The High School Clustering Problem

Ruijters, Anne

A.M.I. Ruijters

The High School Clustering Problem

Master Thesis

12 February 2025

Thesis supervisors:
dr. O. Kanavetas
N.J van der Kooy, MSc

Leiden University
Mathematical Institute

Contents

Abstract

The High School Scheduling Problem (HSSP) is an optimization problem. Each lesson of subject groups must be assigned to a time slot in a schedule. Students and teachers attending these groups cannot be scheduled in the same timeslot twice. Our first step is clustering the subject groups which can be placed in the same timeslot. This thesis examines an existing branch and bound, and a backtracking method to solve the clustering problem. Moreover, it studies two new techniques to improve the algorithm. Both of these new approaches are based on hill climbing algorithms. Finally, these new techniques are tested on a real-life problem. This thesis finds that combining the old technique with hill climbing algorithms has a positive effect on creating an optimal cluster scheme.

1 Introduction

Most people in the Netherlands remember their high school schedules affecting their lives. The impact of schedules on teachers is equally important. Furthermore, schools cannot be overstated given the shortage of teachers.

Given that, in the Netherlands, a school is attended by approximately 650 students and has around 75 employees, creating schedules for high schools is not an easy task(Ministerie van Onderwijs, Cultuur, en Wetenschap, n.d.), (Onderwijs, n.d.), (Rijksoverheid, n.d.). A schedule is necessary for both students and teachers. In the past, this was done by hand, and the schedules were handed out on paper. Many schools have requirements such as customized schedules, extra optional subjects, and the amount of free periods. Furthermore, there is a high teacher shortage in The Netherlands (Ministerie van Onderwijs, 2023). Lastly, in the Netherlands many teachers work part-time, making it even more complex. These aspects make scheduling highly complex and very time consuming.

Creating a schedule by hand will probably not lead to the best one possible, especially if we keep additional requirements of the school and teachers in mind. A suboptimal schedule could have a negative effect on teacher satisfaction. This could increase the teacher shortage problem.

A company that helps high schools optimize their schedules is Zermelo. They provide both software and training. To stay ahead of their competitors, they must continue improving the speed and quality of their algorithms. The goal of this thesis is to optimize and improve the quality of the cluster scheme.

The High School Scheduling Problem (HSSP) was first addressed by Carter and Laporte (1992), where they focused on classroom assignment (Carter & Tovey, 1992). In 1996, Carter and Laporte elaborated on the timetabling of subjects (Carter, Laporte, & Lee, 1996). They published a paper stating the HSSP is NP-hard or NP-complete, depending on the constraints of the problem and described some algorithms that have been applied to this problem (Carter & Laporte, 1998).

It is beneficial to find subject groups that can be taught at the same time to reduce the search space of possible schedules, due to the complexity of the problem and the various possible outcome. Those groups are placed on the same line in a so-called *cluster scheme*. This process is called clustering, and the associated optimization problem is named the *clustering problem.* Van Kesteren (1999) introduced a backtracking algorithm to make it more efficient to create and optimize clusters (van Kesteren, 1999).

In the backtracking algorithm, random permutations are used. This could lead to missing solutions in the neighborhood. The following question arises:

Is a hill climbing algorithm more efficient than backtracking to optimize a cluster scheme?

The aim of this thesis is to figure out if a form of hill climbing is such an algorithm. This is a local search algorithm. When a better solution is found, it is immediately accepted. The algorithm tries to optimize the new solution by changing this new solution until a better one is found within a neighborhood.

To reach this goal, first, we will describe the High School Scheduling Problem in more detail in section 2. Then, in section 3, we define the general clustering problem itself. In section 4 we will elaborate on two methods for optimization. In section 5, we will describe the clustering problem as we will be studying in this thesis, and specify the considered constraints. This model is similar to the one used at Zermelo. The potential improvements of the algorithm are described in section 6. Eventually, in section 7 and 8, we will present the results and the conclusion.

2 High School Scheduling Problem

In this section, the High School Scheduling Problem is described. Since this problem is NP-hard, a method like clustering is used to make it easier to find an acceptable schedule (van der Kooy, 2017).

2.1 Problem description

In the High School Scheduling Problem (HSSP) we are given the number of *lessons*, which consists of a number of *students* and a *teacher* who instructs education in a certain *subject*. Every lesson has to fit into a schedule, which consists of *timeslots*. There are usually multiple timeslots a day. In Dutch high schools it usually concerns 8 or 9 timeslots per day, 5 days a week.

We want the lessons to fit in the schedule "as good as possible". This is evaluated by using different constraints imposed upon the schedule. The constraints are divided into *hard* and *soft constraints*. A hard constraint could be no students have more than one lesson at the same time, and a soft constraint that a student cannot have more than seven lessons at the same day. The hard constraints needs to be satisfied. Moreover, for every unmet soft constraint, a penalty is imposed. We call a schedule a "better" schedule when it has a lower overall penalty. The constraints are known beforehand and can differ from school to school due to different preferences.

A teacher and students are fixed to a lesson, so it is only possible to rearrange the lesson, and not the teachers or students. Classrooms are assumed to be assigned after the optimization of the lesson schedule. Moreover, it is assumed that an initial schedule is known where all hard constraints are satisfied. Therefore, there is always a solution for the HSSP problem. Various research has been done on this problem which has resulted in suitable algorithms to tackle this problem (van der Kooy, 2017).

In Dutch schools, students of higher grades follow elective courses and thus have highly individual schedules. Therefore *compact schedules*, timetables without free periods, are not feasible for these grades. These are timetables which do not contain any free periods. Moreover, most teachers work part-time which make the dutch HSSP even more complex (G. Post, Ahmadi, & Geertsema, 2010).

Various research has been done on the scheduling problem for Dutch high school. The goal is to make an "as good as possible" schedule. In other words, the objective is to minimize the total penalty. In the paper of Van der Kooy this is defined as a problem where the penalty implied by the soft constraint is minimized where none of the hard constraints can be violated and all lessons need to be scheduled. Numerous algorithms and their performances are provided. Note that the problem is an NP-hard problem (van der Kooy, 2017).

3 The Clustering Problem

In the higher grades, students at Dutch high schools have, besides compulsory courses, elective courses. To make the scheduling less complex, we want to know which lessons can be taught at the same time. This process is called *clustering.* This is done prior to the scheduling. In this phase, it is allowed to switch students from groups; they are not fixed to a lesson (G. F. Post & Ruizenaar, 2004).

3.1 Definitions

Students in high er grades (in Dutch "bovenbouw"), choose a *curriculum* that consists of compulsory and optional subjects. The required courses will not be considered for the clustering. We assume that the curriculum choices are known beforehand. Each subject can consist of one or more *groups* depending on the number of students attending this subject.

During the clustering process we try to find groups which can be scheduled on the same timeslots. Such subject groups can then be placed on the same *clusterline.* All clusterlines combined form the *cluster schemes* ($\mathcal{L}$) for every grade. Each student and teacher on one clusterline has to be able to attend the lessons of their subject group, so their availability has to be taken into account.

Each clusterline contains several subject groups. Within a clusterline, the group that contains the most lessons determines the *clusterline-length.* For example, when geography, with 3 lessons a week, and history, with 2 lessons a week, are on the same clusterline, this line has length 3. The *cluster scheme-length* is the total length of all clusterlines (G. F. Post & Ruizenaar, 2004).

3.2 Problem description

When clustering, we take into account on which days the teachers are available. Moreover, we assume that we have a weekly schedule, which means that for instance every Monday we have the same scheme. In practice, this can vary due to, for example, a testweek. However, in general a school's main goal is to schedule the regular lessons.

As described before, in high grades, students in Dutch high schools have elective courses. They can choose from 4 different *profiles* in which these electives are contained. The problem is that scheduling all these courses can become very complex and many free periods can occur which is not desirable.

Most schools in the Netherlands have around 600 students and approximately 2000 lessons to schedule which leads to roughly $1.3 \cdot 10^{3204}$ variations [1]. When

[1] When a school needs to plan the 2000 lessons in 40 timeslots, this naively leads to $40^{2000} = 1.3 \cdot 10^{3204}$ possible ways to schedule these lessons.

a school has 7 sections, like 4VWO or 5HAVO, where the subject groups are clustered, this will lead to roughly $3.1 \cdot 10^{1938}$ possible ways [2] Therefore, making a cluster scheme makes scheduling less of a complex problem.

To address this problem it is assumed that the following is known:

- The choice of subjects for each student;
- The number of weekly lessons of each subject;
- The number of groups of each subject.

The number of groups depends on the number of students attending the subject. The maximum number of students attending a group is set prior to clustering by the school management. When there are more students attending the course, the group will be divided in one or more groups. When the group is too small, the board can decide to remove the group, or to merge it with a small group of another grade with the same subject. Then, the clusterlines of these groups will be merged, which can make the timetabling harder. Moreover, we assume an initial cluster scheme is provided where all hard constraints are satisfied. Since the clustering is done prior to the scheduling, we moreover assume that no lessons are scheduled yet (G. F. Post & Ruizenaar, 2004).

3.2.1 Hard Constraints

In a cluster scheme, it is required that all subject groups are clustered. However, there could be other demands, which are *hard constraints*. When such a constraint is not satisfied, the cluster scheme becomes invalid. There are no general hard constraints, but these are some examples:

- A maximum length of the cluster scheme. Due to the maximum teaching-hours in a week, there could be a maximum number of schedule-positions taken by the cluster scheme. When a school has 40 regular time slots per week, for instance, it is desired the cluster scheme-length is at most 40.
- There can be at most one group of a subject on a clusterline. For example, when group 2 of Dutch is on line 1, then group 3 of Dutch cannot be placed on this line.

[2] Suppose we have 7 sections for which we can cluster. Each section has a cluster scheme with 10 clusterlines with average clusterline-length 3. This leads to approximately $40^{7 \cdot 10 \cdot 3} = 2.7 \cdot^{336}$ possible ways to schedule the lessons. Assuming that roughly 1000 of the lessons are not clustered, the total amount of ways the lessons can be scheduled are $40^{7 \cdot 10 \cdot 3} \cdot 40^{1000} = 3.1 \cdot^{1938}$.

3.2.2 Soft Constraints

In research on clustering, there are no general constraints. However, there are examples of commonly occurring constraints which are not always used as hard, but as soft constraints. These are used when it is not desirable to have a certain characteristic, but it does not make the cluster scheme invalid. Some examples of soft constraints are:

- Collisions of teachers and students. When a student, or teacher, appears more than once on one clusterline, we call it a collision. This can also be considered as an hard constraint. For example when the school board does not want a student to make a decision between on which of two subjects to participate.

- The balance of the subject groups. For example, there are 60 students registered for the subject history. When there are 2 available groups, a classification of 30 students per group is desired.

- The maximum length of the cluster scheme could also be, instead of a hard constraint, a soft constraint where it is penalised when the cluster scheme-length is larger than a certain predefined number.

3.3 Problem definition

The main goal is to minimize the length of all cluster schemes, since this will lead to more possibilities to schedule the clusters; there is more freedom. Since it is our goal and not a hard constraint that it has to be a certain length, this goal is considered as a soft constraint.

Another goal is to balance the amount of students in each group. For example, group 1 of biology has 29 students and 11 students attend group 2 of biology. The goal is to get two groups of around 20 students which is the best balance. It has to be taken into account that there is a maximum size of a group which cannot be exceeded (G. F. Post & Ruizenaar, 2004).

For each grade the problem can be formulated mathematically. Here j is used for the subject, i for the clusterlines, and k for the student in this grade. A decision is made to set a subject on a clusterline, which is denoted by

$$x_{ij} = \begin{cases} 1 & \text{if a group of subject } j \text{ is in clusterline } i, \\ 0 & \text{otherwise,} \end{cases}$$

and

$$y_{ijk} = \begin{cases} 1 & \text{if student } k \text{ takes subject } j \text{ which is in clusterline } i, \\ 0 & \text{otherwise.} \end{cases}$$

Moreover, the following notation is used:

$$
\begin{array}{ll}
L_j & \text{the number of lessons of one group of subject } j, \\
G_j & \text{amount of groups of subject } j, \\
S_{jk} & \begin{cases} 1 & \text{if student } k \text{ chooses subject } j, \\ 0 & \text{otherwise,} \end{cases} \\
A_j & \text{average amount of students in a group of subject } j.
\end{array}
$$

3.3.1 Objective function

The objectives are to minimize the total cluster scheme-length, the variations in group sizes and the amount of non-fitting students. Mathematically this means we try to solve

$$
\begin{array}{lll}
\textbf{Given} & \alpha, \beta, \gamma, L_j, A_j, S_{jk}, G_j & \\
\textbf{Min} & \alpha \sum_i \max_j (L_j x_{ij}) & \\
 & +\beta \sum_{i,j} x_{ij} \left(\sum_k y_{ijk} - A_j\right)^2 & \\
 & +\gamma \sum_{j,k} \left(S_{jk} - \sum_i x_{ij} y_{ijk}\right) & \\
\textbf{s.t.} & \sum_i x_{ij} = G_j & \forall j.
\end{array}
$$

The only constraint is that all subject groups need to be placed in the cluster scheme. This minimization problem is, apart from the non-linear term in the objective function, an integer linear program. The constants α, and β and γ are chosen by importance of the corresponding objective. However, since minimizing the total clusterlength is the main goal, α is chosen much greater than β and γ so that $\alpha >> \beta, \gamma > 0$ holds. The maximum group size is not considered in the constraints, as this is taken into account in the term $\sum_{i,j} x_{ij} (\sum y_{ijk} - A_j)^2$ (G. F. Post & Ruizenaar, 2004).

4 Existing Heuristics for Solving the Clustering Problem

There are an exploding number of possible ways to cluster a school section. Since going through all the possible outcomes is time-consuming, this section provides two existing heuristic algorithms for the minimization problem of subsection 3.3.1. First, the branch and bound method of G.F. Post & Ruizenaar is described and a pseudocode is provided. Second, we highlight the most important aspects of the backtracking algorithm by van Kesteren.

4.1 Branch and Bound Method

One of the attempts to solve an exploding number of possibilities for minimization problems is the branch and bound method, which works by eliminating symmetric lines and ordering subjects and groups to create fewer possible outcomes. For simplicity, the algorithm is used on a slightly more specific minimization problem than described above. The amount of non-fitting students is not minimized, but has become a hard constraint. Therefore, we apply $\gamma = 0$ and a hard constraint is added to limit the maximum cluster scheme-length M. The approach used here is analogous to (G. F. Post & Ruizenaar, 2004).

$$
\begin{array}{lll}
\textbf{Min} & \alpha \sum_i \max_j(L_j x_{ij}) + \beta \sum_{i,j} x_{ij}(\sum y_{ijk} - A_j)^2 & \\
& \sum \max_j(L_j x_{ij}) \leq M, & \\
& \sum_i x_{ij} = G_j & \forall j, \\
& \sum_i y_{ijk} = S_{jk} & \forall j, k, \\
& y_{ijk} \leq x_{ij} & \forall i, j, k.
\end{array}
$$

A branch and bound method is used to solve this optimization problem. The basic concept is to start with the first group of subject j and find the (next) possible clusterline for this group. When a possible line is found, we try the next group of the subject until all its groups are placed. Then we try to assign all the students to the groups. When this is impossible, we try to assign the last group to another line and try again. The process iterates through the subject groups until an assignment is successfully made. Like this, all groups of subjects are placed on the clusterlines and their students are assigned. With this technique, we can find many solutions for the problem. However, this algorithm has a bad performance, since it goes through all possible group arrangements. To avoid this, we will make use of the symmetry, compactness and order of students and subjects. The pseudocode is given in algorithms 1, 2 and 3.

Firstly, we classify the subject groups of the most complicated student k into separate clusterlines. The most complicated student is the one who chose the most subjects with the maximum amount of groups and thus provides maximum information for the clustering process. We call a subject with n groups a "n-grouper". For example, when the maximum number of groups per subject is three, the student with the most 3-groupers is placed first. This means its groups

are fixed to the first clusterlines and the student is assigned to these clusterlines.

Next, we begin placing groups on the clusterlines. This time the order, in contrast to the basic heuristic, is nonrandom. We start by assigning the 1-groupers. When we start placing a second 1-grouper, we only need to check if they have a student in common instead of assigning all the students. If this is the case, the 1-groupers should be placed on separate lines. Then, the 2-groupers are placed with a similar technique. If a 2-grouper includes a student already assigned to two 1-groupers, it must be placed on a different line.

Additionally, we check if assigning a subject group to a specific line could lead to an invalid solution before we assign students to groups. This is done by calculating the maximum number of students that could be applied to a group of subject j in line i, denoted by S_{ij}.
First of all, it should be at least the minimum group size m_j. Recall that A_j is the average number of students and G_j is the number of groups of subject j. The total number of students who follow subject j is thus $G_j A_j$. Subtracting the maximum we could have per group of j (denoted by M_j), we get

$$m_j = G_j A_j - G_j M_j + M_j = G_j A_j - (G_j - 1)M_j.$$

Thus, we need $S_{ij} \geq G_j A_j - (G_j - 1)M_j$. If this is not the case, we go to the next line without trying to assign the students.
Logically, the cluster scheme should contain enough possibilities to place all the students with that subject into a group. Therefore we need

$$\sum_i S_{ij} = G_j A_j \quad \forall j.$$

When this is not the case, we do not even try to place the subject in the scheme and return to the previous subject to rearrange its position (G. F. Post & Ruizenaar, 2004).

Once all groups of a subject are assigned to a clusterline, we allocate the students to the groups. We begin with a random order of the students. Each time a student cannot be assigned, they are prioritized by moving them up in the order. In this way, it is faster and clearer whether we are able to place the "troublemakers" in one of the groups. During this process, we ensure that the group size does not exceed the predetermined maximum number of students (G. F. Post & Ruizenaar, 2004).

In the basic heuristic, there is no limit in the number of lines we can use for the solution. Since we want to minimize the cluster scheme-length, which is the sum of the clusterline-lengths M_i, it is beneficial to have a minimum number of lines. Let the minimum number of non-empty lines be I^*, and N_k the number of subjects chosen by student k. Since student k needs to be able to attend all his lessons, the subjects are placed on different lines. Therefore, we have

$$I^* \geq N_k.$$

Recall that the total cluster scheme-length can be at most M. We have

$$\sum \max_j (L_j x_{ij}) \leq M.$$

Suppose $l = \min_j L_j$. Student k chose N_k subjects with in total L_k number of lessons. Since student k needs to be able to attend his lessons, these lessons are all placed on different clusterline. Therefore, each clusterline has a length of at least this subject's number of lessons. We have

$$\sum_i^{N_k} M_i \geq L_k.$$

The total length can be at most M. This gives

$$M \geq \sum_1^{N_k} M_i \geq L_k.$$

Note that $M_i \geq I^* l$ and $\sum_i^{N_k} \geq N_k l$ hold, since the clusterline-length is at least equal to l. Combining this, we get

$$\sum_i M_i - \sum_i^{N_k} M_i \geq (I^* - N_k) l$$

$$\Rightarrow \sum_i M_i \geq \sum_i^{N_k} M_i + (I^* - N_k) l.$$

Recall that $\sum_i^{N_k} M_i \geq L_k$. Therefore, the following holds

$$M \geq \sum_i M_i \geq L_k + (I^* - N_k) l.$$

Since we have no interest in empty clusterlines, we take in mind that we need the following number of clusterlines:

$$I^* = \min_k (M - L_k)/l + N_k.$$

Therefore, every time a group is placed on the next possible line, it is checked if the number of lines does not exceed I^*.

This already limits the number of possible solutions, but it is possible to reduce them further. The basic heuristic makes it possible to have the same subjects together on one line but on other clusterlines. For example, when a solution is a cluster scheme with $j = 1, 2$ on line 1, and $j = 3, 4$ on line 2, another solution is $j = 3, 4$ on line 1 and $j = 1, 2$ on line 2. In practice, this is the same outcome and we would like to remove this. Therefore, we make sure to use the

compactness of the cluster scheme. A group can only be placed on the ith line when lines 1 until $i-1$ are nonempty. Moreover, we can use the interchangeability of the groups. When a group is placed on line i, the next group will not be placed on lines $i-1, i-2, ...$ These two techniques remove most of the symmetry.

Finally, when the outcome is a valid solution, we want to minimize the cost of the cluster scheme. Since the second part of the minimization problem punishes unbalanced groups, we want to equalize them as much as possible. A greedy heuristic is used to balance them.

Firstly, as shown in algorithm 3, we find the line i and subject j of which $S_{ij} - A_{ij}$ is minimal and negative. In other words, we try to find the subject group with the biggest gap between the average and the possible number of students. This has the greatest impact on the cost and therefore we assign them as much as possible to this group. When finished, the group is not considered again. We repeat this until there is no i and j for which $S_{ij} - A_{ij} < 0$ holds.

Secondly, we consider all the remaining groups, this time in order of the biggest gap between $\sum_k y_{ijk}$, the number of students attending the subject j on line i, and A_j. Again, we start with the highest difference and assign all possible. Continue this with all other subjects until all are finished or above average.

We can easily check the impact of these efficiency methods by considering an example. Suppose we have an instance with 100 students, each student attending 6 optional subjects. They can choose from 17 subjects, of which 10 subjects have only one group, 6 consist of 2 groups, and 1 subject has 3 groups. Pessimisticly, we need as many lines as there are subject groups, so $10 \cdot 1 + 6 \cdot 2 + 1 \cdot 3 = 25$ lines. The number of possible group arrangements with 25 lines will be

$$25^{10} \cdot \binom{25}{2}^6 \cdot \binom{25}{3} \approx 1.6 \cdot 10^{32}.$$

Consider the scenario where $I^* = 8$ and the most complicated student attends six 3-groupers. By applying the techniques described above, we get approximately $6.8 \cdot 10^{11}$ possible outcomes. Optimistically, a solution is found every millisecond. This would indicate that running the algorithm still takes 22 years. We do note, however, that this algorithm is significantly faster in classifying the students to the subject groups.

Algorithm 1 Branch and Bound Method

```
// Precondition: mostcomplicatedstudent is the student who chose most subjects with the maximum number of groups.
// We begins with the groups of the 1-groupers, then of the 2-groupers, etc.
assign 1 group of each of most complicated student's subject to different clusterlines
assign the most complicated student to the placed groups
active group := first group
while first group is not tried in all lines do
    PlaceGroup(active group)
end while
Balance(cluster scheme)
```

Algorithm 2 Pseudocode of PlaceGroup

```
function PlaceGroup(active group)
    Find next possible line for the active group
    if (next clusterline is clusterline-number I* + 1
 or  current line is empty) then
        active group := previous group
    else
        if ([active group is 1-grouper and active group shares a student with placed 1-groupers]
or [active group is 2-grouper and shares a student with placed 1-groupers]
or [the clusterline-number is lower than that of a group of the same subject]
or [the maximum assignable students is lower than the minimum group])
then
            ≪ do nothing ≫
        else
            if group is last group of subject then
                if maximum assignable students for placed groups is more than students in this subject then
                    Try to assign students in subject
                    if all assignments are possible then
                        active group := next group
                    else
                        Move the error causing student one place up in the order
                    end if
                else
                    ≪ do noting ≫
                end if
            else
                active group := next group
            end if
        end if
    end if
end function
```

Algorithm 3 Pseudocode of Balance

```
function BALANCE(cluster scheme)
    while S_ij − A_ij < 0 for a (i, j) do
        Find (i, j) for which S_ij − A_ij is minimal
        Assign as many students as possible to this subject group
        Remove subject group from list
    end while
    while Σ_k y_ijk < A_j for a (i, j) do
        Find (i, j) for which Σ_k y_ijk − A_j is minimal.
        Assign as many students as possible to this subject group
        Remove subject group from list
    end while
end function
```

4.2 Simple Backtracking Algorithm

The assignment of students to subject groups can be done with a simple backtracking algorithm. We need all possible assignments and partial assignment in order to make an "as good as possible" cluster scheme. These partial assignments depend on the order of placement for which a certain permutation Π of the subjects is used. For the assignment algorithm we assume we can make use of the following functions, with student t and his set of chosen subjects P,

- FirstSubject(t), which returns the first element in P,
- LastSubject(t), which return the last element in P,
- NextSubject(s, t), which returns the element following subject s in P,
- PreviousSubject(s, t), which returns the element before subject s in P.

The elements in the set P are ordered according to the permutation Π. If the subject does not exist, for example subject s is the last subject and the function NextSubject(s, t) is used, a dummy subject s' is returned. The boolean IsSubject(s) will return **false** for the dummy subject and **true** for all other subjects.

Anonther function used is NextOpenclusterline(f, s) which returns the next clusterline which is not occupied by another group of the subject. It depends on the subject and on an assignment function f. The algorithm for the assignment is a simple recursive backtracking function.

First of all, the next assignment up to s needs to be found for student t. We assume that it is known that the chosen subjects by student t, the partial assignment function f, and the next assignment function up to s or up to PreviousSubject(s, t). We begin with checking if the subject s is indeed a subject. If this is not the case there is no success, otherwise we check if we can place

the subject group on the next clusterline with NextOpenCluster(f, s). If there is, there is success, otherwise we check if we can find the next assignment for the previous success, and then check again if we can find a free clusterline for subject S. This is repeated until a clusterline is found, or all previous subject are checked.

This is used to find the first complete assignment, so a cluster scheme with all subjects of student t. We begin with $f(s) = 0$ for all subjects s, define s as the first subject of t, and set success as true. While s is still a subject, success is true, and we search for the next assignment up to s, and set s as the next subject of student t.

Finally, we can search for the next complete assignment, knowing that f is a complete assignment function for t. We then just search for the next assignment up to the last subject of t using this f.

The order of the students is dependent on the number of possible assignments they have. When a student has the least possible assingment, he is considered the "hardest", since his flexibility is low. The backtracking described above is therefore done in order beginning with the student with the least assignments. When a student has multiple valid assignments, the assignment with the lowest increase of the objective function is chosen (van Kesteren, 1999).

5 The Clustering Problem at Zermelo

In this section, we elaborate on how the clustering problem is handled at Zermelo. This company is specialized in creating software to make schedules for Dutch high schools. To stay ahead of their competitors, they aim to improve and optimize the process of making an sufficient cluster scheme. At this company they use so called penalties to determine whether or not the cluster scheme is good enough. Therefore, first, we will define the penalty function on which the quality of the cluster scheme is based. Next, the backtracking algorithm they use is explained.

5.1 Problem definition

To determine the quality of a cluster scheme, a penalty function is defined. In this subsection, we mention which soft and hard constraints are considered.

5.1.1 Soft constraints

In section 3.2 we mentioned that a cluster scheme can have hard and soft constraints. When a soft constraint is not satisfied, the cluster scheme stays valid. However, it is still desired that such a constraint is met. Therefore, a penalty is given for each violated soft constraint. The height of the penalty depends on the importance of this constraint. The following are four categories of soft constraints used in the software of Zermelo.

1. *Cluster scheme*
    - *Cluster scheme-length*
      There is a maximum length of the whole cluster scheme. It is undesirable to have a cluster scheme-length greater than the total timeslots per week. Additionally, fewer timeslots in the cluster scheme allows for more scheduling flexibility for subject groups.
    - *Cluster-space*
      Even if the cluster scheme-length is shorter than the total timeslots per week, scheduling all subject groups might still be impossible. For instance, due to the availability of the teachers, the two-hour subjects mathematics and geography can only be taught on different days.
      This means that although the subject groups have no students in common, they cannot take place at the same time. Therefore, this line needs four schedule-positions instead of two. Moreover, this can cause free periodss for the students in these classes; When geography is given, the students of mathematics have no class to attend and vice versa.
      Thus, a penalty is given in case the cluster scheme has to be be scheduled on more timeslots than the cluster scheme-length.
    - *Non-groupable Students*
      In Dutch high schools, students have the choice to take an extra

subject. It is beneficial for these students to be able to attend the lessons of this subject, but not required. When it is not possible to classify such a student, a penalty is given.

- *Main Groups*
  Some lessons are assigned to a main group, which contain exactly the same students. Most of these groups are assigned as a main group beforehand. However, it can turn out that a subject group could contain exactly the same group of students as the main group. This makes scheduling more flexible, because it can take place at the same time as another main group. This has a positive effect on each lesson, so a negative penalty is counted per lesson of such a subject group.

2. *Clusterlines*

   - *Missing students*
     If a student is not present in a clusterline, they will not have a lesson simultaneously with the other subject groups of this line. This may result in free periods in between lessons, which, while tolerated in Dutch high schools, is not ideal. That is why there is a penalty for each student not present on a clusterline. However, this is not done for the line which has the most absent students; these classes could be placed on the first or last hours of the day.
   - *Freedom*
     The cluster scheme can be positioned in several ways. Due to the availability of teachers, most subject groups are not available on all timeslots. Groups are placed on one clusterline in order to be scheduled simultaneously. This is only possible if their combined availability matches the maximum lessons of a subject group placed on this line. The freedom-constraint gives a penalty when the availability is lower than the maximum lessons.
   - *Dayroom*
     Like the freedom, this constraint is related to the availability of the subject groups on a clusterline. Most high schools prefer to have one lesson of a subject group per day, except for two consecutive hours. When it is not possible to place the lessons of the subject groups on different days, a penalty is given for each day it lacks.
   - *Overflow*
     It is possible that some groups are already scheduled. When this is the case, there is a chance that there are more scheduled timeslots than the clusterslength. This is called an overflow. This could happen when not all teachers on one clusterline have the same availability during a week.
   - *Ceilings*
     It is an option to set a maximum clusterline-length. Clusterline 1, for example, has a maximum length of 3. This could be done to force

a solution where the one-hour groups are placed on the same line. When this constraint is not met, a penalty will be given.

- *Educational*
  A board or teacher of a high school can prefer in which way their classes are scheduled. For example, biology has to be given in one block of two consecutive lessons and two separate lesson, all on different days. On a clusterline there can be several groups with different preferable schedules, which results in a penalty. These constraints are called educational.

3. *Groups*

   - *Classifications*
     For classifications, constraints commonly used are limits of the group size and limiting the variation between group sizes of the same subject.

4. *Students*

   - *Student classifications*
     These are constraints to do with how students are classified in groups. For example, a teacher does not want to teach a certain student, or two student cannot be placed in the same group.

In section 5.4, we will describe mathematically the penalties of the categories *Cluster scheme-length, Freedom,* and *Classifications*.

5.1.2 Hard constraints

In addition to soft constraints, there are also hard constraints. When such a constraints is violated, the cluster scheme becomes invalid. Before these constraints can be defined, we need to introduce some variables.

Notation 1. Given are the definitions of the variables

S	the set of students,
S_{jk}	$\begin{cases} 1 & \text{if student } k \text{ chooses subject } j, \\ 0 & \text{otherwise,} \end{cases}$
T	the set of all timeslots,
V	the set of subjects,
L_j	the number of lessons of one group of subject j
G	the set of groups. This tuple is comprised of a class, teacher and a subject. this set is denoted by $\{(\mathcal{S}, v, t) \mid \mathcal{S} \subseteq S, v \in V, t \in T\}\}$
G_j	we have $G_j := \{g \in G \mid g \text{ is a group of subject } j\} \subseteq G$. we denote $\lvert G_j \rvert$ for the number of groups of subject j,
A_j	average amount of students in a group of subject j,
M_j	maximum amount of students in a group of subject j,
Z	the set of clusterlines.

Moreover, the decision variables are

$$x_{ij} = \begin{cases} 1 & \text{if a group of subject } j \text{ is in clusterline } i, \\ 0 & \text{otherwise,} \end{cases}$$

$$y_{ijk} = \begin{cases} 1 & \text{if student } k \text{ takes subject } j \text{ which is in clusterline } i, \\ 0 & \text{otherwise,} \end{cases}$$

and

$$w_{ijt} = \begin{cases} 1 & \text{if teacher } t \text{ teaches subject group } j \text{ which is in clusterline } i, \\ 0 & \text{otherwise.} \end{cases}$$

In the software used by Zermelo, there is one hard constraint in the cluster scheme: *collisions*. When a student or teacher is appointed to a specific clusterline more than once, the scheme becomes invalid.

Since a student cannot attend subject groups which are classified on the same line i, the hard constraint is

$$\sum_j y_{ijk} \leq 1 \quad \forall i, k.$$

A teacher could teach different subjects, which can result in being classified in a clusterline more than once. Since a teacher cannot teach two classes at the same time, the other hard constraint is

$$\sum_j w_{ijt} \leq 1 \quad \forall i, t.$$

Moreover, each student needs to be classified in their chosen subjects. We have

$$\sum_i y_{ijk} = S_{jk} \quad \forall j, k.$$

Lastly, to ensure for each subject that the predefined number of groups is placed, we require

$$\sum_i x_{ij} = G_j \quad \forall j.$$

When these constraints are violated the cluster scheme becomes invalid.

5.2 Penalty function

Not only the total clusterlength and the variation of the group sizes are minimized, but it takes into account all the constraints of the categories described in section 5.1.1. To consider all of them, a penalty function is formulated, starting with the definition of a cluster scheme.

Definition 1. A cluster scheme is a function

$$cs : G \to Z,$$

such that groups of the same subject v are placed on different clusterlines. We denote a cluster scheme by $\mathcal{L}$.

A cluster scheme is considered "good" when it has a low overall penalty.

Notation 2. $\mathcal{C}$ is the set of all possible soft constraints.

Definition 2. Let $\Phi : \mathcal{P}(G \times Z) \times \mathcal{P}(\mathcal{C}) \to \mathbb{R}_{\geq 0}$ be the *penalty function* of a cluster scheme $\mathcal{L} \in \mathcal{P}(G \times Z)$ subject to soft constraints $C \subseteq \mathcal{C}$(van der Kooy, 2017).

Below, several subsets of $\mathcal{C}$ are described. When constraints from such a subset are violated they create a penalty in the corresponding category as described in section 5.1.1.

1. $\mathcal{C}_{CS}(\mathcal{L})$, with $\mathcal{L}$ the whole cluster scheme. These penalties are all part of the *cluster scheme-length* category.
2. $\mathcal{C}_{CL}(z)$, with $z \in Z$ a clusterline. Each penalty given for a constraint of the *clusterlines* category is based on the classification of the clusterlines.
3. $\mathcal{C}_G(g)$, with $g \in G$ a subject group. These constraints belong to the *Groups* category and apply on the classification of these groups.
4. $\mathcal{C}_S(s)$, with $s \in S$ a student. Each penalty in the *Student-classifications* category is based on in which group he is classified.

The sum of these penalties is the total penalty of the cluster scheme. In other words, we have

$$\Phi(\mathcal{L}, \mathcal{C}) = \Phi(\mathcal{L}, \mathcal{C}_{CS}(\mathcal{L})) + \Phi(\mathcal{L}, \mathcal{C}_{CL}(z)) + \Phi(\mathcal{L}, \mathcal{C}_G(g)) + \Phi(\mathcal{L}, \mathcal{C}_S(s)).$$

This is indeed equal to the total penalty, since all the constraints are tied to only one single evaluator in a single category.

5.3 Objective function

The penalty as described above, and the hard penalties described in section result in the following minimization problem

$$\begin{array}{lll} \text{Given} & G_j, S_{jk}, & \\ \text{Minimize} & \Phi(\mathcal{L}, \mathcal{C}) & \\ \text{S.t.} & \sum_j y_{ijk} \leq 1 & \forall i, k, \\ & \sum_j w_{ijt} \leq 1 & \forall i, t, \\ & \sum_{i,j} y_{ijk} = \sum_j S_{jk} & \forall k, \\ & \sum_i x_{ij} = |G_j|, & \forall j. \end{array}$$

5.4 Penalty-functions of three soft constraints

In this subsection, we elaborate on three constraints and denote them mathematically. This will be done for the constraints of the *cluster scheme-length, Freedom,* and *Classifications*. We will focus on these constraints, since these three are all of high importance for a cluster scheme. First of all, a low length of the cluster scheme makes sure there are more possible ways to schedule the subject groups. Besides, the availability of groups of the same line should match to make sure they can be scheduled simultaneously. At last, classifications are of importance to guarantee a high quality of education and satisfy the teachers.

5.4.1 Cluster scheme-length

The cluster scheme-length is the total timeslots needed to schedule the subject groups of the cluster scheme. The length of one line is the maximum amount of lessons of a group on that clusterline, written as $\max_j L_j x_{ij}$. Therefore, the penalty for the length is

$$\mathcal{C}_{\text{length}} = \alpha \cdot \sum_i \max_j (L_j x_{ij}),$$

where $\alpha \geq 0$ is the weight of the penalty. Since the most important goal is to have an as low as possible cluster scheme-length, this weight is much higher than that of the other constraints.

5.4.2 Freedom

Given is the availability of the subject groups. This can be different for each group, since teachers can have different preferences for their days and times off. This is known beforehand, and denoted by

$$\Upsilon_{gt} = \begin{cases} 1 & \text{if subject group } g \text{ can be taught on timeslot } t, \\ 0 & \text{otherwise.} \end{cases}$$

with $g \in G$ and $t \in T$.

When a cluster scheme is made, we want to schedule the groups of a line simultaneously. Thus, the availability of these groups needs to be known. We denote this as

$$\upsilon_{gti} = \begin{cases} 1 & \text{if subject group } g \text{ is } \textbf{not} \text{ present on line } i, \\ & \textbf{or } \Upsilon_{gt} = 1, \\ 0 & \text{otherwise.} \end{cases}$$

The question is, if the availability of the clusterlines is equal to or more than the timeslots needed. For each clusterline, we have the penalty

$$\mathcal{C}_{\text{freedom}} = \beta \cdot \max \left\{ 0, \max_i (L_i x_{ij} - \sum_t \prod_g \upsilon_{gti}) \right\},$$

where the height of weight β should reflect that this is an important constraint and should not easily be violated.

5.4.3 Classifications

Classrooms have a maximum capacity which needs to be kept in mind during the clustering process; there cannot be more students classified to a group than this maximum. When this is violated, a penalty is given. Moreover, teachers prefer regular size classes, and will probably not like it if they have to teach a class with 30 people, and another teacher has a class of only five students. The penalty for these classifications is therefore the difference with the average group size and the students currently classified. Given is M_j, the maximum capacity, and A_j, the average class-size of subject j. For the whole cluster scheme, we have

$$\mathcal{C}_{\text{classifications}} = \sum_{i,j} \gamma_1 x_{i,j} \left(\sum_k y_{ijk} - A_j \right)^2 + \gamma_2 \left(\max\{0, \sum_k y_{ijk} - M_j\} \right)^2,$$

where γ_1 and γ_2 are the penalty weights. It depends on the school how important these classifications are, and thus how high these weights are.

5.5 Backtracking Algorithm

Since the clustering problem is NP-complete, we need a heuristic algorithm. Examples are (meta)heuristics like a greedy algorithm, hill climbing, simulated annealing, tabusearch and backtracking. At Zermelo they decided to use a backtracking algorithm.

At every step, the groups of a certain subject are placed in the cluster scheme. When all possible classifications result in overlapping students, backtracking is used and the previous step is revised accordingly. When another placement is possible, we proceed to the next step. Otherwise, we go another step back. This process is pursued until all subject groups are assigned to a clusterline and the cluster scheme is valid. A permutation is used for the order of placing the subjects, where the variables with the least number of possibilities are placed first.

After a predefined amount of time, the solution is stored and the process starts over with a blank cluster scheme and with a random permutation. Every time a valid scheme is made, the penalty is compared with the lowest penalty so far. If the penalty is improved, the related cluster scheme is stored. This process is done until the customer decides to stop the optimization. This could be after an hour without much progression or when the penalty and cluster scheme-length have satisfactory values.

6 Improving Cluster schemes with Hill Climbing

For the clustering problem, a heuristic algorithm is needed. The idea rises to use another heuristic than backtracking. In this section, two hill climbing heuristics, single movement and single shift, are explained. Moreover, a pseudocode is provided.

6.1 Single Movement

In this subsection, we will first describe the algorithm of the single movement, then describe some functions used, and finally provide and elaborate on the code used.

6.1.1 Description

In the section, we present an algorithm to search for small changes of the cluster scheme to improve the penalty induced by the cluster scheme. This is done by moving subject groups to empty spots in the cluster scheme. In other words, the group can be placed on another clusterline if there is no other group of that subject on that position.

In this new scheme, it could happen that a student is present on a line twice. Therefore, the students are reclassified to their chosen subjects. Then, it is checked if the cluster scheme is valid. This is the case when all students can be classified to their chosen subjects, all groups are placed, and when students and teachers are not placed on a line more than once.

Finally, the new penalty is calculated. If this is the lowest so far, this scheme is seen as the new cluster scheme. This means that everytime a change is made, there is an improvement.

We suppose that we can use the following functions

- GetElapsedTime(), returning the time since the algorithm has been called,
- GetPenalty(cs), returning the penalty of the cluster scheme cs. This is the sum of all the penalties given for the soft constraints that are not satisfied in this cluster scheme,
- Getclusterline(v, g), returning the clusterline on which the group g of subject v is placed,
- EmptyLine(z, v), which returns true when there is no group of subject v present on line z,
- MoveToLine(z, g), moving group g to line z,

- SortRandom(l), returning the list l sorted randomly,
- SortBetter(l), sort the list l with highest improvement first,
- CorrectlyClassifiedStudents(cs), returning false when there is a student, or teacher, who is not correctly classified. For example, the student, or teacher, is present in a clusterline more than once, or is not classified in all his chosen subjects,
- CreateBackup(cs), stores the current cluster scheme cs.
- RestoreBackup(), which replaces the cluster scheme with the back-up made with the function CreateBackup(cs).

6.1.2 Pseudocode

The algorithm of *Single Movement* is divided into two main functions: Movesubject groups(*maxtime*) and FindMovingsubject group(*success*). In the latter function, potential moves are listed. The move with the best improvement is executed. The former function checks whether this penalty is an improvement and whether the maximum time is not exceeded. Finally, there is a small check function FindValidMoves in which moves are validated.

In algorithm 4, we start with a significant penalty to make sure the first valid cluster scheme is the best until now. To find such a valid solution, we use the function *FindMovingsubject group*, given in algorithm 5. First, we start creating a backup to make sure we can go back to the original cluster scheme if no valid solution is found. Then, we start at the first subject and its first group and check on which line it is now. The first clusterline is checked to see whether it is the same as the subject group's line. If not, we check if it is an empty spot for the subject group. This means that no other subject group is placed here. When this is the case, we could shift this subject group. We try this for every subject group and check all the clusterlines if the group could be moved there. Each of these possible shifts is placed in the list *movingCandidates*.

When all subject groups are checked, the list of candidates is sorted randomly and the first candidate-shift is executed. We need to check if this results in a valid cluster scheme with the function given in algorithm 6. Here, the subject group is placed on the new line and checked if the cluster scheme is valid. If not, the students are reclassified. Again, it is checked if this results in a valid scheme. When this is the case, we check whether or not this move results in an improved solution by checking the new overall penalty. In case an improvement is found, the move is added to the successful moves list such that it can be executed at the end of this algorithm. We execute this for the first predetermined number of candidates. In the end, we execute the best successful shift and return success is set to true to the function *Movesubject groups*. When a success is found, the best penalty found so far is updated and we try it again. The algorithm

stops looking for new successful shifts if no improvement is found or when the predetermined time elapses.

Algorithm 4 Pseudocode of Move subject groups

```
function Movesubject groups(maxtime)
// Precondition: maxtime is the maximum runtime after which the
// algorithm stops searching for a lower penalty. The cluster scheme is de-
noted by cs.
// Ensure that there are no incorrectly classified students.
success ← false
bestTillNow ← 1.000.000.000.000
stop ← false
while not stop do
    Findmovingsubject group(success)
    if success then
        quality ← GetPenalty(cs)
        if quality < bestTillNow then
            bestTillNow = quality
        end if
    else
        stop ← true
    end if
    t ← GetElapsedTime()
    if t > maxtime then
        stop ← true
    end if
end while
```

Algorithm 5 Pseudocode of Find Moving subject group

```
function Findmovingsubject group(success)
// Precondition: the number of subjects ns,
and the number of clusterlines nl are known beforehand.
//Precondition: width is the predetermined number of random shifts to try
// The function determines which movement of which subject group results
in
// the best improvement of the cluster scheme.
success ← false
orignalPenalty ← GetPenalty(cs)
CreateBackup(cs)
for v = 1 to ns do
    for g = 1 to NumberOfGroupsOf(ns) do
        currentLine ← Getclusterline(v, g)
        for c = 1 to nl do
            if currentLine is not candidateLine c then
                if EmptyLine(c, v) then
                    add ((v, g), c) to list movingCandidates
                end if
            end if
        end for
    end for
end for
SortRandom(movingCandidates)
for count = 1 to width do
    if FindValidMoves((v, g), c) is true then
        penaltyAfterMove ← GetPenalty(cs)
        improvement ← originalPenalty − penaltyAfterMove
        if improvement > 0 then
            add ((v, g), c, improvement) to list succesfulmoves
        end if
    end if
    RestoreBackup()
    count++
end for
SortBetter(succesfulmoves)
if size of succesfulmoves > 0 then
    ((bestsubject, bestgroup), bestline) ← first element of succesfulmoves
    FindValidMoves((bestsubject, bestgroup), bestline))
    success← true
end if
return success
```

Algorithm 6 Pseudocode of Find Valid Moves

```
function FindValidMoves(group, candidatLine)
// This function checks if moving subject group group to the candidateline results in a valid cluster scheme.
CreateBackup(cs)
MoveToLine(candidatLine, group)
if Not CorrectlyClassifiedStudents(cs) then
    ReClassify(cs)
end if
if Not CorrectlyClassifiedStudents(cs) then
                                ▷ If it is still not valid after reclassification
    RestoreBackup()
    return false
else
    return true
end if
```

6.2 Single Shift

Another way to optimize the cluster scheme is to switch two groups of the same subject with each other. In case most students match, this could cause a new, valid scheme.
Due to the performance of the algorithm, only a limited amount of movements is checked for a valid classification. In case they are valid, the penalty is calculated. Eventually, the best improvement is executed.

Besides the assumed known functions described in section 6.1.1, there are two extra functions used to find improvements of changing two groups of clusterline.

- AlreadyInvestigated($(v, g), (i, j)$), returning true if the change of (i, j) with (v, g) is already tried,

- ReClassify(cs), classifying the students to their chosen subject in the new cluster scheme.

6.2.1 Pseudocode

Since reclassifying the subject is time-consuming, algorithm 8 randomly selects a predefined number of shifts. These selected shifts are tested in algorithm 9. Algorithms 8 and 9 are used in algorithm 7 which checks for improvements and ensures the maximum time is not exceeded.

To ensure the first valid cluster scheme is accepted as the "best till now", we set the initial penalty to -1. In algorithm 7, when a valid scheme is found, we check whether we have a valid scheme test (meaning the penalty is minus

1), and whether the quality of the new cluster scheme surpasses the previously found solution. Algorithm 8 is used to determine whether a successful shift can be found. As stated in the pseudocode of algorithm 6, we first select the first subject and its first group and request their clusterline number. Then, we go through all subject groups and check whether these are placed on the same line. In case a subject group is on another clusterline, we add this group to the *movingCandidates* list, together with the original group. This process is repeated for all subject groups.

Again, the list is sorted randomly and the first move is tried. We cannot test every possible move, as this would be too time-consuming. We first check if this shift has already been investigated. Then, we check if the change is valid with algorithm 9. A backup is created to go back to the original cluster scheme. Line 1 belongs to group 1, line 2 to group 2. To shift them, we let group 1 move to line 2, and group 2 to line 1. When this is not a valid cluster scheme, we first try to reclassify the students over the groups. If it remains invalid, we restore the backup and report that no valid change was found. In that case, we test the next moving candidates. If a valid change is found, we keep this new cluster scheme. Initially, we can choose how many changes we want to test. When this is more than one, we execute the process again until this number of valid new cluster schemes is met.

Finally, we execute the best move found and return success to algorithm 7. The loop starts again to find a new successful cluster scheme. The process stops if no further successful shifts are found or when the predetermined time elapses.

Algorithm 7 Pseudocode of Make Singular Shiftings

```
function MakeSingularShiftings(maxtime)
// Precondition: maxtime is the maximum runtime after which the
// algorithm stops searching for a lower penalty.
// Ensure that there are no incorrectly classified students.
besteTillNow ← −1
success ← false
stop ← false
while not stop do
    FindShiftings(success)
    if success then
        quality ← GetPenalty(cs)
        if bestTillNow < 0 or quality < bestTillNow then
            bestTillNow ← quality
        end if
    else
        stoppen ← true
    end if
    t ← GetElapsedTime()
    if t > maxtime then
        stoppen ← true
    end if
end while
```

Algorithm 8 Pseudocode of Find Shiftings

```
function FindShiftings(success);
//Precondition: width is the predetermined number of random shifts to try
//Function determines which shift results in (the most) improvement
success ← false
orignalPenalty ← GetPenalty(cs)
for v = 1 to ns do
    for g = 1 to NumberOfGroupsOf(ns) do
        b ← Getclusterline(v, g)                                  ▷ current line
        for i = 1 to ns do
            for j = 1 to NumberOfGroupsOf(ns) do
                c ← Getclusterline(i, j)
                if b is not equal to c then
                    add ((v, g), (i, j)) to list movingCandidates
                end if
            end for
        end for
    end for
end for
SortRandom(movingCandidates);
for count = 1 to width do
    if AlreadyInvestigated((v, g), (i, j)) then
        width ++                          ▷ Do not want to check same shift twice
    else
        if FindValidChanges(original,candidate) is true then
            penaltyAfterShift ← GetPenalty(cs)
            improvement ← originalPenalty − penaltyAfterShift
            if improvement > 0 then
                add ((v, g), (i, j), improvement) to list succesfulchanges
                success ← true
            end if
        end if
        RestoreBackup()
    end if
    count++
end for
SortBetter(succesfulchanges)
if size of succesfulmoves > 0 then
    (bestgroup1, bestgroup2) ← first element of succesfulmoves
    FindValidMoves (bestgroup1, bestgroup2)
    success← true
end if
return success
```

Algorithm 9 Pseudocode of Find Valid Changes

```
function FindValidChanges(group1, group2)
// This function checks if changing subject group group1 and subject group
// group2 results in a valid cluster scheme.
CreateBackup(cs)
line1 = Getclusterline(group1)
line2 = Getclusterline(group2)
MoveToLine(line2, group1)
MoveToLine(line1, group2)
if Not CorrectlyClassifiedStudents(cs) then
    ReClassify(cs)
end if
if Not CorrectlyClassifiedStudents(cs) then
                                    ▷ If it is still not valid after reclassification
    RestoreBackup(cs)
    return false
else
    return true
end if
```

7 Results of Hill Climbing Methods

In this section, we present the analyses of the hill climbing methods described in section 6. This is done by the mean and median of the penalty after one hour, and how frequent certain cluster scheme-lengths occur.

The objective is to compare the application of the Single Movement and Single Shift algorithm with the method where only backtracking is used. To test the Single Movement and Single Shift algorithms, a hybrid method is used which combines these two with the backtracking method of subsection 5.5. First, the initial cluster scheme is made using backtracking, and then one of the algorithms is used to optimize the penalty. The probability of which algorithm is used, varies in each situation below. Again, after a certain time, the whole cluster scheme is removed, and rebuilt by the backtracking method.

Since backtracking itself is an optimization strategy, in two of the new methods, backtracking is still used for the optimizing process. In the first test, this is done with 50% chance, and both the single movement and single shift are used with 25% chance. In the second test, they all have the same probability of execution. In the last test, backtracking is just used for the initial schedule, and optimization is done by the hill climbing methods. All these new methods are compared with the original methods in which solely backtracking is used.

For each test, we use the same subject groups and student of the same grade and school. This grade has 31 subjects of which 4 are attended by main groups and thus not clustered. The number of subject groups vary from one to six. Moreover, there are 192 students in this grade. The most number of subjects chosen by one student is 10.
On this grade, each test is executed 100 times, where each test is terminated after one hour.

	Mean of Penalty	Mean of Schedule-length
Original Method	11588167	39,12
B 50, M 25, S 25	11142313	38,87
B 33, M 33, S 33	11171685	38,91
B 0, M 50, S 50	11496436	39,24

Table 1: The mean and schedule-length of the penalty after one hour of running the algorithm. Here, the Original Method is with the backtracking method, and no movement and shifting. The "B" stands for "Backtracking", "M" for "Movement", and "S" for "Shifting". The numbers behind the methods represent the percentage of times the associated code is called.

	Median of Penalty	Median of Schedule-length
Original Method	11252116	39
B 50, M 25, S 25	11249272	39
B 33, M 33, S 33	11250550	39
B 0, M 50, S 50	11277622	39

Table 2: The median of the penalty and schedule-length after one hour of running the algorithm. Here, the Original Method is with the backtracking method, and no movement and shifting. The "B" stands for "Backtracking", "M" for "Movement", and "S" for "Shifting". The numbers behind the methods represent the percentage of times the associated code is called.

In the data shown in table 1 and table 2 show that using 50% backtracking results in the lowest penalty and mean of the schedule-length. Recall that the schedule-length is the cluster scheme-length plus the schedule-positions needed for the subjects attended by the basic groups.

The effectiveness of combining backtracking, moving and shifting can be explained by the locality of each methods's search. Moving a single subject group to another clusterline results in a cluster scheme that remains largely similar to the original one. Shifting two subject group introduces more change but still stays close to the initial solution. Backtracking, however, allows multiple subject groups to be placed on entirely different lines, enabling a broader search. In other words, backtracking can explore near-optimal solution before jumping to a different region of the solution space. Meanwhile, moving and shifting refine the search locally, helping to identify a local optimum.

Although the two methods combining backtracking, shifting and moving have a lower mean than the original method, there is no significant difference. Moreover, note that the median of the schedule-length is the same for all combinations.

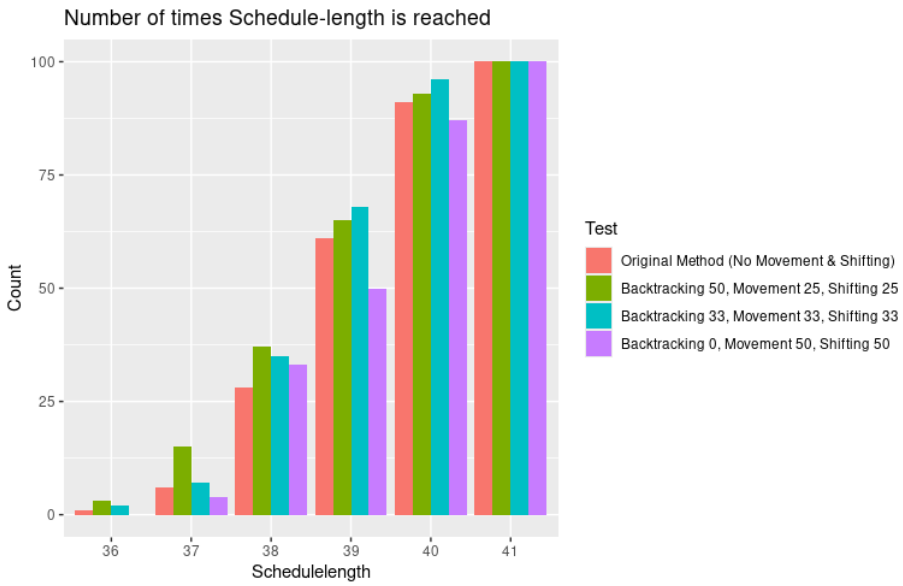

Figure 1: The number of times a schedule-length is reached in one hour. Each test is done 100 times.

We run each test 100 times. Not all of them reach the same schedule-length in one hour. In figure 1 it is seen that all of them reach minimal cluster scheme 41. The tests of the *"Backtracking 0, Movement 50, Shifting 50"* reaches the lengths 37,38,39 and 40 the fewest times. Schedule-length 36 is not reached a single time by this algorithm.
Figure 1 shows that "*Backtracking 50, Movement 25, Shifting 25"* achieves the lower lengths, 36, 37, and 38, the most out of the four methods. This coincides with the intuitive explanation given earlier.

8 Discussion

In this research, our main focus was to investigate the High School Clustering Problem. We have applied the local search functions *Single Movement* and *Single Shift* in which subject groups are placed on a different clusterline and shifted with other groups. We have compared this strategy with the original backtracking algorithm. We found that the strategy to combine backtracking and hill climbing can improve the clusterscheme within our dataset.

To recall, a cluster scheme with a low penalty implies a low cluster scheme-length and thus a low schedule-length. This implies that fewer positions in the schedule are needed. This makes scheduling more flexible and thus less complex to schedule all the lessons. Furthermore, due to the flexibility, it is more feasible to satisfy the tough requirements high schools can have.

It is remarkable that the final schedule-length of a cluster scheme can differ significantly, even when the same method is applied. In figure 1 it is seen that even for the same test, length 36 was only reached a couple of times. It is an interesting question why there is such variation and how the cluster scheme can get to this more optimal situation faster.

One limitation of this study was that it was conducted on a single dataset. While for this specific case the method of combining backtracking with single movement and single shift had a positive effect, it is unclear if this holds in general. It did, however, indicate that this algorithm can result in a more optimal solution ánd that this method was applicable.

Each test is executed for one hour, which is a reasonable amount of time to wait for an optimized cluster scheme. Future research could examine whether extending this duration, for example, 8 hours, would yield significantly more optimized cluster scheme.

For our experiments, we have decided to examine 50 random movements or shifts each time the function *Findmovingsubject group* or *FindShiftings* is called. Investigating the effect of a different amount of movements and shift is left for future research.

The bottleneck for our approach was reclassifying the students to their chosen subjects, as this takes a relatively long time. To avoid this bottleneck, one could only try the valid movements and shifts without reclassifying. Since combining optimizations is likely to have a better result, this could also be done in combination with the reclassifying method.

Another improvement could be to examine traingle-switch, moving subject group a to the line of subject group b, moving b to the line of subject group c, and c to the line of a, is beneficial to add. Moreover, other polygon-switches

could be examined. The downside is that the amount of possible switches increases drastically, which could negatively impact the speed of the algorithm.

To generalise this research, our methodology could be tested on multiple sections of different schools to give a better understanding of the overall effectiveness of this method.

In conclusion, more optimal cluster scheme were found by a combination of backtracking, single shift, and single movement during the clustering process. Even though I have not shown that this method is beneficial for all clustering problems for all schools, this thesis shows that it could be an option for a scheduler to test.

9 Symbols & Notation

A_j	Average amount of students in a group of subject j,
$\mathcal{C}$	The set of all possible soft constraints
G	The set of groups. This tuple is comprised of a class, teacher and a subject. This set is denoted by $\{(\mathcal{S}, v, t) \mid \mathcal{S} \subseteq S, v \in V, t \in T\}\}$
G_j	The set of groups of subject j, denoted by $G_j := \{g \in G \mid g \text{ is a group of subject } j\} \subseteq G$.
HSSP	High School Scheduling Problem,
I^*	The minimum number of non-empty cluster lines,
L_j	The number of lessons of a group of subject j,
$\mathcal{L}$	The cluster scheme,
M	The maximum length of the cluster scheme,
m_j	The minimum group size of a group of subject j,
M_j	The maximum capacity of subject j,
N_k	The number of subjects chosen by student k,
S	The set of all students,
S_{ij}	The maximum amount of students that can be applied to subject j when placed on line i,
S_{jk}	$\begin{cases} 1 & \text{if student } k \text{ chooses subject } j, \\ 0 & \text{otherwise,} \end{cases}$
T	The set of all timeslots,
V	The set of all subjects,
w_{ijt}	$\begin{cases} 1 & \text{if teacher } t \text{ teaches subject group } j \text{ which is in clusterline } i, \\ 0 & \text{otherwise,} \end{cases}$
x_{ij}	$\begin{cases} 1 & \text{if a group subject } j \text{ is in clusterline } i, \\ 0 & \text{otherwise,} \end{cases}$
y_{ijk}	$\begin{cases} 1 & \text{if student } k \text{ takes subject } j \text{ which is in clusterline } i, \\ 0 & \text{otherwise,} \end{cases}$
Z	The set of clusterlines,
v_{gti}	$\begin{cases} 1 & \text{if subject group } g \text{ is } \textbf{not} \text{ present on line } i, \\ & \quad \textbf{or } \Upsilon_{gt} = 1, \\ 0 & \text{otherwise.} \end{cases}$
Υ_{gt}	$\begin{cases} 1 & \text{if subject group } g \in G \text{ can be taught on timeslot } t \in T, \\ 0 & \text{otherwise.} \end{cases}$

References

Carter, M. W., & Laporte, G. (1998). Recent developments in practical course timetabling. In E. Burke & M. Carter (Eds.), *Practice and theory of automated timetabling ii* (pp. 3–19). Berlin, Heidelberg: Springer Berlin Heidelberg.

Carter, M. W., Laporte, G., & Lee, S. Y. (1996). Examination timetabling: Algorithmic strategies and applications. *Journal of the operational research society*, *47*(3), 373–383.

Carter, M. W., & Tovey, C. A. (1992). When is the classroom assignment problem hard? *Operations Research*, *40*(1-supplement-1), S28–S39.

Ministerie van Onderwijs, Cultuur, en Wetenschap. (n.d.). *Aantal vo-scholen.* Retrieved from `https://www.ocwincijfers.nl/sectoren/voortgezet-onderwijs/instellingen/aantal-vo-scholen`

Ministerie van Onderwijs, e. W., Cultuur. (2023). *Feiten en cijfers over leraren en het lerarentekort.* Retrieved from `https://open.overheid.nl/documenten/dpc-91b57e282681f06e0e7f38caa087c4bb464b045e/pdf`

Onderwijs, D. U. (n.d.). *Onderwijspersoneel vo in aantal personen.* Retrieved from `https://duo.nl/open_onderwijsdata/voortgezet-onderwijs/personeel/in-aantal-personen.jsp`

Post, G., Ahmadi, S., & Geertsema, F. (2010). Cyclic transfers in school timetabling. *OR spectrum*, *34*, 133–154.

Post, G. F., & Ruizenaar, H. (2004). Clusterschemes in dutch secondary schools.

Rijksoverheid. (n.d.). *Prognosecijfers leerlingendaling.* Retrieved from `https://www.rijksoverheid.nl/onderwerpen/leerlingendaling/prognosecijfers-leerlingendaling`

van der Kooy, N. (2017). The high school scheduling problem: Improving local search & fairness evaluation.

van Kesteren, B. (1999, 8 26). The clustering problem in dutch high schools changing metrics in search space.