Solving a highly constrained Dutch school timetabling problem

Wesley Enrico Jacobs

Master Thesis

Supervisor: Han Hoogeveen
Second Supervisor: Marjan van den Akker

Utrecht University

Computing Science
Utrecht University
May 2021

Abstract

Constructing a timetable for high schools in the Netherlands comes with many problems. One of the most prominent problems is how to assign students to classes, specifically for students that are in their second half of their education. Commonly, class assignments are made during the creation of cluster schemes. We investigate a local search method to create cluster schemes and compare this against the existing heuristic. Additionally, we create a new method to create the class assignments, which focuses on maximizing the number of classes that can be scheduled together. We make comparisons between the class assignments produced by this method and the class assignments gained from creating cluster schemes. This comparison is made by creating initial schedules using the class assignments and improving them with use of local search.

Acknowledgments

I am deeply grateful to Robin de Munck from Pontes Oranjeweg for providing me with the data used during this thesis, as well as providing helpful insights into defining the quality of a school timetable.I would also like to extend our sincere thanks to Han Hoogeveen, supervisor from the University Utrecht, without whom this thesis would not be possible. I very much appreciate Denise Pieper and Erik Sander, who provided helpful insights during the writing of this thesis.

Contents

Chapter 1

Introduction

During high school, it is not uncommon for students to complain about their timetable. These complaints can be about having too many idle hours, doubly scheduled lessons, or simply lessons being scheduled on the last timeslot of the day. Although these points are taken into consideration when the schedules are created, how come there are still so many issues?

To answer this question, we first define what the school timetabling problem is. The problem consists of three core components: students, teachers, and rooms. Each student follows a list of subjects, such as mathematics, biology or history. The students need to attend a number of lessons for all subjects they follow. The lessons of a subject are followed as a class, a group of students, and all lessons are normally given by the same teacher. Each lesson needs to be assigned both a timeslot and a room. This needs to be done in such a manner that all students, teachers, and rooms are only scheduled once per timeslot. If this is not the case, a clash occurs. The primary goal of the school timetabling problem can be summarized as scheduling all lessons without any clashes.

However, this is not all there is to the problem. Every school has its own set of preferences that add complexity to the problem. There are a couple of common constraints that are required in almost every instance. Teachers might be unavailable on certain days, or subjects may require specific rooms, such as labs, gyms or music rooms. Other subjects may need double hours, two lessons scheduled in a row, to be taught properly. While not an absolute requirement, both students and teachers alike prefer not to have idle hours. As such, a common request is to minimize these, resulting in a compact schedule.

Another issue that adds complexity is the situation in the Netherlands. Here, students have very varied choices in their elective subjects. As this is the case, it is likely that two elective subjects share at least one student. This student should still be able to follow both subjects, so the classes of these subjects the student is in cannot be scheduled at the same timeslot. This also brings us to another problem. If many students follow a subject, there will be multiple classes for that subject. Deciding which students are assigned to which class can make a crucial difference in the quality of the solution.

So as to answer the question posed at the start, there are many remaining complaints about the schedule due to the many components of the problem, each adding a new layer of complexity. As a measure to make scheduling easier, schools in the Netherlands commonly make use of cluster schemes. A cluster scheme consists of multiple groups of classes. The classes within such a group, called a cluster line, are always scheduled together, on the same timeslot. This in turn reduces the complexity, as there are less possible variations. Constructing a good cluster scheme is a problem on its own. However, this can be handled separately of the scheduling problem. We will further expand on cluster schemes in Section 1.1.

In this thesis, we look into solving a highly constrained school in the Netherlands. First, we go over some terminology used throughout this thesis in Section 1.1. Secondly, we examine literature in the area of school timetabling in general, followed by literature on cluster schemes. In Section 2.1 we introduce the constraints of our problem instance, while we dive further into the specifics of the instance in Section 2.2. Afterwards, we go in detail on our approach to the problem in Section 3. In Section 4 we discuss the results we find using the discussed approach. Finally, we finish with a concluding statement in Section 5.

1.1 Terminology

In this section, we go over some core terminology used in this thesis. A list of the terms is also provided at the end of this section. To start, we expand on the situation in the Netherlands. High schools have 3 levels of difficulty: vmbo, havo and vwo. These levels respectively study for 4, 5 and 6 years. The first few years are called the lower years, the first 2 years for vmbo and first 3 for havo and vwo. During the lower years, students within the same base class generally follow all of their subjects together. Once students enter the upper years, they choose a profile. A profile consists of the mandatory subjects, profile specific subjects, and elective subjects. Scheduling the profile specific subjects and elective subjects is the crux of the problem, which cluster schemes attempt to address.

Idle/Free hours
As aforementioned, idle hours are to be avoided. However, a distinction between idle hours and free hours must be made. If an unscheduled timeslot has no preceding or succeeding lessons during that day, it is a free hour. Otherwise, there is both a lesson before and a lesson after the timeslot, which makes it an idle timeslot. A compact schedule minimizes the number of idle hours, but has no regard for the number of free hours.

Double Hours
Some subjects require double hours. A double hour is defined by a teacher teaching a class the same subject for two consecutive hours. A common subject that requires double hours is physical education. For most subjects however,

multiple lessons with one class on the same day is undesirable, as no homework can be given between lessons. If it does happen that two lessons are scheduled on the same day, it is often preferred for there to be a double hour rather than two separate lessons.

Cluster schemes

Cluster schemes are used to help solve the timetabling problem. They do so by fixing the combinations of classes that can be assigned to the same timeslot. Each set of classes that are to be scheduled together is called a cluster line. During scheduling, instead of finding multiple classes to schedule on a timeslot, we only need to find one cluster line to schedule.

Each class has a number of lessons that need to be scheduled. As such, each cluster line needs to be scheduled multiple times as well. Considering that all lessons of each class need to be scheduled, a cluster line needs to be scheduled a number of times equal to the highest number of lessons of all classes in the line. The number of times a cluster line needs to be scheduled is referred to as the length of the cluster line. Similarly, the length of a cluster scheme is the sum of the lengths of the cluster lines.

Note that not all subjects in a cluster line need to have an equal number of lessons. As a result, sometimes classes do not have a lesson, even though it is in the scheduled cluster line. This causes students of that class to have either a free or idle hour.

Cluster line 1	Economics 2	Biology 1	History
Cluster line 2	Geography	Physics	Economics 1
Cluster line 3	Biology 2	Mathematics 2	Chemistry
Cluster line 4	Mathematics 1	German	Music
Cluster line 5	Informatics 1	French 2	
Cluster line 6	French 1	Informatics 2	

Table 1.1: An example cluster scheme. The numbers indicate different classes of the same subject.

List of terms

- Subject: A subject that is taught to students, such as mathematics, economics or english.
- Class: A group of students that follow all lessons of a subject together.
- Base Class: A group of students that follow all their main subjects together.
- Level: a large group of students in the same year and same level of schooling, such as first year of vwo or the third year of havo.
- Lesson: A teacher teaching a class about a subject.

- Clash: When a teacher, student or room is scheduled multiple times in the same timeslot
- Free hour: An empty timeslot at the start or end of the school day, in which the student or teacher does not have to be at school.
- Idle hour: An empty timeslot with lessons scheduled both earlier and later in the day.
- Double hour: A lesson that takes place during two consecutive hours.
- Compact schedule: A schedule focused on minimizing idle hours.
- Cluster scheme: A set of cluster lines.
- Cluster line: A set of classes that are to be scheduled at the same timeslot.
- Cluster line length: number of times requires to completely schedule the cluster line.
- Cluster scheme length: sum of the lengths of each cluster line in the cluster scheme.

1.2 Literature Study

In Stichting et al. [18], instances of the class-teacher timetabling problem are classified into one of three categories: school, course or examination timetabling. The main goal of all three is the same: to avoid teacher, student and room clashes. In the case of school timetabling, classes have either no or complete overlap of students. In other words, a class follows all of their subjects together. This is not the case for course timetabling, in which students can have more variation in their subjects and creating such classes is not possible. Completely avoiding student clashes is often not seen as reasonable, and instead the focus shifts to minimizing the overlap of lectures that have students in common. Examination timetabling once again completely avoids overlap, and additionally tries to spread out the exams of students as far as possible. Completely avoiding overlap can be possible, as this category commonly has less exams to schedule than course timetabling has lessons. According to these definitions, the problem in the Netherlands that involves the upper years is most similar to course timetabling. As for the lower years, the problem fits the definition of school timetabling perfectly.

Although we now have classifications for our problem, using two different classifications for one problem is not ideal. Instead, Drexl and Salewski [5] made a distinction between school timetabling and academic course timetabling, based on the compactness requirements of the schedule. School timetabling often values compact schedules highly, while the opposite holds for course scheduling. Going by these definitions, the problem falls under School timetabling.

In the rest of this section, we will first investigate common methods to solve the school timetabling problem, as defined by Stichting et al. [18], henceforth called the class-teacher timetabling problem to avoid confusion. We start with this as this problem is more commonly found in other countries, and as such has been researched more in depth. Afterwards, we discuss some research on cluster schemes, which are used on the school timetabling problem as defined by Drexl and Salewski [5].

1.2.1 Class-Teacher timetabling

The problem we are interested in is not directly solvable with methods applicable to the class-teacher problem. As mentioned earlier, the class-teacher timetabling problem is defined by classes having either complete or no overlap of students. A class of students follows all of their lessons together with that group of students. In most instances, the composition of classes is part of the input data to the algorithm. In the school timetabling problem, this would be akin to scheduling the lower years. For the upper years however, the problem is only similar if a cluster scheme is used and students are already assigned to classes. This is because a cluster line can be seen as a very large class that requires multiple rooms and teachers. Many methods that we will see are already capable of handling such cases or can easily be modified to do so. As such, we investigate the solution approaches used for the class-teacher problem.

Pillay [11] made an overview of research on the class-teacher problem, in which both the approach and the considered constraints are shown. Evolutionary algorithms are the most popular method, followed by local search-based methods, with integer programming being a close third. A few comparative studies exist, although most research is devoted to creating new methods or improving existing ones. The latter can be done by, for example, improving the neighbourhood search in local search-based methods. If a new method is created, it is commonly compared to existing methods. The improvements within a method, however, are often only compared to solutions within the same methodology. As such, comparisons between methods are scarce.

A more recent survey by Tan et al. [19] focuses on the advancement of the field, considering the different methodologies. They take note of the recent notable papers for each methodology. From this, they find a shift in popularity from local search-based methodologies to mathematical optimization methods, such as linear programming. Additionally, they discuss several proposed data formats for the problem, taking their respective data sets in consideration.

The International Timetabling Competition placed its focus on the class-teacher timetabling problem in 2011. For this competition, a xml-standard for datasets was agreed upon [13]. At the time of writing, about 50 datasets are publicly available, along with their solutions and respective objective scores. Some datasets from the Netherlands are available, which include a cluster scheme for solutions to use.

In the rest of this section we will go over some of the more popular approaches. We examine two categories of metaheuristics, the first based on evo-

lutionary algorithms, and the second based on the principle of local search. Afterwards, we evaluate the more exact approach of linear programming.

Evolutionary Algorithms

Evolutionary algorithms are population-based metaheuristics. They function by keeping a set of solutions, known as the population. With each iteration, also called a generation, the population changes, using the following principles found in evolution: selection, crossover, and mutation. In the selection phase, individuals from the population are selected to pass on their genes to the next generation. From these individuals, crossover is used to create new solutions by combining existing solutions. Once new solutions have been created, mutation can be used to make a small change inside each solution. Commonly, a form of elitism is used to ensure that the overall quality of the population does not degrade. This is achieved by, for example, maintaining the best solution for the next generation.

For the class-teacher timetabling problem, evolutionary approaches can be split into two categories. The first category applies the evolutionary principles to a timetable. The second category makes use of a string of operations, which is used to create the timetable. The evolutionary operations are applied on the string of operations, instead of on the timetable itself. We will now discuss a paper from either category, after which we discuss other research in lesser detail.

Wilke et al. [22] used an approach from the first category. They chose to represent a timetable as a sequence of timetables, one for each class. This has two distinct advantages: a class can only be scheduled once per timeslot, and crossover between solutions is more intuitive. Crossover on a complete timetable consisting of all lessons is difficult to accomplish. With this data structure, crossover can consist of interchanging class-timetables between solutions. This is similar to crossover operations used on bit strings, a commonly used data structure, and as such, existing operators can be used. In their specific problem, teachers are to be assigned to lessons. For this, each teacher expresses a preference for each lesson, indicating whether they want, or do not want to give that lesson. Although the authors do not specifically mention when teachers are assigned to lessons, it can be assumed that it is done before or during the creation of the initial solutions. This assumption comes from their crossover and mutation operators, which do not change a lesson's assigned teacher.

Their algorithm starts by creating a population of initial solutions. An initial solution is created by assigning their lessons to random timeslots for each class. This is done in such a manner that there is only one lesson per timeslot for each class. They then iteratively repeat the procedure of evolutionary algorithms of selection, crossover, and mutation. The two best solutions of a generation are automatically selected. For the other solutions, fitness proportionate selection is used. This makes it so that the probability that a solution is selected is equal to its fitness divided by the total fitness scores of all solutions. The fitness of a solution is proportional to the violations of constraints, with a higher fitness

indicating a higher quality. Note that normally, the quality of a solution is higher when the score, computed as a sum of penalty points, is lower. Due to the nature of their instance, which use teacher preferences and some contradictory soft constraints, there is a lower bound for the quality of a solution. The exact conversion from score to fitness is not declared.

Once the parents are selected, they use a variety of crossover operators to create the new solutions, such as one-point crossover and uniform crossover. As for mutation, two different operators are used, both only capable of making small changes within a class-timetable. The first can swap the content of two timeslots. The second operator changes the assigned room of a lesson.

A problem that occurred is that solutions were often infeasible. To mitigate this, they added repair operators. These repair operations focus on fixing infeasibilities. For example, if a teacher has two lessons scheduled on one timeslot, one of the lessons is randomly moved to a new timeslot. They invoke the repair operators when the overall quality of the population does not increase for a number of generations. This is likely done because the repair operators are computationally more expensive than the normal iterations.

In addition to these repair operators, they added a reconfiguration operator. This operator randomly changes the probability that another operator is used. If we take the crossover operators as example, it could be that there is a probability of 40% to run one-point crossover, and a 20% chance to run uniform crossover. After the reconfiguration operator is used, this could change to 30% and 30% respectively.

This algorithm was then used to solve the timetabling problem of a German high school. According to them, the theoretical optimal solution would have a score of roughly 15.000. They found that while running the algorithm with just selection, crossover and mutation resulted in a score of about 20.000. When including the repair operators, they found a significant increase in quality, resulting in score of about 17.500. Finally, when they included the reconfiguration operator, the resulting solution had a score of about 16.500.

Raghavjee and Pillay [14] used a sequence of operations to create a timetable, and as such, their approach falls under the second category. In this category, the evolutionary operators modify a sequence of operations. This sequence is then used to create a timetable. Each sequence is then given a score, which is equal to the score of the timetable that is created using it. Before we go into the different operations that can be found in a sequence, it is useful to know what the constructed timetable looks like. In their implementation, a timetable is represented as a two-dimension matrix, in which a row represents a timeslot, and a column represents a class. Each cell of the matrix is either empty, or has a single lesson assigned to it, represented as a teacher-room pair. This structure is very similar to the sequence of class timetables that Wilke et al. [22] used. As such, the advantages are similar. In this case, a class can once again only be scheduled once per timeslot. As for the operations that can be done on the timetable, they used seven different operations:

- A) Allocate: Schedules a lesson to a feasible period with lowest increase in score. If there is no feasible period, allocate to a random period instead.
- D) Deallocate: Removes a random lesson from the schedule.
- 1) Two violation mutation: Swaps the contents of two periods which cause constraint violations.
- 2) One violation mutation: Swaps the content of a cell causing constraint violations with a random cell.
- 3) Random swap: Swaps the content of two random cells.
- 4) Row swap: Swaps the contents of two random rows.
- 5) One violation row swap: Swaps the contents of a row with a cell which causes a constraints violation and a random row.

To preserve the assignments of lessons to their respective classes, the swaps are only done within a column. The operations are used to both create and improve the timetables. The sequence of operations determines both which operations are done, and when. As mentioned, the sequences are the individuals that are modified by the evolutionary operators. They first select the parents using tournament selection. Mutation is then done by randomly selecting an operation and replacing it with a random operation. For crossover, they use the cut and splice operator. This operator takes two individuals, each a list of operations, and creates two new individuals. To do so, a random point in each individual is selected. The first new individual is then created by taking the part of the first old sequence before the point, and the part after the selected point in the second old sequence. The second new individual is created in the same way, but then the first part of the second individual is taken, and the last part of the first sequence. For example, if two sequences, $ADADAAA341$ and $21444ADADA$ are used, and points 4 and 8 are selected, the resulting individuals are $ADADDA$ and $21444ADAAAA341$.

Their implementation of the algorithm is split into two phases. The first phase is dedicated to finding a feasible timetable, and as such, the quality of an individual is linked to the hard constraint violations. Once a feasible timetable is found, it is stored. They then create a new population, and the second phase is started. This phase focuses on improving the stored timetable. The operations in a sequence are applied to the timetable, and the quality of that sequence is based on the improvement, with regard to the soft constraints. The best timetable that is found in phase 2 is then the resulting solution of the algorithm.

They compared the results of this approach to different methods using an artificial data set with small instances. On this data set, they found that their approach always found a feasible schedule. In comparison, the simulated annealing, neural network and genetic algorithm approaches did occasionally find

an infeasible solution. They did not compare the scores of the best found solutions between methods. When testing the algorithm on a data set of Greek high schools, it was outperformed by particle swarm optimization and simulated annealing approaches.

We have seen an example from both categories. Further research on the first category can be found in papers by Nurmi and Kyngäs [9] and Odeniyi et al. [10]. Nurmi and Kyngäs [9] use hill-climbing in combination with a tabu list to create their offspring, and investigate the use of an adaptive scoring method. Odeniyi et al. [10] integrate parts of simulated annealing into their approach, by using the acceptance function to determine whether a new solution will replace an old solution in the population. They compared this approach to generic simulated annealing, and found better results for two tests cases on a Nigerian school.

As for the second category, Bufé et al. [1] used a queue of events as their data structure. Each event corresponds to one lesson or a group of lessons. This queue represents the order in which the events will be scheduled. They then use a deterministic timetable builder to create a timetable from this queue. Similarly, Domrös and Homberger [2] also used a queue of lessons to be scheduled. They iteratively improve a single solution using only mutations, making this approach closer to local search than evolutionary algorithms. It is, however, still worth mentioning here, as using an order of operations is somewhat unique to evolutionary algorithms. This algorithm was successfully used for the International Timetabling Competition in 2011, reaching the finals of the competition.

Local Search

In contrast to evolutionary algorithms, local search algorithms usually keep track of a single solution. This solution is iteratively modified with small changes, so that a good solution can be found. This is similar to the mutation part of evolutionary algorithms. The set of solutions that can be found from the current solution is called the neighbourhood. Common neighbourhoods consist of swapping the timeslots of two lessons, moving a timeslot of a lesson, changing the assigned room, or a combination of these. This is akin to the mutation operator used in evolutionary algorithms. Multiple local search techniques have been used to solve the class-teacher timetabling problem, such as tabu Search, simulated annealing and hill climbing.

Zhang et al. [24] investigated an alternative neighbourhood compared to the commonly used single swap between timeslots. Instead, they investigated swapping multiple lessons between two timeslots. They do so by iterating over all possible swaps between these two timeslots in a random order. Each swap is checked by the acceptance function of simulated annealing, which determines if it is applied or not. They found that using this neighbourhood over a single swap resulted in more consistent quality of solutions. Furthermore, the higher

quality solutions are obtained faster.

Fonseca et al. [6] also used simulated annealing, but combined it with the usage of iterated local search. They first run simulated annealing, resetting the temperature a number of times. Then, if they still have computing time left, they run an iterated local search process to further improve the solution. Both processes utilise the same neighbourhood, consisting of:

- Swapping timeslots assigned to lessons
- Moving a lesson to a different timeslot
- Swapping the teachers assigned to lessons
- Assigning a new teacher to a lesson
- Kempe Chain: A chain of movements of lessons between two timeslots, in such a manner that the next move in the chain removes all conflicts created by the previous move.
- Reassigning Resource times: Changes the order of subjects in a teacher's schedule. For example, if a teacher teaches class 1, 2 and then 3, it could become, 3, 1 and then 2.

They implemented this algorithm for the International Timetabling Competition in 2011. It managed to improve the best solutions found thus far in the first round and had the overall best performance in the second and third rounds.

Souza et al. [17] opted to use tabu search, and placed their focus on an improvement procedure. This procedure is invoked when the tabu search stumbles upon a schedule without clashes. Before we go into this procedure, it is important to know that their timetable is represented as a timetable for each teacher. In this data structure, only one lesson can be scheduled per timeslot, so no teacher clashes can exist. While their timetable places its focus on the teacher's schedules, the improvement focuses on the classes. This is done by zooming in on a single class and creating a graph.

To start, a vertex is created for each timeslot the class is scheduled in. The vertex represents the timeslot. These vertices are connected by directed edges. An edge exists from v_1 to v_2 if the teacher scheduled on v_1 is available on v_2. An edge from v_1 to v_2 represents moving the lesson on v_1 to v_2. Each edge is assigned a cost, computed by taking the difference in solution score if the move it represents is performed. If this graph contains a cycle with a negative cost, a feasible schedule with a lower cost exists. This schedule can then be created by executing the moves that the edges in the cycle represent.

Using this improvement procedure improved the attained results significantly. This allowed the program to find results within seconds, of a quality that could not be attained without the improvement procedure.

Integer Linear Programs

Integer linear programs are models used to solve the problem in an exact manner. In contrast to the previous two methods, a solver for a linear program will find the optimal solution. Unfortunately, this comes with a significant increase in computation time, with instances of the class-teacher timetabling problem being unsolvable in a reasonable time frame. This is mostly due to the binary constraints on variables. If these are dropped, the program can be solved. The score of the solution we find is then a lowerbound of for the instance with binary constraints. To mitigate the unreasonable computation time, problems are often decomposed into smaller subproblems. This can significantly increase the computation time, but the drawback is that the found solution is no longer guaranteed to be optimal.

Santos et al. [15] used a column generation approach for finding lower bounds. This approach focuses on the individual timetables of teachers. More specifically, they create timetables for teachers, separated further by day. As such, the binary decision variable in the main program is λ_{tdj}, in which t is the teacher index, d is the day index, and j is an identifier for the specific allocation. λ_{tdj} has a value of 1 if the allocation is used, and 0 otherwise. Each allocation is accompanied by a constant x_{tdjcp}, which indicates if class c is taught on period p, with the other variables the same as before. Furthermore, a cost constant is used to track the costs of each allocation. To ensure that all lessons are scheduled, the main program makes us of constants r_tc, indicating how many lessons teacher t teaches class c. It should also be noted that they do not consider room assignments in their problem instance.

The main program handles three different constraints:

- Exactly one allocation is selected per teacher per day
- Each class is only scheduled one per timeslot
- All lessons are scheduled

Using the variables λ_{tdj} and constants x_{tdjcp}, they are respectively formulated as:

$$\sum_{j} \lambda_{tdj} = 1 \quad \forall t, d \tag{1.1}$$

$$\sum_{t} \sum_{j} \lambda_{tdj} x_{tdjcp} \leq 1 \quad \forall d, c, p \tag{1.2}$$

$$\sum_{d} \sum_{j} \sum_{p} \lambda_{tdj} x_{tdjcp} = r_{tc} \quad \forall t, c \tag{1.3}$$

The pricing problem is solved to find new allocations for each day, for each teacher. While finding new allocations, three constraints need to be respected:

- A lesson can only be scheduled on a timeslot if the teacher is available on that timeslot
- At most 1 lesson can be scheduled per timeslot
- A maximum of m lessons can be given to each class

By solving the pricing problem, we find new allocations that the main program can make use of. This approach allowed them to determine optimal solutions for instances that were available for a long time.

A similar approach was used by Dorneles et al. [4]. Their approach also makes use of column generation, however, instead of creating schedules per teacher per day, they create complete week schedules for teachers. An inherent advantage of this is that scheduling all lessons is easier to accomplish. In the approach by Santos et al. [15], the main program has to find an arrangement of day schedules in such a manner that this constraint is fulfilled. In contrast, if one creates a complete schedule, it is only required that it contains all lessons.

To find new schedules, they also solve the pricing problem. A difference, however, is that they take a graph approach to create new schedules. For each teacher, a directed graph is created. A path through this graph represents a schedule for that teacher. This graph is split into different sections, one for each day. Each day has a number of vertices equal to the number of timeslots + 1. Imagine an example instance with 4 timeslots, and as such, 5 vertices for a given day, numbered 1 to 5. The edges from vertex 1 to vertex 2 represent the lessons that can be given in the first timeslot. These edges are directed and can only go to a vertex with a higher number than the one it originates from. Idle hours are also included in this. An edge from vertex 2 to vertex 4 would be equal to an idle hour in timeslot 2, and a lesson in timeslot 3. To simulate free hours, days can start and end at any vertex. With this, any path through this section of the graph is a daily schedule, and by combining the path through all sections, a complete schedule is formed.

The pricing problem is then solved by finding a path in this graph. They do so with a linear program, using a variable x_{ta}, indicating that teacher t uses the edge a. This approach was compared to the algorithm by Santos et al. [15]. They found similar or better lower bounds, with a significant reduction in computing time.

Dorneles et al. [3] investigated a new way to solve the linear program. Ordinarily, a solver such as CPLEX is used to solve a linear program. In their approach, they continuously fix most of the variables, allowing only a small subset of variables to be changed and optimized by the program. They consider three different ways to fix variables, either fixing a number of classes, teachers, or days. By iteratively changing which variables are allowed to be optimized, a good solution can be found. As the algorithm progresses, the size of the subset to be optimized increases. For example, at first only the schedule of class A is free to be optimized. In the next iteration, the schedule of class A is also fixed,

but now the schedule of class B can be optimized. Once all classes have taken a turn, the program can move to optimizing multiple classes at the same time.

As mentioned, they considered three different ways to fix the variables. They found that fixing the days provided worse results than fixing the classes or teachers. Between these two, the best results were found by using a combination of fixing class schedules and teacher schedules. When compared to the general purpose solver CPLEX, they found an overall increase in quality, even when CPLEX was allotted ten hours of computing time, compared to the ten minute running time of their algorithm.

Comparative Studies

Seven algorithms are compared using a real-word problem by Wilke and Killer [20], including an evolutionary algorithm, tabu search and simulated annealing. While the evolutionary algorithm initially improved the solution at a rapid rate, simulated annealing ended on a better solution, while also requiring significantly less time to compute. Tabu search was slightly faster than simulated annealing, but found the worst solution of the three algorithms. Ultimately, they found that simulated annealing produces the best solutions, while maintaining a comparatively good computing time.

In Wilke and Ostler [21] a comparison between tabu search, simulated annealing, an evolutionary algorithm and a linear program with branch and bound is made. They compared these algorithms on an instance of a German high school. Tabu search could find good solutions quickly but had a violation of a hard constraint in every solution. Similarly, the evolutionary algorithm could not find a feasible solution, but was able to find a good solution, albeit slower than tabu search. The linear program using branch and bound produced the worst solution, while requiring the longest running time. Simulated annealing was the only algorithm capable of producing a feasible solution, with a running time between tabu search and the evolutionary algorithm.

1.2.2 Cluster Schemes

To be able to properly make use of a cluster scheme, it must adhere to some conditions. Two obvious requirements are that both students and teachers only appear once per cluster. Furthermore, each cluster line should include as many students as possible. Ideally, all students are included in a cluster line, so that none of them have an idle hour. However, if the classes in a cluster line have varying numbers of lessons, the students of a class with fewer lessons have an idle hour at one of the scheduled timeslots, so that needs to be taken into consideration as well. Finally, if we include the assignment of students to classes in the generation of the cluster scheme, there should not be a large disparity in number of students assigned to each class of a subject. Commonly, multiple cluster schemes are created. This is done as usually, students do not overlap between years and level of education. As such, creating a separate cluster scheme for each year of each level has very little influence on the quality of the cluster

scheme. This decreases the number of possible cluster schemes we can create, allowing for faster computation.

Post and Ruizenaar [12] use a heuristic enumeration algorithm to create cluster schemes for a school in Enschede. To do so, they first define the length of a cluster scheme as the number of timeslots the cluster scheme needs to be scheduled completely. This can also be defined as the sum of how often each cluster line needs to be scheduled. Each cluster line needs to be scheduled a number of times equal to the maximum a class in that line needs to be taught. For example, in a cluster line with 3 subjects that respectively need to be taught 2, 3 and 3 lessons, the cluster line needs to be scheduled 3 times. As such, cluster schemes with larger length require more timeslots to be scheduled, which leads to more potential idle time when a student is not scheduled in a cluster line. For this student, the timeslot is either an idle or free hour, depending on when on the day it is scheduled. If a cluster scheme has a larger length, the proportion of lines at the start or end of a day decreases. In turn, this increases the likelihood of idle hours. As such, they argue that the quality of a cluster scheme depends on its length, and a lower length indicates a higher quality cluster scheme. This metric takes both maximizing the number of students in a cluster line and possible differences in number of lessons into consideration.

The main goal of their algorithm is to minimize the length of the cluster scheme. During the creation of the cluster scheme, students are also assigned to the different classes of the subjects. They want each class of a subject to have a similar number of students, which is their secondary objective. The results of this method were compared to the commercial package that was used at the school. They found that their implementations found cluster schemes with lower or equal length. This method was later used in Haan et al. [7] to create a complete schedule for the same school.

Wood and Whitaker [23] investigated a similar problem for schools in New Zealand. Students there have a restricted choice in elective subjects due to computational constraints on the schedule. The authors wanted to confirm whether these computational problems still exist, and investigated a hypothetical situation where students have free choice over their elective subjects. In this situation, teachers are always available. To approach this problem, they first constructed a cluster scheme, then assigned students to the groups, and finally created a schedule using it.

Before they can create the cluster scheme, they set up an approximation for the number of student clashes between two subjects. They do so by considering how many groups the subjects have. If one of the subjects has two or more groups, they assume that student clashes can be avoided by scheduling the groups in different periods. If both subjects have exactly one group, the expected number of student clashes is equal to the number of overlapping students. This approximation is used when they create the cluster schemes.

The teacher assignment to classes is required as part of the input. They do so because most schools pre-assign teachers to classes. Furthermore, each

class has its own maximum number of students. They assume that a feasible clustering exists with no idle hours for students, and as such, limit the number of cluster lines to the maximum number of subjects that students can follow.

The cluster scheme is created using simulated annealing, which they initiate by randomly assigning groups to cluster lines. The neighbourhood is then composed of random interchanges of groups between lines. During this process, the following constraints are considered:

- All groups are placed in a cluster line
- The number of groups in a line is smaller than or equal to the number of rooms available

The cluster schemes are given a score, based on the number of teacher clashes and the expected number of student clashes using the aforementioned approximation. The goal of the algorithm is to minimize the score, while complying with the hard constraints. Once a good cluster scheme is found, they assign the students to the groups. This is done by taking a random student, assigning it to groups using the Hungarian assignment algorithm, and repeating until all students have been assigned.

After the groups are constructed, they assign the cluster lines to timeslots. They argue that the problem is similar to an incomplete Latin rectangle; a matrix in which a set of numbers need to be placed in such a way that every number appears only once per row and column. The matrix has a number of rows equal to the available periods per day, and a number of columns equal to the available days. The goal is then to place each cluster line in such a manner that it appears only once per row, and once per column. Each cluster line is required to be scheduled as many times as needed, so a number of times equal to its length.

The results were compared against schedules created by hand. They found that the program can construct timetables of similar quality compared to those created manually. A key difference is that the program can create schedules significantly faster, with the process taking a couple of minutes, compared to the days it takes to create one by hand.

Although implemented for the academic course scheduling problem, Shatnawi et al. [16] used a modified data mining algorithm to generate clusters. They modified the FP-growth algorithm [8], which is normally used to find frequent item sets in a transactional database, such as which items are frequently bought together in case of a store. This modified algorithm first sorts the subjects by number of students in descending order. For each student, they then sort the student's subjects based on the found order. This sorting is important for the creation of a tree data structure. The tree starts with an empty node, known as the root. The other nodes in the tree represent subjects, and each node keeps track of a counter. A subject can occur multiple times in the tree, however, two children of the same node cannot have the same subject. For each student's sorted list of subjects, they repeat the following procedure on the tree:

Taking a student's sorted list of subjects, they start with the first subject $S1$. To start, they look at the children of the root node. If one of the children represents $S1$, its counter is increased by 1. If there is no representing node, a new child of the root is created for $S1$, with the counter set to 1. For the second subject $S2$, we look at the children of the node of $S1$. The same process is done, if there is a child node representing $S2$, its counter is increased by 1, and otherwise a new node is created. This procedure is done for all subjects of a student.

Once the end of a student's list of subjects is reached, they return to the root, and restart the process with the next student. This is repeated until all students have been examined.

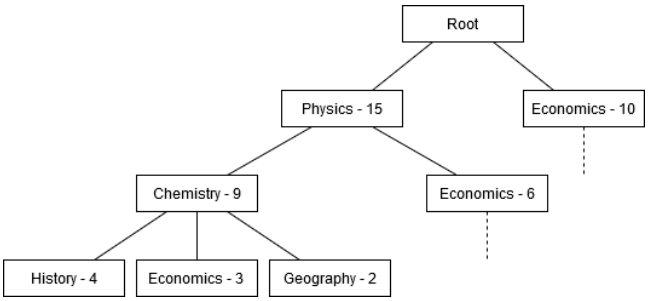

Figure 1.1: Example of a part of a tree constructed by the modified FP-growth algorithm

This process results in a tree in which each complete path represents one or multiple student's chosen set of subjects. Each level of this tree can be used as a cluster line. However, the number of students assigned to each group in the line varies greatly, with some groups having more students than should be allowed, while others have too few. The latter is fixed by searching for a different cluster line in which they can combine this group with another of the same subject. Groups with too many students are split into multiple groups.

Unfortunately, we do not think this algorithm is fit for the school timetable problem. In many cases of school timetabling, the number of groups is already set, and thus less flexible than the above algorithm would require. Furthermore, due to students having more subjects, there are many more unique combinations of subjects. This makes it so that the tree becomes very wide, with each node representing very few students. This in turn requires us to find new cluster lines for most groups, which defeats the point of the algorithm.

Chapter 2

Problem

2.1 Problem Description

The problem of interest is the school timetabling problem, specifically for high schools in the Netherlands. In general, schools are divided into three levels, vmbo, havo and vwo. Each level has its own curriculum with different subjects the students must follow. For example, vwo students are required to choose one specific type of mathematics from a variety of options, whereas this is not necessarily the case for havo and vmbo. Once a student reaches the upper years, starting from 3rd year of vmbo or 4th year of havo and vwo, they get to choose a basic profile and elective subjects. While the profiles do apply some restrictions on possible elective subjects, many different combinations are possible. This allows for a large number of classes with overlapping students, which makes scheduling more difficult. To counteract this, a common strategy is to make use of cluster schemes.

The specific instance of interest is 'Pontes', a secondary school in Goes, the Netherlands, with the data being from the academic year 2019-2020. They feature a timetable with 9 timeslots per day, and 5 days in a week. Each lesson is to be given by a prespecified teacher. For the lower years, the classes that follow each lesson are given, while for the upper years, the students have yet to be assigned to classes. It is, however, known how many classes are required for each subject and who they are taught by. Other than the regular constraints such as room requirements and schedule quality, some more specific constraints are called for. The complete list of constraints is:

Hard Constraints

1. Teachers can only be scheduled once per timeslot
2. Teachers have their requested number of days off per week
3. Rooms can only be scheduled once per timeslot

4. Some lessons require specific rooms such as gyms, music rooms or labs
5. Some lessons require multiple consecutive timeslots
6. Lower year students can only be scheduled once per timeslot
7. Lower year students have no scheduled idle hours
8. Some lessons have fixed timeslots
9. Upper year students are able to follow all their main subjects

In addition to these constraints, which define if a schedule is feasible, we have a number of soft constraints that determine the quality of the schedule:

Soft Constraints

10. Minimize number of missed lessons for upper year students
11. Minimize number of idle hours for upper year students
12. Minimize number of idle hours for teachers
13. Minimize teacher's movement between lessons
14. Minimize students moving between floors between lessons
15. Lessons of a subject should preferably be taught in a room dedicated to that subject
16. Lessons of a subject should be spread out over the week
17. Minimize the number of lessons scheduled in the 9th timeslot

The first six constraints are relatively basic constraints of the problem. Constraint 2 usually has specific days off for each teacher, but because this data was not made available, we limit this to their requested number of days off. Due to how heavily constrained the problem is, it may not always be possible for students to attend all of their lessons. As such, upper year students are allowed to miss lessons from their elective subjects, albeit at a heavy penalty.

The design of the school building plays a large role in the scheduling problem. Firstly, the building is divided into different sections for each subject. Preferably, a subject is given in a room associated with that subject, or as close by as possible. Secondly, when many students move between floors at the same time, the stairs may become congested, which can cause them to be late for their lessons. It is therefore requested that the movement of students between the three floors is minimized. If a student has to move between the first and third floor or vice-versa, it is counted as two floor movements. During a school day, there are two breaks, during which the students move to the ground floor. This movement does not count towards the quality of the schedule. Teachers have voiced a very similar preference but included movement in general. For them,

it is preferable to teach in the same classroom multiple times in a row, without having to move at all. This preference varies per teacher, but we have decided to include this. It is important to note that a higher priority is placed on teaching in a room associated with the subject, rather than minimizing movement of teachers.

Of course, the overall quality of everyone's schedule is important as well. As such, idle hours are penalized for both upper year students and teachers. Finally, the lessons of a subject are preferably spread out over the week. With this, teachers can properly assign homework between lessons. This also somewhat prevents students from having the same subject multiple times on the same day.

While in many schools the lower years have all of their classes together, it is not quite the case at this school. Here, the lower year students have extra curricula they can opt for. For example, there is an art track, in which the students have different music lessons, as well as acting or dancing. There may not always be enough students to form a complete class for these curricula. As such, some classes may consist of students with varying curricula. For example, a class may consist of students with a regular curriculum, as well as students from the art track. While these students follow most of their lessons together, there are some classes which only feature students from one curriculum. This, in combination with the fact that lower years are not supposed to have idle hours, and the available days for teachers, makes scheduling the lower years significantly more difficult than usual.

2.2 Data

In this section, we further describe the specific instance of the school timetabling problem we tackled in this thesis. While we discussed the problems induced by the constraints in Section 2.1, this section places its focus on the data itself.

To start, the lower years consist of 29 different classes. There are four different curricula for the lower years: Regular, Art, Gymnasium and Technasium. The goal of the school is to create classes consisting of about 30 students. However, there may not be enough students in each curriculum to create classes with enough students. As such, some classes may have students from multiple curricula. Additionally, some students may have a fast track program, which can be combined with different curricula. Similarly, students in the fast track program have additional subjects, which replace some lessons of the shared schedule between curricula. Accounting for this, we end up with a total of 69 different groups of students, each group sharing the exact same curriculum. These groups are spread out over 35 different base classes, consisting of a total of 862 students.

Looking at the upper years, we find a total of 719 different students. Each of these students has chosen a profile and accompanying elective subjects. To better illustrate what these choices look like, we translated the profile choice paperwork for vwo into English, which can be found in Appendix A. Seeing as there are so many options, it is not strange that there are 435 unique sets of

subjects among these students. The school also already assigned students to base classes, which follow their mandatory subjects together. Accounting for these base classes, we find a total of 532 unique schedules.

Across all years, there are a total of 1975 lessons that must be scheduled, of which 1095 are for the lower years. Of these lessons, 66 lessons are already assigned to a fixed day and timeslot. The 1975 lessons are taught by 124 different teachers. Most teachers can be scheduled flexibly, with an average of 4 available days, compared to the average of 16 lessons per teacher. However, there are 9 teachers who are only available for the number of days strictly required by their lessons. For example, a teacher with 15 lessons is required to teach at least two days.

Chapter 3

Approach

As this problem has now become more complex, we will break it down into multiple parts. First, we try to find a feasible solution. This feasible solution should adhere to all hard constraints. We begin by focusing on the lower years, as it is important they have as few idle hours as possible. This is exceptionally useful because teachers with the most constraints in availability tend to teach in the lower years. Once we have made a schedule for the lower years, we create a feasible assignment for the upper years, making sure to avoid clashes with the schedule of the lower years. Doing so gives us a feasible solution that does not yet take the soft constraints into account. We take these soft constraints into consideration by using a local search algorithm starting with this feasible solution. In the rest of this section, we go over our approach in more detail.

3.1 Initial Schedule Lower Years

While the lower years are overall easier to schedule, they do come with their own set of restrictions. In our specific problem, there are three constraints that pose significant issues.

1. Lower Year students should not have any idle hours
2. Parts of a class may follow different curricula.
3. Each teachers is only available for a limited number of days

These constraints are especially troublesome when put together. For clarity, we first define lessons as standard or extracurricular. Lessons are standard when all parts of a class follow that lesson, and we define them as extracurricular when not all parts of a class follow the lesson. For the first and second constraint, there are two possible ways to adhere to both: all extracurricular lessons are scheduled either at the start or end of the day, or the other parts of the class also attend an extracurricular lesson during the same timeslot. However, both

methods are hindered by the third constraint. Teachers that teach the extracurricular subjects often teach multiple classes, which makes it difficult to place all extracurricular classes at the start or end of the day. It is also common that parts of two different classes have a extracurricular lesson together, which makes them even more difficult to schedule. Although it was requested for lower year students to have no idle hours, in recent years this has not been successfully applied. As such, some idle hours may be required, although they should be minimized.

Initially, we assign the lessons to the timetable in such a manner that the limited availability of teachers is adhered to. We also consider the lessons that require multiple consecutive hours by scheduling those hours on the same day. In this specific instance, the only subject that requires consecutive hours is Physical Education (PE), of which all lessons should preferably be scheduled consecutively. As such, we limit the PE lessons to one day. We make this assignment by means of an integer linear program (ILP), with the main decision variable x_{lds}, defined as follows:

$$x_{lds} = \begin{cases} 1, & \text{if lesson } l \text{ is assigned to day } d \text{ on timeslot } s \\ 0, & \text{otherwise} \end{cases} \tag{3.1}$$

As each lesson needs to be scheduled exactly once, we add the following constraint:

$$\sum_{d,s} x_{lds} = 1 \quad \forall l \tag{3.2}$$

As mentioned, a large issue is that different parts of a class can have different schedules. To handle this, we introduce the definition of a group; students who follow the exact same schedule. Each group can only be scheduled once per timeslot. We make use of an auxiliary variable, g_{gds}, to track this. This variable tracks the number of lessons the group g is part of in each timeslot. To do so, we first find the set L_g, defined as the set of all lessons that group g attends. Using L_g, we define g_{gds} as follows:

$$g_{gds} = \sum_{l' \in L_g} x_{l'ds} \quad \forall d, s \tag{3.3}$$

And to prevent groups from being scheduled multiple times on the same timeslot, we add the constraint:

$$g_{gds} \leq 1 \quad \forall g, d, s \tag{3.4}$$

We then repeat this process for the teachers. Instead of the set L_g, we now introduce the set L_t, the set of lessons that teacher t teaches.

$$t_{tds} = \sum_{l' \in L_t} x_{l'ds} \quad \forall d, s \tag{3.5}$$

$$t_{tds} \leq 1 \quad \forall t, d, s \tag{3.6}$$

As teachers are only available a limited number of days, represented by A_t, we introduce variables d_{td} indicating that teacher t teaches on day d:

$$d_{td} = \begin{cases} 1, & \text{if teacher } t \text{ is assigned to a lesson on day } d \\ 0, & \text{otherwise} \end{cases} \tag{3.7}$$

d_{td} should be set to 1 if t teaches on any timeslot on day D:

$$d_{td} \geq t_{tds} \quad \forall t, d, s \tag{3.8}$$

teacher t can teach on up to A_t days:

$$\sum_{d} d_{td} \leq A_t \quad \forall t \tag{3.9}$$

We apply the same principle again for the PE lessons:

$$p_{gd} = \begin{cases} 1, & \text{if group } g \text{ has a Physical Education lesson on day } d \\ 0, & \text{otherwise} \end{cases} \tag{3.10}$$

$$p_{gd} \geq x_{lds} \text{if g attends lesson l and lesson is PE} \quad \forall d, s \tag{3.11}$$

$$\sum_{d} p_{gd} \leq 1 \quad \forall g \tag{3.12}$$

For each day, we now have an assignment of lessons to timeslots. However, this assignment does not yet consider idle hours for groups, nor the double hours. To amend this, we use a simple local search algorithm on each day assignment. This local search algorithm is a simplified version of simulated annealing, in which the acceptance probability α is static. To measure the quality of a schedule we calculate a score based on the number of idle hours, and the number of PE lessons that are not scheduled as a double hour. For the neighbourhood, we make use of a single variation operator. This operator takes a lesson l and tries to assign it to a new timeslot. However, we want to ensure that the solution remains feasible. To accomplish this, we fix any clashes generated by moving l by moving the lessons that cause the clashes to the original timeslot of l, essentially swapping l with a number of different lessons. We then check if moving these lessons to the original timeslot does not cause any clashes. If they do, the move is not accepted. If there are no clashes, we check the quality of the schedule. If the score increases, the move is only accepted with probability α, and otherwise rejected.

3.2 Initial Schedule Upper Years

After constructing the schedule for the lower years, we shift our focus to the upper years. This introduces a new component to the problem: students have yet to be assigned to classes. This new dimension of the problem massively expands the search space. Furthermore, the feasibility of a schedule strongly depends on the class assignment. Preferably, we would like to decompose the problem into the class assignment problem and the scheduling problem. Unfortunately, measuring the quality of a class assignment is difficult. While balancing the number of students for classes of a subject is important, there are other constraints which are influenced by the class assignment which we cannot directly measure, such as idle hours for students. This can only be measured once the classes are scheduled. As aforementioned, a common strategy that is used is to create a cluster scheme. A cluster scheme allows us to take some constraints, such as idle hours, into consideration. Furthermore, a class assignment is usually made during the creation of a cluster scheme. However, due to the limited available days for teachers, cluster schemes can be difficult to work with. As such, we investigate an alternative method to assign students to classes. Afterwards, we go into detail on the clustering algorithm by Post and Ruizenaar [12], which we use as a comparison.

Pair based class assignment

Although we cannot directly measure the influence of the class assignment on the timetabling constraints, what we can somewhat measure is how freely we can schedule a class. Because classes cannot be scheduled together if they share one or more students, classes can be scheduled more freely if there are more classes it does not share any students with. If classes can be scheduled more freely overall, the scheduling problem has more feasible solutions. Following this reasoning, we try to find a class assignment with as many pairs of classes without any overlapping students as possible. Henceforth, we define a *c-pair* as two classes without any overlapping students.

We approach this problem by finding a set of c-pairs, from which we can construct the class assignment. As our goal is to assign students to classes, we only consider c-pairs from the same year and level of education. It is important that the set of c-pairs we find allows us to create a feasible class assignment. For a class assignment to be feasible, it needs to adhere to two requirements. Firstly, all students need to be assigned to a class of each subject they follow. Secondly, the classes of a subject each need to have a similar number of students assigned to them.

We redefine a c-pair as a class and a subject, in which the class has no students that follow the subject. This definition is stricter than before, as we now require no student from an entire subject, rather than just the students in a class. However, this definition does allow us to check the feasibility of a set of c-pairs more easily. Now, since a class is not allowed to have any students from a subject, all overlapping students between subjects need to be placed in the other classes of the subject of the class. If this would require us to place more

students in these classes than allowed, the class assignment is infeasible.

This expands to multiple classes of the same subject. If a subject has three classes, and a set of students cannot be in either two of the classes, they need to be assigned to the third class. Finally, if a student cannot be assigned to any of the three classes, the assignment is also infeasible. Let $\bar{A}$ be the set of students that cannot be placed in class A. For a subject with three classes, A, B and C, and a maximum class size of 30 students, we obtain the following rules:

$$|\bar{A}| \leq 60 \tag{3.13}$$

$$|\bar{B}| \leq 60 \tag{3.14}$$

$$|\bar{C}| \leq 60 \tag{3.15}$$

$$|\bar{A} \cap \bar{B}| \leq 30 \tag{3.16}$$

$$|\bar{A} \cap \bar{C}| \leq 30 \tag{3.17}$$

$$|\bar{B} \cap \bar{C}| \leq 30 \tag{3.18}$$

$$|\bar{A} \cap \bar{B} \cap \bar{C}| \leq 0 \tag{3.19}$$

While this method takes all students that cannot be assigned to a class into account, we have not yet discussed how we compute $\bar{A}$ if class A is involved in multiple c-pairs. To do so, we iterate over all c-paired subjects of A, and add all students of these subjects together. These students cannot be placed in class A, however, if the students do not follow the subject of A, this is already the case. As such, we intersect this with the set of students of the subject of class A. To summarise, we take the union of students of each c-paired subject, and intersect this with the students of the subject of class A.

Let us examine a more fleshed out example. We consider the subject of mathematics, which has 71 students spread out over three different classes. At some point, we arrive at the following c-pairs for the classes:

Math1: Physics & Chemistry
Math2: History, Geography & Economy
Math3: Economy

To use the counting rules, we first find the students who cannot be assigned to each of the classes, respectively noted as $\overline{\text{Math1}}$, $\overline{\text{Math2}}$ and $\overline{\text{Math3}}$. If we look at Math1, we find the students who follow both mathematics and physics, as well as the students that follow both mathematics and chemistry. We add these two sets together to find $\overline{\text{Math1}}$. Now these students cannot be in Math1, so they must be assigned to either Math2 or Math3. This was previously defined in rule 3.13. However, as we now have a total of 71 students, the maximum number of students in $\overline{\text{Math1}}$ is 48, since we want balanced classes. The exact same principle applies to the classes Math2 and Math3, with their respective c-pairs.

If none of these three rules are violated, we move on to the rules that look at two classes. For the first one, we find the students that are in both $\overline{\text{Math1}}$ and $\overline{\text{Math2}}$. These students must be assigned to Math3, so there can be at most 24 students. If we look at $\overline{\text{Math2}}$ and $\overline{\text{Math3}}$, we see that economy is paired with both, so all students who follow both economy and mathematics must be assigned to Math1.

Now we have one more rule, which says that there should be no students who are in $\overline{\text{Math1}}$, $\overline{\text{Math2}}$ and $\overline{\text{Math3}}$. If this is the case, the student cannot be assigned to any of the classes, which is obviously infeasible. If we were to add economy as a c-pair to Math1, all students that follow economy cannot be assigned to any class. Adding this c-pair makes the class assignment infeasible, unless there are no students who follow both economy and mathematics.

To find the c-pairs, we initially opted for a greedy algorithm. This algorithm would continuously add c-pairs, provided that it would still create a feasible class assignment. However, the resulting class assignment could not be scheduled within the allowed timetable. As such, we opt for a local search heuristic. This changes two things compared to the greedy algorithm. Firstly, we add a probability β to remove a random c-pair from the set. Secondly, instead of looking for a set with as many c-pairs as possible, each new c-pair is weighed. We presume it is more beneficial if two classes can be scheduled decently freely, rather than one class with many options, while the other class cannot be scheduled with any other classes. The weight we use is thus based on how many c-pairs each class already has. We calculate the score for one class as $\log_2(1 + \#\text{c-pairs of the class})$. The total score of the set is then the sum of the scores of each class. Following this, the neighbourhood is as follows:

- Add a random c-pair
- Remove a random c-pair

Once we establish the c-pairs, we assign students to the classes using an integer linear program. By using the rules we established, this can be done separately per subject. We make use of the decision variable x_{cs}, defined as follows:

$$x_{cs} = \begin{cases} 1, & \text{if student } s \text{ is assigned to class } c \\ 0, & \text{otherwise} \end{cases} \tag{3.20}$$

Note that if a student cannot be assigned to a class, x_{cs} is always set to 0 for that class-student combination. To ensure that the classes are balanced, we make use of a constraint that sets the total students allowed in a class to the average students per class + 1.

$$\textstyle\sum_s x_{cs} \leq \text{Avg} + 1 \quad \forall c \tag{3.21}$$

Running this ILP gives us our class assignment, which we will use to schedule the lessons. However, before we discuss how we schedule the lessons, we examine a different method to create the class assignment.

Enumeration Heuristic

While our algorithm focuses on freedom of scheduling, the algorithm by Post and Ruizenaar [12] finds a cluster scheme that can be scheduled in as few timeslots as possible. To reiterate, a cluster scheme groups classes together, which will be scheduled on the same timeslots. Each such group of classes is called a cluster line. The length of a cluster scheme is the number of timeslots required to schedule the whole cluster scheme. Similarly, the length of a cluster line is the number of timeslots required to schedule all classes in the line, which is equal to the highest number of lessons any class in the line has. During the creation of the cluster scheme, students are assigned to classes. We will make use of this class assignment to create an initial solution.

To create the cluster scheme, Post and Ruizenaar [12] employ a heuristic enumeration scheme. For this, we first discuss how we judge the quality of a cluster scheme. Their main goal is to minimize the length of the cluster scheme, while their secondary goal is to balance the number of students in classes of a subject. Combining this, they formulate the quality of a cluster scheme as:

$$\begin{aligned} \text{Score(scheme)} = \quad & \alpha \cdot \text{Length(scheme)} + \\ & \beta \sum_{\text{subjects}} (\sum_{\text{classes}} \#\text{students in class} - \text{average per class})^2 \end{aligned} \tag{3.22}$$

Here, α and β represent the difference in importance between the length and the balance. These should be set in such a manner that the length is of higher importance. Note that adding more classes to the scheme can only increase the score. Because of this, we can use a branch and bound-like technique with the quality definition in 3.22.

Before we create the cluster scheme, it should be noted that we create a clustering for each year of each level. This can be done without much difference in the total clustering, as there are no overlapping students between these. While a point can be made for the teachers, which do overlap, the search space is drastically smaller when split, allowing us to find the cluster scheme significantly faster. Additionally, only the elective subjects are clustered. This is a common strategy, as the mandatory subjects are already assigned to base classes. Scheduling all base classes on the same timeslot encompasses all students, and by not fixing them as a cluster scheme we retain the freedom to schedule the mandatory subjects in other combinations.

To initialize the enumeration heuristic, we first sort the subjects based on the number of classes it has, in ascending order. This is the order in which we will assign the subjects. Subjects with fewer classes are more constrained in their scheduling, so handling them earlier on shrinks the search space further. We then iterate over the subjects in this order. The first subject we encounter is likely to have only one class. Seeing as we have no cluster lines thus far, we create a new cluster line and add the class to this line. The next subject is also likely to have one class. For this assignment, we now have two options, either add it to the existing cluster line, or create a new cluster line and add it to the new line. Both of these options are explored, given that there are no student or

teacher clashes. Later subjects will have more options, but the sample principle applies for all their options.

Once we reach subjects with multiple classes, assigning classes to lines becomes more complicated. The assignment is still done in a very similar manner. For example, if we have two existing cluster lines and we want to assign a subject with two classes, we have six different assignments, as illustrated in Figure 3.1.

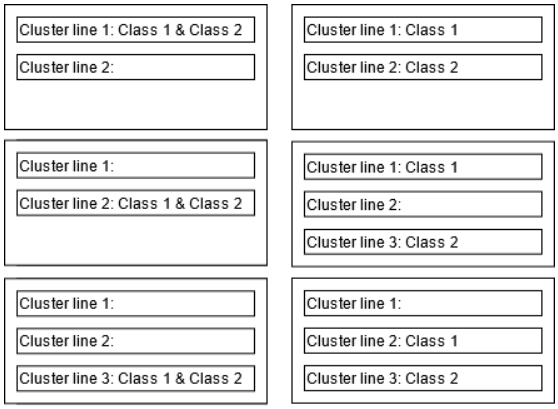

Figure 3.1: Example of how a subject with two classes can be a assigned to three different cluster lines

For each of these assignments, we greedily assign the students of the subject to the classes. We do so by iterating over all students and assigning them to a class if possible. Preferably, we assign the student to the class that contains the fewest students thus far. However, a student may already be assigned on this cluster line, so this is not always possible. If a student cannot be assigned to any class, then this cluster line assignment is deemed infeasible and we stop further expanding on this assignment.

Once all the students of a subject have been assigned, we calculate the score of the current assignment. If the current score is higher than the score of a completed cluster scheme we found earlier on in the algorithm we can stop expanding this assignment, as the score can only increase. For this to function properly, new assignments are explored in a depth-first manner.

Algorithm 1: Heuristic Enumeration Algorithm

```
Sort subjects on class count
Enumurate(empty solution, first subject)

Function Enumurate(solution, subject):
    Find all combinations of assigning classes to lines
    if all subjects assigned then
        return solution
    end
    foreach line assignment combination do
        if all students can be assigned to a class then
            if score ≤ best score then
                return Enumurate(current solution, next subject)
            end
        end
    end
```

Our implementation of this algorithm is exactly as defined in the paper by Post and Ruizenaar [12]. However, some improvements do exists, such as assigning students using a max flow problem instead of greedily.

Local Search

Although Post and Ruizenaar [12] employ techniques to improve the speed of the enumeration, computing the cluster schemes may still require a fair amount of time. Additionally, the cluster schemes it finds are likely not optimal, due to the order the subjects are assigned in and the greedy assignment of students to classes. As an alternative, we consider a local search algorithm to create the cluster scheme. This algorithm uses the same quality formulation, denoted in 3.22. We surmise that this can create cluster schemes of very similar quality, while requiring less computation time.

As such, we create a simulated annealing algorithm, with a neighbourhood consisting of two operators:

- Move a class to a different cluster line
- Move a student of a subject to a different class

In addition to the quality measure denoted in 3.22, we increase the score of a scheme based on the number of student clashes and teacher clashes. For the heuristic enumeration algorithm, this was unneeded, as such schemes are discarded. Finally, we decrease the temperature linearly over time.

Creating the initial schedule

Once we have found the class assignments, whether by our method or the method discussed by Post and Ruizenaar [12], we can start scheduling the upper years. Much like with the lower years, we first look for a feasible schedule using

an ILP. The ILP we use largely resembles that which was used for the lower years, but instead of looking for group clashes, each class is assigned a set of classes that the class cannot be scheduled with. This gives us the following ILP:

$$x_{lds} = \begin{cases} 1, & \text{if lesson } l \text{ is assigned to day } d \text{ on timeslot } s \\ 0, & \text{otherwise} \end{cases} \tag{3.23}$$

$$\sum_{d,s} x_{lds} = 1 \quad \forall l \tag{3.24}$$

For the classes that cannot be scheduled together, we use the constraints:

$$x_{lds} + x_{l'ds} \leq 1 \quad \forall l, d, s, l' \in \text{lessons } l \text{ cannot be scheduled with} \tag{3.25}$$

Unfortunately, in our instance, completely avoiding student clashes makes finding a feasible initial schedule highly unlikely. As such, we allow lessons to be scheduled together even if this creates student clashes. Preferably, we would like to minimize these clashes, however, doing so would drastically increase the computation time required to find the initial schedule. As a result, we allow two lessons to be scheduled together if they create a minimal number of student clashes, but we do not track or minimize the total number of clashes. This is done by simply loosening the restrictions on 3.25.

For teacher availability, we use the same method as in the lower years. Furthermore, if scheduling a lesson on a timeslot would create a clash with either the fixed lessons or the lessons of the lower years, x_{lds} is forced to have a value of 0.

3.3 Improving the schedule

As seen in Section 1.2.1, many algorithms have been used for the class-teacher problem. From these, we initially limit our choice to one of the better performing algorithms. We consider three different candidates: an evolutionary algorithm, a local search-based algorithm, or a integer linear program.

Each of these algorithms can start from an initial schedule. However, making changes to the class assignments is troublesome using integer linear programs. As such, we drop these from our consideration. The advantages of evolutionary algorithms and local search are very similar. Evolutionary algorithms have the considerable downside that the variation operators are problem specific and require a large amount of problem specific knowledge. Alternatively, one could use a standard data structure, which allows common techniques to be applied. This does, however, require a mapping from the problem instance to such a data structure. Furthermore, parameters such as population size need to be investigated, as the best values differ for each problem. A similar issue is apparent in local search algorithms, which require good parameters for escaping

local optima. Additionally, a good neighbourhood search is required, which is a problem specific operator.

Simulated annealing has been shown to have better results than both evolutionary algorithms and tabu search, as discussed in Section 1.2.1. For this reason, simulated annealing will be the algorithm of choice for improving the initial timetable.

The neighbourhood structure we use in our simulated annealing is composed of multiple operators.

1. Move lesson to a new timeslot and/or day
2. Move a student between classes of a subject
3. Swap all lessons between two timeslots
4. Swap all students between classes of a subject
5. Move all lessons of a teacher scheduled on a day to another day

The first two operators mentioned are fairly basic operators to alter the timetable. Operators 3 and 4 are extensions of the first two, increasing the range of the operator. These operators create larger changes in the timetable, which are unlikely to be achieved using just the first 2 operators. Finally, operator 5 is included to allow us to divert more from the initial schedule. Without this operator, moving lessons of highly constrained teachers is difficult, as they either create a clash or exceed their number of allowed days. The latter is assigned a high cost, which makes it less likely that such a move would be accepted. If we move all lessons of a teacher on a day to a new day, we do not exceed their maximum allowed number of days, which is more likely to be accepted.

After each move, we check the quality of the timetable. If it is an improvement, we accept the change, if this is not the case, we only accept the change with a probability determined by the simulated annealing process. We determine the quality of the timetable using the constraints found in Section 2.1. However, due to the complexity of the problem, we decided to drop the assignment of rooms to lessons. While this was requested, the importance of these constraints are lower than the overall quality of the schedule of the lessons. As such, we only consider the maximum number of lessons on a timeslot, so as to not require too many rooms.

To encourage lessons of a class to be spread out over the week, we employ multiple measures. First, we penalise scheduling more than 2 lessons of a class on the same day. Secondly, we look at the number of days between the first lesson in the week and the last lesson in the week, which we will call the dayspan. For example, if a class has a lesson on tuesday, and a lesson on friday, it has a dayspan of 4. Preferably, we want a class to have a dayspan equal to the number of lessons + 1, unless a class has only one lesson. Doing so allows teachers to assign homework, for which students will have at least one or two days to complete.

Although we want to minimize the idle hours, we would rather balance out the idle hours rather than stacking them on a few students or teachers. To do so, higher numbers of idle hours for a student or teacher are penalised harder. We take the number of idle hours raise it to the power of an exponent. The outcome of this is multiplied with the penalty for idle hours. Students and teachers use different penalties and exponents for idle hours.

$$\text{Penalty(Idle Hours)} : \#idle\ hours^{\text{idle exponent}} \cdot \text{idle hour penalty} \tag{3.26}$$

During the creation of the initial class assignment, we mentioned that we want the classes of a subject to have a balanced number of students. This requirement still stands during the improvement phase. During this phase, we calculate the penalty for unbalanced classes in the same manner as during the creation of the initial class assignment.

Chapter 4

Results

Previously, we discussed four different methods for creating the class assignments. In this section, we compare these methods. We primarily do so by comparing the final timetables after improving the initial timetables produced by these four methods using local search. In addition to the four methods, we include a completely random initial timetable.

However, before we make this comparison, we use the quality measure defined by Post and Ruizenaar [12] to compare the cluster schemes created by the enumeration heuristic and the local search variant. Although this quality definition may not necessarily be indicative of a good class assignment, it is still an interesting comparison to make.

Other than investigating the varying methods for creating the class assignments, we also experiment with varying running times for creating the cluster schemes using local search. Similarly to comparing the methods, we focus our comparison on the final timetables.

All tests are run on an Intel(R) Core(TM) i7-6700K CPU @ 4.00GHz.

4.1 Cluster scheme Comparison

First, we compare the local search version of the method by Post and Ruizenaar [12] to their heuristic enumeration algorithm using their quality measure. Although this quality measure is not necessarily indicative of a good resulting timetable, it should be the case that cluster schemes with lower length are easier to schedule. Initially, we run the local search for 5 minutes for each level of education. Throughout the results, each local search algorithm is run 3 times, with the results averaged. The enumeration heuristic works in a deterministic manner, so it is only run once. It should be noted that we established a hard limit on the length of cluster schemes created by the enumeration heuristic to restrict the running time. For this, we chose a length of 32. We went for this as we require about 10 timeslots for scheduling the main subjects. The results of this can be seen in Table 4.1. Later, in Section 4.2 we compare different running

times of the local search algorithm.

Level	Score HE	Time HE (min)	Score LS	Time LS (min)
m3	343,0	0:19	338,2	5
m4	321,0	0:04	321,4	5
h4	404,0	3:31	401,3	5
h5	478,2	43:38	337,8	5
v4	301,4	0:00	257,7	5
v5	316,0	0:02	370,3	5
v6	310,6	0:01	310,4	5

Table 4.1: Comparison between the heuristic enumeration (HE) and local search (LS) method to create the cluster schemes according to the score formulation by Post and Ruizenaar [12] with $\alpha = 10$ and $\beta = 0.25$.

Overall, the local search version creates cluster schemes with a lower score. Other than this, there are two notable details. First, the computation time for the enumeration heuristic is quite short, but takes a while longer for h4 and a significant longer time for h5. Additionally, these cluster schemes have a high score compared to the other levels. Secondly, although the local search variant overall produces better cluster schemes, it is not the case for v5. To explain these occurrences, we examined the produced cluster schemes in greater detail.

Heuristic enumeration score and running time
Examining the cluster schemes of levels h4 and h5 created by the enumeration heuristic further, we find that high score is due to severely unbalanced classes. For example, German at the h4 level has one class with 34 assigned students, while the other class has 2 students. This is most likely due to the heuristic using multiple greedy approaches. With this in mind, we further examined the cluster schemes of the other levels. Although less prevalent, we find some classes that are highly unbalanced.

Furthermore, we reckon that the long running time is also caused by the high score. During the algorithm, we prune partial cluster schemes if their score is higher or equal to the score of the current best complete cluster scheme. If the score of the best cluster scheme is higher, less pruning is done, resulting in a significantly longer running time.

To allow more flexibility during the creation of the cluster schemes, we increased the bound on the length to 38. However, this did not necessarily improve the found cluster schemes. Due to the greedy assignments, it is possible that we find cluster schemes with worse scores than with a lower bound on length, as can be seen in Table 4.2. With this increase in score, we also found an increase in running time, most significantly for h4, which took more than 15 hours to compute.

Score difference for level v5
We move on to examining the difference in score between the two methods for

Level	Score Max 32	Time Max 32 (min)	Score Max 38	Time Max 38 (min)
m3	343,0	0:19	310,0	0:10
m4	321,0	0:04	361,0	0:02
h4	404,0	3:31	629,0	878:16
h5	478,2	43:38	510,2	43:18
v4	301,4	0:00	301,4	0:00
v5	316,0	0:02	380,0	0:01
v6	310,6	0:01	310,6	0:01

Table 4.2: Comparison between the heuristic enumeration method bounded by a maximum cluster scheme length of 32 and 38.

the level v5. We quickly find that the length of the cluster scheme created by local search is quite high, ranging from 35 to 38. Although this is not the case for the enumeration heuristic, they only accomplish this by abusing a few unbalanced classes. A large issue is the difficulty of scheduling for this level. Compared to the other levels, there are fewer students and fewer subjects with multiple levels, making it more difficult to create a good cluster scheme. The only level with a similar problem is v6, however, they have fewer lessons in total, allowing us to still find a scheme with a manageable length.

A bounded local search

To amend the high lengths of the cluster scheme found by local search, we added an additional restrained to the local search. By increasing the score per length to 100 for lengths larger than 32, we create a soft bound on the length of the cluster schemes. We compare the resulting cluster schemes and their respective lengths in Table 4.3. Note that in this table, the length of a cluster scheme always counts as 10 towards the score, even if during the bounded local search it may count as more.

	Unbounded		Bounded	
Level	Score	Length	Score	Length
m3	338,2	33,7	313,1	31,0
m4	321,4	32,0	320,7	32,0
h4	401,3	36,0	345,1	32,0
h5	337,8	33,7	321,1	32,0
v4	257,7	25,7	264,3	26,3
v5	370,3	36,7	320,5	32,0
v6	310,4	31,0	307,0	30,7

Table 4.3: Comparison of the cluster schemes computed by local search, one bounded using an extra penalty for lengths above 32, and one unbounded

We can see that on most levels, the cluster schemes are of similar quality, which is to be expected. For the levels h4, h5 and v5, we find a decrease in length, without creating severely unbalanced classes. This is very beneficial,

however, it does require some problem specific knowledge to set the bound to a value that does not heavily skew the balance, while still enforcing lower length.

4.2 Local Search Running Times

Next, we investigate the effect of the running time on the resulting cluster schemes created by local search. Previously, we used runs spanning 5 minutes per level, however, shorter runs may also suffice. In this section, we will compare the running times of 1 minute, 2 minutes and 5 minutes. We do this for both the unbounded and bounded version. We compare the running times in two different manners. First, we compare the score of the cluster scheme using the quality measure by Post and Ruizenaar [12], similarly as in the previous section. Next, we make a comparison based on the resulting timetables, both initially and after improvement using local search. Once again, each noted result is the average of 3 runs.

	Unbounded			Bounded		
Level	1 minute	2 minutes	5 minutes	1 minute	2 minutes	5 minutes
m3	360,2	338,7	338,2	316,8	310,1	313,1
m4	360,7	347,4	321,4	324,2	320,7	320,7
h4	422,0	398,5	401,3	351,8	348,5	345,1
h5	371,1	367,8	337,8	321,1	333,6	321,1
v4	281,3	257,7	257,7	281,0	267,7	264,3
v5	375,3	370,5	370,3	320,5	320,5	320,5
v6	310,4	310,4	310,4	310,4	310,4	307,0

Table 4.4: Comparison between multiple running times for the local search algorithm

As we can see in Table 4.4, there is a larger difference between a running time of 1 minute and 2 minutes, compared to a the difference between 2 minutes and 5 minutes. Although runs allocated 5 minutes do still outperform, the difference with the 2 minutes runs is slight. Most commonly, the only difference is a slightly better balance of the class assignment.

As mentioned earlier, while the quality measure by Post and Ruizenaar [12] is interesting, it does not directly translate to how well a cluster scheme can be scheduled. As such, we compare the scores of the initial timetables constructed using the found class assignments, as well as the score of the final timetables after improvement using local search. These results can be seen in Table 4.5.

For both the unbounded and bounded variant, taking more time to create the cluster schemes results in a lower score, for both the initial timetables as well as the final timetables. Furthermore, we can see that the bounded variant of local search results in better final timetables for all running times. These results however, are not statistically significant at $p = 0.05$; the lowest p-value is 0.1, between the 2 and 5 minutes of the unbounded local search.

	Initial timetable		Final timetable	
	Score	σ	Score	σ
1 minute unbounded	31219035	413267	1137926	121277
2 minutes unbounded	30754323	4852107	1135627	53362
5 minutes unbounded	30510642	885867	1051802	17807
1 minute bounded	32647457	982680	1040658	120403
2 minutes bounded	31125833	1737742	1023301	11503
5 minutes bounded	23120102	2939399	994763	86195

Table 4.5: Scores with standard deviations (σ) of both initial and final timetables after improvement, for different running times of the local search algorithm to create cluster schemes.

4.3 Initial Schedules

To find the effectiveness of each initial schedule and class assignment, we compare the final schedules after improvement using local search. The parameters for the local search are the same for each run. The probabilities for each operator can be found in Table 4.6. For further details on how the score is computed, a breakdown of the best found timetable is included in Appendix B.

Operator	Probability
Move a lesson to a different timeslot	70%
Move a student between classes of a subject	25%
Swap all lessons between two timeslots	3%
Swap all students between classes of a subject	1%
Move all lessons of a teacher scheduled on a day to another day	1%

Table 4.6: Probability of each operator in the local search for timetable improvement.

We run two different comparisons for the schedules. First, we do five runs of 1 hour long for each method, each starting with a new initial timetable for each run. Secondly, we do one run that lasts 8 hours for each method. This run will result in a timetable that is further optimized than the runs with running time of 1 hour. In addition to the four different methods we use for creating an initial class assignment, we add a comparison with a completely random initial class assignment and schedule. In total, we compare five different methods to create class assignments: The heuristic enumeration (HE) by Post and Ruizenaar [12], a local search (LS) variation of the enumeration heuristic, a bounded local search (LSB) variant, the assignment based on maximal pairs, and a completely random assignment.

1 hour runs
To analyze the differences in each method, we created five different initial timetables for each method. However, as the quality of these timetables does not reflect

how easily the timetable is improved upon, we run a local search to improve each timetable. The average result of each method can be found in Table 4.7.

	Initial timetable		Final timetable	
Method	Score	σ	Score	σ
LS	33079524	2503696	1108928	79847
LSB	31158073	672183	1061743	16264
HE	15655114	1067048	1086163	49762
Pairs	17066964	1242826	1060216	84610
Random	419756314	4270100	4191253	74340

Table 4.7: Scores with standard deviations (σ) of both the initial timetables and the final timetables after a running time of 1 hour.

If we first analyze the initial timetables, we see that interestingly, the initial timetables for the heuristic enumeration and pair based class assignment are better than the ones created from the local search cluster schemes. However, the difference between the final timetables of any of these four methods is not significant enough to distinctly call one method better than another. What can be said, is that the randomly initialised timetable performs significantly worse than the other methods.

8 hour runs

While running the improvement phase for 1 hour results in a decent timetable, there is still room for improvement. By extending the running time to 8 hours, we should be able to find further progress in the quality of the timetable. The results of these runs can be found in Table 4.8.

Method	Initial score	Final score
LS	34291911	893976
LSB	31387990	710004
HE	15023008	759788
Pairs	16466525	810146
Random	427537877	2867308

Table 4.8: Scores of the resulting timetables after a running time of 8 hours.

Here we can once again see that starting with a decent initial timetable is a significant improvement over starting with a completely random one. Compared to the runs with a duration of 1 hour, there is a larger variance between the resulting timetables of each method. However, as we only did a single run with this duration, a fair comparison cannot be made. Nonetheless, we can compare the methods on a different aspect; the time required to construct the initial timetable.

Time to create the initial timetable

With the parameters we used, constructing an initial timetable using the pair

based method takes the least amount of time. Although the computation time of both local search methods is variable, we used 5 minutes per level as we used this value during our computations. However, by limiting the computation time to 2 minutes per level, the construction time is similar to the pair based method. Nevertheless, both the local search methods and the pair based method benefit from a static computation time. Although the heuristic enumeration method finds most cluster schemes quickly, it can get stuck on difficult instances, resulting in a long computation time and poorly balanced classes.

Method	Time to construct initial timetable (min)
LS	39
LSB	39
HE	48
Pairs	16

Table 4.9: Comparison of the total time taken to create initial timetables for each method.

Chapter 5

Conclusion

During this thesis, we experimented with a local search variant of the method by Post and Ruizenaar [12]. According to their quality measurement, we found that using local search produces better and more consistent cluster schemes, albeit requiring a longer running time for most levels of education. The cluster schemes constructed using local search mostly improved on the balance of the classes.

Although the enumeration heuristic requires a significantly lower computation time for most levels, it may take an undesirable amount of time if the instance is difficult. For these instances, it may be more beneficial to have a fixed running time, which the local search variant provides. Finally, another trait of the enumeration heuristic is that the quality of the cluster scheme varies with the bound on the length of the cluster scheme. It may be the case that a higher bound will result in a lower quality cluster scheme, which is troublesome.

As such, for constructing the cluster scheme, we recommend either using the local search variant or running both the local search variant and the enumeration heuristic at the same time. For the latter, we would take the cluster scheme that is computed faster, although it is advised to check the balance of the classes of the enumeration heuristic.

In our instance of the problem, making use of cluster schemes provided to be a difficult assignment due to the restrictions in teacher's schedules. As such, we used a local search algorithm to improve upon an initial timetable. This initial timetable is constructed using an initial assignment of students to classes. We compared different methods to create this assignment of students to classes. First, we used both the aforementioned heuristic enumeration algorithm and local search variant to create cluster schemes, from which we specifically used only the class assignment. Additionally, we investigated a completely new method to produce class assignments. By maximizing the number of classes that can be scheduled together, we allow the timetable to more easily explore the search space during the improvement stage. While we found that adding a good soft bound to the length of a cluster scheme created using local search

can lead to better cluster schemes, we cannot confidently say that any discussed method is better than another.

In conclusion, we investigated two new methods, a local search for creating cluster schemes, and a pair based method that exclusively creates a class assignment. The local search is capable of creating cluster schemes more consistently at a static computation time. Previously, the computation time could ramp up indefinitely depending on the problem and the parameters given.

Although we cannot say that the pair based method to create class assignments results in a higher or lower quality timetable, it does consistently have the lowest time to compute the class assignments compared to all other discussed methods.

5.1 Future Research

In future works, multiple facets of the methods discussed in this work can be improved upon. In this study, we explored a new method of assigning students to classes. This method is based on a counting argument, which may be overly strict. Future research could be done on improving the counting argument, allowing for more pairs to be found. It could also be interesting to create cluster schemes using the class assignment found using this method, instead of assigning students during the creation of the cluster scheme.

Furthermore, we examined the work by Post and Ruizenaar [12]. We found that while their method is effective, it is restricted by the greedy assignment of students to classes and set order in which to assign the subjects. This could be improved by allowing the class assignments to change later on in the algorithm, or assigning students based on the flexibility of the class, instead of how balanced the classes currently are.

Chapter 6

Bibliography

[1] M. Bufé, T. Fischer, H. Gubbels, C. Häcker, O. Hasprich, C. Scheibel, K. Weicker, N. Weicker, M. Wenig, and C. Wolfangel. Automated solution of a highly constrained school timetabling problem - preliminary results. pages 431–440, 04 2001. doi: 10.1007/3-540-45365-2_45.

[2] J. Domrös and J. Homberger. An evolutionary algorithm for high school timetabling. *Practive and Theory of Automated Timetabling (PATAT 2012)*, pages 485–488, 2012.

[3] Á. P. Dorneles, O. C. de Araújo, and L. S. Buriol. A fix-and-optimize heuristic for the high school timetabling problem. *Computers & Operations Research*, 52:29–38, 7 2014.

[4] Á. P. Dorneles, O. C. de Araújo, and L. S. Buriol. A column generation approach to high school timetabling modeled as a multicommodity flow problem. *European Journal of Operational Research*, 256:685–695, 02 2017. doi: 10.1016/j.ejor.2016.07.002.

[5] A. Drexl and F. Salewski. Distribution requirements and compactness constraints in school timetabling. *European Journal of Operational Research*, 102:193–214, 1996.

[6] G. Fonseca, H. Santos, T. Toffolo, S. Brito, and M. Souza. Goal solver: A hybrid local search based solver for high school timetabling. *Annals of Operations Research*, 08 2014. doi: 10.1007/s10479-014-1685-4.

[7] P. Haan, R. Landman, G. Post, and H. Ruizenaar. A case study for timetabling in a dutch secondary school. pages 267–279, 01 2006. doi: 10.1007/978-3-540-77345-0_17.

[8] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. *SIGMOD Rec.*, 29(2):1–12, May 2000. ISSN 0163-5808. doi: 10.1145/335191.335372.

[9] K. Nurmi and J. Kyngäs. A framework for school timetabling problem. 01 2007.

[10] O. A. Odeniyi, E. O. Omidiora, S. O. Olabiyisi, and C. A. Oyeleye. Mathematical programming model and enhanced simulated annealing algorithm. *Asian Journal of Research in Computer Science*, pages 21–38, 5 2020. doi: 10.9734/AJRCOS/2020/v5i330136.

[11] N. Pillay. A survey of school timetabling research. *Annals of Operations Research*, 218, 02 2013. doi: 10.1007/s10479-013-1321-8.

[12] G. Post and H. Ruizenaar. *Clusterschemes in Dutch secondary schools.* Number 1707. University of Twente, Department of Applied Mathematics, 2004.

[13] G. Post, S. Ahmadi, S. Daskalaki, J. Kingston, J. Kyngas, C. Nurmi, C. Gogos, N. Musliu, N. Pillay, H. Santos, and A. Schaerf. An xml format for benchmarks in high school timetabling. *Annals of Operations Research*, 194:385–397, 04 2012. doi: 10.1007/s10479-010-0699-9.

[14] R. Raghavjee and N. Pillay. A genetic algorithm selection perturbative hyper-heuristic for solving the school timetabling problem. *ORiON*, 31, 04 2015. doi: 10.5784/31-1-158.

[15] H. Santos, E. Uchoa, L. Ochi, and N. Maculan. Strong bounds with cut and column generation for class-teacher timetabling. *Annals of Operations Research*, 194:399–412, 04 2012. doi: 10.1007/s10479-010-0709-y.

[16] S. Shatnawi, F. Albalooshi, and K. Rababa'h. Generating timetable and students schedule based on data mining techniques. *International Journal of Engineering Research and Applications*, 2:1638–1644, 07 2012.

[17] M. Souza, N. Maculan, and L. Ochi. *A GRASP-Tabu Search Algorithm for Solving School Timetabling Problems*, pages 659–672. 01 2004. ISBN 1384-6485. doi: 10.1007/978-1-4757-4137-7_31.

[18] C. Stichting, M. Centrum, and A. Schaerf. A survey of automated timetabling. *Artificial Intelligence Review*, 13, 01 1996. doi: 10.1023/A:1006576209967.

[19] J. Tan, S. L. Goh, G. Kendall, and N. Sabar. A survey of the state-of-the-art of optimisation methodologies in school timetabling problems. *Expert Systems with Applications*, 09 2020. doi: 10.1016/j.eswa.2020.113943.

[20] P. Wilke and H. Killer. Comparison of algorithms solving school and course time tabling problems using the erlangen advanced time tabling system (eatts). *PATAT 2010 - Proceedings of the 8th International Conference on the Practice and Theory of Automated Timetabling*, pages 427–439, 01 2010.

[21] P. Wilke and J. Ostler. Solving the school timetabling problem using tabu search, simulated annealing, genetic and branch & bound algorithms. *PATAT 2008 - Proceedings of the 7th International Conference on the Practice and Theory of Automated Timetabling*, 8 2008.

[22] P. Wilke, M. Gröbner, and N. Oster. A hybrid genetic algorithm for school timetabling. pages 455–464, 12 2002. doi: 10.1007/3-540-36187-1_40.

[23] J. Wood and D. Whitaker. Student centred school timetabling. *Journal of the Operational Research Society*, 49(11):1146–1152, 1998. doi: 10.1057/palgrave.jors.2600628.

[24] D. Zhang, Y. Liu, R. M'Hallah, and S. Leung. A simulated annealing with a new neighborhood structure based algorithm for high school timetabling problems. *European Journal of Operational Research*, 203:550–558, 06 2010. doi: 10.1016/j.ejor.2009.09.014.

Appendix

Appendix A

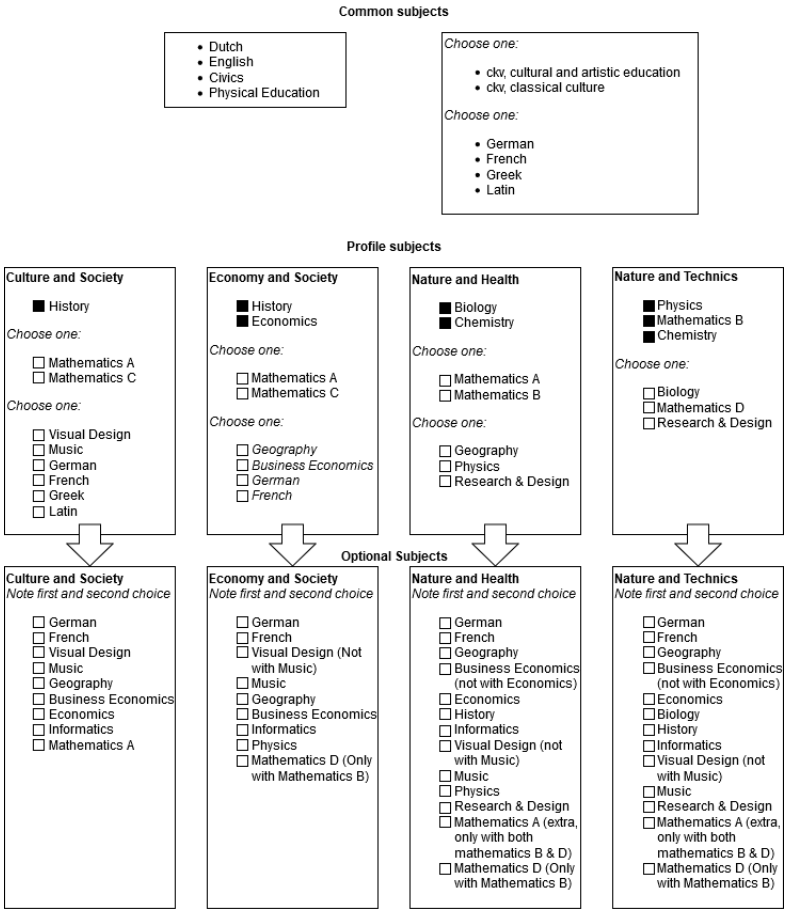

Figure 6.1: Profile choice form VWO

Appendix B

Number of lessons scheduled on 9th hour	0
Score for lessons scheduled on 9th hour:	0
Score for balanced number of students for classes	10980
Score for balancing lessons across the week	165045
Total score for upper years	481040
Total score for lower years	27000
Total score for Teachers:	25939
Total score	710004

Table 6.1: Summary of score breakdown of best found timetable

	Upper Years		Lower Years		Teachers	
	Clashes	Idle Hours	Clashes	Idle hours	Clashes	Idle Hours
Total	16	2827	0	11	0	496
Average	0.03	3.93	0	0.16	0	4
σ	0.18	1.90	0	0.43	0	2.92
Highest	2	9	0	2	0	13

Table 6.2: Detailed summary of the number of clashes and idle hours of the upper year students, lower year groups and teachers