School Timetabling in Theory and Practice

Irving van Heuven van Staereling

VU University, Amsterdam
Faculty of Sciences

December 24, 2012

Preface

At almost every secondary school and university, some weeks or months before school starts, there is always one person who stands before an enormous puzzle, and it is always the same one. After spending days to schedule all lectures one by one, he finally found a timetable which should be acceptable and ready to publish. However, he knows that probably many improvements could have been possible, but that he simply does not have the time, insight or perhaps motivation to find them.

School timetable construction can be an extremely difficult task and usually consumes a large amount of time. Although some have a better intuition for the problem than others, it is most of the time practically impossible to construct a timetable that satisfies every teacher and student. For example, it always occurs that students have idle hours between their lectures (which many students experience as a time waste), or worse, have courses with clashing lectures. Such conditions affect the choice of student's curriculum or even the student's performance; one can imagine that two scheduled exams on a day could affect a student's concentration and thus his chances of success. In other words, finding a good timetable is not only desirable, but may also be essential. At many school institutes, timetables are constructed by hand, which implies (for mathematicians) that improvements should be possible. This research paper will investigate this conjecture by providing an overview and contribution regarding the theory and practice of school timetabling, with the ambition to improve the currently available timetabling software.

This work is part of my Master's degree program Business Analytics at the VU University, where each student is required to perform a research regarding a specific problem motivated by practice. The choice for this subject is primarily driven by my own experience and interests; it was my impression that the timetables I personally encountered had a lot of room from improvement. For example, I frequently noticed that two or three simple adjustments in a timetable could avoid overlapping lectures or reduce idle hours between lectures, and it was for me a very disturbing idea that it would probably take only one smart insight to save 40 students from waiting 2 hours for their next lecture. Furthermore, I discovered during my Bachelor program that the courses within operations research interested me the most, and finding a good timetable is due to numerous constraints and preferences (of teachers, students and rooms) a perfect, complex optimization problem.

As last, but most important, I would like to extend my gratitude to my supervisor dr. Sandjai Bhulai. His ideas, efforts and enthusiasm have helped me enormously in making this research paper. Thanks Sandjai!

Irving van Heuven van Staereling
December 2012

Summary

This research paper is concerned with the theory and practice of school timetabling. An overview and contribution has been provided on both the theoretical and practical side, which resulted in a satisfying addition to the theory of timetabling and new, important insights to construct better timetables in practice.

The theory of timetabling comprises problems that indeed are less complicated than the problems in practice, but a theoretical analysis provides a fundamental understanding of the complexity of timetabling. The most basic timetabling problems (almost without any special constraints) can be solved easily using elementary graph theory. However, when other realistic possibilities are added (such as unavailabilities of teachers), the problem becomes strongly NP-hard, meaning that it is unlikely that the problem can be solved fast in theory. An own contribution to the theory concerned a variant in which only the number of lectures on a specific time unit needs to be maximized, which is proved to be strongly NP-hard as well.

The logical result is that the problems in practice also are very hard to construct due to many various constraints. Such constraints include requirements such as the prevention of clashing lectures, but also preferences such as the reduction of high workloads on a day. To construct a timetable that can take any type of constraint into account, this research paper proposes a new heuristic: the tournament heuristic. This heuristic consists of three components: an extensive initialization technique, a local search method and a "survival of the fittest" mechanism. Furthermore, the heuristic can take the personal timetable of every single student into account, including priorities of the constraints that need to be fulfilled.

This heuristic has been tested on experimental data that is based on data of the VU University. In fact, the problem instance could even be more difficult than is encountered in reality. The results show that the heuristic can find a timetable of very good quality within only 100 seconds. For example, the number of clashing lectures for students can reduced with 30% to 70% compared to standard timetabling methods that are used in practice. Similar numbers are achieved for other performance measures, such as the prevention of high workloads and idle hours. Depending on the time and preferences the school or university has to construct a timetable, different techniques need to be applied to find the best timetable. Even though some parts of the heuristic have been considered before, the combination of existing and new ideas have lead to the satisfying results.

For a detailed explanation of the practical accomplishments, the reader is referred to Chapter 3 (explanation of the heuristic) and 4 (experimental results). The theoretical part, Chapter 2, does not need to be read for a full understanding of the practical part, since the used theoretical insights are recapitulated when required.

Contents

Chapter 1

Introduction

This research paper provides an extensive analysis of the theory and practice behind school timetabling. Not only the most important findings in the research field are discussed, but also new own findings regarding the theory of timetabling are presented. However, the main piece of this report is the presentation of a new heuristic, that has been tested on realistic experimental data.

The focus will mainly be put on timetabling for secondary schools and universities, but this does not imply that the discussed techniques are solely applicable to schools; one could imagine that the same setting holds in e.g., hospital planning, where students are replaced by patients and teachers by doctors. In this chapter, the reader will be introduced to the field of school timetabling by illustrating the practical relevance of the research today. In Section 1.1, the background of school timetabling is provided to characterize the context of the problem. The goals and research questions are formulated in Section 1.2 whereupon an outline of this research paper is given in Section 1.3.

1.1 Background

The construction of school timetabling for secondary schools and universities has been recognized as a difficult problem for decades. Informally, the problem can be formulated as finding the best allocation of lectures between teachers and students over a finite number of time units and rooms, while satisfying a (large) number of various constraints. These constraints are mostly simple (e.g., a teacher may not teach on Tuesday), but it is usually hard to satisfy a large combination of multiple constraints in which many classes and teachers are involved. Even though the problem can be extremely large and complex, school timetables are nowadays mainly constructed by hand, simply because the available software is not satisfactory. Both free and paid timetabling programs have their limitations in terms of computation time, the lack of functionalities or simply the poor quality of the timetables. In fact, there currently exists no software which is considered as the "holy grail", making further research and development in the field the more interesting.

In practice, the construction of school timetables consists of two distinct phases [9]. The first step concerns the creating of the curricula of each class including

the assignment of resources (such as teachers or equipment) to the courses/classes and the number and types of lectures per course. The second step deals with the assignment of these lectures to rooms and times. In this research paper, the focus will only be put on the second phase and regard the information in the first step as given. After all, the first step is generally performed by coordinators and/or teachers of the educational institution.

Furthermore, the constraints can be distinguished in two different types: *requirements* and *preferences*. Requirements (or hard constraints) need to be fulfilled at all cost, because no education can be given otherwise. When a teacher is not available on a specific day, then his lectures clearly may not be assigned to that day. Preferences (or soft constraints) need to be fulfilled maximally, as long as the hard constraints remain unaffected. These preferences only increase the luxury and satisfaction of the classes and teachers. For example, students dislike waiting for their lectures, so it is in their interests to minimize the number of "idle hours" between lectures. In many secondary schools in Europe, the timetable fulfills the hard constraints, but has a lot of room for improvements regarding the soft constraints. The problem for universities is so complex that even not all hard constraints can be satisfied. It is not unusual that students have to choose between two courses because they have clashing lectures, or that lectures are scheduled in rooms with not enough space.

As last, it is worth mentioning that school timetabling problems come in several variants. The generally accepted classification of the timetabling problem [19] distinguishes the problem in three main classes:

- **School timetabling:** the weekly scheduling for all the classes of a school, avoiding teachers meeting two classes at the same time and vice versa.
- **Course timetabling:** the weekly scheduling for all lectures of a set of university courses, minimizing the overlap of lectures of courses having common students.
- **Examination timetabling:** the scheduling for the exams of a set of university courses, avoiding overlap of exams of courses having common students, and spreading the exams for the students as much as possible.

Note that the three classes are not distinct; some timetabling problems can fall in between two of the classes. Particularly the first classification method is considered throughout this research paper, but a heuristic will be proposed that is able to take the characteristics of all three problems into account.

1.2 Goals and research questions

As mentioned in Section 1.1, the realistic assumption is made that the list of events for which the classes and teachers are already given, such that the problem becomes the assignment of events to rooms and time slots. Generally, the timetable also has to be constructed within a specific amount of time (minutes, hours, days or even weeks), so there is a need for a method which generates a timetable of good quality

in proportion to the running time. These aspects provide the basis for the following main research question that will be investigated in this research paper:

Given a list of events, requirements and preferences for a school timetable, how can the best timetable be constructed within a reasonable amount of time?

Even though the intuition behind the research question should be clear, it is intentionally formulated such that new subquestions arise. For example, how can the "best" timetable of all possible timetables be defined? Or even better, when is one timetable is better than another? A large difficulty to answer this subquestion is that the quality of a timetable is subjective. It may well be that one student prefers that two lectures are spread over two days for an optimal focus during the lectures, while another prefers two lectures on the same day to obtain a free day. Also, it is not clear yet what all possible (realistic) constraints and preferences are for the problem, and what in practice is considered as "a reasonable amount of time". These subquestions will be considered throughout the research paper before the main research question is answered.

1.3 Outline

The outline of this research paper is as follows. To obtain a thorough understanding of the complexity behind the construction of timetables, the problem will initially be formulated in its simplest form. Subsequently, several variants and extensions of the basic problem are discussed until the basic problem is extended into a complex, practical problem. For that reason, this research paper is split up a theoretical and a practical part.

Chapter 2 is concerned with the theory of timetabling. A survey will be provided with reference to previous works that have shown under which conditions the timetabling problem is easy or hard to solve. Furthermore, a new theoretical finding is included regarding a specific variant of the timetabling problem. For this part, the reader is required to have a good understanding of combinatorial optimization, such as (integer) linear programming, graph theory and reducibility among NP-hard problems. Several combinatorial optimization problems are mentioned and used to prove NP-hardness of timetabling problems, and even though some of these problems are explained shortly as part of the proof, a formal definition of these problems can be found in Appendix A.

The practical aspects of timetabling will be discussed in remainder of the research paper. In Chapter 3, an overview is given containing the best-known approaches at this moment, but also their strengths and shortcomings are formulated shortly. The practical part consists of an own contribution of heuristics including implementation, after which its effectiveness will be demonstrated using realistic experimental data in Chapter 4. Knowledge on evolutionary algorithms such as local search is advantageous, but not required. As last, the most important findings are summarized in Chapter 5 including suggestions for future work.

Chapter 2

School timetabling in theory

This chapter considers the theory behind school timetabling construction, and starts with a formulation of the most general case of the timetabling problem in Section 2.1 which serves as a fundamental basis for the remaining sections. Section 2.2 consists of an overview of proposed optimal algorithms for easy (polynomially solvable) variants of this general case, whereas Section 2.3 considers previous works that prove which variants are hard (strongly NP-hard). Finally, Section 2.4 contains an own contribution to the theory, a proof of strongly NP-hardness of a variant of the timetabling problem.

2.1 The Class-Teacher problem

The basics of timetabling theory consider decision problems, meaning that only the question is asked whether a feasible timetable exists (i.e., satisfying the hard constraints without taking preferences into account). One of the first fundamental definitions of such a school timetabling problem has been provided in [24] and is formulated as follows:

Class-Teacher

Given: A set of classes $C = \{c_1, \ldots, c_m\}$, a set of teachers $T = \{t_1, \ldots, t_n\}$, p time units and an $(m \times n)$ matrix R, where r_{ij} is the number of lectures that have to take place between class c_i and teacher t_j.

Goal: Determine whether there exists a timetable of at most p time units s.t. all lessons are assigned while no class or teacher is involved in more than one lecture per time unit.

Note that only 3 specific constraints are mentioned so far:

1. Every lesson needs to be taught.
2. Every class cannot be involved in more than one lecture per time unit.
3. Every teacher cannot be involved in more than one lecture per time unit.

More formally, Class-Teacher can be formulated as an Integer Programming problem using the following binary variables and constraints:

$$x_{ijk} = \begin{cases} 1, & \text{if class } c_i \text{ and teacher } t_j \text{ meet at time unit } k, \\ 0, & \text{otherwise.} \end{cases}$$

$$\sum_{k=1}^{p} x_{ijk} = r_{ij}, \qquad i = 1, ..., m; j = 1, ..., n, \tag{1}$$

$$\sum_{j=1}^{n} x_{ijk} \leq 1, \qquad i = 1, ..., m; k = 1, ..., p, \tag{2}$$

$$\sum_{i=1}^{m} x_{ijk} \leq 1, \qquad j = 1, ..., n; k = 1, ..., p, \tag{3}$$

$$x_{ijk} \in \{0, 1\} \qquad i = 1, ..., m; j = 1, ..., n; k = 1, ..., p. \tag{4}$$

Note that an objective function is redundant, since decision problems are only interested in finding feasibility and not optimality. It turns out that one can determine very quickly whether a feasible timetable exists for any instance of Class-Teacher. To see this, firstly define Δ as the maximum number of lessons in which any student or teacher is involved, i.e.:

$$\Delta = \max \left\{ \max_{i=1,...,m} \sum_{j=1}^{n} r_{ij}, \max_{j=1,...,n} \sum_{i=1}^{m} r_{ij} \right\}.$$

Recall that Class-Teacher asks the questions whether a feasible timetable exists using at most p time units.

Theorem 1. *There exists a solution for* Class-Teacher *if and only if* $\Delta \leq p$.

Proof. The original proof is provided in [24] and is based on a bipartite graph representation. Given an instance of Class-Teacher, construct a bipartite graph $G = (C, T, R)$ where the classes and teachers represent the nodes, and node c_i and t_j is connected by r_{ij} parallel edges. Using this representation, one can formulate the problem as an edge coloring problem, which is an assignment of colors to the edges such that no two adjacent edges have the same color. Every color represents a different time unit. Furthermore, note that Δ is equal to the maximum vertex degree of the bipartite graph. König's Line Coloring Theorem [14] states that the required number of colors for a feasible edge coloring in a bipartite graph always equals its maximum vertex degree, meaning that a feasible timetable exists if and only if $\Delta \leq p$. □

To find a feasible timetable, one should iteratively construct matchings that contain edges incident to all vertices with maximum degree and remove these matched edges (give these edges color i in iteration i). This leads to a decrease of 1 of the maximum degree of the graph. Hence, this procedure must terminate in exactly Δ iterations, meaning that exactly Δ colors are required to assign colors to all edges (and thus solve the timetabling problem).

2.2 Well-known easy timetabling problems

In combinatorial optimization, the term "easy" for a problem is used if there exists an algorithm that can solve a problem fast in theory. More formally, a problem is easy if any instance of the problem can be solved in polynomial time of the input. As seen in the previous section, the basic timetabling problem is easy. This section provides three of the most well-known extensions that are shown to be easy as well.

2.2.1 Limited rooms

In Section 2.1, the assumption was made that it was possible to schedule any number of lessons at any time unit. This implies an infinite number of available rooms, which clearly is unrealistic. In fact, the room capacity is the largest bottleneck for many high schools and universities. Therefore, an extension of Class-Teacher is considered in which at most ρ identical rooms can be used (with $\rho \in \mathbb{N}$) during every time unit by adding the following constraint:

$$\sum_{i=1}^{m}\sum_{j=1}^{n} x_{ijk} \leq \rho \qquad\qquad k = 1, ..., p \quad (5)$$

Fortunately, it is also for this extension possible to determine a priori whether a feasible timetable exists. To see this, firstly define λ as the total number of lessons to be taught, i.e.:

$$\lambda = \sum_{i=1}^{m}\sum_{j=1}^{n} r_{ij}$$

Lemma 1. *There exists a solution for* Class-Teacher *including the limited rooms constraint if and only if* $\lceil \frac{\lambda}{p} \rceil \leq \rho$.

The proof is provided in [3], but for an intuition of timetabling complexity, the initial insight of the proof is given. This insight notes that whenever there are λ lessons to be given in a p-time unit timetable, that at least $\lceil \frac{\lambda}{p} \rceil$ rooms are required in at least one time unit. One can show that it is always possible to find a timetable in which λ lessons are scheduled in p time units s.t. at most $\lceil \frac{\lambda}{p} \rceil$ rooms are occupied each time unit, which directly results in Lemma 1 (see reference).

2.2.2 Daily maximum

Class-Teacher considers a timetable of p subsequent time units, while practice deals with five daily timetables (Monday until Friday). This does not necessarily have to be a problem, since the chained timetable easily can be split into 5 pieces. However, it would be ideal for students and teachers if the number of lectures is limited to a specific maximum for every day, which is the extension to be considered in this subsection. Redefine x_{ijk} as the number of lectures in which class c_i and teacher t_j meet at day (instead of time unit) k. Moreover, define a_i and b_j as the maximum number of lectures in which respectively class c_i and teacher t_j may be

involved with on any of the p days (instead of time units). Now, this extension can be formulated as an Integer Programming problem by using constraint (1), (2) and (3) plus the following constraints:

$$\sum_{j=1}^{n} x_{ijk} \leq a_i, \qquad i = 1, ..., m; k = 1, ..., p, \tag{6}$$

$$\sum_{i=1}^{m} x_{ijk} \leq b_j, \qquad j = 1, ..., n; k = 1, ..., p, \tag{7}$$

$$x_{ijk} \in \mathbb{N}_0, \qquad i = 1, ..., m; j = 1, ..., n; k = 1, ..., p. \tag{8}$$

Lemma 2. *There exists a solution for* Class-Teacher *including the daily maximum constraints if and only if* $\sum_{j=1}^{n} x_{ijk} \leq p \cdot a_i$ *for* $i = 1, ..., m$ *and* $\sum_{i=1}^{m} x_{ijk} \leq p \cdot b_j$ *for* $j = 1, ..., n$.

The proof is provided in [24] and also uses a bipartite graph representation and edge coloring formulation. In this case, the goal is to assign p colors to every edge such that no more than a_i and b_j edges are adjacent to respectively node c_i and t_j have the same color. It is not hard to see that the minimum number of required days is equal to the maximum number of days any teacher or class needs, i.e.:

$$p = \max \left\{ \max_{i=1,...,m} \left\lceil \sum_{j=1}^{n} \frac{r_{ij}}{a_i} \right\rceil, \max_{j=1,...,n} \left\lceil \sum_{i=1}^{m} \frac{r_{ij}}{b_j} \right\rceil \right\}.$$

One can show that this minimum always can be achieved which directly leads to Lemma 2 (see reference).

2.2.3 Balanced workload

A related extension is the balancing of lectures over the days. The reasons for doing this ought to be clear; the maximum work for students and teachers on a day is then minimized, leading to a balanced workload. In other words, this extension considers constraint (1) including:

$$\left\lfloor \sum_{j=1}^{n} \frac{r_{ij}}{p} \right\rfloor \leq \sum_{j=1}^{n} x_{ijk} \leq \left\lceil \sum_{j=1}^{n} \frac{r_{ij}}{p} \right\rceil, \qquad i = 1, ..., m; k = 1, ..., p, \tag{9}$$

$$\left\lfloor \sum_{i=1}^{m} \frac{r_{ij}}{p} \right\rfloor \leq \sum_{i=1}^{m} x_{ijk} \leq \left\lceil \sum_{i=1}^{m} \frac{r_{ij}}{p} \right\rceil, \qquad j = 1, ..., n; k = 1, ..., p, \tag{10}$$

$$\left\lfloor \frac{r_{ij}}{p} \right\rfloor \leq x_{ijk} \leq \left\lceil \frac{r_{ij}}{p} \right\rceil, \qquad i = 1, ..., m; j = 1, ..., n; k = 1, ..., p. \tag{11}$$

Recall that p now represents the number of days instead of the number of time units.

Lemma 3. *There exists a solution for* Class-Teacher *including the balanced workload for any* p.

The proof is also provided in [24] and is an extension of Theorem 1 and Lemma 2.

2.3 Well-known hard timetabling problems

The extensions in Section 2.3 are unfortunately three of the few realistic extensions that are easy. Many of the other realistic extensions are hard, meaning that there is no known algorithm that can solve all possible instances of the problem fast (in theory). More formally, a problem is hard (or strongly NP-hard) when there exists no polynomial time algorithm that can solve the problem, unless P = NP. In the coming subsections, three of the most well-known hard extensions of Class-Teacher are discussed. Even though the reader is referred to the original works for the complete proofs of NP-hardness, an intuitive explanation (or the initial step) of the proofs will be provided to illustrate the complexity of school timetabling. Knowledge of reducibility among NP-hard problems is required.

2.3.1 Lectures involving multiple classes

In high schools and universities, students have the freedom to choose a part of their courses. This means that the scheduler needs to take into account that two lectures cannot be given at the same time, because there is a group of students that might want to attend both lectures. The complexity for universities is much higher as students have a larger freedom concerning the choice of their curriculum, and especially for this case, extra (hard) constraints need to be fulfilled. The first task is to seek a time slot such that two classes need to have the same lecture on the same moment. However, assigning this lecture to a certain time slot might influence the possibilities for the other lectures that need to be scheduled for the concerning classes. In other words, there are strong (direct, but also indirect) dependencies between classes that make the timetabling problem not trivial to solve.

Lemma 4. Class-Teacher *including the possibility that lectures involve multiple classes is strongly* NP*-complete.*

The proof is provided in [8] and is based on a reduction from the decision variant of the strongly NP-hard problem Graph K-Colorability. In this problem, a graph $G = (V, E)$ and an integer K are given. The goal is to determine whether there exists a coloring of at most K colors s.t. all vertices are assigned a color and $Color(u) \neq Color(v)$ for every $\{u, v\} \in E$. In other words, for every edge, the color of the two belonging nodes must be different.

Given an instance of Graph K-Colorability, one can construct an instance for the timetabling problem s.t. every vertex $v \in V$ represents one course with only one lecture, and the edges $\{u, v\} \in E$ represent classes attending course u and v. Clearly, if the two vertex colors of every edge (i.e., the time units of the lectures of the class) are different, then the timetable is feasible. Showing that the finding of a K-coloring in Graph-K-Colorability is equivalent to finding a timetable of K time units is then straightforward.

2.3.2 Merging lectures in rooms

Merging different lectures in the same room is normally not an option. However, there may be cases in which merging classes in one room is acceptable, efficient and thus desirable. For example, it is usual in examination timetabling that two different courses have their exam in the same room. This saves staff cost (the number of required supervisors could be halved) and is efficient in terms of space. The largest bottleneck is to group the exams, such that the sum of the sizes of the classes does not exceed the room capacity. Note that the problem itself is not exactly a timetabling problem, but is part of the timetabling process.

Lemma 5. Class-Teacher *including the option to merge classes in one room is strongly* NP*-complete.*

The proof is based on a very simple reduction from the decision variant of the strongly NP-hard Bin Packing problem. In this problem, a bin capacity V, an integer B and a list of values $A = \{a_1, \ldots, a_n\}$ are given, where $a_i \in [0, V]$. The goal is to determine whether there exists a partition of A into at most B sublists such that each sublist sums to at most V.

Given an instance of Bin Packing, one can construct an instance for the timetabling problem s.t. the sizes of the classes are represented by the list of values in A and there are B identical rooms available of capacity V. Showing that the finding of a valid partition of at most B sublists in Bin Packing is equivalent to finding a valid assignment for the classes to B identical rooms is then straightforward.

2.3.3 Unavailabilities of teachers

In practice, many teachers are not available at all time units when lectures can be given; both high schools and universities employ part-time teachers, who are only available on specific days. Hence, this extension of Class-Teacher considers the possibility that teachers are some time units are unavailable for teaching (these time units are given). Note that this problem is slightly different than the daily maximum constraints, as that extension considered a fixed maximum for all days, whereas unavailabilities may differ per day.

Lemma 6. Class-Teacher *including unavailabilities of teachers is strongly* NP*-complete.*

The proof is provided in [11] and is based on a reduction from the strongly NP-complete problem 3-Satisfiability, but is due to its length omitted. In this problem, a set of Boolean variables $X = \{x_1, x_2, ..., x_n\}$ and a set of clauses $C = \{c_1, c_2, ..., c_m\}$ is given, where each clause is a disjunction[1] of exactly 3 variables $\in X$ and a Boolean formula $F = c_1 \wedge c_2 \wedge \cdots \wedge c_m$. The goal is to determine whether there exists a truth (True or False) assignment to $x_1, ..., x_n$ s.t. $F =$ True. One can show that the timetabling problem is strongly NP-complete already when there are only 3 time units. The problem becomes easy when classes are always available and every teacher is available for exactly two time units (see reference).

[1] A disjunction is a Boolean formula containing only "or"-operators, e.g. $c_i = (x_3 \vee \neg x_5 \vee \neg x_6)$.

2.4 Maximizing lectures on a time unit

In this section, a new timetabling variant is discussed. The problem is motivated by the fact that timetabling problems are extremely difficult, and clever solution methods are required. The subproblem of the timetabling problem that is considered only wants to maximize the number of lectures on a specific time unit. Even though the lectures that need to be scheduled on multiple days are given, it is a natural approach to maximize each time unit individually. This could provide the basis of a heuristic that firstly maximizes the amount of lectures on time unit 1, subsequently on time unit 2, and so on.

Theorem 2. *Maximizing the amount of lectures on a time unit in* Class-Teacher *including the possibility that lectures involve multiple classes is strongly* NP*-hard.*

Proof. The proof is based on a reduction from the decision variant of the strongly NP-hard problem Maximum independent set. In this problem, a graph $G = (V, E)$ and an integer K is given, and the goal is to determine whether there exists an independent set I of size at least K. An independent set I is a subset of the vertices such that there is no edge between any two of the vertices in I, i.e., $I \subseteq V$ s.t. $\{u, v\} \notin E$ for every $\{u, v\} \in I$.

Given an instance of Maximum independent set, construct the following instance for the timetabling problem:

- Every vertex $v \in V$ represents one course with exactly one lecture.
- Every edge $\{u, v\} \in E$ represents a class attending course u and v.
- The objective for the timetabling problem is to determine whether it is possible to schedule at least K lectures on one specific time unit.

If Yes-instance for Maximum independent set, then schedule every lecture represented by a vertex (in I). Because I is an independent set, there is no edge (class) between the vertices (lectures) in I, meaning that there is no class that has to attend both lectures. Therefore, the schedule is feasible for all classes and the number of lectures scheduled is equal to at least K.

If Yes-instance for the timetabling problem, then there are at least K lectures scheduled on the specific time unit. Now pick the corresponding vertices (lectures) in I for the Maximum independent set instance. By definition, $|I| \geq K$ and there is no edge between any pair of vertices in I. Otherwise, there would have been a class (edge) following both lectures (i.e., connected to both vertices), which would contradict to the assumption that the timetabling problem had a Yes-instance. Hence, I is an independent set of size at least K. □

In other words, simply maximizing the number of lectures on one specific time unit is already strongly NP-hard, meaning that even intuitive approaches to acquire suboptimal timetables are not possible. Knowing the complexity of the problem, timetabling solvers are forced to consider heuristics that cannot provide any performance guarantee with respect to the quality of the solution, but still can be extremely powerful. Such heuristics will be considered in the next chapters.

Chapter 3

School timetabling in practice

In this chapter, the complete, practical setting of the timetabling will be researched, by firstly providing a short overview of known approaches in Section 3.1. After that, Section 3.2 formalizes the required definitions, assumptions and constraints, after which the problem can be defined formally. This provides the basis for the crown of this research paper to solve the timetabling problem, the tournament heuristic. This heuristic consists of three components, described in Section 3.3, whereupon two practical remarks are made in Section 3.4.

3.1 Previous work

School timetabling is due to its practical relevance and complexity a research topic of relatively large interest, and has therefore been investigated by many researchers. The two most well-known surveys can be found in [15] and [19], that also discuss the open research questions. The direction of this research paper is mentioned in [19], being the direction of standardization. Every educational institution has its own rules and constraints, making it more difficult to create a program that is applicable for any type of school and university.

Many researchers have proposed algorithms after reducing the timetabling problem to a graph coloring problem (such as in [2, 16, 22, 23, 24]), but only consider relatively simple constraints. For example, it is very difficult to take the minimization of idle time units into account. Fortunately, there exist approaches which can take almost any type of constraint into account such as evolutionary algorithms, simulated annealing and tabu search, of which an comparison is made in [7]. A large part of such approaches use a weighted penalty function that define the quality of a timetable as the weighted sum of violated constraints; this is also explained in the next sections, as this is similar to the approach in this research paper.

Finally, it is worth mentioning that the survey in [15] notes that a distinction can be made in so-called one-stage and two-stage algorithms. One-stage algorithms optimize both hard and soft constraints at the same time (see e.g., [1, 5, 7, 18, 20]). On the other hand, two-stage algorithms firstly optimize hard constraints, and optimize soft constraints when a feasible timetable is found (see [4, 6, 13, 23, 25] for different variants of such two-stage algorithms).

3.2 Problem formulation

3.2.1 Definitions

Before formulating the TIMETABLING PROBLEM, define the following:

Definition 1. *A **student set** is a set of students that have the identical set of lectures (curriculum) that it wants to attend.*

This term will replace the earlier used term "class" because of the following reason. It could be that a specific class contains multiple specializations; for example, the class "Economics" has multiple specializations (such as "Finance", "Accountancy", etc.). These specializations are in the same class and both share some courses, but also have a lot of courses not in common. The use of the concept of student sets makes the model more accurate, since it is now known more precisely when a specific set of students is available or not (because the set has the identical curriculum).

With this small definition, it is possible to define the input of the problem formally. Even though this notation is not used extensively in the remainder of this research paper, the following list provides an overview of all ingredients that are required for a timetabling problem:

- $S = \{S_1, ..., S_x\}$ is the set of student sets (with different student set sizes).
- $T = \{T_1, ..., T_y\}$ is the set of teachers.
- $R = \{R_1, ..., R_z\}$ is the set of rooms (with different room capacities).
- $E = \{E_1, ..., E_n\}$ is the set of events to schedule. Each event E_i consists of:
  - A set $w_{E_i} \subseteq \{1, ..., w\}$, the weeks in which the event has to be scheduled.
  - A set $S_{E_i} \subseteq S$, the set of student sets that are involved with the event.
  - A set $T_{E_i} \subseteq T$, the set of teachers that are involved with the event[1].
- $C = \{C_1, ..., C_m\}$ is the set of constraints. Each constraint C_j consists of:
  - A resource $\rho_{C_j} \in (S \cup T \cup R)$, being the student set, teacher or room that is involved with the constraint.
  - A type τ_{C_j}, the type of constraint (clarified in the next subsection).
  - A penalty $\pi_{C_j} \in \mathbb{R}^+$, indicating the degree of satisfaction per violation of the constraint.
- w is the number of weeks of the timetabling instance.
- d is the number of days of the timetabling instance.
- u is the number of time units per day of the timetabling instance.

The reason why the term "time unit" is chosen instead of hour, is due to the applicability of the model. Some schools teach in blocks of 45 minutes, while others teach in blocks of 60.

[1] Usually, $|T_{E_i}| = 1$ (i.e. the number of teachers that are involved in an event equals one).

3.2.2 Constraints

As mentioned in Section 1.1, requirements (or hard constraints) are constraints that need to be fulfilled at all costs, while preferences (or soft constraints) need to be fulfilled if possible. In the next paragraph, an overview is given of the six important types that the constraint set C could contain.

1. Avoid unavailabilities (requirement) Clearly, the largest priority for the school or university is that every event takes place, which clearly cannot happen without teacher. Hence, the events need to be scheduled such that every student set and teacher is available, including an appropriate room. Note that also student sets could be unavailable, e.g., due to excursions, or weekly obligations at other educational institutes.

2. Avoid clashes (requirement/preference) A constraint which is almost as important as the previous, is the constraint to avoid clashes of compulsory courses for all student sets (requirement). For optional courses, it is no disaster to have clashes; the student set could pick one of them, but not both (preference). Therefore, optional courses should have a lower penalty compared to compulsory courses. In other words, this constraint also guards the degree of choice that student sets have concerning optional courses.

3. Distribute events of course (preference) This constraint is close to being a requirement, but is sometimes accepted. Most courses contain more than one lecture per week, and it is preferred that these are spread over the week rather than having both lectures on the same day.

4. Avoid high day load (preference) Lectures require a lot of concentration from the students, and this concentration decreases over the day. Ideally, the number of lectures on a specific day does not exceed a specific number. When exams would be considered instead of lectures (examination timetabling), this constraint would be even more desirable (and therefore have a higher penalty).

5. Avoid idle time (preference) Many students experience idle time (the time between their lectures) as a waste of time and loss of concentration for the last lecture. A perfect timetable for the students would contain no idle time, such that the days are as short as possible.

6. Avoid working day (preference) This constraint only applies for students. It could for example be that students have two different lectures on two different days, meaning that they also have to travel two different days for just one lecture. In this case, most (but not all) students prefer to have these two lectures on the same day, such that they have a day off. For teachers, this constraint is less relevant, since they usually are present at the education instance every day anyway.

Recall that a constraint C_j consists of a resource ρ_{C_j} (a specific room, student set or teacher), a type τ_{C_j} and a penalty π_{C_j}. It is important to note the reason for making a distinction between different resource types; after all, idle time units for student sets could be less desirable for teachers. But also the weight for two specific, different student sets could differ. One could argue that different weights for different student sets need to be used for the same constraint, e.g., by taking the number of students in the set into account. For example, an idle hour for a student set containing 100 students is less satisfactory than an idle hour for a single student.

Note that only six constraints are mentioned, but that much more constraint exist. For instance, many university have room types (e.g., computer rooms), which add a new dimensions to the problem. Even though the heuristic proposed in the next section can handle any type of constraint, they are not included in the remainder of this research paper. Otherwise, the results would depend on too many parameters and constraints, making it harder to understand the value of the results.

As last, the penalties per constraint also have an interpretation. For a specific student set S_i, the constraint "avoid idle time" could have a penalty of 2, and the constraint "avoid working day" a penalty of 5, which implies the following. Students are in this case willing to have one or two idle hours, if it provides them a free day. However, the students are willing to sacrifice a free day (total penalty 5) if it prevents three idle time units (total penalty 6).

3.2.3 The timetabling problem

Basically, the goal of the timetabling problem is to find a solution σ that simply assigns every event $e_i \in E$ to a specific collection of $|w_{E_i}|$ time units and rooms, where $|w_{E_i}|$ is the number of weeks in which event e_i needs to be scheduled.

Denote $f(C_j, \sigma)$ as the number of violations that solution σ causes for constraint C_j. For example, if C_j is a "avoid idle time"-constraint for a specific student set, and solution σ causes five idle time units for that student set, then $f(C_j, \sigma) = 5$. Clearly, if $f(C_j, \sigma) = 0$, then the constraint is satisfied, but $f(C_j, \sigma)$ can never be smaller than 0. Knowing this, the complete timetabling problem can be defined as follows:

Timetabling problem

Given: A set of student sets S, teachers T, rooms R, events E, constraints C and three integers w, d, u, respectively the number of weeks, days and time units per day.

Goal: Find an solution σ that minimizes the total penalty function $\Pi(\sigma, C) = \sum_{C_j \in C} \pi_{C_j} \cdot f(C_j, \sigma)$.

Clearly, this combinatorial optimization problem is strongly NP-hard, since the timetabling problem with only unavailabilities of teachers is proven to be strongly NP-hard (see 2.3.3). This problem contains unavailabilities and many other hard and soft constraints, and the goal is to find optimality instead of feasibility (in contrast to most theoretical timetabling problems).

3.3 The tournament heuristic

The proposed heuristic consists of three components: an initialization, local search and a survival of the fittest mechanism. Even though parts of the heuristic are intuitive, the combination has not been suggested before. Especially many of the initialization methods have not been discussed before extensively, while their influence will prove to be striking. The three components of the heuristic are listed below.

3.3.1 Qualification

The first step of the heuristic basically considers several types of initialization methods. An initialization is simply a first attempt to assign every event to a specific time slot and room as good as possible. Before listing several smart ways in finding an initial solution, it is required to introduce the following:

Definition 2. *The **tightness** of an event E_i, $tight(E_i)$, is the number of time units at which there is an involved student set or teacher that is not available **or** no room exists with enough capacity for the student sets.*

Definition 3. *The **flexibility** of an event E_i, $flex(E_i)$, is the number of time units at which every involved student set and teacher is available **and** a room exists with enough capacity for the student sets.*

In other words, the tightness is the number of time units at which the event cannot be scheduled without large problems (unavailabilities or clashes), while the flexibility is the reversed. Recall that d and u are respectively the number of days and time units per day, meaning that $d \cdot u$ the the number of time slots at which an event can be scheduled.

Corollary 1. $d \cdot u = tight(E_i) + flex(E_i)$ *for all* $E_i \in E$.

Initialization involves two different components: the order at which the events will be scheduled (ordering method) and the way these events are assigned to time slots (assignment method). For example, it is possible to order the events on tightness and to assign these events on this order to a random time slot. The investigated four ordering and four assignment methods are discussed below.

Random (ordering method) The first ordering method is self-explanatory; the order in which the events will be scheduled is random.

Simple tightness-based (ordering method) The simple tightness-based ordering is perhaps the most intuitive. Note that the tightness of an event is a measure of how hard it is to schedule. Since the events with the highest tightness are the most difficult to schedule, it is more intelligent to (try to) schedule these first. After all, if these would be scheduled last, then there would be even less possibilities for this tight event, which could result in clashes that could have been prevented. Therefore, this method orders the events on descending order of tightness beforehand, and schedules the tightest first.

Dynamic tightness-based (ordering method) When more events are assigned to time slots, the tightness of events could increase. Consider a student set S_i which is involved with two different events E_1 and E_2. If the first event is scheduled on a specific time slot, then it could be that the $tight(E_2)$ increases by one, because the student set is not available anymore on that specific time slot. Note that $tight(E_2)$ does not necessarily have to increase, if there was another student set or teacher involved with E_2 not available anyway on that specific time slot. For this reason, the dynamic tightness-based ordering method dynamically updates the tightness of each event, and schedules the one with the highest tightness.

Size-based (ordering method) This method orders the events on number of involved student sets. If there are more student sets involved, then there are more timetables to take into account, which is easier to handle at the start of creating an initial solution. When there are multiple events with the same number of involved student sets, the tightest is chosen (dynamically).

Random (assignment method) For this first assignment method, a remark has to be made because it is not entirely random. Clearly, the idea is to assign the event to a random time slot. However, the current implementation also takes two constraints into account, being the "avoid unavailabilities" and "avoid clashes" constraint.

This assignment method firstly checks at which time slots the event can be scheduled without clash and unavailability of anyone and chooses a random time slot of this set of time slots. However, if the set is empty, then a new attempt is done in which clashes are allowed (but no unavailabilities), and randomly assigns to a time slot which causes the least number of clashes. In case such a time slot also does not exist (i.e., there exists no time unit at which the student sets and teachers are available or there exists no room with enough capacity), the event is not scheduled. In practice, the event should then be scheduled outside the normal time units (e.g., in the evening), which regularly occurs in universities due to the lack of room capacity.

Fixed (assignment method) The previously mentioned remark also holds for this method. But rather than assigning the event to a random time slot, it chooses a fixed ordering. Thus, if the event can be scheduled on the first time unit on Monday, then the method does so. Otherwise, it checks the second time unit on Monday, third time unit, and so on.

The first time slots on the first days does not necessarily have to be the fixed order. One can argue to schedule in the middle of the day first (instead of the start of the day), to reduce the probability of creating many idle time units further on, and this is also the order that has been implemented. For instance, if there are three time units per day, the method firstly tries to assign to the second time unit on Monday (if not possible, second time unit of Tuesday and so on). If the "middle" of the schedule is not possible, then the first or third time units of the days are tried.

Lowest tightness increase (assignment method) The motivation for this assignment method is the question whether there is a time slot which could cause the least problems (potential clashes) further in the initialization. The only reason why clashes occur, is when the tightness of a specific event is too high. For that reason, this method assigns the event to the specific time slot that increases the tightness of all other events the least.

Lowest penalty (assignment method) The final but perhaps also most logical method assigns the event to the time slot with the lowest penalty. This assignment method is arguably also the one that (unconsciously) is used in practice, when employees need to make a timetable manually. The scheduler needs to assign a specific event and can choose several time slots, but needs to take multiple factors into account (clashes, idle time units, etc.). These factors have different priorities and clearly, the scheduler wants to choose the time slot causing the least trouble.

It is important to note that the ordering method and assignment method are truly separate. One could select the event with the highest tightness (dynamic tightness-based ordering method) and assign this event to a random time slot. The same argument holds for all other combinations, thus there are 16 different initialization methods in total, which are potential "participants" for the tournament heuristic. Every initialization method is further referred to as (*Ordering method*, *Assignment method*), e.g. (Random, Lowest penalty) or (Dynamic tightness, Fixed).

To fully explain the qualification mechanism, it is necessary to know that the tournament heuristic in further phases include techniques that:

- improve an initial solution including some randomization (such that different local optima can be found).
- quickly finds the best solution of a collection of k initial solutions, $\sigma_1, \ldots \sigma_k$, the "participants" of the tournament.

These two methods are discussed in more detail in the following two subsections. With this information, the qualification of the tournament heuristic is also concerned with the choice of the most-promising participants, which can be created using any of the above mentioned ordering and assignment methods. However, if it turns out that one of the initialization methods is superior, then all "participants" in the tournament heuristic could be created using that specific initialization method.

Determining the best initialization methods can simply be done as follows. For each of the 16 methods, generate n initial solutions, and improve these until a local optimum is found (using the technique which will be discussed shortly). The initialization method(s) that produced the solution(s) with the lowest penalty, is/are the best initialization method(s). As a remark, it is recommended to choose for each initialization method the solution with the best penalty rather than the average penalty. For example, the (Random, Random) method could perform poor on average, but also could produce extremely good outliers. In that case, the (Random, Random) method is the most interesting to investigate, because it potentially yields the best solution (given a large amount of time).

3.3.2 Recursive local search

Given an initial solution σ, there usually is still a lot of room for improvement. Improvement techniques are also considered in other works (e.g., in [20]), and such techniques are similar to the approach considered for the tournament heuristic. To find improvements, the heuristic basically moves a specific event from one time slot to another, to obtain a slightly adapted solution σ'. If the new penalty is smaller than the old penalty, i.e., if $\Pi(\sigma', C) \leq \Pi(\sigma, C)$, then keep σ'. Otherwise, continue with the old solution σ and reject the move. Note that σ' is also accepted when the penalty is the same, since this could help escaping from local optima. This procedure is repeated until either a specific time limit is reached or no improvement is made after a specific amount of time.

So far, the heuristic only considers one move at the time. Recursive local search implies to do multiple steps at once (to escape from local optima). It could be beneficial to move a specific event only if another event is also moved. The current implementation considers a maximum of thinking one step ahead, meaning that two moves are attempted at the same time (in [20] defined as a "double move"). Clearly, triple moves and further also can be considered, but this could increase the computation time significantly. The most important reason to disregard this, is because double moves already do not show a significant improvement compared to single moves.

One could think of a smart selection of events to move, and a smart search of the time slots to which these events should be moved to. However, there are good arguments to do this randomly. This means that the local search takes a random event, and try to move this event to a random possible time slot, and see whether improvements are made. This allows for a larger diversification in the participating solutions, which is important for a further phase of the tournament heuristic. Also, it saves computation time; after all, when a search has to be done to find the best time slot, the search runs in $\mathcal{O}(d \cdot u)$ instead of $\mathcal{O}(1)$. But for the sake of completeness, both search methods have been investigated, but the random search proves to be more effective because a greedy approach results usually in a bad local optimum. Both are hill climbing techniques (solutions with a higher penalty are never accepted), but the last-mentioned method searches explicitly for the largest improvement.

Tabu list A small but worthwhile addition to the heuristic concerns a tabu list of events. After all, the heuristic randomly takes an event and searches improvements. Whether this improvement was successful or not, it is potentially more rewarding to try a different event to move, rather than the same one. Hence, the tabu list (with fixed capacity) of events contains the events which may not be moved in the next move, e.g. 50% of the number of events. In other words, if the number of events $n = 1000$, and a specific event E_i has just been considered to move, then it will not be considered again in the next 500 moves (this event is then declared taboo).

3.3.3 Knock-out race

In this phase, the heuristic will improve and evaluate the potential quality of the initial solutions against each other in a very fast way, and requires k initial solutions. These k solutions can be made using a combination of the techniques mentioned in the previous section. For instance, a participating solution could be one where the events were ordered and assigned randomly, the (Random, Random) method. Any of the other 16 combinations can be used for the initialization of participating solutions, and also specific initializations may be considered multiple times. After all, local search does not converge constantly to the same local optimum, since randomization is invoked in the search.

The heuristic starts with a set of initial solutions $\Omega = \{\sigma_1, \sigma_2, ..., \sigma_k\}$ (or set of participants) and optimizes all solutions to find its local optimum using the recursive local search method, as discussed in the previous subsection. However, for some of these solutions, the heuristic will discover that its quality is poor and is doomed to fail (i.e., improvements would still result in a bad local optimum). Therefore, the heuristic will stop improving these solutions (these solutions are removed from the race), such that it can dedicate its valuable time to improve the more promising solutions.

Initially, the heuristic spends a specific amount time t_1 (say, in seconds) in which all k solutions are improved using the recursive local search method in parallel. In other words, all solutions are changes with the use of randomization (in the local search) to converge to a local optimum. Subsequently, the heuristic removes a specific amount of solutions $r_1 < k$ from Ω, being the r_1 solutions with the highest penalty. These solutions are the least promising solutions after t_1 seconds and are therefore not interesting enough to investigate further. After that, the heuristic optimizes the remaining $k - r_1$ solutions in Ω in parallel for t_2 seconds, after which the worst $r_2 < k - r_1$ are removed. This procedure is done p times (the number of phases), meaning that the user of the heuristic needs to determine three parameters:

- The set of initial solutions $\Omega = \{\sigma_1, \sigma_2, ..., \sigma_k\}$.
- The computational time vector $t = (t_1, \ldots, t_p)$. Note that $\sum_{i=1}^{p} t_i$ is the total amount of time the knock-out race runs.
- The removing quantity per phase vector $r = (r_1, \ldots, r_{p-1}, 0)$. Note that $\sum_{i=1}^{p-1} r_i = k - 1$ is required to let the heuristic terminate with one assignment (the assignment with the lowest penalty).

Note that the p time periods need to be provided, but $p - 1$ removing quantities. Clearly, the choice of the above parameters determine the quality of the final solution. For example, if t_1 is too small and r_1 is high, then it could be some solutions are deleted from the race too quickly. Or to call them informally, solutions with a "slow start" but with good potential, will never reach their potential. For that reason, it is recommended to choose t_1 the largest, such that each solution can get close enough to a local optimum. The choice of r_1 is to the user of the heuristic, but $r_1 \geq \frac{1}{2}k$ is recommended since the knock-out race does not save a lot of time otherwise. See the next page for the pseudo-code for the knock-out race.

Heuristic 1: Knock-out race of the tournament heuristic

Input: $S, T, R, E, C, w, d, u, \Omega = \{\sigma_1, \sigma_2, ..., \sigma_k\}, t = (t_1, \ldots, t_p), r = (r_1, \ldots, r_p)$
set startingTime $\leftarrow$ currentTime

for $i = 1$ to p
    while currentTime $<$ startingTime $+ \sum_{j=1}^{i} t_j$
        for every solution σ_i in Ω
            try to improve σ_i
        end for
    end while

    remove the $\min\{r_i, |\Omega| - 1\}$ solutions with highest penalty from Ω
end for

set $\sigma^* \leftarrow \arg\min_{\sigma_i \in \Omega} \Pi(\sigma_i, C)$
return σ^*

The step "**try** to improve σ_i" is simply one step in the recursive local search, i.e., moving a random event in σ_i to a random possible time slot. If the new penalty is not higher, then keep the adjustment (otherwise, move the event back to its original time slot).

3.4 Practical remarks

Efficiency It is very important to mention that all the mentioned constraints and heuristic can be implemented in various ways. Clearly, some implementations are computationally faster than others and the efficiency of the implementation can have crucial effects on the performance. Most of the work in creating this research paper has been dedicated to a good implementation method, such that the tournament heuristic (particularly, the recursive local search) can be executed as fast as possible.

Practical implementation tricks The current implementation also includes features that assign events to rooms and time slots intelligently. For example, if a room has to be found for an event, it tries to find the smallest possible room. After all, it would be inefficient to place a class of 30 students in a room with a capacity for 200. Otherwise, a problem could arise further in the assignment, when a class of 150 students needs to be scheduled, while only a room with a capacity for 40 is available. Also, the assignment methods firstly check whether there is a possible time slot without clashes. If this does not exist, it tries the find the time slot which creates the minimum number of clashes for the involved student sets.

Chapter 4

Experimental results

This research paper focuses on data which is based on realistic statistics, and considers an instance that could even be more difficult to solve than in practice. Section 4.1 lists the chosen input (general information, events, student sets, teachers, rooms), after which the performance of the tournament heuristic is visualized in Section 4.2.

4.1 Experimental data

To test the effectiveness and practical applicability of the proposed heuristic, it is of vital importance that the datasets (instances) are realistic and must contain every type of problem that is faced in practice. For example, there may not be too much rooms since no overload in rooms would occur otherwise. After all, the room capacity is very often a bottleneck in practice (especially in the "middle" of the day). There is unfortunately no large, complete, directly usable dataset (e.g., from an university) provided for this problem online. This is understandable, since this would require an university to note the preferences and constraints of every room, student set and teacher, which would become an administrative chaos.

Nevertheless, to replicate a realistic problem as good as possible, this research paper bases its experimental data on statistics from the VU University. Some important information (such as the total number of courses) can be found online, and these number provide a basis for this experimental data. There are, however, several matters that are not known; for example, the number of students per course, or unavailabilities of teachers. For this reason, several assumptions will be made regarding the courses (events), rooms, student sets, teachers and constraints, to be explained below.

Moreover, the timetabling instance will be made slightly harder than the statistics from the VU mention, to compensate for constraints that are unknown. In practice, there could be preferences of teachers for specific time slots, and such constraints need to be compensated to have (approximately) the same complexity of the real problem. For example, the VU University contains approximately 24403 students, but this experimental data will consider as if there are 30000 students to make the problem harder. It could even be that the used data is more difficult, but this will probably not be of large influence.

A summary of the used data is shown below, based on the statistics that can be found on the VU website[1] and others.

Input variable	Value
# weeks w	2
# days d	5
# time units per day u	4
# lectures	4500
# student sets	1200
# teachers	1500
# rooms	195

Table 4.1: Used data

The actual dataset includes also extra constraints, such as unavailabilities of teachers, room capacities and sizes of student sets. These characteristics are based on realistic assumptions and samples from the VU University. The complete information regarding the experimental data including the arguments why these are chosen, can be found in Appendix B.

The used constraints and their penalties are necessary to obtain a good understanding of the results, and are therefore shown below. These six types of constraints are explained in 3.2.2 and the penalties are chosen based on personal preference. A small distinction has been made regarding the day load; for students, a day load of 3 time units (5 hours and 15 minutes) is undesirable, but a day load of 4 time units (7 hours) is even more unsatisfactory. Hence, different penalties for different day loads are invoked, and also a distinction between teachers and student sets is made.

Constraint type	Resource	Penalty per violation
Avoid unavailability	Every student set and teacher	10000
Avoid clashes	Every student set and teacher	10000
Spread events of course	Every student set	100
Avoid day load ≥ 2	Every teacher	20
Avoid day load ≥ 3	Every student set	10
Avoid day load ≥ 3	Every teacher	50
Avoid day load ≥ 4	Every student set	25
Avoid idle time unit	Every student set	2
Avoid working day	Every student set	3

Table 4.2: Constraint penalties

In other words, the priority of this experimental data lies in the assigning of lectures (events) such that no unavailabilities or clashes occur. Spreading the events of a course (i.e., not having two lectures of a course on the same day) has the second-highest priority and the remainder should be self-explanatory.

[1] Annual report: `http://www.vu.nl/en/Images/Student-ombudsman-annual-report-2011_tcm12-313060.pdf`

4.2 Performance

Before presenting the quality of the best timetables, it is crucial to observe how fast an initial timetable improves using the recursive local search (without the tournament heuristic), and which initialization methods are better than others. The results of such experiments are shown in the first part of these results, after which that information will be used for further application of the tournament heuristic. All experiments were performed on a personal computer with a 2.67 Core-i5 processor and 4.00 GB of main memory.

4.2.1 Anytime behavior

The anytime behavior illustrates the best objective value (total penalty) over time for a specific evolutionary algorithm, which in this case concerns the recursive local search. For example, if an initial solution was created using the (Random, Random) method, then the local search should be able to reduce the number of clashes and undistributed events significantly. The speed at which these two performance measures are reduced, is visualized below (averaged over 10 runs).

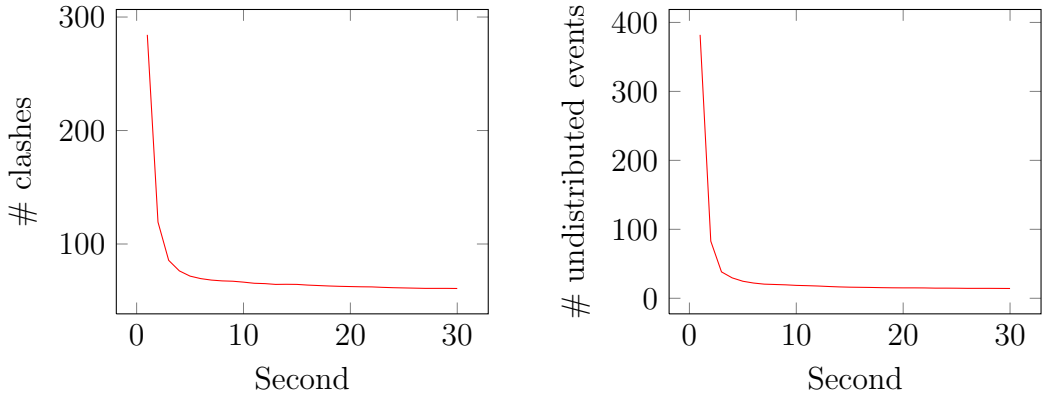

Figure 4.1: Average reduction of the number of clashes and undistributed events using recursive local search

The most interesting result is that more than 90% of the total reduction is already achieved in the first 10 seconds, while the remaining 10% of the improvement is obtained until a local optimum is found (only the first 30 seconds are shown above). Such curves, also known as the anytime behavior, are typical for evolutionary algorithms [10]: an enormous improvement is shown in the beginning, while flattening out later on. The reason for this phenomenon is that improvements are harder to find, when the assignment is already close to its local optimum. For every of the 16 initialization methods, it took at most 120 seconds to reach its local optimum (see Appendix C.1), showing the same anytime behavior as above.

Also, it is worth mentioning that clashes are more difficult to avoid. The (Random, Random) method converges (on average) to 60 clashes and 14 undistributed events. Clashes are harder to minimize since some student sets have too many lectures, but some lectures can be swapped to avoid undistributed lectures.

See below for two different performance measures: the number of days with a load larger than 3 or 4 for only student sets.

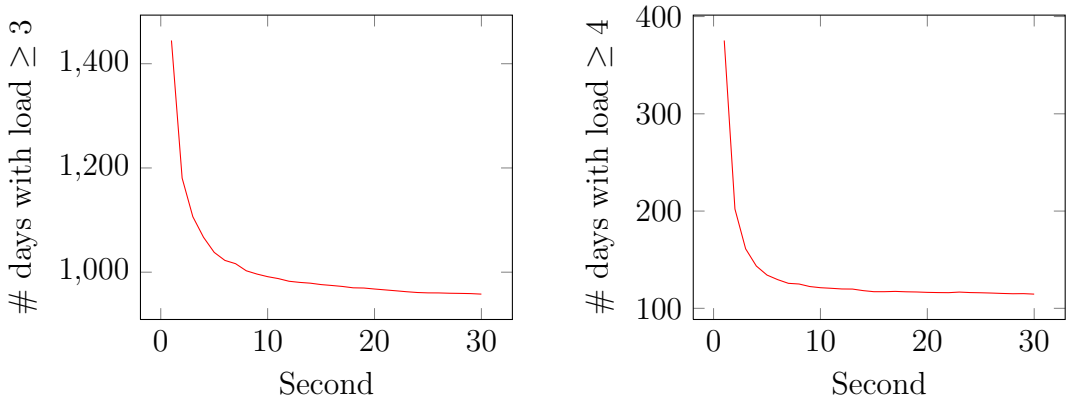

Figure 4.2: Average reduction of high day loads using recursive local search

These performance measures show the same anytime behavior and should be self-explanatory. One point of interest is that the curve is slightly less sharp (decreases slower) than the curve of the first two hard constraints. This is due to the fact that the first two constraints have a higher penalty per violation. In other words, if a clash can be prevented by increasing the day load for student sets, then the local search does so. This insight is more recognizable for the number of working days.

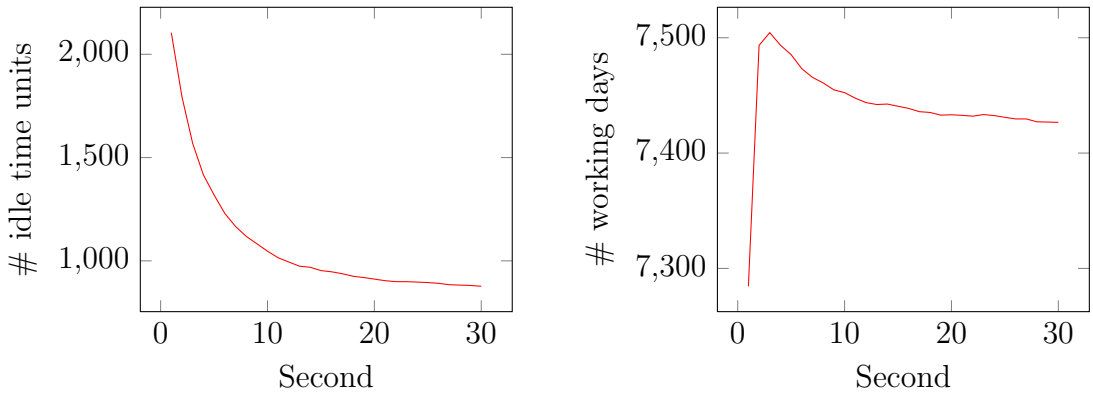

Figure 4.3: Average reduction of idle time units and working days using recursive local search

Recall that the reduction of the number of working days for student had a very low penalty. If it is possible to prevent a day load of 3 or 4 by adding an extra working day, then the local search does so. This is exactly what can be observed well in the plot of the number of working days. Since the prevention of high day loads has higher priority, this is optimized at the expense of the number of working days. After the most important performance measures reach its local optimum, then some moves can be done to improve also on the minor objective functions, which explains the anytime behavior of the number of working days. The number of idle time units is less affected, since it depends on fewer other constraint types.

4.2.2 Comparison of initialization methods

To apply the tournament heuristic, it is essential to know how the participants should be selected, i.e., which method(s) should be chosen to create initial solutions. For this reason, 10 runs for each of the 16 initialization methods (total 160 runs) are performed to see which methods prove to be worth investigating or are doomed to fail. Below is an overview of the best and average total penalty $\Pi(\sigma, C)$ for each method (in thousands and rounded, for readability). Also the standard deviation are shown, for which the reason is clarified below.

	Random	Fixed	Lowest penalty	Lowest tight.
Random	Best: 522 Mean: 653 St.dev.: 90	Best: 421 Mean: 531 St.dev.: 62	Best: 450 Mean: 574 St.dev.: 81	Best: 361 Mean: 438 St.dev.: 21
Simple tight.	Best: 462 Mean: 534 St.dev.: 60	Best: 419 Mean: 387 St.dev.: 38	Best: 419 Mean: 426 St.dev.: 8	Best: 361 Mean: 399 St.dev.: 21
Dynamic tight.	**Best: 271** **Mean: 298** **St.dev.: 25**	**Best: 241** **Mean: 246** **St.dev.: 5**	Best: 277 Mean: 278 St.dev.: 0	Best: 251 Mean: 269 St.dev.: 9
Size-based	Best: 491 Mean: 573 St.dev.: 53	Best: 481 Mean: 539 St.dev.: 38	Best: 601 Mean: 625 St.dev.: 16	Best: 343 Mean: 363 St.dev.: 13

Table 4.3: Local optima (penalties) per initialization method in thousands

From this table, it can be very well argued that the two bold-printed initialization methods are the better than the others. The (DT, Fixed)[2] method has the lowest mean thus performs best on average, which is a clear reason to declare this as a good initialization method. The most likely reason for this good performance is the fact that this method creates a good buffer on specific time slots. If the assignment method always places events on a fixed order (firstly time unit 1, then time unit 2, etc.), then the last time unit (say, time unit $d \cdot u$) is as empty as possible. Therefore, clashing events can be moved to these time slots during local search, to avoid clashes (recall that the avoidance of clashes had the highest priority).

However, arguably the most interesting and important observation is that the (DT, Random) method performs not extremely well on average (a penalty of 298, which is worse than the mean of 3 other methods), but produces good outliers. The reason for this is because randomization creates diversity, i.e., more differences between the initial solutions. The local optima that are found are then more spread over the sample space of all solutions. This is in contrary to the (DT, Fixed) method, where the initial solutions are identical to each other (the randomization in the local search causes differences in the found local optima). Also, the fact that a dynamic tightness-based ordering is used, ensures that the local optima are of good quality.

[2]DT = Dynamic tightness ordering method.

For these reasons, one could argue that the (DT, Fixed) method is most appropriate when a good timetable needs to be found in a short amount of time. After all, this method has the lowest mean and a small standard deviation, implying that this method yields the most certainty for a good timetable immediately. However, if there is available amount of time is large, then one could also consider the (DT, Random) method, hoping on an extremely good outlier. Due to these arguments, only the mentioned two initialization methods are considered from this point. But to give away a glimpse of the conclusion: there is one other initialization method which could be more interesting for other universities, but this is elaborated on further at the final comparison.

Whether the standard deviation for the (DT, Random) method is high enough to actually produce better timetables than the (DT, Fixed) initialization method could do, depends on the amount of time. A very interesting question would be how much attempts statistically are required for the (DT, Random) method to produce better solutions than the (DT, Fixed) method would do in the same amount of time. To make a plausible statement concerning this issue, a histogram is firstly provided concerning the penalty distribution. Based on 60 runs for the two selected methods, 50 runs on top of the runs that already were available, the penalty distribution is as follows.

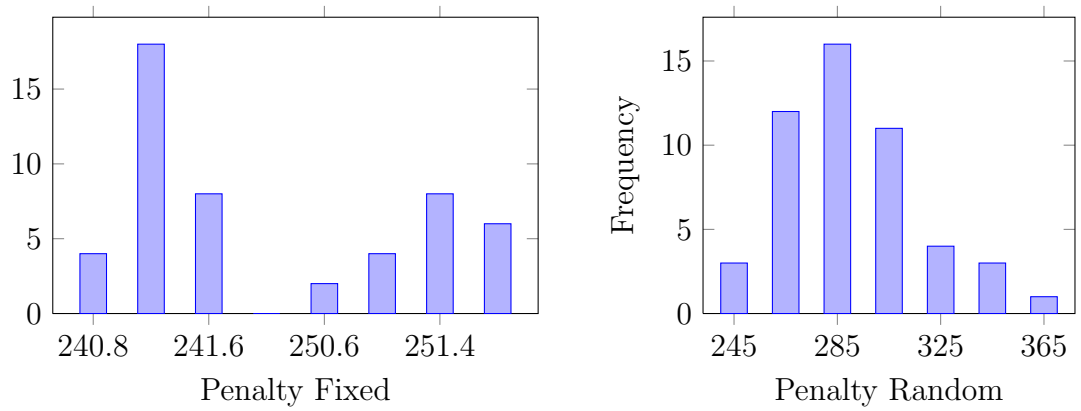

Figure 4.4: Penalty (in thousands) distribution of the best two initialization methods

The penalty of (DT, Random) has a mean penalty of 292 and a standard deviation of 27. The data is normally distributed, which has been verified using the Shapiro-Wilk test [21]. The penalty of (DT, Fixed) has mean 245 and standard deviation 4. The histogram shows fewer characteristics of a normal distribution, but there are good reasons to believe why the penalty theoretically should be normally distributed (see Appendix C.2 for these insights).

But the important observation concerns the large standard deviation of the penalty for an optimized (DT, Random) solution. This suggests that this method delivers more outliers and potentially could deliver a better solution. Clearly, the solution with the lowest penalty is searched, but this could take too much attempts and consume a lot of time. This is exactly the reason why the knock-out race is included in the tournament heuristic, since this method is expected to find the best solution out of many solutions quickly.

4.2.3 Performance of the tournament heuristic

As last, the final component of the tournament heuristic will be analyzed, the knock-out race. It is required to have k initial solutions out of which the most promising solution should be found. Even though it is not necessary, two knock-out races have been considered: one with $k = 1000$ (DT, Random) solutions as participants, while the other starts with the same amount of (DT, Fixed) solutions. This is done to compare the best solution of both initialization methods. Recall that based on 10 runs, the (DT, Fixed) method produced the best instance so far (see Table 4.3).

As mentioned in Section 4.2.1, more than 90% of the possible improvements are already achieved in the first 5 to 10 seconds. For that reason, the time that is dedicated to the first phase of the knock-out race $t_1 = 10 \cdot k$, meaning that every solution will at least be optimized for at least 90%. Usually, more than 95% of the improvements are achieved within 10 seconds. After the first round, it is time saving to remove a large percentage of solutions with a relatively bad quality. Therefore, 900 of the 1000 will be removed, whereupon an extra 50 seconds is dedicated to the remaining 100 solutions. All promising solutions have then almost reached their full potential. After that, only the best solution will be optimized entirely further to its local optimum. To summarize:

- The number of initial solutions $k = 1000$
- The number of knock-out phases $p = 3$
- The removing quantity per phase $r = (900, 99, 0)$.
- The time vector $t = (1000 \cdot 10, 1000 \cdot 50, 1 \cdot 100) = (10000, 5000, 100)$. Note that the total duration equals $10000 + 5000 + 100 = 15100$ seconds ≈ 4.2 hours.

This implies that only one solution is left after 2 rounds, which is optimized for exactly 160 seconds using the recursive local search.

Note already how much time the knock-out race saves. Normally, it takes between 40 and 100 seconds to reach the local optimum (see Appendix C.1 for a distribution of the running time that justify this statement). Instead of optimizing 1000 solutions to their local optimum, which would take on average 70000 seconds, only 15100 seconds are used. The required time could even be more reduced when $t_1 = 5 \cdot k$ is found to be enough as well. Of course, there is a higher risk that the best potential solution is knocked out too early, but the risk remains small since the largest improvements are made in the start.

These two tournaments both have produced a winning solution, further to be called "Winner Random 1000" and "Winner Fixed 1000". These are declared as the best solutions that can be found for the two initialization methods based on 1000 initial solutions. Also, the best solution for each initialization method is shown when only 10 solutions are optimized, i.e., the best solutions that are also listed in Table 4.3. These solutions are further referred to as "Winner Random 10" and "Winner Fixed 10". The table on the next page compares the performance of these solutions by listing all performance measures of interest, including the average penalty for all other initialization methods (also shown in Table 4.3).

Init. method	Clash	Undist.	DL ≥ 3	DL ≥ 4	Idle	Work	Pen.
Winner Random 1000	**18**	17	937	94	847	7438	**233**
Winner Fixed 1000	19	12	956	90	609	7451	241
Winner Random 10	22	16	932	90	845	7455	271
Winner Fixed 10	19	15	945	88	603	7440	241
Random, Random	60	13.1	949	113	840	7419	653
Random, Fixed	48	14.6	943	108	862	7422	531
Random, Lowest pen.	52	8.4	951	110	819	7409	574
Random, Tightness	39	12.7	940	103	745	7446	438
ST, Random	48	19	953	106	849	7400	534
ST, Fixed	34	10.3	935	104	809	7435	387
ST, Lowest pen.	38	3.1	932	93	656	7414	426
ST, Tightness	35	13.1	944	102	711	7424	399
DT, Random	25	15.7	951	97	835	7423	298
DT, Fixed	20	14.9	949	91	604	7431	246
DT, Lowest pen.	23	**3**	**868**	**76**	**560**	7454	278
DT, Tightness	22	16.4	954	104	665	7412	269
Size, Random	52	13.7	955	121	817	7416	573
Size, Fixed	49	7.9	942	116	704	7437	539
Size, Lowest pen.	58	6.8	935	121	598	**7371**	625
Size, Tightness	31	16	967	108	666	7421	363

Table 4.4: Final performance comparison

The bold-printed values are the best found values for the specific performance measure. Three important observations can be made based on the above table, that each provide possibly new, but important insights for timetabling in practice.

The first observation concerns the fact that the (DT, Fixed) method produced a better solution than the (DT, Random) method when only 10 attempts could be done. However, using 1000 attempts, the (DT, Random) method is able to produce a slightly better solution than the (DT, Fixed) method. The reason for this phenomenon is that randomization causes diversification in the "participating" solutions, meaning that more diverse local optima are found. Many of these local optima are bad, but with some luck, the local optimum is better than would have been possible with the (DT, Fixed) method. As also can be seen in the above table, the best (DT, Fixed) solution out of 1000 runs is almost just as good as the best solutions out of 10 runs[3]. The drawback is that, as mentioned, a certain degree of luck is required when the (DT, Random) method is applied, which fortunately occurred in the above results. Many attempts should be done to be fortunate enough in finding such an extremely good local optimum, which could take (too) much time. Yet, this is exactly the reason why the tournament heuristic includes a knock-out race, since this technique can save time in finding an extremely good local optimum.

[3]As a small remark, the "Winner Fixed 1000" has a slightly better penalty than "Winner Fixed 10", 240722 vs. 241226, but is due to rounding not visible in the above table.

Furthermore, the table shows that the (DT, LP) method[4] scores best on four of the seven performance measures, but this method did not belong to the two investigated methods. Recall that the number of clashes had the highest priority; every violation increases the total penalty with 10000. The (DT, LP) method produces at least four more clashes, making the penalty much higher than the two methods that participated in the knock-out race.

The point is that (perhaps ethical) questions could arise regarding the selected weights, when comparing the performance measures of the (DT, LP) method to the "Winner Random 1000" solution. On average, the (DT, LP) method produces 5 more clashes, but prevents 14 more undistributed events, 69 high day loads, 17 extremely high day loads, and 287 idle time units for the student sets. One could argue that it is more fair that 5 student sets have a clash extra, such that many other students have fewer high day loads and idle time units. Clearly, the heuristic considers the "Winner Random 1000" solution currently as superior due to the large weight on the "avoid clashes"-constraint. But when this constraint would have a lower weight, then it would be most likely that the (DT, LP) method would have been superior. In other words, determining the weights on the constraints must be done with great care, because less desirable timetables could be produced otherwise.

The last question of interest is whether the heuristic also produces better timetables than in practice. After all, the ambition of this research paper is to improve the currently available timetabling software. To make a good comparison, it is necessary to have an intuition on how timetables are constructed in practice. In many schools and universities, timetables are constructed manually by placing the events one by one on the best slot and improve the timetable afterwards. This suggests a "lowest penalty" assignment method in practice, which also has been suggested in 3.3.1. Improving the timetable afterwards, suggests actually exactly the recursive local search algorithm (trying to move an event from one slot to another, and see whether improvements are made). A fair assumption is then that the scheduler performs the local search just as good as a computer would do; humans normally think more (smart) moves ahead, but a computer consistently performs much more moves in a short amount of time. Furthermore, on large educational institutes, the events are probably ordered randomly because it is too difficult and time-consuming to order thousands of events on tightness (especially not dynamically). Also, many schedulers schedule according the the first-come, first-served principle: the teachers that send their preferences first, are scheduled first. Thus basically, the events that need to be scheduled are randomly ordered.

This suggests a (Random, LP) initialization in practice, with a similar local search method afterwards. If this is compared to the (DT, LP) method, one can observe that many improvements are possible. The number of clashes, undistributed events, extreme high day loads and idle hours could be reduced by respectively 63%, 64%, 30.9% and 31.6%, which is a significant improvement. The practical method can also be compared to the actual winner "Winner Random 1000", which improves the most important performance measure slightly better with a reduction of 65%.

[4]LP = Lowest penalty. Recall that the (DT, LP) method orders the events dynamically on tightness, and assigns the events to the slot causing the lowest increase in penalty

However, this specific solution has more undistributed events and idle time units, while the reduction in extreme high day loads is also not extreme. In practice, it could therefore be more convincing to present the results of the (DT, LP) method that, roughly speaking, reduces the 4 most important performance measures with 30% to 70%.

Under these realistic assumptions, one can argue that the tournament heuristic indeed produces better timetables than in practice. Furthermore, an extensive analysis of initialization techniques has not been considered before in the current literature (especially not in combination with optimization heuristics), while this proves to be of the largest influence. The knock-out race is a component of timetabling optimization that could not be found in the current research, while this can save an enormous amount of time under a reasonable assumption that no optimality has to be lost.

Chapter 5

Conclusions and future work

5.1 Conclusions

This research paper has investigated the theory and practice of school timetabling which has lead in both aspects in very interesting results.

The theory of timetabling concerns many problems that are based on the Class-Teacher problem, where a set of lessons need to be assigned to time slots, without clashing lectures. This can be solved easily using a bipartite graph representation. Easy extensions of the problem include a upper bound on the number of rooms, invoking a daily maximum of lessons per day and balancing the work loads. However, most of the timetabling problems are strongly NP-hard. For example, when classes can have the freedom to choose lectures, when exams can be merged in rooms or when teachers are unavailable. An own proof of strongly NP-hardness has been provided for a variant in which the lectures on a specific time unit needs to be maximized, via a reduction from Maximum Independent Set.

The practical part presents a new heuristic that can be applied in any school or university: the tournament heuristic. This heuristic consists of an extensive initialization, a local search method and a "survival of the fittest" mechanism. The local search method is standard, yet very effective. The exact survival of the fittest mechanism has not been proposed before (even though variants could exist), but proves specifically for timetabling to be very time saving. However, the results have shown that the biggest improvements can be obtained from a good initialization.

It turned out that the best initialization method should order the events (lectures) dynamically on tightness (see 3.3.1). Based on this order, one could assign the events according to a fixed order, randomly or on the time slot that achieves the lowest penalty. Each of these methods have their own advantages and disadvantages and the choice of the most appropriate method depends on the available amount of time, but mainly on the preferences of the educational institute. Compared to timetables in practice, it can be very well argued that the heuristic realizes an impressive reduction between 30% and 70% concerning the number of clashes, high workloads, undistributed lectures over the week and number of idle time units. Under several realistic assumptions, one can therefore conclude that the heuristic indeed produces better timetables than in practice.

5.2 Future work

Even though the achieved results are very satisfactory, future work is always possible. The first point concerns the experimental data, since only 1 timetabling instance has been considered so far. The main reason for this is that already has been explained in the final comparison when one method is better than another, and that the tournament heuristic will always discover this if enough attention is paid to the qualification phase. However, it could be interesting to find out which initialization methods are more suitable for specific instances. For example, it could well be that a complete random initialization method could perform best given a reasonable amount of time, when a timetabling instance for example easily suffices all hard constraints anyway.

Another interesting addition to the recursive local search could be an efficient, effective method to reduce the number of clashes. Currently, the heuristic "thinks" one move ahead. However, if 10 different student sets want to attend a specific lecture, then one move ahead is not enough. What actually should be done, is an attempt to make sure that all these 10 different student sets have no lecture on a specific time unit, say Tuesday 11:00. When this is done, the lecture can be scheduled at that moment of time without any clash, but could require thinking 10 steps ahead, which could explode in computation time. It would be interesting to investigate whether one can determine fast for which time units this is possible.

Appendix A

Strongly NP-hard/complete problems

Throughout the research paper, several strongly NP-hard problems are used to prove hardness of miscellaneous variants of the timetabling problems. The formal definitions of the used problems are listed below in alphabetical order, but can also be found in [12] or [17].

3-Satisfiability

Given: A set of Boolean variables $X = x_1, x_2, ..., x_n$, a set of clauses $C = c_1, c_2, ..., c_m$ where each clause is a disjunction of exactly 3 variables of X and a Boolean formula $F = c_1 \wedge c_2 \wedge ... \wedge c_m$.

Goal: Determine whether there exists a truth assignment to $x_1, ..., x_n$ s.t. $F =$ True.

Bin packing

Given: A bin capacity V and a list of values $A = \{a_1, ..., a_n\}$ where $a_i \in [0, V]$

Goal: Minimize the sublists s.t. each sublist sums to at most V and the sublists form a partition of A.

Graph K-Colorability

Given: A graph $G = (V, E)$.

Goal: Minimize the number of colors s.t. all vertices are assigned a color and $Color(u) \neq Color(v)$ for every $\{u, v\} \in E$.

Maximum independent set

Given: A graph $G = (V, E)$.

Goal: Maximize the size of an independent set I, i.e. a set of vertices $I \subseteq V$ s.t. $\{u, v\} \notin E$ for every $\{u, v\} \in I$.

Appendix B

Complete experimental data

B.1 General

The VU University has structured the school year in six periods, of which period 1 and 2 comprise respectively September-October and November-December. Some courses are taught in both the first and second period (i.e., there are dependencies between the first and second period), but there are no dependencies regarding other periods. Therefore, the timetables for period 1 and 2 are made separately from the timetable in other periods, and this is also the timespan that is tested in this research paper. In these two periods, approximately 40% of the courses are taught.

The assumption is that the timetable per period stays the same, meaning that only a timetable for two weeks (where one week is used the entire period) needs to be made. In practice, this is not the case, as some lectures are only given once (or excursions are only in a specific week). Optimizing on this feature can easily be implemented, but is not considered in this specific dataset for simplicity.

Furthermore, the VU University teaches five days per week (Monday until Friday) and has specific slots at which lectures are taught: from 09:00 until 10:45, 11:00 until 12:45, 13:30 until 15:15 and 15:30 until 17:15. For the sake of simplicity, these time units are referred to as "time unit 1" until "time unit 4". Actually, some lectures also occur in the evening (e.g., when there is not enough room space), but this will be not included in this dataset to prevent misleading results. To summarize: $w = 2$, $d = 5$ and $u = 4$.

B.2 Courses and events

In the year 2012-2013, students from the VU University were able to enroll for 4492 courses[1]. As mentioned, only the first two periods (40%) are considered on the VU, meaning that approximately 1800 courses are taught between September and December. The actual number of courses that should be considered is actually even less, since a substantial amount of these courses do not require any lecture (e.g. a thesis for a specific Bachelor's program). To make the problem harder

[1] Obtained from the student's portal, VUnet.

(to compensate for unknown information), the experimental data comprises 2000 courses, each having a different amount of lectures per week. These amounts are listed below and are based on a sample from 40 random courses on the VU.

Lecture amount	Number of courses
1	200
2	1200
3	500
4	100
	2000

Table B.1: Lecture amounts per course

Of these 2000 courses, 900 are taught in only the first period, 900 are taught in only the second period while the remaining 200 are taught in both periods. These numbers are based on the same random sample of 40 courses. Note that the total number of lectures (events) is equal to $200 \cdot 1 + 1200 \cdot 2 + 500 \cdot 3 + 100 \cdot 4 = 4500$.

B.3 Student sets

In total, 1200 student sets are considered, where each student set has a size (number of students) of 10, 20, 30 or 40. There are 300 student sets with any of the four different student set sizes, thus $30 \cdot 10 + 30 \cdot 20 + 30 \cdot 30 + 30 \cdot 40 = 30000$ students are involved in the instance. This is (as mentioned before) more than the 24403 students that the VU actually has, but it makes the problem harder for the previously mentioned reason. The large amount of student has influence on the room constraints, since it will be harder to find rooms with enough capacity for the students. The number of students per course are given in the table below.

Number of students	Number of courses
300	100
200	150
100	200
50	300
20	1000
10	250
	2000

Table B.2: Number of students per course

To clarify: the above table implies that there are 100 courses that have 300 students. These 300 students are composed of a random collection of the 1200 student sets of which the total sum of student sets sums to 300. Each student set has at most five courses per period (thus at most ten in total).

B.4 Teachers

Recall that the data consists of 2000 courses, where each course is coordinated by exactly one teacher. Based on a random sample of 20 teachers of the VU, it turned out that active teachers teach on average 1.25 courses per two periods. This has been slightly increased to 1.33 courses, meaning that there are 1500 teachers that teach 2000 courses. See below for their availabilities.

Number of available days	Number of teachers
5	1000
4	100
3	50
2	50
1	300
	1500

Table B.3: Availabilities per teacher

Recall also that some courses had multiple lectures per week. If a course had e.g., three lectures, then the course is not assigned to a teacher with only one or two available days. Otherwise, the constraint "spread events of course" can never be fulfilled.

B.5 Rooms

At January 2012, the VU had 196 lecture rooms available and approximately 34 computer rooms[2]. The computer rooms are removed from the set to make the problem harder. As mentioned earlier, even though the heuristic can take different room types into account, this feature is disregarded to make the results more interpretable. The capacity of the rooms including the number of rooms with that capacity is listed below and is based on the room sizes of the VU (with some small "roundings" such that only 5 different capacities need to be considered).

Capacity	Number of rooms
300	5
200	10
100	20
50	50
25	100
	195

Table B.4: Number of rooms and their capacities

Note that only 5 rooms have enough capacity for the 100 courses with 300 students.

[2]This information is publicly obtainable via the VU's website, `http://www.vu.nl/`.

Appendix C

Extra performance measures of results

C.1 Running time of best initialization methods

For completeness, the running time distribution of the best two initialization methods are provided below.

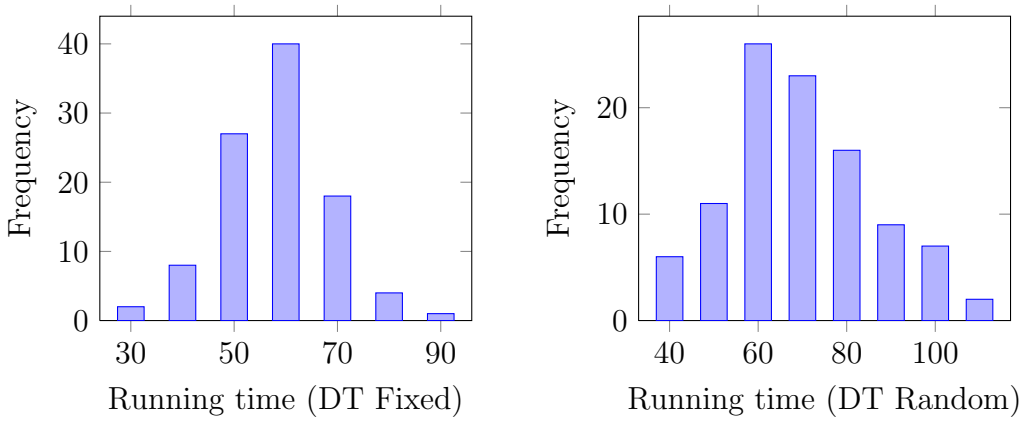

Figure C.1: Running time distribution of the best two initialization methods

The running time of (DT, Random) has mean 73,5 seconds and standard deviation 15 seconds. The running time of (DT, Fixed) has mean 64,8 seconds and standard deviation 12.8 m. The reason why the running time of (DT, Random) is higher, is due to the fact that more improvements are possible. After all, an initial solution in which randomization is used should have more room for improvement than an initial solution that already is close to a local optimum. The initial solutions produced by (DT, Fixed) on the other hand, do not include any randomization and are created with the use of smart rules and are therefore closer to specific local optimum. These graphs are shown merely to mention this observation, and to show that finding a local optimum is done fast. A local optimum is found when no improvements can be found anymore for 3 seconds. For completeness, the data is normally distributed, which has been confirmed using the Shapiro-Wilk test [21].

C.2 Dynamic tightness, Fixed

See below for the distribution of extra performance measures of the (Dynamic tightness, Fixed) initialization method after local search.

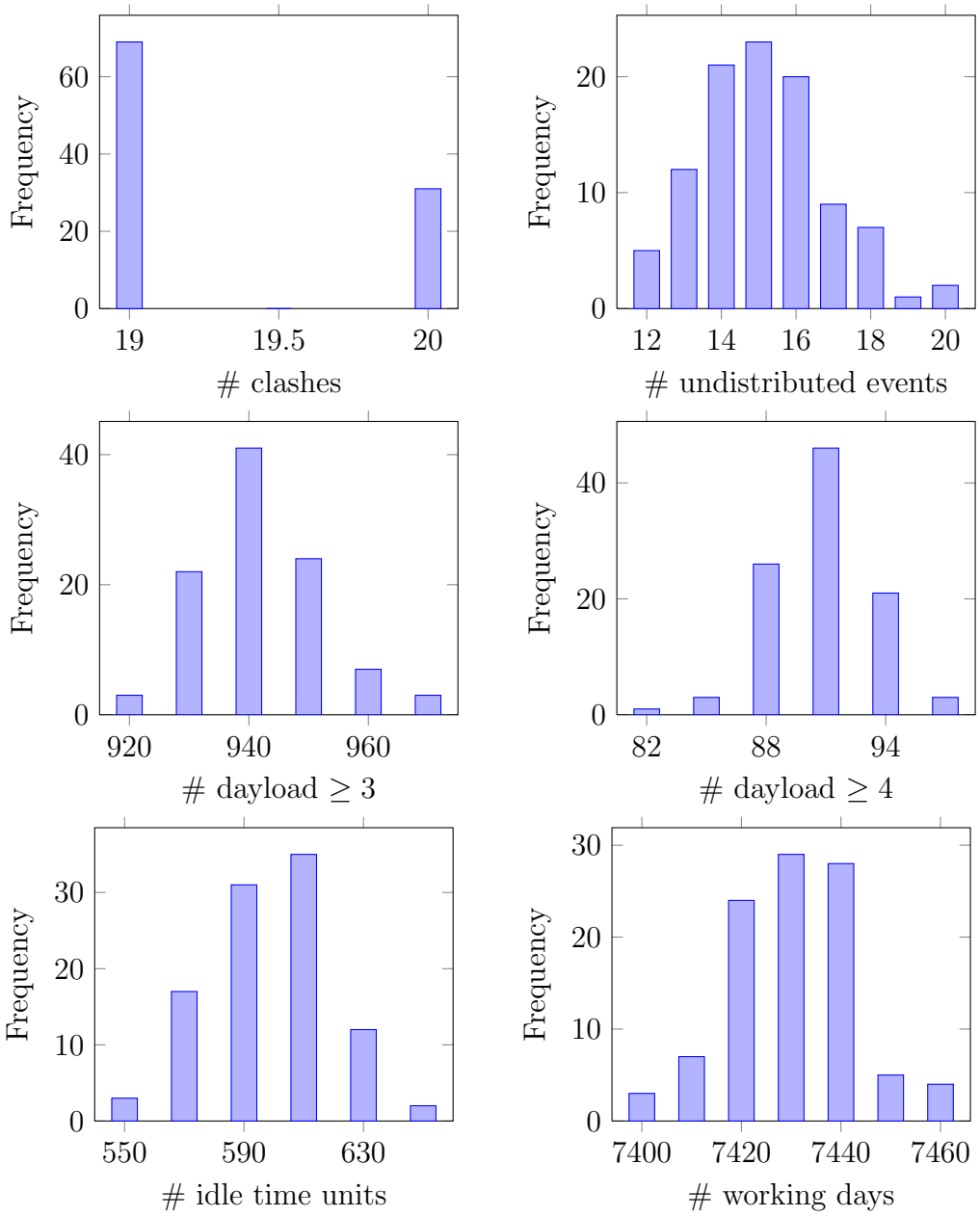

Figure C.2: Performance measure distribution for the (Dynamic tightness, Fixed) initialization method

All samples have been tested on normality using the Shapiro-Wilk test [21], which confirmed that these indeed are normally distributed, apart from the clashes. Apparently, the (Dynamic tightness, Fixed) method is close to a local optimum where the number of clashes can either only be 19 or 20, but since the penalty is a weighted sum of the above data, it is assumable that the penalty is also normally distributed.

Bibliography

[1] D. Abramson, H. Krishnamoorthy & H. Dang, *Simulated annealing cooling schedules for the school timetabling problem*, Asia-Pacific Journal of Operational Research, 1996.

[2] N. Balakrishnan, *Examination scheduling: a computerized application*, Omega, 1991.

[3] J. A. Bondy & U. S. R. Murty, *Graph Theory with Applications*, American Elsevier Publishing Company Inc., 1976.

[4] E. Burke, Y. Bykov, J. P. Newall & S. Petrovic, *A time-defined approach to course timetabling*, Yugoslav Journal of Operations Research, 2003.

[5] M. Carrasco & M. Pato, *Metaheuristics: computer decision-making*, E. Burke and W. Erben (editors), Practice and Theory of Automated Timetabling III, 2001.

[6] S. Casey & J. Thompson, *GRASPing the examination scheduling problem*, E. Burke and W. Erben (editors), Practice and Theory of Automated Timetabling IV, 2002.

[7] A. Colorni, M. Dorigo and V. Maniezzo, *Metaheuristics for high-school timetabling*, Computational Optimization and Applications, 1997.

[8] T. B. Cooper & J. H. Kingston, *The Complexity of Timetable Construction Problems*, Proceedings of the first International Conference on Practice and Theory of Automated Timetabling, 1995.

[9] M. A. H. Demptser, D. G. Lethridge & M. A. Ulph, *School timetabling by Computer - A technical History*, Educational Research, 1973.

[10] A. E. Eiben & J. E. Smith, *Introduction to Evolutionary Computing*, Springer, 2007.

[11] S. Even, A. Itai & A. Shamir, *On the Complexity of Timetable and Multicommodity Flow Problems*, SIAM Journal on Computing, 1976.

[12] M. R. Garey & D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H. Freeman & Company, 1979.

[13] K. Socha, J. Knowles & M. Samples, *A max-min ant system for the university course timetabling problem*, M. Dorigo, G. Di Caro and M. Samples (editors) Proceedings of Ants 2002 - Third International Workshop on Ant Algorithms, 2002.

[14] D. König, *Über graphen und ihre anewendung auf determinantentheorie und mengenlehre*, Mathematische Annalen, 1916.

[15] R. Lewis, *A Survey of Metaheuristic-based Techniques for University Timetabling Problems*, OR Spectrum, 2007.

[16] N. K. Mehta, *The application of a graph coloring method to an examination scheduling problem*, Interfaces, 1981.

[17] C. H. Papadimitriou & K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*, Dover Publications, 19898.

[18] A. Schaerf, *Tabu search techniques for large high-school timetabling problems*, Proceedings of the Thirteenth National Conference on Articial Intelligence, 1996.

[19] A. Schaerf, *A Survey of Automated Timetabling*, Artificial Intelligence Review, 1999.

[20] A. Schaerf, *Local Search Techniques for Large High-School Timetabling Problems*, IEEE Transactions on Systems, Man, and Cybernetics, 1999.

[21] S. S. Shapiro & M. B. Wilk, An analysis of variance test for normality (complete samples), Biometrika, 1965.

[22] S. M. Selim, *An algorithm for producing course and lecture timetables*, Computers & Education, 1983.

[23] J. M. Thompson and K. A. Dowsland, *A robust simulated annealing based examination timetabling system*, Computers and Operations Research, 1998.

[24] D. de Werra, *An introduction to timetabling*, European Journal of Operations Research, 1985.

[25] E. Yu & K-S. Sung, *A genetic algorithm for a university weekly courses timetabling problem*, International Transactions in Operational Research, 2002.