then the consequences of the max destroy parameter ξ is discussed and some observations are made regarding the ALNS implementation. Finally the best ALNS found solutions are put through post processing. Throughout both the pairwise and non-pairwise implementations are tested. Note that the numbering of destroy and repair methods/pairs is shown in Tables B.1 - B.3 in Appendix B.2.

6.3.1 Benchmarking

In order to benchmark the ALNS implementation it is run on all datasets 10 times using the same initial solution (the solutions found in Section 6.1) and the best schedule cost is examined. Parameters are set as found through parameter tuning (see Table 6.7). On the ALL dataset the heuristic is allowed to run for 3 hours and 1 hour on the others. This is because, from a practical perspective it would be allowable to have the heuristic running for 3 hours (or more), but generally on most datasets, no improvements are found after the first hour. The results are shown in Table 6.8; for the ALL dataset the best schedule cost found after 1 hour is also shown.

These results show, that the implemented heuristic is very stable and finds good timetables. The one hour non-pairwise ALL runs have the largest average gap and spreading with 8.49% and 21.79 respectively. However within two additional hours these numbers are brought down to 3.20% and 13.22. For all other datasets the largest gap and spreading is found for the non-pairwise IKH with values of 3.28% and 10.25. All runs on the INM dataset result in timetables with the best known cost, showcasing that this is the smaller and easier instance to solve. In fact for each run the best timetable is found within the first 30 seconds of the search.
For all datasets the difference in performance between pairwise and non-pairwise

ALNS is small but the pairwise version is is better, consistently having both a smaller average gap and spreading. The only exception is the ISE dataset where the spreading is slightly lower for the non-pairwise ALNS.

Dataset	Pairwise	Solution Values										$E(\%)$	σ	Best Known
IKH	No	540	536	543	573	543	542	545	543	545	533	3.28	10.25	527
IKH	Yes	539	545	546	537	542	536	550	545	539	530	2.64	5.56	527
IMT	No	280	283	283	280	283	283	283	283	286	280	0.86	1.80	280
IMT	Yes	283	283	283	280	280	280	283	283	280	283	0.64	1.47	280
INM	No	110	110	110	110	110	110	110	110	110	110	0.00	0.00	110
INM	Yes	110	110	110	110	110	110	110	110	110	110	0.00	0.00	110
ISE	No	498	495	495	492	499	495	498	496	504	495	0.96	3.10	492
ISE	Yes	492	496	492	501	492	505	503	492	492	498	0.87	4.88	492
ALL	No	1,067	1,093	1,063	1,104	1,067	1,043	1,048	1,107	1,050	1,077	8.49	21.79	988
ALL	Yes	1,070	1,080	1,082	1,054	1,052	1,042	1,054	1,040	1,060	1,058	7.21	13.57	988
ALL (3h)	No	1,007	1,030	1,024	1,039	1,016	995	1,006	1,023	1,020	1,036	3.20	13.22	988
ALL (3h)	Yes	1,010	1,032	1,017	1,003	1,026	1,011	1,014	996	1,010	1,014	2.56	9.79	988

Table 6.8: The objective value for each evaluation run as well as average gap, $E(\%)$, and spreading, σ, for each of the datasets. Results are shown for both the one hour and three hour runs on the ALL dataset.

6.3.2 Consequence of the Max Destroy Parameter

During parameter tuning all “better” parameter combinations had the lowest tested value of 10 for the max destroy parameter ξ. In fact, measured on lowest average gap, for the ISE dataset the 59 best parameter combinations had $\xi = 10$ while on the ALL dataset it was the 24 best. Here the consequences of ξ are investigated.

Both the pairwise and non-pairwise ALNS is run for three hours on the ALL dataset with ξ values 10, 50 and ∞ (no limit). The other parameters are set based on the parameter tuning, such that $\alpha = 0.999$ and $w = 0.01$ for all runs. For the pairwise runs the λ values for $\xi = 10$, 50 and ∞ are 0.9, 0.1 and 0.1 respectively and for non-pairwise λ is set to 0.9, 0.5 and 0.5. Through parameter tuning these λ values are identified as best for the given ξ values with α and w fixed.

The timetable cost per second for all runs is shown in Figure 6.1. For the same ξ settings, the pairwise and non-pairwise heuristics have very similar total cost progressions, with an initially very steep curve that flattens and seems to converge asymptotically. All curves approximately start to flatten after about 500 seconds and ξ is the deciding parameter as to how fast/deep the initial “dive” is. This is likely because a smaller ξ value greatly limits the size of the solution neighborhood and thereby allows for a quicker, but less thorough, search. In Figure 6.2 the timetable total cost for each iteration is shown. This shows that both cost progression for both $\xi = 50$ and $\xi = \infty$ is very similar and that for these settings each iteration requires more time. In three hours, the search gets through between approximately 42,000 and 47,500 iterations for $\xi = \infty$ and through around 143,000 iterations for $\xi = 50$. This is not a lot compared to the 340,000 to 396,000 iterations that the $\xi = 10$ runs get through. However, all runs start to show the same flattening curve after about 10,000 iterations and therefore, it would require a very long time for the heuristic to reach solutions of the same quality for $\xi = 50$ or $\xi = \infty$ as for $\xi = 10$. Also notice that after 10 minutes (600 seconds) there is a noticeable difference in the cost between all ξ settings, indicating that allowing for 10 minute parameter tuning runs is sufficient to at least distinguish better performing ξ settings.

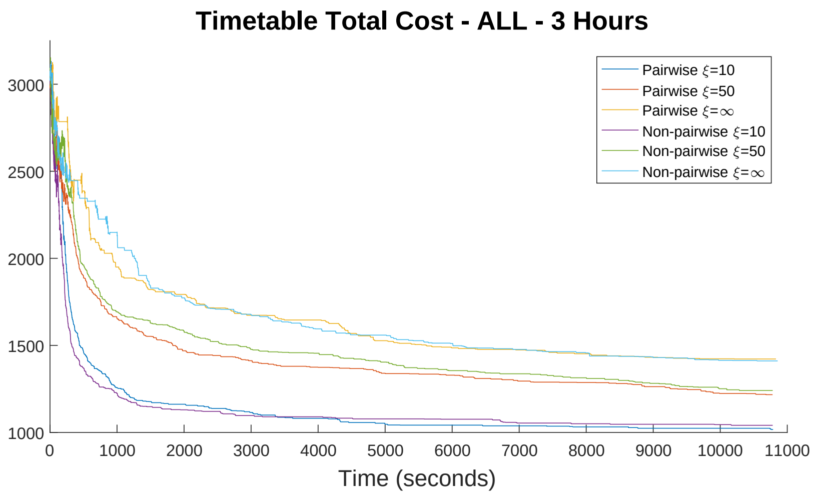

Figure 6.1: The current timetable total cost for each second of a three hour runs on the ALL dataset with different ξ settings.

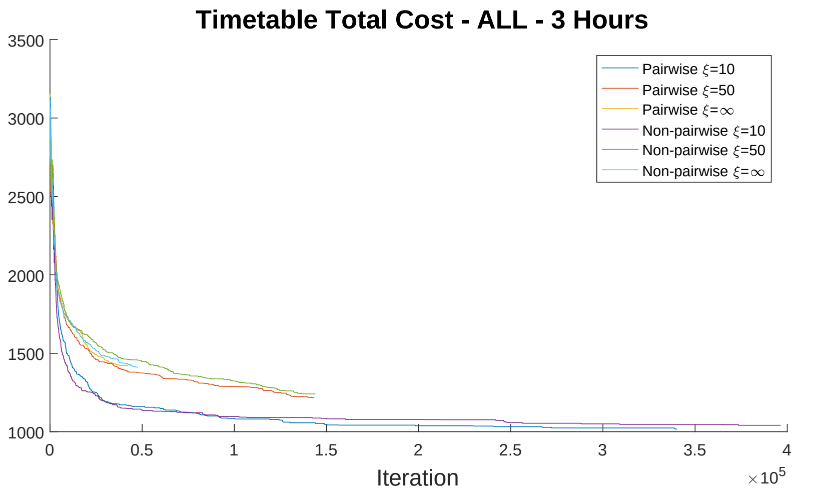

Figure 6.2: The current timetable total cost for each iteration of a three hour runs on the ALL dataset with different ξ settings.

In the cost per second plot all curves are very "ragged" in the beginning due to simulated annealing acceptance criteria, as worsening solutions are more likely to be accepted early in the search (see Section 5.1.5). While the initial cooling phase is longer (in time) for $\xi = \infty$, the curves still do not become as smooth as for the curves for the other ξ settings. This is explained by the time requirement of some of the iterations. With no limit on the degree of destruction, large parts of the solution may be destroyed and repaired using the more time consuming Strict Greedy Schedule heuristic. In fact, in the pairwise and non-pairwise $\xi = \infty$ runs examined, there are 37 and 42 iterations respectively that require more than a minute. This may be acceptable if there is potential for a much improved solution, but the data shows that this did not happen.

The average time requirement dependent on ξ for all destroy-repair pairs on the ALL dataset is shown in Table 6.9. In general there is an increased time requirement for larger values of ξ, but for pair 6 the increase is especially large with an average of almost 9 seconds per iteration for $\xi = \infty$. The longest time taken for pair 6 is 127.96 seconds for just one iteration. Pair 6 first unscheduled a random day and then uses the Strict Greedy Heuristic to repair. It is not unusual for some days to have more than 300 exams when timetabling the ALL dataset and therefore fully rescheduling that many exams is quit time consuming.

Lastly, reducing the degree of destruction to a maximum of 10 exams effectively make some destroy methods equal. For example, there are destroy methods that unschedule random exams in the interval $[3, 10]$, $[11, 25]$ and $[26, 40]$. For $\xi = 10$, the latter two will always remove 10 exams from the schedule. Therefore the methods will perform equally well and and the probability for removing 10 random exams is effectively doubled since it present in the method pool twice and both will have the same weight. This is apparent through the reward distribution and performance of matching destroy methods after a whole run.

ξ	0	1	2	3	4	5	6	7	8	9	10	11
10	0.02	0.01	0.05	0.01	0.05	0.01	0.04	0.01	0.04	0.01	0.02	0.01
50	0.03	0.01	0.16	0.02	0.50	0.04	0.71	0.04	0.04	0.01	0.02	0.01
∞	0.03	0.01	0.19	0.02	0.61	0.04	8.98	0.14	0.03	0.01	0.01	0.01

ξ	12	13	14	15	16	17	18	19	20	21	22	23
10	0.41	0.02	0.56	0.03	0.00	0.00	0.02	0.01	0.04	0.01	0.04	0.01
50	0.69	0.04	1.05	0.05	0.00	0.00	0.04	0.01	0.35	0.03	0.05	0.01
∞	0.79	0.04	1.12	0.05	0.00	0.00	0.04	0.01	0.33	0.03	0.03	0.01

Table 6.9: The average time requirement (seconds) for each destroy-repair pair for different ξ settings.

6.3.3 Temperature Cooling

For the ALL dataset, the best parameter combination found has the two parameters related to the simulated annealing acceptance criteria at their lowest tested values; $\alpha = 0.999$ and $w = 0.01$. This indicates that immediate intensification is rewarded which corresponds to the pattern seen in the previous section where the total cost curve quickly dives. Figure 6.3 shows the acceptance probability progression of different absolute worse values for the first 5,000 iterations. The start temperature is 43.77 as calculated using $w = 0.01$ on the initial solution cost value. The figure shows that the probability of accepting worsening solutions quickly becomes very low even for small differences. After 3,004 iterations the probability of a new solution with a total cost that is just 10 larger is only 1%. After 5,000 iterations the probability of accepting a solution with just 1 more in cost is 3.35%. In three hours with the most time consuming ξ setting the heuristic gets through around 42,000 iterations, which means that in most iterations the simulate annealing acceptance criteria is negligible and practically only improving solutions are accepted. The same is the case for the other datasets, even though these are run with a higher w value of 0.05 and 0.10. On the IKH dataset for $w = 0.10$ the start temperature (highest for all runs) is 185.96. With $\alpha = 0.999$ the probability of accepting a worse solution with only one more in cost is less than 1% after 6,751 iterations. In an hour the heuristic gets through more than half a million iterations.

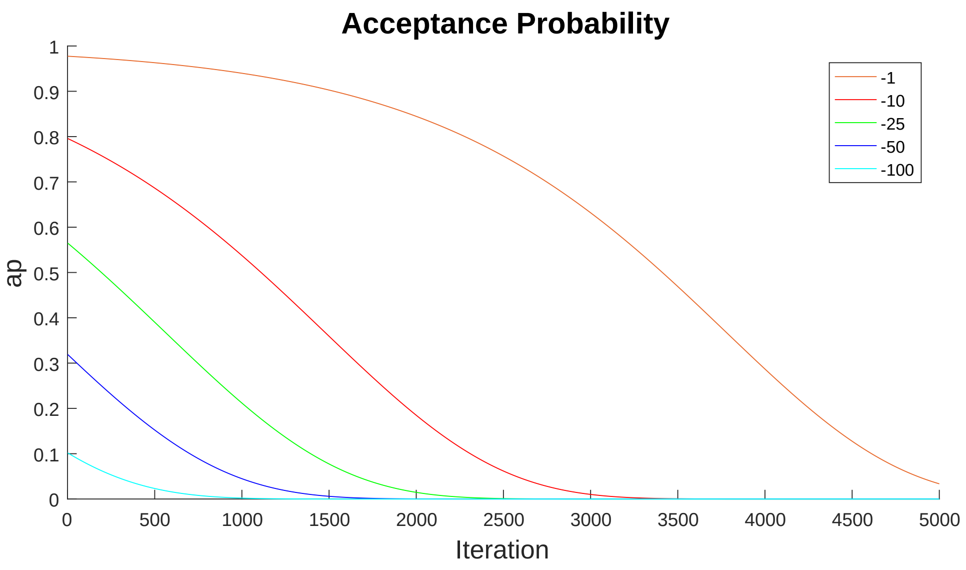

Figure 6.3: The acceptance probability progression for new worse solutions of different cost changes for $T_{start} = 43.77$ and $\alpha = 0.999$.

One additional test is done to further investigate the effects of α. The pairwise ALNS is run with the three different α settings (with $\xi = 10$, $\lambda = 0.9$ and $w = 0.01$). The total cost per iteration is shown in Figure 6.4 with the last worsening accepted solution for each run marked by an asterisk. As expected, the lower α values result in a quicker "dive" but after about 40,000 iterations the three runs have found timetables of about the same cost. The curves follow each other close from then on, but both for $\alpha = 0.99975$ and $\alpha = 0.9999$ the heuristic ends up finding better solutions in the end. However, after 53,093 iterations even for the slowest cooling rate of $\alpha = 0.9999$ the acceptance probability for a timetable with just one more in cost is less than 1%. Ergo beyond this point all runs are practically only accepting improving solutions. Also the last worsening but accepted solution for each run is within the first 46,000 iterations. Therefor the performance of each run in the second half of all iterations is more likely to be due to randomness. It seems fitting that $\alpha = 0.999$ is used for cooling, as this provides better solutions quicker and results in solutions of equal quality in the end.

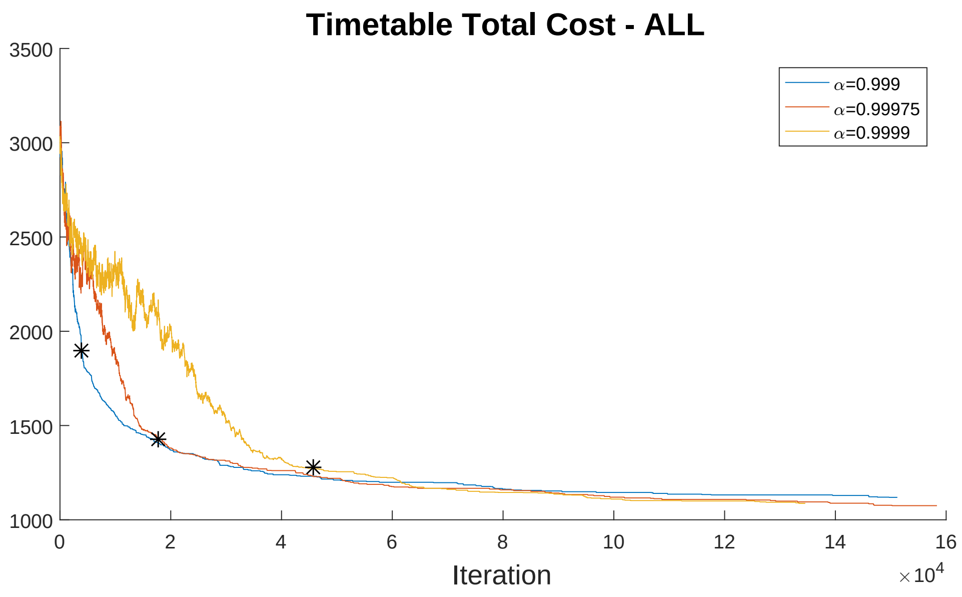

Figure 6.4: The total timetable cost using the pairwise ALNS on the ALL dataset with three different α settings. An asterisk marks the last worsening solution accepted for each run.

6.3.4 Performance of Destroy and Repair Methods/Pairs

Since there is so many iterations where non-improving solutions are found, the analysis of destroy and repair methods/pairs is limited to the first 3,000 iterations of the runs on the ALL dataset. The reward distributions (in the first 3,000 iterations) for all pairs with both $\xi = 10$ and $\xi = \infty$ are shown in Figure 6.5 (data shown in Table C.34 and C.35 in Appendix C.3). For $\xi = \infty$ pairs 8, 9, 16, 17, 22 and 23 receive a disproportionately large number of ω_2 rewards, i.e. worsening but accepted solutions.

Of special note is pair 16 and 17 which resulted in only worse but accepted solutions. The two pairs both destroy exams with late timeslots (in random order) and rebuild with the Strict Greedy Schedule and Greedy Schedule heuristic respectively. Pair 8 and 9 both unschedule the day with the least number of exams and pair 22 and 23 unschedule the first day with exams scheduled. The fact that these pairs are chosen so often is likely because, they often lead back to the exact same, or very similar, solutions. If the same solution is found then the solution is accepted by the simulated annealing acceptance criteria and reward ω_2 is given. Throughout the search the number of late timeslots is generally very low and very few exams are scheduled on the first day with exams. Therefore the degree of destruction for these pairs is naturally very low and the neighborhood very small. This makes it likely that the repair methods lead back to the same solution. In this implementation, only the timetable total cost is used to evaluate new solution and not whether or not it has been seen before. When the same solution is found, it is accepted and awarded ω_2 which, since the reward settings are taken from (Ropke and Pisinger, 2006), rewards more than if the new solution was better. Therefore the likelihood of choosing the same pair again becomes quite large and thus a reinforcing loop is started.

This shows the importance of keeping track of already visited solutions and “punishing” heuristics that find these. Just looking at the reward distribution for $\xi = \infty$ could also lead to the conclusion, that all of the discussed pairs are bad at finding good solution. However comparing to the reward distribution for $\xi = 10$ gives better insight. Here pair 16 and 17 show the same pattern of only finding non-improving but accepted solutions. Generally throughout the search there is a very limited number of exams with late timeslot and therefore these pairs do not destroy much of the solution. Only late exams are destroyed and room is not specifically for them elsewhere in the schedule, makes it difficult to find improving solutions using the two repair heuristics.

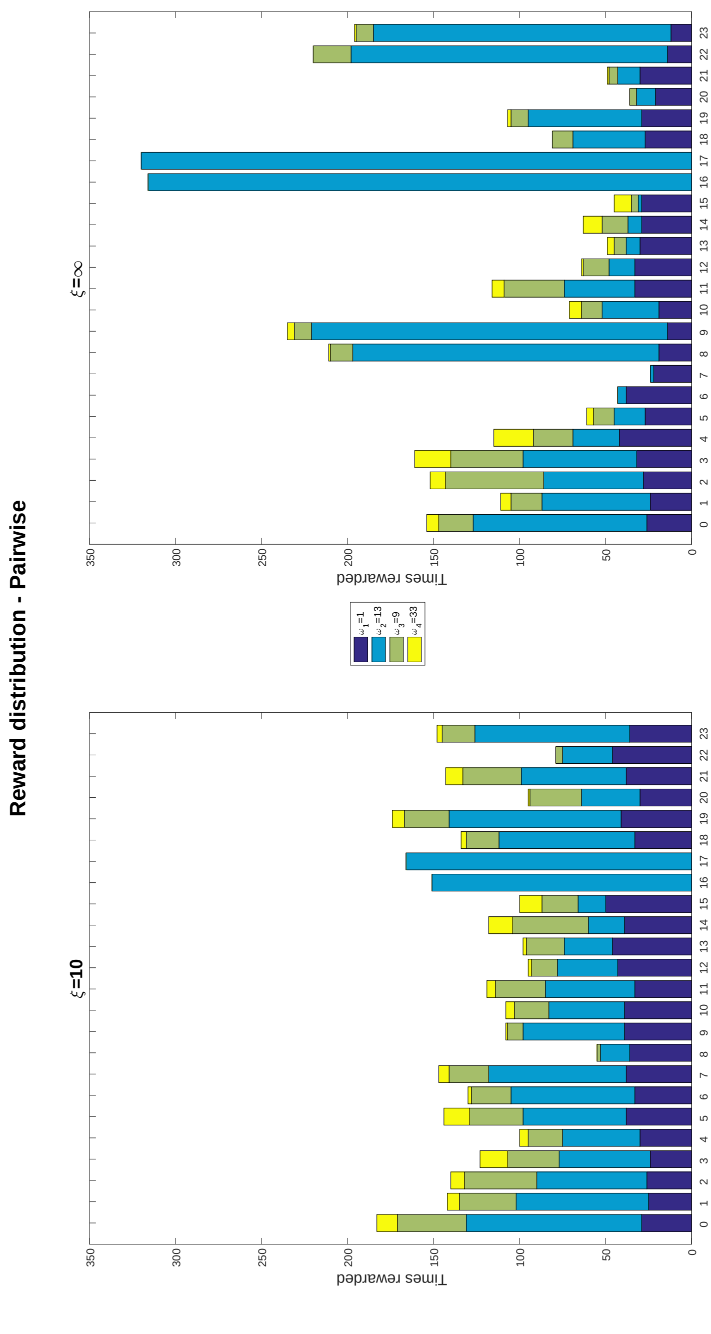

Figure 6.5: The reward distribution for each destroy-repair pair in the first 3,000 iterations for $\xi = 10$ and $\xi = \infty$ when run on the ALL dataset using the pairwise ALNS.

It is expected that the Strict Greedy Schedule heuristic gives better results than the Greedy Schedule heuristic. The results shown in Figure 6.5 cannot confirm this. As seen in Table B.3 the first two pairs are defined using destroy method 0, the third and fourth pair are defined using destroy method 1 and so on. All even numbered pairs are repaired using the Strict Greedy Schedule heuristic and all odd using the Greedy Schedule heuristic. Looking across all pairs there is no clear indications that those using Strict Greedy Schedule outperform their Greedy Schedule counterparts.

The reward distribution for the first 3,000 iterations of the non-pairwise $\xi = \infty$ ALNS run on the ALL dataset is shown in Figure 6.7 (data in Table C.37). Some of the poorer performing destroy methods are 3, 6, 8 and 10. Destroy method 8 unschedules late exams and suffers from the problems discussed earlier. Destroy method 3 unschedules a whole day, destroy method 6 unschedules between 1 and 5 censors based on their CMD cost and destroy method 10 unschedules all exams for between 4 and 10 random censors. Just as for the pairwise version, the destroy method unscheduling a random day performs very poorly. The large degree of destruction combined with lack of "focus" is likely the reason for these destroy methods having lackluster performance. For example destroy method 5 finds the censor with the largest CMD cost and unschedules the rooms this censor uses on the days where he/she is scheduled. This can also be a very destructive strategy, but opens up the problem more up for finding better solutions. The destroy method performance when $\xi = 10$ shows a very different picture (Figure 6.6). Here destroy method 3 (unscheduling a random day) has very good performance. This difference can be explained by the lowered degree of destruction, which directly makes the method more focused; countering the problems present with no maximum destroy limit.

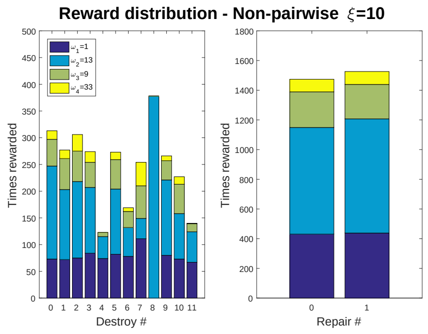

Figure 6.6: The reward distribution for each destroy and repair method in the first 3,000 iterations for $\xi = 10$ when run on the ALL dataset using the non-pairwise ALNS.

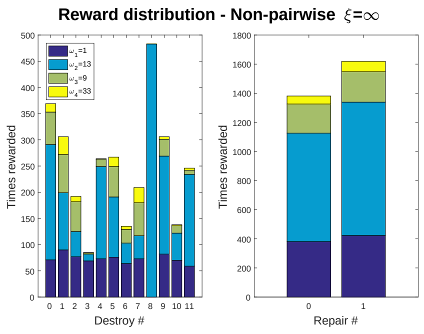

Figure 6.7: The reward distribution for each destroy and repair method in the first 3,000 iterations for $\xi = \infty$ when run on the ALL dataset using the non-pairwise ALNS.

Another consequence of limiting the degree of destruction is making the two repair methods more similar, which is also noticeable in the reward distribution. With less exams to reinsert, there is less possibility for the Strict Greedy Schedule heuristic to use the looping feature which distinguishes it from the Greedy Schedule heuristic. Table C.38 and C.39 show the reward distributions for all iterations with $\xi = 10$ and $\xi = \infty$ restrictively. This data shows, that when the degree of destruction is very low, the two repair methods are rewarded almost identically, while there is a more significant difference with no limit. It may seem that the Greedy Schedule repair heuristic has better performance than the Strict Greedy Schedule heuristic when $\xi = \infty$. However this is likely due to randomness and the fact that worsening but accepted solutions are rewarded as much as they are. This means that the simpler repair heuristic may have "okay" performance and the rewarded well and thus chosen more often. The data supports this, as the Greedy Repair method is rewarded many more ω_1 and ω_2 rewards and only slightly more ω_3 and ω_4 rewards. A consequence of only including two repair heuristics is that an increase in weight (and thereby probability) for one results in an equally large decrease in the probability for choosing the other. Especially in the beginning of the search, this may lead to another reinforcing loop meaning that the Greedy Repair heuristic is chosen more often and thereby have more changes to "get lucky" and find good solutions. The Strict Greedy Schedule heuristic does not have as many chances, but still manages to find almost as many improving solutions as the other method. When the search gets to a point where it almost only finds rejected solutions, an almost cyclical pattern of opposite increase and decrease in the probability of choosing each repair method begins. This causes the heuristic to alternates between the two repair methods.

6.3.5 Adaptive Weights

One of the strengths of the ALNS heuristic is the adaptive layer allowing the search to be guided based on the instance at hand. As discussed in the previous section, the implementation used here has some problems directly related to the reward system used for controlling the adaptive layer. These problems are investigated in more detail as well as the effects of the decay parameter λ.

Figure 6.8 shows the weights for pair 0, 1, 8 and 16 for the first 3,000 iterations for parameter combinations $\xi = 10$, $\alpha = 0.999$, $w = 0.01$ and $\lambda = [0.9, 0.1]$. Pair 0 and 1 both remove 3-10 random exams and reschedule these using the Strict Greedy and Greedy Schedule heuristics respectively. From the previous section it can be concluded, that pair 0 has overall slightly better performance than pair 1. In the first plot of Figure 6.8 it is also clearly visible that, at least for the first 3,000 iterations, pair 0 generally has a greater weight than pair 1. But again, there is no indication that the Strict Greedy Schedule heuristic is better than its' more simple counterpart. The purple line shows the weight of pair 16, which unschedules late timeslots and repairs using Strict Greedy Schedule and never finds improving solutions. The plot shows, the weight for this pair quickly rises to its' maximum of 13 and stays there. Thereby the probability of this pair being chosen in a given iteration is generally greater than either pair 0 or 1, which have much better performance. Throughout

the whole run (approximately 340,000 iterations) the probability of picking either one of the pairs unscheduling late exams (16 and 17) is on average 27.54%. Thereby these pairs are chosen often, resulting in 94,081 "wasted" iterations.
Pair 8 unschedules the slowest day and repairs using the Strict Greedy Schedule heuristic and has poor performance. The figure supports this, showing that the weight for this pair is generally lower than for the better performing pairs 0 and 1.

The two plots in Figure 6.8 also show the effect of decay parameter λ on how sensitive weights are to change. For $\lambda = 0.9$ changes are very slow, which is clearly illustrated by the pair 16 weight progression. Even though this pair is awarded 13 every time, it has to receive 73 rewards (each seen as a "bump" on the curve) before its' weight reaches 13. For $\lambda = 0.1$ weight changes are much more violent, as the second plot in the figure shows. For this setting weights change rapidly, making the adaptive aspect of ALNS more chaotic. For $\lambda = 0.1$, pair 16 is only rewarded three times before reaching its' maximum.

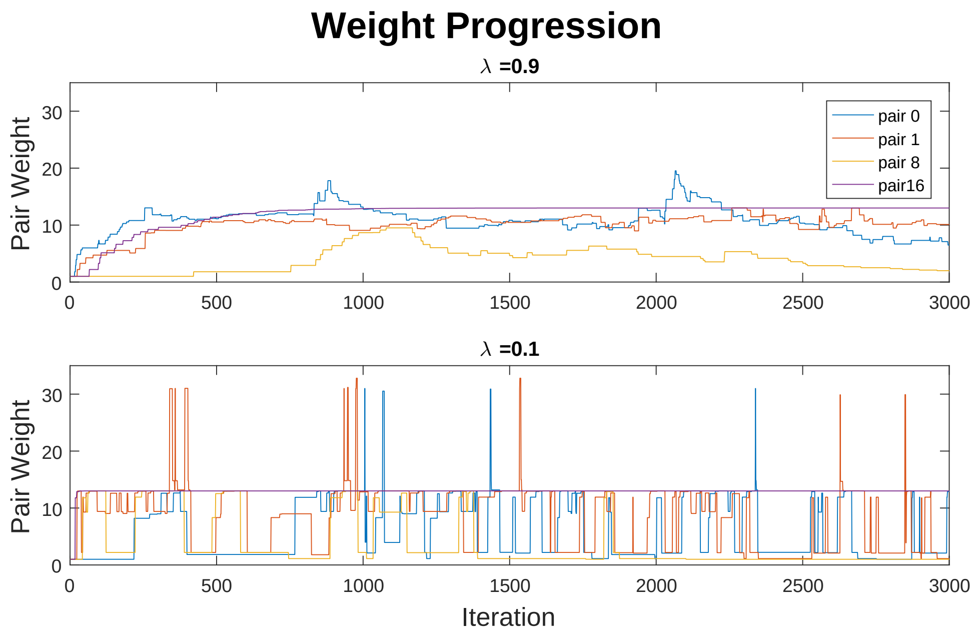

Figure 6.8: The weight progression of pair 0, 1, 8 and 16 in the first 3,000 iterations when run on the ALL dataset with the two λ parameter settings.

6.3.6 Post Processing

As discussed in Section 5.1.7, the One Day Model can be used as a post processing tool for final improvements and evaluation of the solutions found through the ALNS. An overview of the best solution for each dataset found using the implemented ALNS is shown in Table 6.10. Since the One Day Model can only improve objectives on a daily basis, only solutions with room stability, censor gaps and number of late timeslots costs can be affected. Thereby the post processing can only hope to improve the ALNS identified solution for the IKH, ISE and ALL datasets.

Post processing is done by going through all days of the given solution, unscheduling each day and repairing using the One Day Model. If the solution value improves following a repair the day is kept as is and otherwise the changes are undone by rescheduling each exam to its previous allocation.

No improvements are made trough post processing the ISE solution.

For the IKH solution an improvement is found such that there is one less late timeslot and the the total cost of the best known solution for the IKH dataset is thus 527.

For the ALL solution improvements are found on day 17, 18 and 19 which have 176, 195 and 326 exams scheduled respectively. Surprisingly the One Day Model for day 17 cannot be solved to optimality within 8 hours. These tests are run on the DTU HPC using CPLEX with default MIP solver settings and using 20 threads. The initial daily cost of day 17 is 25 which is reduced to 23 with an absolute lower bound of 21. On day 18 the optimal solution is found in just under 1 hour and 45 minutes, improving the initial daily cost of 23 to 21. Finally on day 19, which has more than 130 exams in excess of the other two days, the optimal solution is found in around 3 hours and 30 minutes, resulting in a cost reduction of 21 to 18. In total, the number of censors gaps is reduced by four and the number of late timeslots used by one, resulting in a total cost reduction of 7 and the best know solution for the ALL dataset is thereby 988. The reason for not finding an optimal solution on day 17 even, though it has less exams, is unknown. These runs have only been performed with the default settings and the effects of changing branching and search strategy should be investigated.

An overview of the best known solutions is shown in Table 6.11.

	Dataset	IKH	IMT	INM	ISE	ALL
	Solution Cost	530	280	110	492	995
Best ALNS solution	**RS**	0	0	0	0	4
	CG	0	0	0	0	6
	CMD	26	7	2	21	65
	ND	15	14	6	18	19
	NLT	15	0	0	4	14

Table 6.10: An overview of the best ALNS identified solutions.

	Dataset	IKH	IMT	INM	ISE	ALL
	Solution Cost	527	280	110	492	988
Best known solution	**RS**	0	0	0	0	4
	CG	0	0	0	0	2
	CMD	26	7	2	21	65
	ND	15	14	6	18	19
	NLT	14	0	0	4	13

Table 6.11: An overview of the best known solutions.

CHAPTER 7

Discussion

This thesis consists of two major parts; defining a Mixed Integer Programming model for the Group Exam Timetabling Problem at RUC and solving this problem using the ALNS metaheuristic. The discussion section is therefore split as to deal with the modeling and the solution method separately.

7.1 Modeling

The MIP model developed to solve the Group Exam Timetabling problem at RUC is especially complex due to the model size; proving to be too big to solve using a standard MIP solver such as CPLEX. Also the defined model is somewhat basic and can be improved to increase its' realism and real world usefulness. As it stands now, the model is unlikely to be directly useful for RUC planners.

The model is defined with the focus of building timetables that caters to censors. No distinction is made between internal and external censors, although the actual situation is very different for the two groups of people. For internal censors, room changes, gaps and exams on different days, is not as big a problem as for external censors. Therefore the model could be improved by differentiating between external and internal censors. This can be done by defining sets C_{ex} and C_{in} to identify external and internal censors respectively. Notice that $C_{ex} \cup C_{in} = C$ and $C_{ex} \cap C_{in} = \emptyset$. Now different cost parameters for the different censor types can be introduced to increase the solution quality specifically for external censors. Additionally, constraints can be defined that only enforce restrictions for one of the censor groups.

This idea can be extended further by changing the model to only consider whether or not people are RUC staff. The model is redefined using the set of people P, which includes all external and internal persons associated with exams (not counting students) such that $P_{ex} \cup P_{in} = P$, where $P_{ex} = C_{ex}$ and $P_{in} = C_{in} \cup S$. The aim of the model is to produce good timetable specifically for external censors, and this model change allows for keeping that focus while also removing confusion associated with people whom are both censors and supervisors. This also eliminates model problems such as the fact, that a censor may receive a censor gap penalty if he/she functions as a supervisor in between two exams where he/she is the censor. Different weights should be defined for external and internal persons such that the produced timetables try to consider all people.

It is also possible to introduced new objectives, for example travel expenses. External censors have their their travel expenses related to attending their exams compensated by RUC. Therefore it would be very valuable to minimize the number of days which external censors have exams, i.e. their CMD count. This extra cost could even consider the size of the expenses related to each external censor, such that those whom are more monetarily costly also have a larger CMD cost. Such a change could be implemented by defining a cost p_c^{CMD} or simply by introducing a new parameter p_c^{travel} which is multiplied onto the CMD cost parameter for each external censor.

RUC planners know that supervisors and censors have very different wants and needs when it comes to their individual examination plan. Some would like to get all their exams done as soon as possible; even if that means having exams throughout a whole day with very few breaks. For others such a plan is unimaginable and they want very few examination per day. The model could be improved by the inclusion of personal preferences. The easiest would probably be to allow people (censors and supervisors) to specify their preferences using predefined personality profiles. Such profiles could for example specify, that a given person wants as few exam days as possible or that a person wants a maximum of n exams in a day. Additionally, people could be allowed to indicate which days they wold prefer to have their exams by "rating" all possible examination days what. Then the model should penalize whenever these personal preferences are not met. Also one problem with the current model is that breaks are only enforced when room changes happens between two conflicting exams. Thereby it is possible for a person to be assigned exams throughout an entire day without any breaks. In reality this is unlikely to be acceptable for the people involved, and therefore breaks should be inserted whenever the same people have some number of timeslots scheduled in a row. This could also be defined through personality profiles.

One of the issues when defining the One Day Model is that room stability is counted across days. This objective is included to decrease the number of room changes especially for external censors, whom are likely to not be familiar with RUC campus. Finding the location of a new room is not as stressful when returning on a new day as when between two exams during the same day. Therefore if the external censor has multiple days at RUC, it is not as important that the censor should use the same rooms as he/she did on other days. Rather that the number of room changes during a single day should be minimized. Therefore room stability could be counted on a daily basis without decreasing the practical value of the produced timetable by much. This in turn allows for more freedom in scheduling single days, improving both the solutions found when producing a whole timetable but especially when considering a single day. However, the increased freedom may make it more difficult to find optimal solutions when solving the One Day Model.

The "push exams back" constraint, which enforces that a day can only have exams scheduled if there are exams on the following day, is perhaps unnecessarily strict and causes some issues especially for the construction algorithm used in the ALNS solution method. It is up to RUC and their planners to decide, whether or not it is important that this constraint is enforced, but it is also an option to relax it and still obtain the benefits of its' intention. By redefining the constraint as $b_d \leq b_{d+1}$, it allows for

having empty days in the middle of the timetable, but at the cost of “opening” those days. Therefore the model will still reward a compact schedule at the end of available examination days, but it is allowed to have empty days in the middle if that results in an overall better solution.
An alternative way to include the push back effect, is to have different weight parameters for using different days. For example by introducing weights $p_0^D > p_1^D > \cdots > p_{d_{last}}^D$ for using day $0, 1, \ldots, d_{last}$ respectively and dropping the push exams back constraint. These weights could be multiplied by the number of exams on their respective days, such that the penalty increases depending on the number of exams on the given day instead of simply penalizing the usage of a day. These additions would make it favorable to produce compact timetables at the end of the available examination days and increase the freedom of the model.

The constraint included to ensure that conflicting exams are not held simultaneously is defined for all exams that have conflicting exams. Given the current constraint definition, some redundancy is introduced as multiple constraints will enforce that the same exam pairs cannot be scheduled simultaneously. However the constraint can be tightened even further by considering cliques. In graph theory a clique is defined as a complete subgraph of a given undirected graph, i.e. a subgraph where all vertices are adjacent to each other (Weisstein, 2017). For this problem, a graph can be defined such that exams are vertices and conflicting exams are connect by an edge. Figure 7.1 shows a clique of exams and Figure 7.2 shows a subgraph; both taken from the INM dataset. Instead of defining a constraint for each of the exams in Figure 7.1, it would be sufficient to define a constraint for that clique. The problem is then to identify the cliques for which constraints should be defined, in order to avoid including the mentioned redundancy. This can be done by finding all maximal cliques, cliques that cannot be extended by including more adjacent vertices, and then finding a minimum cover of the edges with regard to the number of cliques. In order to do so, all maximal cliques must first be enumerated, for example by using the backtracking algorithms presented in (Bron et al., 1972). These cliques are then used to solve the Edge Clique Cover problem on the exam conflict graph. This problem is shown to be NP-complete in (Kou et al., 1978), but different heuristics for solving the problem (no guarantee of optimality of course) are also presented.

Changing the conflicting exam constrains to use cliques is sure to reduce the number of require constraints. However, the overall effect of this change is unknown and warrants further research.

Although not directly related to the model, a small note is made on the unavailability data. All unavailability data is randomly generated and may not be very realistic. Of course, it would be best to get real data, but given the nature of said data it is unlikely that data collection is practically possible. The generated exam and censor availability can be very sporadically spread throughout the available examination days. It is more likely, that a person would be available on a few, mostly coherent days, and the generated data should reflect that.

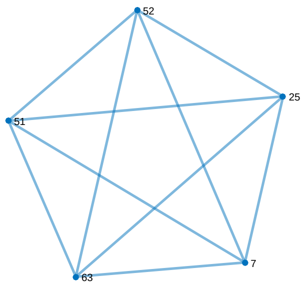

Figure 7.1: A clique of conflicting exams in the INM dataset.

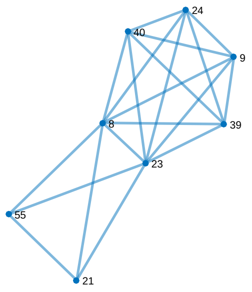

Figure 7.2: A subgraph of the conflicting exam graph for the INM dataset.

7.2 Solution Method

The result section discusses a significant problem with the ALNS implementation; the repeated acceptance of already visited solutions. This means that especially some destroy heuristics with bad performance are rewarded and chosen again and again, resulting in a lot of wasted iterations and time. The implementation could be greatly improved by including a scheme for keeping track of already seen solutions and only accepting those that have not been found before. A straightforward implementation is to hash solutions and store them in a hash table as done in (Ropke and Pisinger, 2006). Then only solution not found in the hast table should be accepted. This will prevent poorly performing heuristics from having large weights because of artificial success.

Even with the above mentioned addition, there are some destroy method which may still have lackluster performance. Much more testing should be done to identify these and the cause for their poor performance. With the given results it is difficult to make comments because of the lack of protection from acceptance of the same solution adds “noise” to the data which is difficult to identify without implementing said protection. However it is certain that some destroy methods should be changed in order to be more effect. For example the destroy method unscheduling all late exams could be changed to also unschedule surrounding exams or the associated censors other exams on the same day. This would give the repair heuristic more freedom and a better chance of finding improving solutions. If heuristics prove to be unsatisfactory they could entirely be removed from the method pool, however if the adaptive layer of the ALNS works correctly, it should be able to handle these cases and ensure that those methods are rarely chosen.

The two repair methods included in the ALNS are very similar, so it would likely be beneficial to develop different types of repair heuristics. One possibility is to implement a Regret$-k$ Repair heuristic as done in (Ropke and Pisinger, 2006), which tries to implement some look-ahead when repairing solutions. This could be an extension of the Strict Greedy Schedule heuristic. The difference is that regret$-k$ repair heuristic considers the k best schedulings for each exam and schedules the exam whose difference in cost between its' best scheduling and the $k-1$ best schedulings is the largest. That is, exams are scheduled where the chance of regret (potential cost) is largest if it is not done.

Another repair method that should be included is the One Day Model. The post processing done using the model shows great promise, improving the best solutions found through the ALNS for two of the datasets. The inclusion of the One Day Model within the ALNS is especially interesting if the proposed model change regarding daily room stability is implemented. Then the One Day Model will be able to produce solutions that are optimal when considering a single day, which it otherwise cannot because of the limitation implemented because of cross day room stability counting. In order to introduce this repair method much more testing is required. It is not included in the current implementation because building the MIP as well as solving it may sometimes be quite time consuming. For it to be usable as part of the ALNS, some balance between time consumption and solution quality should be ensured. Therefore testing should be done to figure out when using the One Day Model would be feasible. One possibility is introducing a exam threshold, such that the One Day Model is only attempted if there is less than a certain number of exams on the given day. However, as the results of the post processing shows, the number of exams is not the only deciding factor as to how long the MIP takes to solve. Therefore more testing is required.

In the implemented ALNS, new solutions can only be accepted if it is feasible, meaning that all exams should be scheduled. The Greedy Schedule repair heuristic should therefore not continue after finding an exam that it cannot schedule and should instead terminate immediately. This should be implemented by changing line 28 of Algorithm 2 to empty the set of unscheduled exams instead of simply removing the exam that could not be scheduled.

The max destroy parameter ξ is a very influential parameter on the overall performance of the ALNS implementation. This parameter is fixed for each run, but it could be interesting to investigate the effects dynamically adjusting the parameter during the search. That is, the ξ parameter could be lowered if more intensification is needed and increased when diversification is needed. Another option is to include different max destroy parameters, such that the same limit is not imposed on all methods. For example, the best ξ value found during testing is 10, which makes some destroy methods equal and thereby some redundancy is introduced. In these cases, there may be a benefit from having separate max destroy limits.

Since the simulated annealing acceptance criteria is relatively quickly put out of play, the search has no possibilities for escaping local optimums in its' later stages. If the search gets stuck, it should be diversified to continue somewhere else in the solution space. A possibility is the introduction of a “reheating” scheme, such that the temperature used in the acceptance criteria is increased when diversification is needed.

Parameter tuning could be done by tuning parameters one at a time as done in (Ropke and Pisinger, 2006), instead of trying all parameter combinations 10 times for 10 minutes. Thereby less parameter combinations will need testing, making it less cumbersome to have each parameter tuning run go on for longer. However through analysis of the results, there is no indication that any one parameter has a sudden positive effect after the initial 10 minutes. Therefore there would likely be no gain from changing how parameter tuning is executed.

Lastly, there is a slight performance gain from using the pairwise version of the ALNS as opposed the non-pairwise version. In benchmarking, the pairwise ALNS almost consistently had a lower average gap and spreading. Pairing the destroy and repair methods allows the adaptive layer of the ALNS to be more specific, resulting in better performance; although the difference is not immense.

7.2.1 Decomposition

Very few parts of the model tie the different days together and there is much potential for using decomposition to solve the Group Project Exam Timetabling problem at RUC. Especially if room stability is changed to be counted on a daily basis. At a master level decisions could be about about what exams should be scheduled on what days. This affects CMD and ND costs and should observe exam availability as well as whatever exam push back constraints may be imposed. Then on a daily basis, the specific scheduling of each exam should be decided, resulting in daily RS, CG and NLT costs. Thereby a Master Problem could be defined as a set partitioning problem of assigning exams to days and then a Sub Problem could be defined for each day, allocating exams to rooms and timeslots. The One Day Model (or a modified version of it) is an obvious choice for a sub problem definition.

A decomposition approach could take the shape of a purely mathematical approach, for example using column generation to find the partitioning of exams. But there is also potential for interplay between the programmatic and mathematical solution methods. Already it is proposed to include the One Day Model directly in the ALNS, where the ALNS effectively produces the sets of exams for each day and a MIP is used for planning the specific days. Although, as seen in the results section, the One Day Model can become too large to be solved in practice. Therefore it would also be interesting to go the other way, where a mathematical set partitioning problem is defined and the ALNS is used to schedule each day.

Exactly because of the problem structure, where there are very few ties between individual days of the timetable, the use of decomposition for solving this problem is very interesting and warrants more research.

7.3 Concluding Remarks

The goal of this thesis is to define a model for the Group Project Exam Timetabling problem at Roskilde University and attempt to solve it. A model is defined which proves to be too difficult to solve directly with standard MIP solvers. Therefore the problem is attempted solved using the ALNS metaheuristic. The implemented ALNS has good results, but some weaknesses have been identified, which undoubtedly have an impact on overall performance. Both in regard to the model and the solution method thoughts for improvement have been given.

In order for the work in this thesis to be directly usable by RUC staff, more development is required in cooperation with RUC. The model does have some weakness, for example the exclusion of general breaks, which means that the produced timetables may not be feasible in reality. Currently, RUC does the group exam timetabling throughout the semester, scheduling exams as the become available for scheduling. This is because they cannot wait until data has been collected for all exams. An initial support tool using the solution method proposed in this thesis, could be designed to help by allowing for iterative planning. For example, exams could be timetabled in waves, where already timetabled exams are fixed whenever a new set of exams are scheduled in the timetable. The work shows promise and could be, in collaboration with RUC, further developed to make a specialized and robust group project examination timetabling tool.

APPENDIX A

The Linear Programming Relaxation

A.1 The Linear Programming Relaxation

For many MIP solution methods, the LP relaxation of a problem needs to be solved repeatably. Therefore, in order for the method to terminate in a timely manner, solving the LP should be relatively easy and quick. An idea of how quickly the LP relaxation can be solved should be a guiding factor in developing a solution method.

In CPLEX six LP optimizers are available: Primal Simplex, Dual Simplex (the default setting), Network Simplex, Barrier, Sifting and Concurrent. All six methods are tested on the LP relaxation for all available datasets, to examine how relevant LP based methods may be to this problem. [1]

Primal Simplex: The "classic" LP solving algorithm. This method may especially work well on problems where the number of variables greatly exceed the number of constraints, or in the case that there is little variance in the cost coefficients.

Dual Simplex: The Simplex algorithm performed on the LP dual. Works well on primal-degenerate problems with little variability in the right hand side coefficients and great variability in the cost coefficients.

Network Simplex: Uses highly efficient network algorithms and works well on problems where a major part consists of a network structure.

Barrier: This method exploits a "primal-dual logarithmic barrier algorithm" and sequentially solves both primal and dual formulations to the problem. Primal feasibility, dual feasibility and the duality gap is used as a progress measure. This method is especially good on large and sparse problems.

Sifting: An extension to the simplex method where first a subproblem consisting of all rows and only a small subset of columns while assuming some lower bound on all other columns. Excluded columns are then added to the problem based on their reduced cost (evaluated using the subproblem solution). This solution method works

[1]details from: `https://www.ibm.com/support/knowledgecenter/en/SSSA5P_12.4.0/ilog.odms.cplex.help/CPLEX/User_manual/topics/uss_solveLP_4.html`

well on problems with a large number of columns compared to the number of rows; especially when an optimal solution is expected to have most variables at their lower bounds.

Concurrent: This solution method launches different solution methods on separate threads; based on the number of threads available. Given one thread, dual simplex is launched (default single thread LP method). With two threads available, the barrier method is run in the second. Primal simplex is run on the third thread. All additional threads are used to run the barrier method in parallel.

Each of the six methods are used to solve the LP relaxation of the problem on all five datasets. Each run is given a wall time limit of 4 hours and can use up to 20 threads and 128GB memory. The results are shown in Table A.1.

LP Method	Primal Simplex	Dual Simplex	Network Simplex	Barrier	Sifting	Concurrent
IKH	-	-	-	-	-	-
IMT	716	1,056	8,861	5,273	2,025	888
INM	5	8	47	74	335	10
ISE	-	-	-	-	-	-
All	-	-	-	-	-	-

Table A.1: Wall time (in seconds) for solving the LP relaxed version of the problem with each LP solution method. Models that could not be built or terminate within 4 hours (14,400 seconds) are shown with "-".

Since the model is to large to be built for dataset IKH and ALL, it is not possible to solve the LP relaxed version of the model on these dataset. Even though the model can be built on the ISE dataset, CPLEX crashes immediately after due to running out of memory. On the remaining two datasets, it the primal simplex outperforms the remaining LP solution methods. The Concurrent method is a multi threaded method using Dual Simplex, Primal Simplex and Barrier method, and it is through the Primal Simplex that the problem is solved. Primal Simplex likely works well on this model exactly because the number of constraints and (non-fixed) variables is so balanced and there is very little variability in the objective coefficients.

However, since it is not possible to solve (or even build) the models on some datasets, solution methods relying on solving the whole model are not very promising.

APPENDIX B ALNS Implementation

In B.1 the pseudo code for the Strict Greedy Schedule repair heuristic is shown. In B.2 overviews of all destroy and repair methods/pairs are given with their used designation/numbering.

B.1 Strict Greedy Schedule Repair Heuristic

Algorithm 4 Strict Greedy Schedule

```
1:  Given set $V$ of all exams to schedule
2:  while $|V| > 0$ do
3:      candidateCost = $\infty$, candidateScheduling = $\emptyset$
4:      for each exam $i \in V$ do
5:          for each day $d$ going backwards through available days do
6:              for each room $r$ with sufficient capacity in order of increasing capacity do
7:                  for $t = 0$ to $t_{last}$ do
8:                      if can schedule $i$ in $\{r, d, t\}$ then
9:                          if cost of scheduling = 0 then
10:                             schedule $i$ in $\{r, d, t\}$
11:                             $V = V \setminus i$
12:                             break to start exam loop again
13:                         else if cost of scheduling < candidateCost then
14:                             candidateScheduling = $\{i, r, d, t\}$
15:                             candidateCost = cost of scheduling
16:                         end if
17:                     else
18:                         if other exam scheduled at this time in this room then
19:                             $t$ = timeslot following end of other exam
20:                         end if
21:                     end if
22:                 end for
23:             end for
24:         end for
25:     end for
26:     if no exam has been scheduled and candidateScheduling found then
27:         schedule candidate using candidateScheduling and remove from $V$
28:     else if no candidateScheduling found then
29:         $V = \emptyset$
30:     end if
31: end while
```

B.2 Destroy and Repair Methods

Number	Name	Description
0	UnscheduleRandomExams(3-10)	Unschedules between 3 and 10 exams picked at random.
1	UnscheduleRandomExams(11-25)	Unschedules between 11 and 25 exams picked at random.
2	UnscheduleRandomExams(26-40)	Unschedules between 26 and 40 exams picked at random.
3	UnscheduleRandomDay	Unschedules all exams from a random day.
4	UnscheduleSlowestDay	Unschedules all exams from the day with the fewest exams.
5	UnscheduleRoomDay-CensorMostCMD	Unschedules all exams from rooms used by the censor with the largest CMD cost on days where the censor is scheduled.
6	UnscheduleCensorsMostCMD(1-5)	Unschedules all exams for a random number of censors (1 to 5) with the most CMD cost.
7	UnscheduleCostlyCensors(1-5)	Unschedules all exams associated with a random number of censors (1 to 5) with the highest total cost.
8	UnscheduleLateExams	Unschedules all exams that uses late timeslots.
9	UnscheduleRandomCensor(1-3)	Unschedules all exams associated with a random number of censors (1 to 3).
10	UnscheduleRandomCensor(4-10)	Unschedules all exams associated with a random number of censors (4 to 10).
11	UnscheduleFirstDay	Unschedules all exams from the first day in the schedule with exams.

Table B.1: Overview of destroy methods with numbering.

Number	Name	Description
0	StrictGreedySchedule	Schedules all unscheduled exams in random order using the Strict Greedy Schedule Heuristic.
1	GreedySchedule	Schedules all unscheduled exams in random order using the Greedy Schedule Heuristic.

Table B.2: Overview of repair methods with numbering.

Pair	Destroy	Repair
0	0	0
1	0	1
2	1	0
3	1	1
4	2	0
5	2	1
6	3	0
7	3	1
8	4	0
9	4	1
10	5	0
11	5	1
12	6	0
13	6	1
14	7	0
15	7	1
16	8	0
17	8	1
18	9	0
19	9	1
20	10	0
21	10	1
22	11	0
23	11	1

Table B.3: Overview of destroy-repair pairs with numbering.

APPENDIX C

Parameter Tuning

Tables showing the ξ and λ overviews for each parameter tuning setup are in C.1. In C.2 all parameter tuning data is found and C.3 shows the reward distributions of some ALNS runs on the ALL dataset.

C.1 Parameter ξ and λ overview

	E (%)			σ		
$\xi \backslash \lambda$	0.1	0.5	0.9	0.1	0.5	0.9
10	107.10	107.76	108.54	59.86	66.07	58.74
25	138.73	140.43	145.61	66.09	63.61	67.33
50	150.23	156.68	159.08	78.89	62.21	62.98
∞	167.65	172.36	177.19	96.44	84.82	103.76

Table C.1: The mean average gap E (%) and spreading σ across all parameter setting for α and w with different ξ and λ combinations when parameter tuning the non-pairwise ALNS on the ALL dataset.

	E (%)			σ		
$\xi \backslash \lambda$	0.1	0.5	0.9	0.1	0.5	0.9
10	9.42	9.66	10.98	16.09	17.01	16.89
25	30.57	32.00	32.38	19.97	20.09	19.76
50	42.63	45.94	46.16	25.88	23.52	26.22
∞	49.00	53.67	59.76	29.94	27.47	28.67

Table C.2: The mean average gap E (%) and spreading σ across all parameter setting for α and w with different ξ and λ combinations when parameter tuning the pairwise ALNS on the ISE dataset.

	E (%)			σ		
$\xi \backslash \lambda$	0.1	0.5	0.9	0.1	0.5	0.9
10	7.71	9.03	7.36	12.53	17.42	13.63
25	29.36	30.54	31.53	20.62	16.30	18.53
50	41.04	43.37	45.79	22.35	25.82	25.04
∞	48.02	52.70	55.39	26.76	27.84	29.09

Table C.3: The mean average gap E (%) and spreading σ across all parameter setting for α and w with different ξ and λ combinations when parameter tuning the non-pairwise ALNS on the ISE dataset.

C.2 Parameter Combination Performance Overviews

C.2.1 Pairwise ALNS - ALL

$\xi = \mathbf{25}$

$\xi = 25$

	$\lambda = 0.1$				$\lambda = 0.5$				$\lambda = 0.9$			
$\alpha\backslash\omega$	0.01	0.05	0.10	0.15	0.01	0.05	0.10	0.15	0.01	0.05	0.10	0.15
0.999	71.13	90.67	108.87	120.50	68.28	108.07	112.20	118.69	74.74	102.71	127.13	134.68
0.99975	128.27	163.88	170.25	171.78	133.22	166.28	171.23	174.50	141.60	169.16	173.34	173.96
0.9999	140.52	164.09	169.04	176.04	148.29	167.34	172.09	176.14	149.97	172.56	172.91	175.02

Table C.4: The average gap (%) for all parameter combinations with $\xi = 25$ on the ALL dataset with the pairwise ALNS.

$\xi = 25$

	$\lambda = 0.1$				$\lambda = 0.5$				$\lambda = 0.9$			
$\alpha\backslash\omega$	0.01	0.05	0.10	0.15	0.01	0.05	0.10	0.15	0.01	0.05	0.10	0.15
0.999	49.67	93.87	131.80	153.88	50.61	92.86	150.03	117.50	74.27	121.71	73.14	122.27
0.99975	80.48	54.83	82.55	63.41	73.28	62.74	52.35	55.73	64.28	76.08	30.85	60.36
0.9999	50.72	34.53	57.24	40.93	81.45	63.06	44.72	68.80	49.78	48.40	38.73	55.89

Table C.5: The spreading (σ) for all parameter combinations with $\xi = 25$ on the ALL dataset with the pairwise ALNS.

$\xi = 50$

	$\xi = 50$											
	$\lambda = 0.1$				$\lambda = 0.5$				$\lambda = 0.9$			
$\alpha \backslash \omega$	0.01	0.05	0.10	0.15	0.01	0.05	0.10	0.15	0.01	0.05	0.10	0.15
0.999	94.24	114.33	140.64	149.78	83.43	127.25	148.03	159.21	92.81	145.26	153.99	171.89
0.99975	133.61	167.16	176.41	179.99	142.35	171.30	178.66	180.73	141.85	175.20	180.41	181.05
0.9999	142.76	171.14	175.05	179.44	148.91	177.58	181.58	184.11	155.35	178.56	181.61	183.88

Table C.6: The average gap (%) for all parameter combinations with $\xi = 50$ on the ALL dataset with the pairwise ALNS.

	$\xi = 50$											
	$\lambda = 0.1$				$\lambda = 0.5$				$\lambda = 0.9$			
$\alpha \backslash \omega$	0.01	0.05	0.10	0.15	0.01	0.05	0.10	0.15	0.01	0.05	0.10	0.15
0.999	38.30	94.16	104.76	115.72	60.52	118.84	104.51	87.46	50.68	112.81	77.34	78.11
0.99975	87.58	52.47	67.54	64.62	92.12	55.40	47.27	58.99	66.00	57.73	56.55	69.43
0.9999	97.78	69.16	67.49	50.49	62.18	56.61	28.75	75.38	68.37	52.65	81.77	63.24

Table C.7: The spreading (σ) for all parameter combinations with $\xi = 50$ on the ALL dataset with the pairwise ALNS.

$\xi = \infty$

	$\xi = \infty$											
	$\lambda = 0.1$				$\lambda = 0.5$				$\lambda = 0.9$			
$\alpha \backslash \omega$	0.01	0.05	0.10	0.15	0.01	0.05	0.10	0.15	0.01	0.05	0.10	0.15
0.999	126.52	163.03	173.68	183.97	125.63	169.76	175.48	185.46	125.19	183.63	184.33	189.79
0.99975	150.37	175.28	182.23	181.91	160.16	181.39	183.66	188.50	163.62	185.02	192.28	187.34
0.9999	154.33	175.81	176.52	179.24	164.70	182.88	186.59	183.99	167.05	184.98	189.60	191.27

Table C.8: The average gap (%) for all parameter combinations with $\xi = \infty$ on the ALL dataset with the pairwise ALNS.

	$\xi = \infty$											
	$\lambda = 0.1$				$\lambda = 0.5$				$\lambda = 0.9$			
$\alpha \backslash \omega$	0.01	0.05	0.10	0.15	0.01	0.05	0.10	0.15	0.01	0.05	0.10	0.15
0.999	113.69	103.74	77.02	60.58	139.66	129.76	104.31	44.70	190.19	102.01	90.66	96.90
0.99975	77.35	79.01	70.41	55.37	101.95	82.26	84.70	122.39	82.70	138.30	61.73	78.07
0.9999	90.45	60.56	59.61	73.41	94.01	61.99	52.67	85.58	58.43	114.51	94.71	71.76

Table C.9: The spreading (σ) for all parameter combinations with $\xi = \infty$ on the ALL dataset with the pairwise ALNS.

C.2.2 Non-pairwise ALNS - ALL

$\xi = \mathbf{10}$

	$\xi = 10$											
	$\lambda = 0.1$				$\lambda = 0.5$				$\lambda = 0.9$			
$\alpha\backslash\omega$	0.01	0.05	0.10	0.15	0.01	0.05	0.10	0.15	0.01	0.05	0.10	0.15
0.999	45.93	54.72	57.70	56.95	49.75	47.63	52.75	55.62	40.07	47.59	53.43	52.72
0.99975	79.51	133.35	143.42	146.85	88.09	138.20	145.38	146.58	93.80	137.81	149.68	147.29
0.9999	120.46	146.18	150.32	149.82	124.43	147.77	149.26	147.66	131.13	149.57	150.50	148.90

Table C.10: The average gap (%) for all parameter combinations with $\xi = 10$ on the ALL dataset with the non-pairwise ALNS.

	$\xi = 10$											
	$\lambda = 0.1$				$\lambda = 0.5$				$\lambda = 0.9$			
$\alpha\backslash\omega$	0.01	0.05	0.10	0.15	0.01	0.05	0.10	0.15	0.01	0.05	0.10	0.15
0.999	51.28	59.59	64.50	62.61	60.31	79.25	60.72	48.41	42.05	58.41	72.79	44.24
0.99975	59.06	54.73	78.68	80.17	122.38	69.88	56.48	54.14	76.19	90.82	59.29	52.79
0.9999	74.22	33.94	50.73	48.76	66.56	64.57	51.34	58.84	43.61	74.65	45.31	44.74

Table C.11: The spreading (σ) for all parameter combinations with $\xi = 10$ on the ALL dataset with the non-pairwise ALNS.

$\xi = 25$

	$\xi = 25$											
	$\lambda = 0.1$				$\lambda = 0.5$				$\lambda = 0.9$			
$\alpha\backslash\omega$	0.01	0.05	0.10	0.15	0.01	0.05	0.10	0.15	0.01	0.05	0.10	0.15
0.999	75.01	88.22	98.65	118.84	68.67	88.74	104.92	119.57	77.99	97.05	123.36	127.89
0.99975	124.25	158.83	171.00	174.68	128.09	164.91	171.09	175.81	136.48	170.80	173.58	172.80
0.9999	138.15	168.53	171.89	176.66	146.04	171.19	172.31	173.74	149.52	170.24	170.72	176.93

Table C.12: The average gap (%) for all parameter combinations with $\xi = 25$ on the ALL dataset with the non-pairwise ALNS.

	$\xi = 25$											
	$\lambda = 0.1$				$\lambda = 0.5$				$\lambda = 0.9$			
$\alpha\backslash\omega$	0.01	0.05	0.10	0.15	0.01	0.05	0.10	0.15	0.01	0.05	0.10	0.15
0.999	60.47	67.94	82.48	93.71	51.33	88.39	90.19	76.10	83.70	88.18	133.52	99.55
0.99975	56.99	86.57	54.47	47.82	78.10	63.79	52.82	39.78	44.12	40.46	61.08	33.88
0.9999	65.63	48.99	75.44	52.58	55.00	44.32	77.58	45.91	47.80	56.42	57.77	61.43

Table C.13: The spreading (σ) for all parameter combinations with $\xi = 25$ on the ALL dataset with the non-pairwise ALNS.

$\xi = 50$

	$\xi = 50$											
	$\lambda = 0.1$				$\lambda = 0.5$				$\lambda = 0.9$			
$\alpha\backslash\omega$	0.01	0.05	0.10	0.15	0.01	0.05	0.10	0.15	0.01	0.05	0.10	0.15
0.999	90.06	114.25	136.59	143.54	84.33	132.54	153.70	159.02	85.38	130.82	156.42	160.51
0.99975	133.24	160.71	173.50	179.01	139.27	175.08	175.70	179.77	145.44	174.61	179.21	181.21
0.9999	143.31	171.56	176.32	180.66	146.53	175.71	176.35	182.22	156.27	177.26	179.39	182.51

Table C.14: The average gap (%) for all parameter combinations with $\xi = 50$ on the ALL dataset with the non-pairwise ALNS.

	$\xi = 50$											
	$\lambda = 0.1$				$\lambda = 0.5$				$\lambda = 0.9$			
$\alpha\backslash\omega$	0.01	0.05	0.10	0.15	0.01	0.05	0.10	0.15	0.01	0.05	0.10	0.15
0.999	59.49	116.03	95.56	95.25	41.17	69.56	74.42	105.55	77.22	76.91	88.71	62.20
0.99975	99.18	110.95	62.00	49.58	56.30	62.31	45.35	39.45	52.34	63.42	38.70	26.33
0.9999	40.28	101.91	68.83	47.57	80.44	50.42	57.77	63.82	61.34	86.07	53.56	68.94

Table C.15: The spreading (σ) for all parameter combinations with $\xi = 50$ on the ALL dataset with the non-pairwise ALNS.

$\xi = \infty$

$\xi = \infty$

	$\lambda = 0.1$				$\lambda = 0.5$				$\lambda = 0.9$			
$\alpha \backslash \omega$	0.01	0.05	0.10	0.15	0.01	0.05	0.10	0.15	0.01	0.05	0.10	0.15
0.999	128.35	164.06	169.94	179.94	122.37	164.63	183.82	186.33	137.99	169.43	184.61	187.15
0.99975	149.59	174.30	178.91	186.62	152.86	177.03	183.97	182.85	161.31	189.66	183.59	183.66
0.9999	153.42	171.04	175.50	180.08	158.54	185.36	185.68	184.87	168.30	187.24	190.28	183.04

Table C.16: The average gap (%) for all parameter combinations with $\xi = \infty$ on the ALL dataset with the non-pairwise ALNS.

$\xi = \infty$

	$\lambda = 0.1$				$\lambda = 0.5$				$\lambda = 0.9$			
$\alpha \backslash \omega$	0.01	0.05	0.10	0.15	0.01	0.05	0.10	0.15	0.01	0.05	0.10	0.15
0.999	154.45	119.85	144.14	54.61	147.93	119.70	95.66	100.04	231.94	143.14	94.79	118.81
0.99975	96.64	97.75	102.58	79.64	69.96	61.76	63.23	77.53	91.49	73.83	93.91	59.23
0.9999	96.26	56.95	66.47	87.95	94.49	83.51	34.68	69.30	77.14	54.26	93.65	112.88

Table C.17: The spreading (σ) for all parameter combinations with $\xi = \infty$ on the ALL dataset with the non-pairwise ALNS.

C.2.3 Pairwise ALNS - ISE

$\xi = \mathbf{10}$

	$\xi = 10$											
	$\lambda = 0.1$				$\lambda = 0.5$				$\lambda = 0.9$			
$\alpha \backslash \omega$	0.01	0.05	0.10	0.15	0.01	0.05	0.10	0.15	0.01	0.05	0.10	0.15
0.999	5.18	6.89	6.95	5.69	4.31	4.17	3.92	4.67	3.96	4.00	3.54	4.70
0.99975	6.18	5.63	6.22	5.59	4.94	5.14	5.00	6.30	4.98	4.27	4.57	5.73
0.9999	5.91	9.35	18.46	31.04	4.47	10.08	20.63	42.28	4.19	15.22	31.48	45.14

Table C.18: The average gap (%) for all parameter combinations with $\xi = 10$ on the ISE dataset with the pairwise ALNS.

	$\xi = 10$											
	$\lambda = 0.1$				$\lambda = 0.5$				$\lambda = 0.9$			
$\alpha \backslash \omega$	0.01	0.05	0.10	0.15	0.01	0.05	0.10	0.15	0.01	0.05	0.10	0.15
0.999	8.73	16.36	15.78	10.95	9.20	9.37	8.10	7.35	10.69	6.40	8.06	7.40
0.99975	10.50	9.25	6.65	10.47	9.61	8.46	6.04	6.62	5.16	8.41	13.71	9.80
0.9999	10.94	12.16	33.66	47.65	5.06	18.67	48.54	67.11	5.35	43.81	42.31	41.60

Table C.19: The spreading (σ) for all parameter combinations with $\xi = 10$ on the ISE dataset with the pairwise ALNS.

$\xi = \mathbf{25}$

$\xi = 25$

	$\lambda = 0.1$				$\lambda = 0.5$				$\lambda = 0.9$			
$\alpha \backslash \omega$	0.01	0.05	0.10	0.15	0.01	0.05	0.10	0.15	0.01	0.05	0.10	0.15
0.999	15.41	14.70	14.88	13.84	15.47	12.87	13.19	13.27	10.16	12.78	11.83	12.28
0.99975	14.67	15.43	19.09	21.81	14.07	17.40	16.67	19.98	11.87	14.04	19.17	18.98
0.9999	16.00	48.68	81.59	90.81	15.83	62.30	84.23	98.76	16.06	70.85	91.28	99.23

Table C.20: The average gap (%) for all parameter combinations with $\xi = 25$ on the ISE dataset with the pairwise ALNS.

$\xi = 25$

	$\lambda = 0.1$				$\lambda = 0.5$				$\lambda = 0.9$			
$\alpha \backslash \omega$	0.01	0.05	0.10	0.15	0.01	0.05	0.10	0.15	0.01	0.05	0.10	0.15
0.999	15.80	16.61	17.93	12.27	21.04	16.12	15.83	15.29	10.56	12.32	11.85	14.42
0.99975	11.98	22.51	20.32	13.89	15.38	16.63	23.19	20.05	10.62	17.48	24.35	21.75
0.9999	13.42	28.99	31.82	34.11	18.51	36.34	20.21	22.46	17.41	21.87	36.61	37.89

Table C.21: The spreading (σ) for all parameter combinations with $\xi = 25$ on the ISE dataset with the pairwise ALNS.

$\xi = \mathbf{50}$

$\xi = 50$

	$\lambda = 0.1$				$\lambda = 0.5$				$\lambda = 0.9$			
$\alpha \backslash \omega$	0.01	0.05	0.10	0.15	0.01	0.05	0.10	0.15	0.01	0.05	0.10	0.15
0.999	24.41	22.52	20.91	26.54	23.68	22.97	24.78	21.67	20.41	20.24	21.99	23.60
0.99975	21.16	24.17	32.09	35.61	21.40	27.52	37.42	48.11	19.67	24.33	38.60	54.98
0.9999	24.49	81.44	98.39	99.78	25.59	87.97	102.22	107.91	27.28	84.82	107.62	110.43

Table C.22: The average gap (%) for all parameter combinations with $\xi = 50$ on the ISE dataset with the pairwise ALNS.

	$\xi = 50$											
	$\lambda = 0.1$				$\lambda = 0.5$				$\lambda = 0.9$			
$\alpha \backslash \omega$	0.01	0.05	0.10	0.15	0.01	0.05	0.10	0.15	0.01	0.05	0.10	0.15
0.999	20.93	21.51	23.09	25.12	14.29	17.54	11.41	28.92	22.62	15.13	25.33	20.20
0.99975	23.88	18.59	23.77	35.75	20.68	21.39	30.08	50.09	20.68	23.62	39.75	45.10
0.9999	16.82	43.22	20.28	37.54	14.10	30.79	18.94	24.01	17.33	31.50	32.87	20.48

Table C.23: The spreading (σ) for all parameter combinations with $\xi = 50$ on the ISE dataset with the pairwise ALNS.

$\xi = \infty$

	$\xi = \infty$											
	$\lambda = 0.1$				$\lambda = 0.5$				$\lambda = 0.9$			
$\alpha \backslash \omega$	0.01	0.05	0.10	0.15	0.01	0.05	0.10	0.15	0.01	0.05	0.10	0.15
0.999	27.48	27.87	28.82	29.65	28.41	27.66	27.22	22.97	27.58	26.75	25.91	24.86
0.99975	27.58	32.24	43.96	50.71	26.65	35.83	53.03	72.72	25.96	48.31	73.64	86.18
0.9999	32.68	85.28	99.72	102.03	36.28	94.25	107.24	111.83	45.30	103.21	112.95	116.46

Table C.24: The average gap (%) for all parameter combinations with $\xi = \infty$ on the ISE dataset with the pairwise ALNS.

	$\xi = \infty$											
	$\lambda = 0.1$				$\lambda = 0.5$				$\lambda = 0.9$			
$\alpha \backslash \omega$	0.01	0.05	0.10	0.15	0.01	0.05	0.10	0.15	0.01	0.05	0.10	0.15
0.999	17.93	20.74	19.20	26.86	15.35	18.76	22.15	16.66	21.83	29.25	23.06	16.79
0.99975	18.55	22.93	50.86	47.42	23.42	26.40	43.57	54.25	18.85	40.81	43.89	42.80
0.9999	38.98	36.43	35.08	24.35	25.55	20.45	31.48	31.53	32.34	27.85	31.95	14.64

Table C.25: The spreading (σ) for all parameter combinations with $\xi = \infty$ on the ISE dataset with the pairwise ALNS.

C.2.4 Non-pairwise ALNS - ISE

$\xi = \mathbf{10}$

	$\xi = 10$											
	$\lambda = 0.1$				$\lambda = 0.5$				$\lambda = 0.9$			
$\alpha \backslash \omega$	0.01	0.05	0.10	0.15	0.01	0.05	0.10	0.15	0.01	0.05	0.10	0.15
0.999	6.71	5.73	5.73	6.50	5.10	5.06	3.92	5.37	4.21	3.01	3.64	4.13
0.99975	6.65	4.11	6.67	6.24	5.06	5.61	5.69	5.04	4.94	4.67	4.37	4.80
0.9999	5.41	6.91	8.09	23.76	5.24	7.64	17.93	36.73	3.41	8.31	24.31	18.52

Table C.26: The average gap (%) for all parameter combinations with $\xi = 10$ on the ISE dataset with the non-pairwise ALNS.

	$\xi = 10$											
	$\lambda = 0.1$				$\lambda = 0.5$				$\lambda = 0.9$			
$\alpha \backslash \omega$	0.01	0.05	0.10	0.15	0.01	0.05	0.10	0.15	0.01	0.05	0.10	0.15
0.999	9.20	9.25	9.63	10.81	9.28	10.90	6.12	6.18	6.62	6.68	10.06	7.35
0.99975	8.99	5.67	12.50	7.59	9.51	10.93	6.68	5.49	9.08	12.92	5.92	8.72
0.9999	13.92	9.95	19.54	33.35	11.84	9.66	45.16	77.27	8.80	12.21	37.33	37.89

Table C.27: The spreading (σ) for all parameter combinations with $\xi = 10$ on the ISE dataset with the non-pairwise ALNS.

$\xi = \mathbf{25}$

	$\xi = 25$											
	$\lambda = 0.1$				$\lambda = 0.5$				$\lambda = 0.9$			
$\alpha \backslash \omega$	0.01	0.05	0.10	0.15	0.01	0.05	0.10	0.15	0.01	0.05	0.10	0.15
0.999	15.12	15.53	12.99	13.37	13.90	12.99	12.62	13.92	11.89	12.54	13.56	12.60
0.99975	12.48	15.20	16.14	19.90	12.09	15.18	16.54	15.79	9.49	14.11	16.50	17.74
0.9999	15.04	47.17	78.58	90.73	13.41	61.10	84.39	94.53	14.57	67.93	89.35	98.03

Table C.28: The average gap (%) for all parameter combinations with $\xi = 25$ on the ISE dataset with the non-pairwise ALNS.

	$\xi = 25$											
	$\lambda = 0.1$				$\lambda = 0.5$				$\lambda = 0.9$			
$\alpha\backslash\omega$	0.01	0.05	0.10	0.15	0.01	0.05	0.10	0.15	0.01	0.05	0.10	0.15
0.999	15.46	14.74	13.95	12.54	16.35	13.52	12.20	11.78	15.17	16.70	14.20	15.43
0.99975	17.14	18.62	25.30	24.67	13.63	8.78	15.62	17.01	8.38	22.17	15.55	13.98
0.9999	16.48	38.94	24.84	24.77	10.80	21.81	32.94	21.18	18.10	25.63	32.93	24.17

Table C.29: The spreading (σ) for all parameter combinations with $\xi = 25$ on the ISE dataset with the non-pairwise ALNS.

$\xi = 50$

	$\xi = 50$											
	$\lambda = 0.1$				$\lambda = 0.5$				$\lambda = 0.9$			
$\alpha\backslash\omega$	0.01	0.05	0.10	0.15	0.01	0.05	0.10	0.15	0.01	0.05	0.10	0.15
0.999	22.60	24.74	23.58	23.48	22.34	19.57	21.52	23.35	19.51	19.96	20.45	18.41
0.99975	22.50	25.18	27.99	30.98	19.86	26.10	31.02	39.63	19.53	23.07	32.32	61.69
0.9999	22.93	78.80	92.15	97.54	25.06	82.91	103.21	105.83	29.35	91.81	104.04	109.33

Table C.30: The average gap (%) for all parameter combinations with $\xi = 50$ on the ISE dataset with the non-pairwise ALNS.

	$\xi = 50$											
	$\lambda = 0.1$				$\lambda = 0.5$				$\lambda = 0.9$			
$\alpha\backslash\omega$	0.01	0.05	0.10	0.15	0.01	0.05	0.10	0.15	0.01	0.05	0.10	0.15
0.999	20.57	17.37	23.06	16.44	24.13	14.46	15.83	13.85	19.75	12.88	16.52	27.00
0.99975	15.84	17.29	26.45	38.27	15.75	21.29	19.55	47.70	15.39	24.59	22.79	44.11
0.9999	14.50	23.31	31.91	23.16	24.71	39.31	41.67	31.62	27.61	26.51	34.40	28.79

Table C.31: The spreading (σ) for all parameter combinations with $\xi = 50$ on the ISE dataset with the non-pairwise ALNS.

$\xi = \infty$

	$\xi = \infty$											
	$\lambda = 0.1$				$\lambda = 0.5$				$\lambda = 0.9$			
$\alpha\backslash\omega$	0.01	0.05	0.10	0.15	0.01	0.05	0.10	0.15	0.01	0.05	0.10	0.15
0.999	29.27	28.05	27.30	29.02	29.00	28.05	26.52	24.39	24.55	23.90	26.26	23.82
0.99975	27.44	33.21	39.23	51.50	25.51	34.90	53.72	66.77	22.40	30.30	68.48	77.20
0.9999	29.19	83.60	98.60	99.78	38.05	91.63	105.28	108.54	49.63	94.72	109.96	113.41

Table C.32: The average gap (%) for all parameter combinations with $\xi = \infty$ on the ISE dataset with the non-pairwise ALNS.

	$\xi = \infty$											
	$\lambda = 0.1$				$\lambda = 0.5$				$\lambda = 0.9$			
$\alpha\backslash\omega$	0.01	0.05	0.10	0.15	0.01	0.05	0.10	0.15	0.01	0.05	0.10	0.15
0.999	24.52	21.30	30.90	19.17	26.08	18.67	22.42	21.48	19.04	23.57	25.57	17.89
0.99975	15.30	23.95	22.78	38.80	24.83	16.56	46.15	62.07	20.63	21.60	57.41	37.14
0.9999	24.61	34.83	29.87	35.05	24.52	21.71	18.30	31.32	38.25	40.84	24.23	22.89

Table C.33: The spreading (σ) for all parameter combinations with $\xi = \infty$ on the ISE dataset with the non-pairwise ALNS.

C.3 Reward Distribution

Reward distributions for the first 3,000 and all iterations of ALNS runs on the ALL dataset.

Pair	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
ω_1	29	25	26	24	30	38	33	38	36	39	39	33	43	46	39	50	0	0	33	41	30	38	46	36
ω_2	102	77	64	53	45	60	72	80	17	59	44	52	35	28	21	16	151	166	79	100	34	61	29	90
ω_3	40	33	42	30	20	31	23	23	2	9	20	29	15	22	44	21	0	0	19	26	30	34	4	19
ω_4	12	7	8	16	5	15	2	6	0	1	5	5	2	2	14	13	0	0	3	7	1	10	0	3

Table C.34: The amount of each reward given to all destroy-repair pairs through the pairwise $\xi = 10$ run on the ALL dataset.

Pair	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
ω_1	26	24	28	32	42	27	38	22	19	14	19	33	33	30	29	29	0	0	27	29	21	30	14	12
ω_2	101	63	58	66	27	18	5	2	178	207	33	41	15	8	8	2	316	320	42	66	11	13	184	173
ω_3	20	18	57	42	23	12	0	0	13	10	12	35	15	7	15	4	0	0	12	10	4	5	22	10
ω_4	7	6	9	21	23	4	0	0	1	4	7	7	1	4	11	10	0	0	0	2	0	1	0	1

Table C.35: The amount of each reward given to all destroy-repair pairs through the the first 3,000 of the pairwise $\xi = \infty$ run on the ALL dataset.

Destroy/ Repair	D0	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10	D11	R0	R1
ω_1	73	72	75	84	74	82	78	111	0	80	73	67	431	438
ω_2	174	131	143	123	41	122	54	38	378	141	85	57	718	769
ω_3	50	58	57	47	8	55	30	61	0	36	55	15	240	232
ω_4	16	16	31	20	0	14	7	44	0	9	14	1	85	87

Table C.36: The amount of each reward given to all destroy and repair methods through the first 3,000 iterations of the non-pairwise $\xi = 10$ run on the ALL dataset.

Destroy/ Repair	D0	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10	D11	R0	R1
ω_1	71	90	77	69	73	76	64	73	0	82	70	59	381	423
ω_2	220	109	48	13	176	115	39	44	483	187	52	175	745	916
ω_3	62	73	57	2	14	58	26	63	0	32	14	8	200	209
ω_4	16	34	10	1	1	18	6	29	0	5	2	4	55	71

Table C.37: The amount of each reward given to all destroy and repair methods through the first 3,000 iterations of the non-pairwise $\xi = \infty$ run on the ALL dataset.

Destroy/ Repair	D0	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10	D11	R0	R1
ω_1	24,996	19,208	19,221	22,730	19,809	13,615	9,273	9,845	0	22,070	18,297	19,299	98,968	99,395
ω_2	15,784	5,108	5,039	9,440	23,586	2,122	313	410	100,378	8,300	4214	22,457	98,143	99,008
ω_3	50	58	61	47	8	56	30	61	0	37	60	15	243	240
ω_4	56	45	68	61	2	16	7	51	0	42	95	1	233	211

Table C.38: The amount of each reward given to all destroy and repair methods through all iterations of the non-pairwise $\xi = 10$ run on the ALL dataset.

Destroy/ Repair	D0	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10	D11	R0	R1
ω_1	3,123	1,824	1,584	1,634	2,388	1,791	1,550	1,565	0	2,442	1,528	2,442	10,639	11,232
ω_2	2,306	277	66	113	1,016	287	71	63	18,468	1,044	71	1,158	10,516	14,424
ω_3	62	73	57	2	14	60	26	63	0	32	14	8	200	211
ω_4	116	65	19	1	2	25	7	44	0	52	8	4	147	196

Table C.39: The amount of each reward given to all destroy and repair methods through all iterations of the non-pairwise $\xi = \infty$ run on the ALL dataset.

Bibliography

Bron, C., J. Kerbosch, and H. Schell. *Finding Cliques in a Undirected Graph.* Department of Industrial Engineering, University of Technology, 1972.

Carter, M. W., G. Laporte, and S. Y. Lee. “Examination timetabling: Algorithmic strategies and applications”. In: *Journal of the Operational Research Society* 47.3 (1996), pages 373–383.

Kahar, M. N. M. and G. Kendall. “The examination timetabling problem at Universiti Malaysia Pahang: Comparison of a constructive heuristic with an existing software solution”. In: *European journal of operational research* 207.2 (2010), pages 557–565.

Kou, L. T., L. J. Stockmeyer, and C.-K. Wong. “Covering edges by cliques with regard to keyword conflicts and intersection graphs”. In: *Communications of the ACM* 21.2 (1978), pages 135–139.

McCollum, B. et al. *The second international timetabling competition: Examination timetabling track.* Technical report. Technical Report QUB/IEEE/Tech/ITC2007/-Exam/v4. 0/17, Queen’s University, Belfast, 2007.

Qu, R. et al. “A survey of search methodologies and automated system development for examination timetabling”. In: *Journal of scheduling* 12.1 (2009), pages 55–89.

Ribeiro, G. M. and G. Laporte. “An adaptive large neighborhood search heuristic for the cumulative capacitated vehicle routing problem”. In: *Computers & operations research* 39.3 (2012), pages 728–735.

Ropke, S. and D. Pisinger. “An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows”. In: *Transportation science* 40.4 (2006), pages 455–472.

Shaw, P. “Using constraint programming and local search methods to solve vehicle routing problems”. In: *International Conference on Principles and Practice of Constraint Programming.* Springer. 1998, pages 417–431.

Stidsen, T. and L. Reinhardt. “GRASP and ALNS”. Techincal University of Denmark Course 42137 - Optimization using metaheuristics: Lecture Slides. 2016.

Weisstein, E. W. *Clique From MathWorld–A Wolfram Web Resource*. 2017. URL: `%5Curl%7Bhttp://mathworld.wolfram.com/Clique.html%7D` (visited on December 29, 2017).

Welsh, D. J. and M. B. Powell. “An upper bound for the chromatic number of a graph and its application to timetabling problems”. In: *The Computer Journal* 10.1 (1967), pages 85–86.

Wren, A. “Scheduling, timetabling and rostering—a special relationship?” In: *International Conference on the Practice and Theory of Automated Timetabling*. Springer. 1995, pages 46–75.