The School of Mathematics

THE UNIVERSITY
of EDINBURGH

Algorithms for the real-life examination timetabling problem at The University of Edinburgh

by

Pim Schutman

Dissertation Presented for the Degree of
MSc in Operational Research with Computational Optimization

August 2019

Supervised by
Dr. Sergio García Quiles

Abstract

A new setting for the real-life examination timetabling problem is introduced and applied to The University of Edinburgh. In the real-life examination timetabling problem, an institution faces the problem of scheduling a set of exams within several periods using a set of limited resources. A construction algorithm is proposed that first groups exams in clusters using a saturation degree method. We exploit the non-conflicting property of the clusters by scheduling exams from the same cluster adjacent to each other. The construction algorithm is shown to be a hybrid between a sequential method and a cluster method and a corresponding weighting parameter α is introduced to shift between the two methods. Furthermore, two different improvement algorithms are presented, which model a penalty function that incorporates all soft constraints. A neighborhood where single exams are moved in the timetable is studied. Throughout computational experiments we assess the performance of the proposed algorithms.

Acknowledgements

I would like to express my gratitude to my supervisor, dr Sergio García Quiles, for offering me the opportunity to write my dissertation in this topic. Without his help, guidance and constructive comments it would not have been possible to write this dissertation.

I would like to extend my gratitude to the timetabling team of The University of Edinburgh for providing me with valuable insights on their scheduling procedure, which has helped me to characterize the problem.

Finally, I also would like to thank my family for their support and making it possible for me to study this year abroad in Edinburgh.

Own work declaration

I declare that this thesis was composed by myself and that the work contained therein is my own, except where explicitly stated otherwise in the text.

(Pim Schutman)

Contents

1 Introduction

1.1 Problem Introduction

Among the wide scale of topics studied in the field of Operational Research, scheduling problems have taken a prominent place over the past decades. *The Oxford Dictionary* defines scheduling as *"The arrangement or planning of an event to take place at a particular time"*. Therefore, the practice of constructing schedules comprises a series of decisions to arrange multiple events to different time slots. One could imagine that the possibilities to construct such schedules are practically endless. As a result, finding an optimal solution to a scheduling problem can become remarkably complex. When done manually, the process of optimizing multiple objectives while satisfying large sets of constraints leaves finding feasible solutions to be an undoubtedly tedious task, if not an impossible one. Hence, it is no surprise that such problems have been studied extensively in the areas of Operational Research and Artificial Intelligence.

Different kinds of scheduling problems appear in varying areas. Fields that show up repeatedly are sports scheduling, transportation scheduling, nurse scheduling and educational scheduling. In this dissertation, we will study one particular problem from the latter area. Every end of the semester universities face the problem of scheduling large sets of exams over a fixed number of periods with a limited amount of available resources. Frequently, many hours are spent on this by a timetabling team, regularly taking up multiple full-time jobs. An automated timetabling algorithm would be able to marginally reduce the workload and the number of hours that are spent on scheduling these exams. In this dissertation, we will develop such an algorithm for a specific case of exam scheduling at The University of Edinburgh.

Although the specification of the examination timetabling problem varies across different studies, we can describe its general characteristics. The examination timetabling problem deals with a finite set of exams that need to be scheduled over a finite set of periods with a finite amount of available resources. The final timetable should satisfy a set of constraints fully or at least to some extent. Most studies separate the constraints into two different subsets. The first subset includes all the constraints that cannot be broken at any cost and are also known as hard constraints. The second subset includes all the constraints which may be violated to some extent, even though this is not desired. Such constraints are known as soft constraints and the amount of violation is often measured by some penalty

function. Section 2 provides a full description of all the hard and soft constraints that apply to the examination timetabling problem studied in this thesis.

1.2 Literature Review

1.2.1 Conflict-free Scheduling

Among the first works to address the examination timetabling problem is the work of Broder (1964). Broder describes the problem of conflict-free scheduling, in which the aim is to find a timetable where no student needs to take two exams at the same time. He proposes an algorithm that produces conflict-free timetables by looking at a so-called conflict matrix in which the entries correspond to the overlap of students between two courses.

Another leading paper is by Carter (1986) and surveys most of all early work published on examination timetabling. Carter shows how the structure of conflict-free scheduling is related to the structure of the graph coloring problem. In the graph coloring problem, the objective is to color every node of a graph using as few different colors as possible such that nodes connected by an edge do not have the same color. The minimum different colors needed to color a graph is called the chromatic number and finding the chromatic number is known to be an NP-complete problem (Karp, 1973). A graph corresponding to an examination timetabling problem can be constructed as follows. First, exams that need to be scheduled are represented by nodes. Then, nodes are connected by an edge if at least one student is taking both exams, implying that these two exams cannot be scheduled at the same time. Finding the chromatic number of the graph then corresponds to finding the minimal number of time slots needed to schedule all exams in a conflict-free way. The graph coloring problem has been studied extensively and is still relevant nowadays due to its NP-completeness. Consequently, many studies exist on heuristics that attempt to find good solutions to the graph coloring problem. Carter points out how nearly every paper on conflict-free exam scheduling that existed at that time is some variation of an existing graph coloring heuristic. Table 1 summarizes different heuristics for the graph coloring problem and their applications in relevant works for the examination timetabling problem.

In addition to Table 1, Carter (1986) mentions some other graph coloring heuristics. These are not included in Table 1 as they had not been applied frequently to the examination timetabling problem according to Carter's survey. Carter also points out that the saturation degree method performs the best on the graph coloring problem. A similar approach thus might also be successful when applied to the examination timetabling problem.

Coloring Heuristic	Description	First description of the graph coloring heuristic	Similar examination timetabling heuristics
Largest degree first	First, order the vertices by degree (number of adjacent edges). Then, keep selecting the vertex on top of the list and color this vertex using the first non-conflicting color.	Welsh & Powell (1967)	Broder (1964), Wood (1968)
Largest degree first: fill from top	Again, sort the vertices by degree, but now color all possible vertices starting from the top of the list using the lowest color. Then move to the next lowest color and repeat the procedure, and so on.	Peck & Williams (1966)	Cole (1964), Barham & Westwood (1978), White and Chan (1979)
Largest saturation degree first	Sort the uncolored vertices first by the number of different adjacent colors on the adjacent colored vertices and second, by the degree on the uncolored subgraph. Then, select keep selecting the vertex on top of the list and color this vertex using the first non-conflicting color.	Brelaz (1979)	Mehta (1981)

Table 1: Graph coloring heuristics and their applications to the examination timetabling problem.

It should be noted that some of the above works also considered some other constraints, such as room restrictions and exam spread. However, these constraints mostly were enforced either manually or by applying relatively simple heuristics. Generally, the works in this period laid their emphasis on conflict-free scheduling.

Due to the increasing computer performance, researchers gained more interest in automated timetabling, leading to notably more research output. Consequently, Carter & Laporte (1996) published a survey on the recent developments at the time. More research started focusing on secondary objectives next to the standard conflict-free scheduling. There especially was much focus on minimizing the occurrence of consecutive exams for students. The sections below summarize some of the developments at the time.

1.2.2 Cluster Methods

Carter & Laporte first mention cluster-based approaches. These approaches consist of two separate stages. In the first stage, non-conflicting exams are grouped in so-called clusters using some heuristic. Then, in a second stage, the clusters are assigned to periods by either minimizing an objective function or by satisfying some set of constraints. There exists a broad range of studies that apply this type of approach. Common is to use a ranking rule to rank the exams by difficulty and then select an exam in descending difficulty and place them in a non-conflicting cluster. Examples of ranking rules are ranking by size (White & Chan, 1979), ranking by number of conflicts (Leong & Yeong, 1990), ranking by the product of size and number of conflicts (Lofti & Cerveni, 1991) and ranking by a linear combination of the size and number of conflicts (Johnson, 1990). Then, once the exams are grouped, the clusters need to be sequenced. White & Chan minimize the number of adjacent conflicts, that is, the number of times that a student needs to take exams in consecutive periods. One can define the overlap cost between two clusters as the number of students that are taking an exam in both clusters. When these two clusters are scheduled next to each other, the overlap cost is incurred. Then, finding the minimum total of overlap costs while scheduling all the clusters is similar to solving a travelling salesman problem (TSP). White and Chan exploit this rationale by solving the TSP to find solutions. Arani, Karwan & Lofti (1988) point out another interesting relationship. They show that if no overnights costs are considered and two periods a day are assumed, the second phase of the clustering method reduces to a minimum weighted matching problem. Yet some other technique is to model the second phase as a Quadratic Assignment Problem, as done in Leong & Young (1990) and Lofti & Cerveni (1991).

1.2.3 Sequential Methods

As a second, Carter & Laporte discuss a type of approach which is called sequential methods. Instead of first grouping compatible exams as done in the clustering methods, exams are now scheduled directly into the timetable. The order of scheduling exams is again determined by some predefined ranking rule, such as size or number of conflicts of an exam. The construction phase often is followed by an improvement phase. Typical improvement techniques are 1-opt (moving one exam to another period) and 2-opt (interchanging pairs of exams). Some of the earlier mentioned coloring heuristic-based approaches are examples of sequential methods. Other examples of papers that employ sequential methods are Desroches, Laporte & Rousseau (1978), Laporte & Desroches (1984) and Carter, Laporte & Chinneck (1994).

1.2.4 Generalized Search Approaches

Generalized search approaches can be described as the set of algorithms that start with an initial solution and then try to move around in the global solution space to look for improvements. They can be distinguished from local search approaches as they employ methods to avoid getting stuck in a local search space. They generally do not require a good or feasible initial solution; the idea is to let the algorithm run for a while so that it can move around until it has found a good enough solution. There often is a trade-off between the time and the quality of the solution: generalized search algorithms perform better when run for a longer time.

A common example of a generalized search approach is tabu search. In tabu search, we look into some local neighborhood for improvements to our current solution. To prevent from getting stuck in local neighborhoods, a tabu list is kept with previously visited solutions. These solutions are forbidden to visit again for a certain time. This forces the algorithm to look somewhere else and accept other solutions even if they are worse than previous solutions. Tabu search also has its application to the examination timetabling problem. Hertz (1991) defines a restricted version of an 1-opt neighborhood by random selection and uses tabu search where the tabu list contains seven solutions. Another more practical example can also be found in Boufflet & Négre (1996).

A different example of generalized search is simulated annealing. Simulated annealing also attempts to move away from local optima. It does this by accepting new solutions if they are better than the current solution or by accepting solutions with some probability

if they are worse than the current solution. The probability often is called the temperature and decreases over time to drive the solution into a local optimum. Applications of simulated annealing to examination timetabling can be found in Thompson & Dowsland (1995, 1996). They use a 1-opt and 2-opt neighborhood (also known as Kempe chains) to find solutions using simulated annealing.

Another example of generalized search that Carter & Laporte discuss is genetic algorithms. The idea behind genetic algorithms is to find optima by generating new solutions using crossover techniques at so-called genes of two parent solutions. A coding scheme is used to represent solutions and to make it possible to generate new child solutions. A fitness function determines how good solutions are. Fitter solutions then are chosen with a higher probability for reproduction. Genetic algorithms have been applied before to the examination timetabling problem at The University of Edinburgh by Corne, Fang & Mellish (1993). Populations of size 50 were used and the procedure was repeated for 300 generations and evaluated by a fitness function, which considered conflicting exams and consecutive exams with different weights. The algorithm was implemented and turned out to be more successful than manually scheduled timetables from previous years.

1.2.5 Toronto Benchmark Dataset

In addition to the survey, Carter & Laporte also provided a benchmark dataset which is based on the examination timetabling problem at The University of Toronto. The reason for the introduction of this dataset was that all the existing algorithms at the time were hard to compare since all of them were applied to different instances. To create a more comprehensive and cohesive setting for the examination timetabling problem, Carter & Laporte recommended the use of the Toronto dataset for further research. This rapidly became the standard and for the next decade, most of the work was tested on (modifications of) the Toronto dataset.

Most research in this period was based on new generalized search approaches. Caramia et al. (2001, 2008) exploited the technique of iterated local search, where the search procedure is restarted when certain criteria are met to obtain multiple minima. Casey and Thompson (2003) investigated a special case of iterated local search called GRASP, where the local search procedure is greedy. In addition, genetic algorithms were further explored and modified. Côté et al. (2005) presented a combination of both local search heuristics and a genetic algorithm. Ülker et al. (2007) tested different crossover operations on the genes for the examination timetabling problem. Burke et al. (1998) proposed an im-

provement of the genetic algorithm called the memetic algorithm. Another relatively new type of algorithm is the ant colony algorithm. This algorithm mimics the way ants search for their food by laying pheromones. In every stage, information is obtained during the search and is used in the next stage to find better solutions. The application of this type of approach was investigated by Naji Azimi (2004) and Dowsland & Thompson (2005).

The above-mentioned work is just a small selection of the wide range of studies that have been tested on the Toronto benchmark dataset. A more thorough comparison on the results of the studies can be found in a survey by Qu et al. (2009).

1.2.6 ITC 2007 Benchmark Dataset

Although the existing algorithms were able to provide good solutions for the Toronto dataset in considerable amounts of time, the algorithms faced multiple challenges when tested on real-life datasets. The biggest issue was that practical instances dealt with a lot more constraints which add extra layers of complexity to the problem. The Toronto dataset only focused on conflict-free scheduling and optimizing the exam spread for the students. Hence, for the 2007 International Timetabling Competition, it was decided to include more difficult constraints to bridge the gap between theoretical and practical timetabling. Nowadays, the ITC 2007 benchmark dataset is used more regularly than the Toronto benchmark dataset and this type of examination timetabling is sometimes referred to as the real-life examination timetabling problem. McCollum (2012) provided a full description and mathematical formulation for all the hard and soft constraints of the ITC 2007 benchmark dataset.

Similar earlier proposed approaches have been modified to make them applicable to the ITC 2007 benchmark dataset. These approaches include tabu search methods (McCollum et al., 2010), GRASP metaheuristics (Gogos, Alefragis & Housos, 2008) and local iterated search (McCollum et al. 2010). Additionally, new techniques were also introduced. Some examples are hybrid local search (Müller, 2009), bee colony optimization (Alzaqebah & Abdullah, 2015) and great deluge algorithms (Hamilton-Bryce, McMullan & McCollum, 2013).

The above-mentioned papers have developed algorithms for application to the ITC 2007 benchmark dataset. However, this dissertation will not study the examination timetabling problem in the same setting. Instead, it will use a modification of the ITC 2007 benchmark dataset to make it suitable in the context of examination timetabling for The University

of Edinburgh. In Section 2.3.1, a comparison between the Toronto benchmark dataset, the ITC 2007 benchmark dataset and the dataset used in this thesis is given.

1.3 Contribution of this Thesis

Most of the mentioned approaches either apply to the Toronto dataset or ITC 2007 dataset. Consequently, some of the studied algorithms may have limited applications when used at other institutions. This thesis will contribute to the existing literature by further bridging this gap between theory and practice on examination timetabling. Several algorithms will be introduced to solve the examination timetabling problem for The University of Edinburgh. This requires a new type of characterization of the problem which to our knowledge, has not been studied before. Thus, this thesis will show how to model the examination timetabling problem in a different setting with a new set of constraints. The type of approach that is used in this thesis is not only limited to The University of Edinburgh. Most of the constraints that are modelled in this thesis are trivial for most institutions. This thesis aims to provide a new general setting for the examination timetabling problem. Hence, the ideas that are incorporated in this dissertation can also be extended to other institutions.

2 Problem Characterization

2.1 Timescales

The academic year at The University of Edinburgh consists of two different semesters, the first being from September until December and the second being from January until April. Students regularly take around six courses per semester and most of them include a final examination at the end of the semester. Some of the courses from the first semester are examined in December and all the other courses are examined in April/May. In 2018, this resulted in 47202 sittings in December and 59732 sittings in April/May. The courses that are examined in December need to be scheduled in a period of two weeks including Saturdays and the courses that are examined in April/May need to be scheduled in a period of four weeks excluding any weekends. Additionally, some students need to take resits which are scheduled throughout a period of two weeks. However, this is not considered to be a complex scheduling problem; in 2018 only 2931 resits were taken.

The University of Edinburgh has been working with two different time slots a day: one in the morning and one in the afternoon. Hence in December, we can choose between 24 different periods for scheduling exams. Similarly, for April/May, we end up with 40 different periods. It is assumed that there is enough time to examine any type of course in a given time slot.

2.2 Constraints Formulation

The complexity of the real-life examination timetabling problem can mostly be accredited to a large number of constraints that need to be accounted for. This section describes all the soft and hard constraints that apply to the formulation of this thesis. Also, a comparison between the Toronto formulation, the ITC 2007 formulation and the formulation in this thesis will be given.

2.2.1 Hard Constraints

A summation of hard constraints that apply to this thesis is given below.

- Exam feasibility: All exams need to be scheduled into the timetable.
- Conflict-free: Exams need to be scheduled in a conflict-free way. This means that no student should take two exams at the same time.

- Preassignments: Some exams (such as exams from the vets/medics faculty) can only be scheduled at a certain period.
- Exam coincidence: Some exams need to be taken together with another exam.
- Exam precedence: Some exams can only be taken after another exam.
- No room splitting: Students taking a particular exam cannot be divided over multiple rooms.
- Room capacity: The number of students in an exam room cannot exceed the room's capacity.
- Room type: Exams need to be assigned to the right type of room (regular/computer room).
- Mixed durations: Exams of different durations cannot be scheduled in the same room.

It should be noted that these are many hard constraints to account for. However, this is a realistic setting for a university when it comes to exam timetabling. As this thesis attempts to mimic real-life examination timetabling, it is decided to include all the above constraints as hard constraints.

2.2.2 Soft Constraints

In addition to the hard constraints, this dissertation also deals with a range of soft constraints.

- Two in a row, same day: It is not desired that students need to take two exams in a day.
- Two in a row, overnight: It is not desired that students take an exam in the afternoon and an exam in the morning.
- Period spread: It is not desired that a student takes two exams in x consecutive periods.
- Disability, morning: Some disabled students prefer to only have exams in the morning.
- Disability, afternoon: Some disabled students prefer to only have exams in the afternoon.
- Disability, two consecutive: Some disabled students prefer to not have two exams in consecutive periods.
- Disability, two in a day: Some disabled students prefer to not have two exams in one day.

Observe that a particular assignment of exams can violate multiple soft constraints simultaneously. If a disabled student (no two consecutive exams) has an exam in the morning and the afternoon on the same day, the *"Two in a row, same day"*, *"Period spread"* and *"Disability, two consecutive"* constraints are all violated at the same time. The total penalty that is incurred then is the sum of all the violated soft constraints. An institution can set these penalties to own preference to obtain multiple timetables.

2.2.3 Differences with Toronto and ITC 2007 Formulations

As mentioned before, this thesis contributes to the existing literature by studying a new type of formulation for the real-life examination timetabling problem. Table 2 displays how the formulation in this thesis differs from the existing benchmark formulations.

2.3 Data Description

2.3.1 Input File Format

The input file for the algorithm contains all the necessary information that the algorithm needs to be able to schedule all the exams. First, the input file contains information about the exams. For every exam is specified: an exam identifier, a list of students taking the exam, the duration of the exam in minutes and an indicator of whether the exam should be taken in a computer room or not. Additionally to this type of data, the input file contains data about all the periods and rooms. For the periods, the date and the time of the day are specified and for the rooms, the capacity and an indicator whether it is a computer room are specified. The input file also contains all the relevant information about which exams need to be taken together (exam coincidence) and which exams need to be taken after one another (exam precedence). Also, a list of all the exams that need to be preassigned and their corresponding preassigned period are specified in the input file. Then, the input file also contains information about disabled students and their disability and finally also a list of parameters with institutional weightings for the soft constraints penalties.

2.3.2 Test Dataset

The timetabling team of The University of Edinburgh has been very helpful when it comes to characterizing the examination timetabling problem by providing insights on their current scheduling procedure. Unfortunately, the timetabling team has not been able to

Constraint	Toronto	ITC 2007	This thesis
Exam feasibility	Hard	Hard	Hard
Conflict-free	Hard	Hard	Hard
Large exams as early as possible	-	Soft	-
Preassignments	-	-	Hard
Exam coincidence	-	Hard	Hard
Exam precedence	-	Hard	Hard
Room feasibility	-	Hard	Hard
No room splitting	-	Soft	Hard
Room type	-	-	Hard
Individual rooms (Only one exam in a room)	-	Hard	-
Two in a row, same day	Soft	Soft	Soft
Two in a row, overnight	Soft	Soft	Soft
Two in a day, not consecutive	-	Soft	-
Period spread	Soft	-	Soft
Unfavored periods	-	Soft	-
Unfavored rooms	-	Soft	-
Disabilities	-	-	Soft

Table 2: Comparison of different formulations for the examination timetabling problem.

provide us with a dataset containing all the above described required data for one exam semester. Since testing and validating an algorithm is an essential part of the entire analysis, we thus have come up with a new dataset that is a modification of the ITC 2007 benchmark dataset. It should be noted that the data from The University of Edinburgh is not particularly necessary for the algorithm to work. The ideas that are presented in this thesis are not specifically limited to The University of Edinburgh; the algorithm should be able to perform well on any type of institution that has the same requirements on examination timetabling.

The modified ITC 2007 benchmark dataset consists of a total of 607 different exams that need to be scheduled in seven different rooms over a total of 54 periods. The number

of total sittings is equal to 32380. There is a conflict density of 5.13% between the exams, indicating that 5.13% of pairs of exams have at least one common student taking both exams. 39 of the exams have a duration of 90 minutes, 259 a duration of 120 minutes, 16 a duration of 150 minutes and 293 a duration of 180 minutes. 58 exams are computer exams and need to be scheduled in a computer room. The others are regular exams and need to be scheduled in a regular room. There is only one computer room with a capacity of 260. The other six rooms are regular rooms with capacities of 260, 100, 129, 77, 65 and 111. The dataset also contains exam coincidence and exam precedence constraints. Additionally, the dataset includes information about 30 exams that need to be preassigned to certain periods.

We expect this dataset to be a fair representation of the examination timetabling problem at The University of Edinburgh. The number of sittings in the modified ITC 2007 dataset is comparable to the number of sittings at a regular exam semester at The University of Edinburgh. Even though our dataset considers slightly more periods than The University of Edinburgh does, we expect this dataset to have a more restricting character. This has to do with the small number of available rooms, which is likely to be exceeded by The University of Edinburgh.

3 Construction Algorithm

In this section, we will describe a heuristic that constructs a feasible initial solution for the examination timetabling problem at The University of Edinburgh. We will use a variation of a cluster-based approach.

3.1 First Phase

The first phase of a cluster-based approach consists of grouping non-conflicting exams into multiple clusters. Frequently, the number of available clusters is set equal to the number of different periods so that in the second phase exam clusters can be assigned to the periods one-to-one. An advantage of this method is that it is relatively simple and effective. The one-to-one relation can be exploited in the second phase by for instance modelling it as a TSP, a minimum weighted matching problem or a QAP, as mentioned in Section 1.2.2. However, setting the number of clusters equal to the number of periods also has some drawbacks. The first phase only takes into account conflict-free scheduling when clustering the exams. The second phase then starts looking at how to minimize adjacent conflicts using the groups from the first phase. The problem here is that the number of total adjacent conflicts also depends on how the exams are grouped in the first phase. Thus, we propose a different method on clustering exams by also taking into account the minimization of adjacent conflicts in the first phase. The proposed method is based on the relatedness of the examination timetabling problem to the graph coloring problem. Instead of defining the number of clusters, we first treat the problem as a graph coloring problem. The number of different clusters will then correspond to the number of different colors used.

3.1.1 Graph Coloring Issues

As discussed, many works focus on computing the chromatic number of the color graph which is directly related to the examination timetabling problem. However, when studying the real-life examination timetabling problem, such methods often give undesirable results. One of the problems that show up is that simple coloring algorithms produce clusters of undesirable sizes. Some of the clusters end up containing exams with many more students than the maximum number of students that can be scheduled into one single period. Other clusters end up with too few students. Moreover, the chromatic number often is way smaller than the number of available periods, leaving many periods empty.

The biggest underlying issue here is the room capacity. If we had an infinite capacity for every period, then color heuristics would produce satisfying results. However, this is not the case in real-life examination timetabling where we would rather have a more balanced spread across all the periods to account for the room capacities. This situation is sketched in Figure 1.

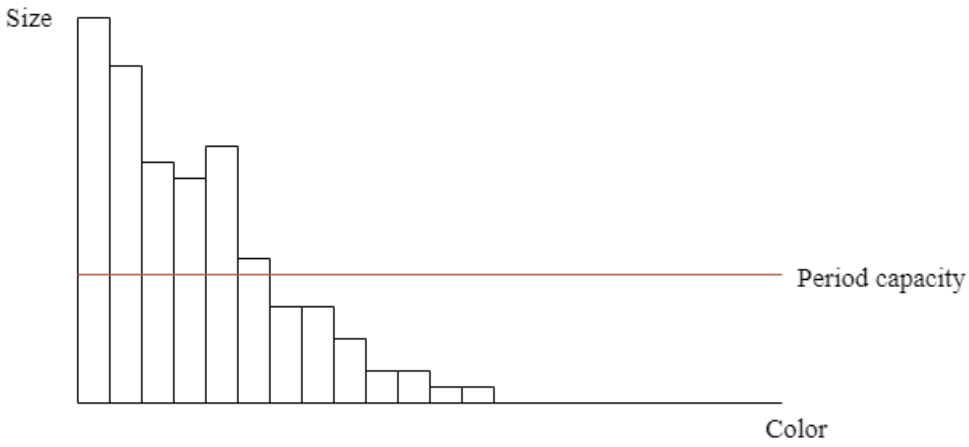

Figure 1: Coloring heuristics produce unbalanced clusters.

Another issue deals with the exams that need to be preassigned to a certain time slot. Such constraints do not allow for much flexibility when scheduling these exams. As we already know the period of a specific exam, we can schedule it directly into our final timetable. We illustrate the above by an example.

Example 3.1: Preassigning exams into the final timetable

Consider a simple examination timetabling problem for a university with 20 different exams that need to be scheduled. For simplicity, all exams are of the same size and we can only schedule three exams per period. The university has 8 periods available for scheduling and has some additional requirements on certain exams. Exam 2 needs to be scheduled in period 2, exam 12 in period 5 and exam 19 in period 7. Table 3 displays the timetable after scheduling the preassignments.

Period	1	2	3	4	5	6	7	8
Exam #1		2			12		19	
Exam #2								
Exam #3								

Table 3: Timetable after scheduling preassignments.

Note that, as we schedule the preassignments, we also need to account for other constraints that deal with room types, durations and capacities. Hence, we cannot schedule the preassignments directly into the first slot as shown in the previous example, but we need to make some other considerations first. We instead will put the preassignments in separate clusters, whose structure we will exploit in a later stage.

We also need to deal with the exams that need to be taken at the same time. We simply group the exams and treat them as one exam for the first phase of the clustering algorithm. In the second stage, we then can schedule the exams separately if needed.

3.1.2 Exploiting Large Clusters

The method proposed in this thesis exploits the non-conflicting property of clusters. Even though the exams from large clusters cannot all be scheduled into one single period, they can be spread out over multiple adjacent periods. Doing this has a big advantage over earlier described approaches. Since the exams from a single cluster are scheduled next to each other, we know that no student has to take two exams in these periods, thus minimizing the number of adjacent conflicts. The above analogy is again illustrated in an example.

Example 3.2: Minimizing adjacent conflicts using large clusters

Consider again the situation from Example 3.1. After applying some coloring algorithm, we end up with three different clusters. Cluster 1 contains exams 1, 2, 5, 7, 11, 13, 14. Cluster 2 contains exams 6, 10, 12, 15, 16, 18, 20 and cluster 3 contains exams 3, 4, 8, 9, 17, 19. Two ways of scheduling these clusters into the final timetable are given in Table 4 and Table 5.

Observe that both timetables are feasible as they both are scheduled conflict-free. This can be seen by noting that every single period is assigned only one type of color. Observe however that timetable 2 is preferred over timetable 1. Consider for example period 2 in

Period	1	2	3	4	5	6	7	8
Exam #1	15	2	3	1	12	18	19	14
Exam #2	16	11	4	5	6	20	9	7
Exam #3		13	8		10		17	

Table 4: Examination timetable 1 for example 3.2

Period	1	2	3	4	5	6	7	8
Exam #1	1	2	14	6	12	20	19	8
Exam #2	5	11		10	16		3	9
Exam #3	7	13		15	18		4	17

Table 5: Examination timetable 2 for example 3.2

both timetables. For timetable 1, there could be some students taking both an exam in period 1 and 2 or period 2 and 3. For timetable 2, we can rule out this possibility since all these exams belong to the same cluster and timetable 2 consequently is expected to contain fewer adjacent conflicts.

3.1.3 Available Spots for Clusters

As the period of the preassigned is fixed, our goal is to schedule the remaining exams around the preassigned exams in the best possible way. We do this by assigning a single color to every period that contains at least one preassigned exam. These colors form the basic clusters for the main scheduling procedure. We then assign the uncolored exams a cluster using a coloring heuristic. However, to prevent exams from grouping all into one single cluster as shown in Figure 1, we take into account another variable, which we call *available spots*. This variable computes for every single cluster how many spots are left for additional exams to join. This number not only depends on how many exams already have joined the cluster but also depends on the spread of the preassigned exams. We again demonstrate this by an example.

Example 3.3: Computing available spots

Consider again the situation from Example 3.1. For every preassigned exam, we compute the number of available spots by counting the number of empty spots that are the closest to that preassigned exam. In a tie, we assign the empty spot to the cluster with the lowest period index. Table 6 demonstrates the procedure

Period	1	2	3	4	5	6	7	8
Exam #1		2			12		19	
Exam #2								
Exam #3								

Table 6: Computing available spots

Note that we have 3 different clusters as we have 3 preassigned exams. It follows that the numbers of available spots left for the blue, green and red clusters are 8, 8 and 5 respectively.

The number of available spots per cluster tells us how many exams we still can add to that cluster if we want to schedule all the exams from the cluster into adjacent periods. In the above example, it is straightforward to compute this number since all the exams are of the same size. However, in the general real-life examination timetabling problem, this is not the case. Hence, we need to resort to other methods to get an approximation of how many exams we can schedule into one cluster. We propose the following estimate for the available spots a_i for a cluster i.

$$a_i = \left\lfloor \alpha p \sum_{r \in R} c_r \right\rfloor$$

where p is the number of periods that are closest to cluster i, c_r the capacity of room r and R the total set of available rooms. Consequently, the estimate does not compute how many exams can be scheduled into a cluster but computes how many students can still be scheduled into a cluster. After adding an exam to a cluster, the number of available spots is recomputed by subtracting the number of exam students from the available spots.

We also specify the parameter $\alpha \in [0, 1]$. This is a parameter that compensates for the fact that a room rarely utilizes its full capacity. Often, seats are left empty in an exam room since there are not enough spaces left for scheduling another exam into that room. Another reason for empty seats is that there are no more suitable exams in the cluster to schedule into that specific room. This can be due to a different room type or a different exam duration.

Now, we can decide which of the clusters is the best to assign the uncolored exam to. Instead of using a greedy approach as done in most coloring heuristics, we first identify

the possible list of non-conflicting clusters that this particular exam can be scheduled into. We then assign the exam to the cluster that has the most available spots. Using this approach, we make sure that we spread out the exams evenly over the whole timetable while still taking advantage of the non-conflicting structure of color groups.

3.1.4 First Phase Algorithm Description

In this section, we will put together the analyses from the previous sections. We formally define the algorithm using a more technical description. For the coloring heuristic, we will use the saturation degree method, as suggested by Carter (1986).

Let us define the symmetric conflict-matrix $C \in \mathbb{R}^{n \times n}$ with n being the number of exams that need to be scheduled. Let the i, j-th entry $c_{ij} \in C$ correspond to the number of students that need to take both exams i and j for $i \neq j, i = 1, 2, \ldots, n, j = 1, 2, \ldots, n$. Additionally, we set $c_{ii} = 0$ with $i = 1, 2, \ldots, n$ for modelling purposes.

Let $G = (V, E)$ by an undirected graph where $V = \{1, 2, \ldots, n\}$ is the set of vertices where a single vertex represents a single exam. E is the set of edges that correspond to the pairs of exams that cannot be scheduled together, i.e. $E = \{e \in (i, j) | i, j \in V, c_{ij} > 0\}$. We define the subset $V_c \subseteq V$ which contains all the colored vertices and the subset $V_u \subseteq V$ which contains all uncolored vertices. Clearly, we have that $V_c = V \setminus V_u$. Let $\delta(v)$ for all $v \in V$ be the degree of vertex v. Furthermore, we define $\delta_u(v)$ as the degree on the uncolored subgraph and $\delta_c(v)$ as the number of adjacent different colors on the colored subgraph. We will refer to these as the uncolored degree and the colored degree respectively. Note that the colored degree is not necessarily the same and is always equal or less than the degree on the colored subgraph. We define the color of a vertex $v \in V$ by $\psi(v)$ where the color is denoted by a number. We set $\psi(v) = 0$ for all $v \in V_u$. The number of students that need to take an exam v is denoted by $n(v)$. Let $P = \{1, 2, \ldots, p\}$ be the set of possible periods that we can schedule exams into with p being the number of total periods. We define $F \subseteq P$ as the set of all clusters where we indicate a cluster by the period of the corresponding preassigned exam. Let $a(f)$ and $e(f)$ denote the number of available spots and elements of a cluster f respectively. Furthermore, let $Preassignments$ be a list of two-dimensional vectors where the elements describe the exam and the period that the exam needs to be assigned to respectively. Similarly, let $Exam\_Coincidences$ be a list of two-dimensional vectors where the elements correspond to the exams that need to be taken together. Using the above notation, we now can specify the saturation degree coloring heuristic which is presented in Algorithm 1.

Algorithm 1: Grouping exams using the saturation degree method

for (v_1, v_2) *in Exam_Coincidence* **do**
  $V = V/v_2$; `/* Group together exam coincidences */`
  $n(v_1) = n(v_1) + n(v_2)$;
  for j *in* V **do**
    $c_{v_1 j} = c_{v_1 j} + c_{v_2 j}$; `/* Update conflict matrix */`
    $c_{j v_1} = c_{v_1 j}$;
  end
end
Construct graph $G = (V, E)$ using the new conflict-matrix;
Set $V_c = \varnothing$ and $V_u = V$;
for *(v, p) in Preassignments* **do**
  Set $\psi(v) = p$; `/* Color the preassigned exams */`
  $V_c = V_c \cup v$;
  $V_u = V_u/v$
end
Set $F = \{p \in P/v \in V_c, \psi(v) = p\}$; `/* Construct clusters */`
for f *in* F **do**
  $a(f) = ComputeSpots(f)$;
end
while $V_u \neq \varnothing$ **do**
  Compute $\delta_c(v)$ and $\delta_u(v)$ for all $v \in V_u$; `/* Saturation degree method */`
  Sort the set V_u first by $\delta_c(v)$ and second by $\delta_u(v)$ in a descending order;
  Set v^* equal to the first element in the set V_u;
  `/* Determine the set of feasible clusters for exam v */`
  for f *in* F **do**
    for v *in* $e(f)$ **do**
      if (v^*, v) *in* E **then**
        break; `/* Cluster is conflicting */`
      else
        candidates.append(f); `/* Non-conflicting clusters */`
      end
    end
  end
  $f^* = 0$; `/* Assign to null cluster if no feasible clusters */`
  $a(f^*) = n(v^*) - 1$; `/* A cluster should have at least` $n(v^*)$ `spots left */`
  for f *in candidates* **do**
    if $a(f) > a(f^*)$ **then**
      $f^* = f$; `/* Select cluster w/ most available spots */`
    end
  end
  $V_c = V_c \cup v^*$; `/* Assign exam` v^* `to cluster` f^* `*/`
  $V_u = V_u/v^*$;
  $e(f^*) = e(f^*) \cup v^*$;
  $\psi(v^*) = f^*$;
  $a(f^*) = a(f^*) - n(v^*)$;
end

3.2 Second Phase

Once we have grouped the exams into clusters using Algorithm 1, we can start assigning the exams into the final timetable. Observe however that not necessarily every exam is assigned to a cluster. Exams for which there were no feasible non-conflicting clusters have been assigned the artificial color 0 by Algorithm 1. We will refer to this cluster as the null cluster. The exams in the null cluster will be scheduled into the timetable after we have exploited the structure of the regular clusters.

3.2.1 Scheduling the Clusters

Even though the periods for a given cluster are known, there are certain considerations that we need to make before we can schedule a particular exam. The first one concerns the room type. Clearly we cannot schedule an exam in a room of the wrong type. The second consideration has to do with durations. We can only schedule an exam into a room if the room is empty or if an exam of the same duration is already scheduled in that room at that time. The third consideration deals with room capacities. As we are not allowed to split exams over multiple rooms, we need to check if there are still enough spots left in the particular slot before we can schedule an exam into a room. Finally, we choose to leave as many periods empty as possible, as we need to schedule the null cluster afterwards. If there are not enough empty periods left, we might be unable to schedule these exams because of their conflicting character. Example 3.4 illustrates why leaving empty periods is beneficial.

Example 3.4: Scheduling clusters

Consider a cluster that is called cluster 6 which contains the exams 1, 2, 3, 4, 5 and 6. Also consider exam 7, which was placed in the null cluster as it was conflicting with exams 4, 5 and 6 and exams in other clusters. Cluster 3 was assigned 3 different periods to schedule exams into. Two different ways of scheduling cluster 6 into the timetable are given in Table 7 and Table 8. We again assume for simplicity that the exams are of the same size and that only three exams can be scheduled per period.

Period	·	5	6	7	·
Exam #1		1	2	3	
Exam #2		4	5	6	
Exam #3					

Table 7: Scheduled cluster timetable 1

Period	·	5	6	7	·
Exam #1		1	4		
Exam #2		2	5		
Exam #3		3	6		

Table 8: Scheduled cluster timetable 2

Both timetables show a feasible way of scheduling cluster 6. However, timetable 2 is preferred over timetable 1 since timetable 2 leaves an empty period for the null cluster where timetable 1 leaves none. When scheduling exam 7, we can indeed see why timetable 2 is favored. Timetable 1 has no non-conflicting spots left for exam 7 and thus exam 7 cannot be scheduled here, whereas timetable 2 leaves a spot for exam 7 in period 7.

Thus, we try to schedule all the exams in a cluster in a feasible way while maximizing the number of empty periods. Our proposed algorithm first selects a period from the assigned cluster periods and schedules as many exams into that period. Then, the algorithm selects a next adjacent period and schedules as many exams into that period. This process continues until all exams in the cluster are scheduled and the cluster is empty. The order of periods that we consider is determined by the distance to the period where we scheduled the preassigned exam. For instance, if we preassigned the exam with cluster periods 5-6-7-8 to period 6, the order of periods in which we will schedule the remaining exams from the cluster is 6-5-7-8. Consequently, period 8 will have the largest probability of being left empty.

Before we schedule the exams from a cluster into a slot, we first order the exams by size. The rationale behind this is that large exams are harder to schedule and hence should be prioritized. When an exam is selected, we look in the set of feasible rooms for the room with the most available spots and we assign the exam to that particular room. The algorithm keeps scheduling exams until it has considered every single exam in the cluster. When there is no feasible room for an exam, the exam is left unscheduled. The same exam will then be reconsidered for scheduling when the next period is selected. This process keeps going until all exams are scheduled or when all periods for that cluster are checked. If a particular exam is left unscheduled after considering every period for that cluster, we put it in the null cluster.

3.2.2 Second Phase Algorithm Description

First, we will introduce some additional notation before we formally define the second phase scheduling algorithm. Let R be the set of rooms. Let $c(r)$ and $t_r(r)$ denote the capacity and the room type respectively for all $r \in R$. Let $a(r, p)$ and $d(r, p)$ denote the available spots left and the duration of room r in period p. Let $d(v)$ and $t_v(v)$ denote the duration and type of an exam respectively for all $v \in V$. Finally, we denote our final timetable by Ξ where $\Xi[i, j]$ corresponds to the list of exams that is scheduled in room i and period j. Algorithm 2 describes the full procedure of scheduling the clustered exams.

Algorithm 2: Scheduling clustered exams into the timetable

```
for p in P do
    for r in R do
        a(r,p) = c(r);                          /* Initialize available spots */
        d(r,p) = 0;                          /* Empty rooms are set to duration 0 */
    end
end
for (v,p) in Preassignments do
    r* = 0;                                  /* First, schedule the preassignments */
    a(r*,p) = n(v) − 1;
    for r in R do
        if t_v(v) == t_r(r) and a(r,p) > a(r*,p) then
            r* = r;                          /* Select room with most available spots */
        end
        Add v to Ξ[r*,p];                                   /* Update timetable */
        a(r*,p) = a(r*,p) − n(v);
        d(r*,p) = d(v);
        e(f) = e(f)/v;
    end
end
for f in F do
    Sort f by exam size;
    Per = GetPeriods(f) ;                    /* Get the periods corresponding to the
     cluster */
    Sort Per by distance from the period where the preassigned exam is assigned to;
    for p in Per do
        for v in e(f) do
            r* = 0;
            a(r*,p) = n(v) − 1;
            for r in R do
                if t_v(v) == t_r(r) and (d(r,p) == d(v) or d(r,p) == 0) and
                  a(r,p) > a(r*,p) then
                    r* = r;
                end
            end
            if r* > 0 then
                Add v to Ξ[r*,p];
                ;                    /* Update timetable if feasible room is found */
                a(r*,p) = a(r*,p) − n(v);
                d(r*,p) = d(v);
                e(f) = e(f)/v
            end
        end
    end
    e(0) = e(0) ∪ e(f) ;    /* Unscheduled exams are put in the null cluster
     */
end
```

3.3 Scheduling the Null Cluster

After all regular clusters are scheduled into the final timetable, we need to schedule the exams that ended up in the null cluster. These exams ended up in the null cluster either since they could not be assigned to a regular cluster in Algorithm 1 or since there was no feasible room available while scheduling clusters in Algorithm 2. The number of exams in the null cluster consequently depends on the number of clusters we started initially with, which again depends on the number of preassigned exams. Consequently, if there are not many exams that need to be preassigned, one could manually preassign some of the exams to create more clusters. This avoids the problem of too many exams ending up in the null cluster.

Also, observe that exams in the null cluster that were rejected initially in the first phase because of their conflicting character might still be feasible in periods where exams have been scheduled in the second phase. Example 3.5 illustrates such a situation.

Example 3.5: Scheduling an exam from the null cluster

Consider a cluster that is called cluster 6 which contains exams 1, 2, 3 and 4, 5. Also consider exam 6, which was placed in the null cluster as it was conflicting with exam 2 and with exams in other clusters. Cluster 6 has a total of two adjacent periods. The scheduling algorithm produced the timetable given in Table 9. We again assume for simplicity that the exams are of the same size and that only three exams can be scheduled per period.

Period	·	5	6	·
Exam #1		1	4	
Exam #2		2	5	
Exam #3		3		

Table 9: Timetable produced by the scheduling algorithm after scheduling cluster 6.

Even though exam 6 was initially rejected in the first phase, we can still schedule this exam in period 6. This is because exam 6 is only conflicting with exam 2 and not with exam 4 and 5. However, this comes at a cost. Some students need to take exam 2 in period 5 and exam 5 in period 6 which are adjacent periods.

The above example illustrated how exams in the null clusters can still be scheduled into the final timetable. Note, however, that sometimes this creates situations in which students need to take two exams in a row. As a result, the approach used to schedule the exams in the null cluster is focused on feasibility and not on the optimization of students' exam spread.

The last part of the construction algorithm looks into the timetable for every exam in the null cluster. First, analogous to Algorithm 2, we sort the exams in the null cluster by size. Then, for the first exam in the null cluster, we obtain all the feasible slots that we can assign the exam to. A slot is defined by a period and a room and is feasible when there are no conflicting exams scheduled into that period, the room type and duration are the same for a given exam and the number of available spots in the room is at least the size of the given exam. Then, for all feasible slots, we select the one with the largest number of available spots and we schedule the exam into that slot. We keep repeating this process until all exams from the null cluster are scheduled into the timetable. Note however, that if an exam from the null cluster cannot be scheduled, we have no feasible solution to the examination timetabling problem. Algorithm 3 summarizes the procedure. After running the previously described three algorithms, all exams have been scheduled into the final timetable.

Algorithm 3: Scheduling the remaining exams in the null cluster

```
Sort the null cluster by exam size;
for v* in e(0) do
    r* = 0;
    p* = 0;
    a(r*, p*) = n(v*) − 1;
    for p in P do
        conflict = False;            /* First check if period is feasible */
        for r in R do
            for v in Ξ[r, p] do
                if (v*, v) in E then
                    conflict = True ;                  /* Conflicting period */
                end
            end
        end
        if conflict == False then
            for r in R do
                if t(v*) == t(r) and (d(r, p) == d(v) or d(r, p) == 0) and
                  a(r, p) > a(r*, p*) then
                    r* = r;    /* Override optimal parameters if current pair
                      (r, p) has more available spots and is feasible */
                    p* = p;
                end
            end
        end
    end
    if r* > 0 and p* > 0 then
        Add v* to Ξ[r*, p*];
        ;                    /* Update timetable if feasible room is found */
        a(r*, p*) = a(r*, p*) − n(v*);
        d(r*, p*) = d(v*);
    end
end
```

3.4 Exam Precedence Constraints

After scheduling all exams, the only type of hard constraint that we have not accounted for are the exam precedence constraints. For The University of Edinburgh, it is relatively simple to incorporate this. For such pairs of exams, it is assumed that they both are taken by the same set of students. As a result, after the final timetable has been constructed, these exams can be switched around if needed to obtain a feasible final timetable.

This idea can be extended to a more general case. If the exams do not share the same students, we can artificially adjust the set of students to the union of both sets for both exams before we start scheduling. In this way, it is always possible to switch around the exams to satisfy the exam precedence constraints. A drawback is that when taking the union of both sets, these exams may become more complex to schedule.

4 Improvement Algorithm

The construction phase primarily aimed at obtaining a feasible solution that satisfies all the hard constraints. As a second, the construction algorithm also looked into the optimization of the students' exam spread. In this section, we will look at the soft constraints in more detail using a penalty function and we propose an algorithm that improves the final timetable by moving exams to different rooms and time slots.

4.1 Penalty Function

First, we need to have a better definition of an improvement. As mentioned before, the performance of a solution is often measured by a penalty function. We can do something similar and define a penalty function for our problem to measure the fitness of a solution. The penalty function then describes to what extent the soft constraints are violated. An improvement corresponds to a move for which a timetable is obtained with a lower penalty score.

We define x_{ij} where the i, j-th entry corresponds to the number of students that need to take an exam in both periods i and j for $i \neq j, i = 1, 2, \ldots, p, j = 1, 2, \ldots, p$. Also, define x_i^1 and x_i^2 as the number of disabled students with requirements *"Morning"* and *"Afternoon"* respectively that need to take an exam at period i. Furthermore, we define x_{ij}^3 and x_{ij}^4 as the number of disabled students with requirements *"No two on the same day"* and *"No consecutive"* respectively that need to take an exam in both periods i and j. Let P_{sd} be a list of two-dimensional vectors where the elements correspond to pairs of periods that are on the same day. Similarly, define P_{on} as a list of two-dimensional vectors where the elements correspond to pairs of periods that are adjacent and separated by a night. We also define P_m and P_a as the set of periods in the morning and afternoon respectively. Furthermore, we denote s as the maximum number of periods between two exams for which the *period spread* constraint is violated. We now can define the penalty function by specifying every component that models a different type of soft constraint.

- Two in a row, same day

$$s_1 = w_1 \sum_{(i,j) \in P_{sd}} x_{ij}$$

- Two in a row, overnight

$$s_2 = w_2 \sum_{(i,j) \in P_o} x_{ij}$$

- Period spread

$$s_3 = w_3 \sum_{j=1}^{s} \sum_{i=1}^{p-j} x_{i,i+j}$$

- Disability, morning

$$s_4 = w_4 \sum_{i \in P_a} x_i^1$$

- Disability, afternoon

$$s_5 = w_5 \sum_{i \in P_m} x_i^2$$

- Disability, no two on same day

$$s_6 = w_6 \sum_{(i,j) \in P_{sd}} x_{ij}^3$$

- Disability, no consecutive

$$s_7 = w_7 \sum_{i=1}^{p-1} x_{i,i+1}^4$$

where we additionally define s_i as the total penalty score for violating the soft constraint of type i and w_i as the weight parameter that specifies the penalty that is incurred for violating the soft constraint of type i. Finally, the total penalty score is given by

$$\text{Total penalty} = \sum_{i=1}^{7} s_i$$

As mentioned before, an institution can create different timetables by adjusting the weights to their preferences.

4.2 Algorithm Description

Using the defined penalty function, we now can quantify the fitness of a solution. The improvement algorithm will look into the neighborhood of a current solution and attempts to find a better solution by moving one exam to another feasible slot. For simplicity, we do not move around exams that are preassigned or are subject to exam coincidence constraints. However, we do move around exams that are subject to exam precedence constraints; these simply can be switched afterwards if the order is incorrect.

The improvement algorithm can be characterized as follows. First, we select the slot that has the most spots left in the corresponding room. Then, we will look in the set of feasible exams for improvements. We now have two options. First, we can use a greedy approach which moves the first feasible exam that qualifies as an improvement. Second, we can use an approach that computes the increase in penalty function for every feasible exam and then moves the exam with the best improvement. The second method results in larger improvements, but also comes with the cost of larger computational times per iteration. In some cases, the set of feasible exams is fairly large and evaluating the penalty function for every possible scenario can become computationally exhausting. After moving an exam to a certain slot, we update the timetable and move on to the next slot. Whenever there are no improvements possible for a selected slot, we remove the slot from a specific list, which we will refer to as *poss_spots*. The algorithm keeps selecting spots until all slots are removed from *poss_spots*. Algorithm 4 shows the improvement heuristic using a greedy approach. The algorithm for the second mentioned approach is very similar to Algorithm 4; the only thing that differs is how the exam v is selected.

Algorithm 4: Greedy improvement heuristic

```
while length(poss_spots) > 0 do
    r* = 0;
    p* = 0;
    a(r*, p*) = 0;
    for (r,p) in poss_spots do
        if a(r, p) > a(r*, p*) then
            r* = r;
            p* = p;                    /* Select slot with most available spots */
        end
    end
    feas = GetFeas(r*, p*) ;           /* A function that returns the set of
     feasible exams for slot (r*, p*) */
    curr_sol = ComputePenalty(Ξ) ; /* A function that returns the penalty
     score for a timetable Ξ */
    for v in feas do
        (r_old, p_old) = GetSlot(v) ;                    /* Obtain old slot */
        Remove v from Ξ[r_old, p_old];
        Add v to Ξ[r*, p*];
        score = ComputePenalty(Ξ);
        move = False;
        ;    /* A boolean variable that states if an exam can be moved */
        if score < curr_sol then
            curr_sol = score ;
            a(r*, p*) = a(r*, p*) − n(v);
            d(r*, p*) = d(v);
            move = True;
            break;      /* Break out of for loop once a better timetable is
             found */
        else
            Remove v from Ξ[r*, p*];              /* Return to old timetable */
            Add v to Ξ[r_old, p_old];
        end
    end
    if move == False then
        Remove (r*, p*) from poss_spots ;     /* No improvements for this slot
         possible */
    end
end
```

5 Computational Analysis and Implementation

In this section, we will apply the algorithms to the test data described in Section 2.3.2. Additionally, we will provide a computational analysis of the performance for different algorithms and parameter settings. The algorithms are written in programming language Python and are implemented using the Spyder Integrated Development Environment (IDE). The experiments are performed on a 2.4 GHz Intel Core i7-5500 with 4 GB of memory running under Windows 10.

5.1 Construction Algorithm Implementation

5.1.1 First Phase

We start with grouping the exams into clusters, as illustrated in Algorithm 1. One of the factors that determines how exams are clustered is the spread of the preassigned exams. The periods which contain preassigned exams form the basis for the clusters. We hence need to check if these preassigned exams are evenly spread and if the number of clusters is sufficiently enough. Table 10 summarizes the data on the preassigned exams.

Period	Preassigned Exam(s)	Adjacent periods	Period	Preassigned Exam(s)	Adjacent periods
2	4	1, 2, 3	27	425	27, 28
5	130, 407, 479	4, 5, 6, 7	29	527	29, 30
9	86, 524	8, 9	31	325	31
10	256	10, 11	32	41	32
13	396	12, 13, 14	33	405	33, 34, 35
16	368, 543	15, 16	38	574	36, 37, 38
17	171	17	39	131, 339	39, 40
18	321	18, 19	42	91, 511	41, 42
21	424	20, 21	43	243	43, 44, 45
23	506, 560	22, 23	47	88	46, 47, 48, 49, 50
24	400	24, 25	54	269	51, 52, 53, 54
26	195	26			

Table 10: Data on preassignments.

The preassigned exams seem to be spread out quite evenly throughout the whole semester and we assume that the 23 different clusters provide an acceptable setting for the algorithms to run properly. If this is not the case, we always can adjust the number of clusters to obtain a more useful setting. For now, we do not need to introduce any artificial preassigned exams to correct for the spread of the preassigned exams.

Before Algorithm 1 is run on the test data, we need to set the parameter α, which was introduced in Section 3.1.3. The parameter α essentially determines the distribution of the exams over the clusters. Low values of α will lead to few exams being assigned to the regular clusters and more exams being assigned to the null cluster. Contrary to this, high values of α will leave more available spots for exams to be assigned into the regular clusters. However, this does not necessarily mean that fewer exams will end up in the null cluster; if too many exams are assigned to the regular clusters, Algorithm 2 will fail to schedule some of the exams and the unscheduled exams will end up in the null cluster after all.

The parameter α also has another important implication. Observe that when $\alpha = 0$, the first phase of the construction algorithm is skipped and all the exams end up in the null cluster. Then, all the exams will be scheduled directly into the final timetable from the null cluster. This approach essentially is the same as a sequential method where exams are ranked by size, as discussed in Section 1.2.2. Thus, the method proposed in this thesis can also be regarded as a hybrid between a cluster and sequential method where the parameter α serves as a weighting factor between the two methods.

To study the exam distribution of the clusters for different values of α, we compute the ratio of sittings that are assigned to a regular cluster to the total sittings and plot these in Figure 2.

We can separate several different patterns in Figure 2. First, observe that for $\alpha = 0$, the graph touches the positive vertical axis. The reason for this is that the preassigned exams are always assigned to a cluster and do not depend on α. Once we start to increase α, we allow exams to be assigned to regular clusters and the ratio slowly increases. Then, as we continue to increase α, the ratio grows linearly with α. The extra available spots caused by the increase in α are directly translated into more exams being assigned to regular clusters.

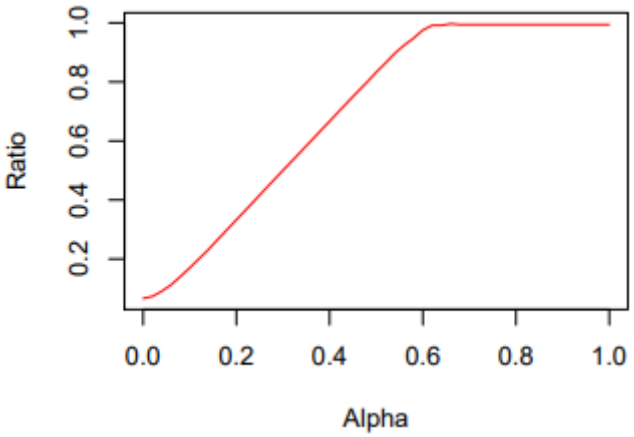

Figure 2: Ratio of regular clustered sittings to total sittings for different α.

The graph also exhibits an interesting relationship once it approaches the ratio of 1.0. First, observe that for large values of α, the number of available spots is not a limiting factor; it only determines the order in which clusters are considered when exams are assigned. However, once we lower α, the number of available spots in certain clusters starts to approach 0. As a result, some exams are not scheduled into a regular cluster, which would have been the case for higher values of α. Note that in this critical region, Algorithm 1 can produce very different clusters for small changes of α due to the low availability of rooms, even leading to a decrease in the ratio for higher values of α. This is also the case in our test data, where the ratio decreases from 0.9968 at $\alpha = 0.66$ to 0.9942 at $\alpha = 0.68$.

Although Figure 2 displays an interesting relationship between α and the cluster distribution, it does not necessarily provide us with an optimal value of α. We first need to schedule all the exams into the final timetable to be able to determine the performance of a solution.

5.1.2 Second Phase Implementation

To continue our analysis of α, we will apply the second phase of the construction algorithm to multiple cluster distributions with different α. We will then determine if the algorithm has produced a feasible solution. If this is the case, we compute the penalty score to determine the performance of the produced timetable. Using this approach, we can evaluate multiple values for α and see which one results in the best timetable. We will use the weight vector $w = (10, 5, 2, 25, 25, 50, 25)$ for violations of the soft constraints, as introduced in Section 4.1. Table 11 summarizes the outcomes for different α and gives

a decomposition of the penalty score.

α	Feasibility	s_1	s_2	s_3	s_4	s_5	s_6	s_7	Total
0	✓	8550	5185	16230	225	300	0	75	30565
0.05	✓	6710	4070	15864	275	275	250	50	27494
0.1	✓	6620	5330	14144	375	250	50	25	26794
0.15	✓	3820	6040	14506	275	350	100	25	25116
0.2	✓	7340	4700	13650	325	400	200	0	26615
0.25	✓	5530	3875	13544	325	425	150	25	23874
0.3	✓	5090	3595	12860	400	325	100	0	22370
0.35	✗	5260	3125	11324	225	225	100	50	20309
0.4	✗	5200	4660	12146	225	325	50	25	22631
0.45	✓	5060	2350	11156	325	300	200	0	19391
0.5	✓	4590	1415	9746	300	250	100	25	16426
0.55	✗	2270	1325	9172	325	300	100	25	13517
0.6	✓	3330	1280	9752	350	300	50	25	15087
0.65	✗	3040	685	8606	300	325	50	0	13006
0.7	✓	2580	545	8008	375	325	100	0	11933
0.75	✗	2720	495	8450	350	325	50	0	12390
0.8	✗	2830	675	8134	300	350	50	0	12339
0.85	✗	3670	1275	9126	350	325	50	25	14821
0.9	✗	2560	695	8260	400	375	50	25	12365
0.95	✗	2750	630	8866	325	300	50	25	12946
1	✗	2750	630	8866	325	300	50	25	12946

Table 11: Penalty score decomposition of timetables for different values of α.

Table 11 reveals multiple interesting results. First, observe that not all constructed timetables are feasible. Especially for higher values of α, we see that less timetables are feasible. This result can be partly explained as follows. For lower α, more exams end up in the null cluster after applying Algorithm 1. When scheduling the exams from the null cluster in Algorithm 2, we look into the entire timetable and select the most suitable slot to schedule the exam into. In contrast to this, when we are scheduling exams from regular clusters, we iteratively select periods and schedule an exam into that specific period if a feasible room is available. The latter approach leads to a less efficient allocation of exams across the slots in the sense of feasibility. As a result, for larger α, we might end up with a set of exams that are not possible to schedule into the final timetable, leading to an infeasible solution. Note however that the above analogy does not always apply. As can be observed in Table 11, the timetable constructed using $\alpha = 0.35$ is infeasible, yet the timetable that

results from the construction algorithm using $\alpha = 0.45$ is feasible. The above reasoning describes a more general trend while the actual final timetable depends on many different factors, such as exam sizes, exam types, room capacities and exam durations.

Another general trend that follows from Table 11 is the decrease in the total penalty score for higher values of α. This result can be directly attributed to the decrease in the penalty scores s_1, s_2 and s_3. Indeed, the penalty scores s_4, s_5, s_6 and s_7 seem to be unaffected by changes in α. This is in line with the design of the construction algorithm; we did not consider any data on the disabilities of students in Algorithm 1 and Algorithm 2 and thus the penalty scores s_4, s_5, s_6 and s_7 should be randomly distributed. Observe that the decrease in penalty scores s_1, s_2 and s_3 is also in line with the design of the construction algorithm. To minimize adjacent conflicts, we introduced clusters of which the members were intended to be scheduled next to each other. Moreover, such an approach would not only significantly reduce the number of adjacent clusters but would also optimize the students' exam spread. Since a student is only required to take at most one exam in every cluster, it is less likely that an exam spread penalty is incurred. The results in Table 11 provide the evidence of this reasoning.

Now, for determining the best value for α, we look at both the feasibility component and the penalty score component. We see that for $\alpha = 0.7$, the total penalty score is minimal. Furthermore, the timetable corresponding to $\alpha = 0.7$ also happens to be feasible. However, this does not necessarily imply that $\alpha = 0.7$ yields the best timetable. Note that the total penalty score is a piece-wise function of α. We have only sampled observations from this function, so it is likely that there exists another value for α that yields a better timetable. To determine the optimal value of α, we thus should evaluate the total penalty for all $\alpha \in [0, 1]$, which is computationally impossible. For the remainder of this thesis, we hence decide to select the timetable that corresponds to $\alpha = 0.7$ to be used in any further analysis.

5.2 Improvement Algorithm Implementation

In Section 4.2, we have proposed two different approaches for the improvement algorithm. The first method moves the first feasible exam to a spot that leads to a reduction in the penalty score. The second method instead evaluates all possible exams and then selects the exam that leads to the biggest reduction in the penalty score. In this section, we apply both methods to the constructed timetable that resulted from Section 5.1 to gain insights on their performances. We will run both algorithms for five hours and see how the penalty score develops over time. The results are displayed in Figure 3.

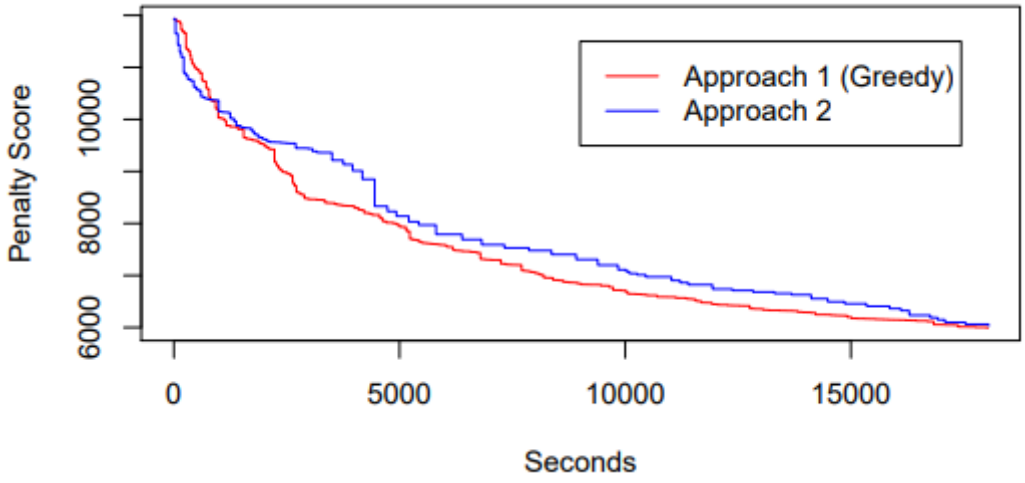

Figure 3: Development of the penalty score over time for different versions of the improvement algorithm.

First, observe that the curve corresponding to the first approach is smoother than the curve corresponding to the second approach. This can be explained by the fact that approach 1 performs more iterations per hour than approach 2. However, the iterations in approach 1 yield smaller improvements on average, which is in line with the design of the algorithms.

From Figure 3 it is not immediately visible which approach generates the best results. Both approaches seem to yield approximately the same penalty score after 5 hours. However, we can say that the first approach is less subjective to changes in the input dataset; it always moves the first improvement. The second approach, however, does highly depend on the input dataset. For input sets that are much larger than our dataset, it may become computationally difficult to check all the exams for every iteration in the second approach.

We thus might prefer to use the greedy approach over the second approach because of its robust character.

Finally, observe that after implementing the construction algorithm and running the greedy improvement algorithm for five hours we end up with a feasible timetable that has a total penalty score of 5986, which boils down to an average penalty of 0.7594 per student. This number is relatively small considering that we are using weight vector $w = (10, 5, 2, 25, 25, 50, 25)$ for the soft constraint penalties.

6 Summary and Concluding Remarks

In this thesis, we have tackled a new setting for the real-life examination timetabling problem that applies to The University of Edinburgh. Multiple new types of hard and soft constraints are introduced alongside a modification of the popular ITC 2007 dataset. A construction algorithm is proposed that exploits the structure of clusters by scheduling non-conflicting exams in adjacent periods. The construction algorithm iteratively schedules the exams and incorporates the structure of the current timetable by looking at the number of available spots. The number of available spots again is tweaked by some parameter α. Using this methodology, the proposed construction algorithm is presented as a hybrid between a sequential method and a cluster method. It is shown that higher values of α provide us with more efficiently constructed timetables that result in lower penalty scores. However, this comes with the cost that higher values for α produce timetables that are more likely to be infeasible. Further analysis reveals that determining the optimal value for α is not a simple task: the performance of the construction heuristic highly depends on the character of the input data. A proposed rule is to select the timetable that corresponds to the α with the lowest total penalty score while still remaining feasibility. In addition, two different versions of an improvement algorithm are presented alongside a penalty function to measure the degree of soft constraint violation. A neighborhood is defined in which one exam is moved at the time. It is shown that both algorithms perform equally well on the test data, but a greedy approach is preferred because of its robust character. Finally, we conclude that the algorithms have accomplished to produce a feasible timetable whose average penalty score per student is relatively low in context of the weighting vector that is used.

This thesis has laid the foundations for a new setting of the real-life examination scheduling problem. However, the work in this thesis can be extended in many ways. First, future research could consider incorporating additional hard and soft constraints. Common constraints that have not been included in this thesis deal with exam exclusions, early scheduling for large exams and period/room preferences. Adding such constraints might add extra difficulties to the scheduling problem, but also allows such models to be more versatile and applicable to different kind of institutions. Such approaches could also give useful insights on how the algorithms perform on other types of data.

Future research may also aim at improving the proposed algorithms. In this thesis, the structure of the preassignments determined in what fashion the clusters were constructed.

In contrast, one could also use other selection criteria, such as approaches based on the chromatic number to form the clusters. Finally, the improvement phase can be altered to obtain better solutions in lower computational times. This dissertation considered a neighborhood where one exam at the time is moved. Future research could look at different neighborhoods such as a neighborhood that switches around two exams. Another way of enhancing the improvement phase is to implement a more advanced search algorithm. Approaches based on tabu search, simulated annealing or great deluge algorithms have been popular recent choices in the field of examination timetabling.

References

[1] Alzaqebah, M., & Abdullah, S. (2015). *Hybrid bee colony optimization for examination timetabling problems. Computers & Operations Research, 54*(0), 142-154.

[2] Arani, T., Karwan, M., & Lofti, V. (1988). *A Lagrangian relaxation approach to solve the second phase of the exam scheduling problem. European Journal of Operational Research, 34*(3), 372-383.

[3] Barham, M. A., & Westwood, B. J. (1978). *A Simple Heuristic to Facilitate Course Timetabling. Journal of the Operational Research Society, 29*(11), 1055-1060.

[4] Boufflet, J. P., & Négre, S. (1996). *Lecture Notes in Computer Science, 1153*, 328-344.

[5] Brlaz, D. (1979). *New methods to color the vertices of a graph. Communications of the ACM, 22*(4), 251-256.

[6] Broder, S. (1964). *Final examination scheduling. Communications of the ACM, 7*(8), 494-498.

[7] Burke, E. K., Newall, J. P., & Weare, R. F. (1998). *Initialization Strategies and Diversity in Evolutionary Timetabling. Evolutionary Computation, 6*(1), 81-103.

[8] Caramia, M. F., Italiano, G., & Dell'Olmo, P. (2001). *Lecture Notes in Computer Science, 1982*, 230-241.

[9] Caramia, M. F., Dell'Olmo, P., Italiano, G. F. (2008). *Novel Local-Search-Based Approaches to University Examination Timetabling. INFORMS Journal on Computing, 20*(1), 86-99.

[10] Carter, M. (1986). *A Survey of Practical Applications of Examination Timetabling Algorithms. Operations Research, 34*(2), 193-202.

[11] Carter, M. W., Laporte, G., & Chinneck, J. W. (1994). *A general examination scheduling system. (EXAMINE). Interfaces, 24*(3), 109-120.

[12] Carter, M. W., & Laporte, G. (1996). *Recent developments in practical course timetabling. Lecture Notes in Computer Science, 1153*, 3-21.

[13] Casey, S., & Thompson, J. (2003). *GRASPing the examination scheduling problem. Lecture Notes in Computer Science, 2740*, 232-244.

[14] Cole, A. J. (1964). *The Preparation of Examination Timetables Using a Small-Store Computer. The Computer Journal, 7*, 117-121.

[15] Corne, D., Fang, H., & Mellish, C. (1993). *Solving the modular exam scheduling problem with genetic algorithms. Proc. of the Sixth International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems.*

[16] Côté, P., Wong, T.K., & Sabourin, R. (2004). *Application of a hybrid multi-objective evolutionary algorithm to the uncapacitated exam proximity problem. Lecture Notes in Computer Science, 3616,* 151-168

[17] Desroches, S., Laporte, G., & Rousseau, J. (1978). *Horex: A Computer Program For The Construction Of Examination Schedules. INFOR: Information Systems and Operational Research, 16*(3), 294-298.

[18] Dowsland, K., & Thompson, J. M. (2004). *Ant colony optimization for the examination scheduling problem. Journal of the Operational Research Society, 56*(4), 426.

[19] Gogos, C., Alefragis, P., & Housos, E. (2008). *A multi-staged algorithmic process for the solution of the examination timetabling problem. In Proceedings of the 7th international conference on practice and theory of automated timetabling.*

[20] Hamilton-Bryce, R., McMullan, P., & McCollum, B. (2013). *Directing selection within an extended great deluge optimization algorithm. In Proceedings of the 6th multidisciplinary international conference on scheduling*

[21] Hertz, A. (1991). *Tabu search for large scale timetabling problems. European Journal of Operational Research, 54*(1), 39-47.

[22] Johnson, D., (1990). *Timetabling University Examinations. Journal of the Operational Research Society, 41*(1), 39-47.

[23] Karp, R. M. (1972). *Reducibility among combinatorial problems. Complexity of Computer Computations*, 85-104.

[24] Laporte, G., & Desroches., S. (1984). *Examination timetabling by computer. Computers and Operations Research, 11*(4), 351-360.

[25] Leong, Y. L., & Yeong, W. Y. (1990). *A Hierarchical Decision Support System for University Examination Scheduling. Working paper.*

[26] Lofti, V., & Cerveny, R. (1991). *A Final-Exam-Scheduling Package. Journal of the Operational Research Society, 42*(3), 205-216.

[27] Peck, J., & Williams, M. (1966). *Algorithm 286: Examination scheduling. Communications of the ACM, 9*(6), 433-434.

[28] McCollum, B., Schaerf, A., Paechter, B., McMullan, P., Lewis, R., Parkes, A. J., et al. (2010). *Setting the research agenda in automated timetabling: The second international timetabling competition. INFORMS Journal on Computing, 22*(1), 120-130.

[29] McCollum, B., McMullan, P., Parkes, A. J., Burke, E. K., & Qu, R. (2012). *A new model for automated examination timetabling. Annals of Operations Research, 194*, 291-315.

[30] Mehta, N. K. (1981). The Application of a Graph Coloring Method to an Examination Scheduling Problem. Interfaces, 11, 57-67.

[31] Müller, T. (2009). *ITC2007 solver description: A hybrid approach. Annals of Operations Research, 172*(1), 429-446.

[32] Naji Azimi, Z. (2004). *Comparison of metaheuristic algorithms for Examination Timetabling Problem. Journal of Applied Mathematics and Computing, 16*(1-2), 337-354.

[33] Qu, R., Burke, E., McCollum, K., Merlot, B., & Lee, L. (2009). *A survey of search methodologies and automated system development for examination timetabling. Journal of Scheduling, 12*(1), 55-89.

[34] Thompson, J. A., & Dowsland, K. (1995). *Variants of simulated annealing for the examination timetabling problem. Annals Of Operations Research, 63*, 105-128.

[35] Thompson, J. A., & Dowsland, K. (1996). *General Cooling Schedules for a Simulated Annealing Based Timetabling System. Lecture Notes in Computer Science, 1153*, 345-363.

[36] Ülker, E., Özcan, E., Korkmaz, E. (2006). *Linear linkage encoding in grouping problems: applications on graph coloring and timetabling. Lecture Notes in Computer Science, 3867*, 347-363.

[37] Welsh, D. J. A., & Powell, M. B. (1967). *An upper bound for the chromatic number of a graph and its application to timetabling problems, The Computer Journal, 10*(1), 85-86.

[38] White, G., & Chan, P. (1979). *Towards The Construction Of Optimal Examination Schedules. INFOR: Information Systems and Operational Research, 17*(3), 219-229.

[39] Wood, D. C. (1968). *A system for computing university examination timetables, The Computer Journal, 11*(1), 41-47.

Appendix

A Python Code

The complete Python code that implements the described algorithms consists of more than 700 lines and thus it is decided to not include the code here. Instead, we provide a link to GitHub page where the code can be inspected:

`https://github.com/s1877621/Master-Thesis`