universität wien

MASTERARBEIT

Titel der Masterarbeit

Adaptive Large Neighborhood Search for the Curriculum-Based Course Timetabling Problem

Verfasser

Alexander Kiefer, Bakk.rer.soc.oec.

angestrebter akademischer Grad

Master of Science (MSc)

Wien, 2013

Studienkennzahl lt. Studienblatt:	A 066 915
Studienrichtung lt. Studienblatt:	Masterstudium Betriebswirtschaft
Betreuer:	O.Univ.Prof. Dipl-Ing. Dr. Richard F. Hartl

Acknowledgement

I would like to thank the people at the chair of Production and Operations Management of the University of Vienna for their support. In particular I want to thank my supervisor O.Univ.Prof. Dipl.-Ing. Dr. Richard F. Hartl for giving me the opportunity to write the thesis at his chair, Alexander Schnell, MSc for his advices and his critical review of the thesis and Mag. Attila Kovacs for his suggestions regarding the improvement of the algorithm's performance. Moreover I want to thank Prof. Andrea Schaerf from the University of Udine for giving me insights about the problem structure. Last but not least I would like to thank my family and friends.

Contents

List of Figures

List of Tables

List of Algorithms

1 Introduction

Timetabling problems are eminently relevant in practice. Petrovic and Burke [2004] state that these problems can be found in various fields, including educational timetabling, nurse rostering, timetabling of public transport systems and timetabling of sport events. In the context of educational timetabling, Schaerf [1999a] defines the problem as scheduling lectures that involve teachers and students in a prefixed period of time, while taking different constraints into account.

Schaerf [1999a] subdivides educational timetabling into *school timetabling*, *examination timetabling* and *course timetabling*. Basically, school timetabling aims to find a weekly timetable for classes, such that no teacher and no class has two lectures at the same time. In examination timetabling and course timetabling overlaps between events with common students should be avoided. A typical objective of exam timetabling is a good distribution of exams for the students over time. The considered time period of course timetabling might be a week, while for exam timetabling it depends on the examination session. Occasionally, minimizing the length of the examination session is part of the objective, as it is the case for some tests by Carter et al. [1996].

A very basic problem of finding a feasible school timetable is considered by Even et al. [1976]. They prove its NP-completeness when teachers and classes might be unavailable at some periods by a polynomial reduction of the 3-SAT problem. Moreover, the authors present a polynomial time algorithm for the special case where every teacher has only two available periods. They state that without unavailabilities a feasible solution always exists and can be found in polynomial time.

Cooper and Kingston [1996] show the NP-completeness of educational timetabling problems in five different ways. For the practically relevant case where teachers have a fixed workload due to their labor contract and the number of lessons varies between subjects, the NP-completeness is shown by a reduction from the bin packing problem. Finding feasible course and exam timetables is NP-complete due to a reduction of graph coloring. The authors note that this connection only requires that the students are free to choose which courses or exams, respectively, they want to attend. These choices result in conflicts between courses, such that they must not be scheduled at the same time to ensure that students are able to attend all of their choices. Possible teacher or room constraints are not essential for the NP-completeness. The connection between graph coloring and timetabling has already been studied before, e.g. by Welsh and Powell [1967].

The complexity results suggest that solving large educational timetabling problems by exact approaches might be impossible in reasonable time. Therefore a metaheuristic approach is implemented in the context of this theses. In particular an *Adaptive Large Neighborhood Search* (ALNS) approach is used. ALNS proposed by Ropke and Pisinger [2006] and Pisinger and Ropke [2007] has been originally used for tackling vehicle routing problems and is an extension of *Large Neighborhood Search* by Shaw [1998]. The algorithm aims to iteratively improve the solution by repetitive destruction and reparation of relatively large fractions of the timetable.

Two variants of course timetabling have been formulated for the *second international timetabling competition* (ITC-2007)[1], including *Post Enrollment Course Timetabling* (PE-CTT) and *Curriculum-based Course Timetabling* (CB-CTT). A detailed description of these problems is given in the technical reports by McCollum et al. [2007a] and Di Gaspero et al. [2007], respectively.

The input for PE-CTT includes data of the students' enrollments to courses. The conflicts between courses are based on the enrollment data. On the contrary for CB-CTT curricula are known in advance and courses have to be scheduled, such that for each period at most one course per curriculum takes place. Other hard constraints that have to be satisfied to achieve a feasible solution require that all lectures are scheduled, the assignment of at most one lecture to a room at the same time and further version-dependent features.

The solution quality is measured in terms of violations of soft constraints representing favorable characteristics of the timetable, e.g. rooms should meet the capacity requirements of the assigned lectures and the schedule of courses of the same curriculum should be compact such that students do not have to wait between lectures. The theses has its focus on the CB-CTT. In particular, the algorithm is tested on the benchmark instances for the CB-CTT track of the ITC-2007.

[1]`http://www.cs.qub.ac.uk/itc2007/`

The outline of this theses is as follows. Section 2 gives a literature review with a focus on algorithms that have been implemented for the ITC-2007 and more recent approaches that have been proposed for CB-CTT and benchmarked on the ITC-2007 instances. Furthermore graph coloring approaches and general concepts for metaheuristics will be described, as subsequent sections refer to these methods. In section 3 the content of the three international timetabling competitions is sketched. Additionally, the ITC-2007 formulation of CB-CTT and its benchmark instances are described in detail. Also some extensions to the problem are mentioned. A mathematical model based on the integer programming (IP) formulations of Lach and Lübbecke [2012] and Burke et al. [2011] is presented in section 4. The *mixed integer programming* (MIP) solver CPLEX is used for solving the ITC-2007 instances to test the effectiveness of the exact approach. Furthermore, it is explained how extensions can be incorporated into the IP formulation. Section 5 describes a solution approach for CB-CTT based on *Adaptive Large Neighborhood Search* (ALNS) by Ropke and Pisinger [2006]. Computational results for the ITC-2007 benchmark instances are presented in section 6. A conclusion is given in section 7.

2 Literature Review

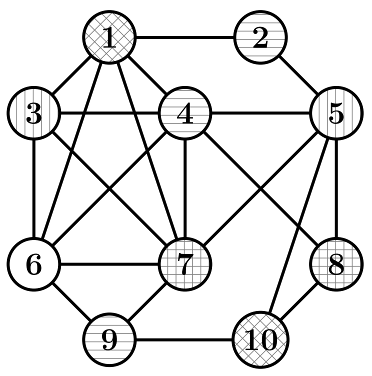
Schaerf [1999a] surveys *automated timetabling* in general, i.e. approaches without human intervention. Besides stating basic timetabling models, practically relevant variants and solution approaches, the author sketches the history of solution techniques for timetabling problems. Early approaches aim to extend a partial timetable by scheduling one lecture after another until a complete timetable is found. These methods typically prioritize lectures that are very constrained, with regard to a particular measure. They basically imitate human manual scheduling. Thus Schaerf [1999a] calls them *direct heuristics*. Later approaches are based on *integer programming*, *network flow* techniques and *graph coloring* methods. Even more recently *constraint satisfaction* and *metaheuristic* approaches are applied. Some new techniques hybridize several metaheuristic concepts, e.g. Müller [2009] and Abdullah et al. [2012].

A more recent survey by Qu et al. [2009] has its emphasis on examination timetabling, while Lewis [2008] focuses on metaheuristics for university timetabling problems. Qu et al. [2009] suggest to classify solution approaches into *graph based sequential techniques*, *constraint based techniques*, *local search based techniques*, *population based algorithms*, *multi-criteria techniques*, *hyper-heuristics* and *decomposition/clustering techniques*. Other types are mentioned by Petrovic and Burke [2004], including *case-based reasoning* and *approaches based on mathematical programming*. Hereafter, only approaches, which are either relevant for the understanding of the operators embedded in the algorithm of this theses or used for comparison in the results section, are described.

2.1 Graph Coloring

The reduction of a university course timetabling problem to graph coloring is described by several authors, e.g. De Werra [1985]. The lectures of each course have to be assigned to periods, in a way that no student has two lectures at the same time. In order to construct a graph each lecture of each course is represented by a node. Two nodes are connected by an edge whenever there is at least one student who wants to attend both of the associated lectures. In particular all lectures of the same course have to be connected with each other. Edges have to be introduced either due to the enrollment data or due to the curricula. Finding a feasible timetable with k periods corresponds to solving a graph coloring problem with k colors, i.e. coloring the nodes in such a way that connected nodes have different colors and only k colors are used in total. The conversion from a graph coloring solution to the final timetable is visualized in Figure 1, borrowed from Lewis [2008]. Each color corresponds to a different period of the timetable.

Other conflicts between courses can be incorporated by introducing additional edges between the corresponding nodes, e.g. if there are courses held by the same teacher. De Werra [1985] shows how unavailabilities and previously fixed assignments can be taken into account. In this case one additional node has to be introduced for each period. Edges have to be added to connect these period-nodes with each other. Whenever a course c has an unavailability at period p edges have to be introduced between all lecture-nodes of c and the period-node of p. On the



Periods				
1	**2**	**3**	**4**	**5**
Event 1	Event 4	Event 3	Event 7	Event 6
Event 10	Event 9	Event 5	Event 8	
	Event 2			

Figure 1: Conversion: Graph coloring - timetabling (Lewis [2008])

other hand, if course c is preassigned at period p edges between the lecture-nodes of c and all other periods $q \neq p$ have to be added.

Carter et al. [1996] analyze algorithmic strategies based on graph coloring for the examination timetabling problem. According to the authors these algorithms typically sort the events in descending order with respect to their assignment difficulty first. Then events are scheduled in a conflict-free way one at a time. Some algorithms have also a backtracking procedure implemented, meaning that already scheduled events can be removed from the timetable to insert another event without causing conflicts. The authors note that whether the initial ordering affects the final outcome cannot be answered in general for all data sets and different researchers come to divergent answers to this question.

Several criteria used for sorting events are listed by Carter et al. [1996]. According to the *largest degree* rule used by Broder [1964] events with the largest number of conflicts with other events are scheduled first. The *saturation degree* rule proposed by Brélaz [1979] prioritizes events with the smallest number of remaining periods for scheduling. For the course timetabling problem Lü and Hao [2010] suggest to adapt the *saturation degree* rule in a way that it takes the number of unscheduled lectures of a course into account. Three other rules are described by Carter et al. [1996]. The *random* rule arranges events randomly, the *largest weighted degree* rule weights conflicts by the number of affected students and for the *largest enrollment* rule events are sorted in descending order with respect to the number of enrolled students. Wood [1968] uses a rule similar to *largest enroll-*

ment. Events are sorted in ascending order according to the number of rooms that satisfy their capacity requirements and the *largest degree* rule is used to break ties.

2.2 Metaheuristics

Lewis [2008] classifies metaheuristic methods into *one-stage optimization algorithms*, *two-stage optimization algorithms* and *algorithms that allow relaxations* and explains their strengths and weaknesses. The difference between one-stage and two-stage optimization techniques is that the the latter decomposes the problem in a way that a feasible solution satisfying all hard constraints is found first while the solution quality with respect to the soft constraints is improved in the second stage. Allowing relaxations refers to relaxing some features of the problem, e.g. temporary assigning events without a feasible period to an artificial period and reducing the number of extra periods later.

One-stage optimization algorithms tackle hard and soft constraints at once and thereby allowing the violation of any constraint. According to Lewis [2008] these approaches typically make use of a weighted sum objective function with sufficiently high penalties for hard constraint violations. Setting the weights in the objective function appropriately seems to be critical for navigating well through the search space. The author highlights the advantages and disadvantages of one-stage optimization. Its benefits include its flexibility, the easy implementation and the possibility to embed it in any technique. However, one-stage optimization techniques might be inappropriate if it is absolutely essential to find a feasible solution, particularly if only few feasible solutions exist. As a possible reason for this, Lewis notes that incorporating soft constraints in the objective function when feasibility is not achieved yet, might direct the search away from feasible regions by trying to reduce soft constraint violations.

Schaerf [1999b] adjusts the infeasibility weight dynamically in his local search algorithm for high school timetabling. After each segment of ten moves the weight might be adjusted depending on the number of infeasible solutions. If all solutions have been infeasible the weight is increased. On the contrary the weight is decreased if all solutions have been feasible.

Bellio et al. [2013] propose a one-stage *Simulated Annealing* (Kirkpatrick et al. [1983]) approach, incorporating and extending the ideas by Bellio et al. [2012] and Ceschia et al. [2012]. The cost function is a weighted sum of the penalties for soft constraint violations and high penalties for violations of some hard constraints. The neighborhoods are specified by the operators *lecture move* and *lecture swap*. An important aspect of their work is the statistical method for the parameter tuning, aiming to find either fixed parameter values that suit all instances or correlations with the instances' features in order to predict the parameter values for each instance individually. In particular, the statistical analysis is performed only on a set of artificial instances. The parameter setting is then predicted automatically for the benchmark instances.

Two-stage optimization algorithms try to find a feasible solution first and reduce soft constraint violations while maintaining feasibility afterwards. As noted by Lewis [2008], no weights for infeasible assignments have to be specified. These approaches might be beneficial if feasibility is very important. On the other hand, two-stage algorithms might be inefficient and ineffective when the search space is very constrained because these algorithms are only allowed to move in feasible regions. To overcome the disadvantages when facing very constrained problems, Lewis [2008] suggests to take shortcuts by allowing infeasible solutions. Furthermore, if a feasible solution is barely achievable and some hard constraint violations are accepted, one-stage optimization algorithms are probably a better choice in order to incorporate a trade-off between soft and hard constraint violations.

Lewis [2008] concludes that each principle has its benefits and drawbacks and neither is generally superior. However, he notes that the first international timetabling competition might have influenced researches towards two-stage algorithms. The reason for this is that only algorithms were accepted that were able to generate feasible solutions for all benchmark instances, as stated on the competition's website[1]. The scores for ranking the algorithms were based on the soft constraint violations.

An example for two-stage algorithms is the hybrid approach by Müller [2009]. He is the winner of the ITC-2007 of the tracks about CB-CTT and examination timetabling and performed well on the PE-CTT track. The results and the order-

[1]`http://www.idsia.ch/Files/ttcomp2002/oldindex.html`

ing of the five best participants are available on the competition's website[1]. His construction algorithm schedules events one at a time, where events that are hard to schedule are prioritized. Moreover the algorithm allows the removal of already scheduled events to avoid conflicts. In the improvement phase a *Hill Climbing* algorithm, a *Great Deluge* algorithm (Dueck [1993]) and a *Simulated Annealing* approach alternate. Several neighborhood operators are implemented, including *time move*, *room move* and *event move* and also some tailor-made operators for the different tracks.

Lü and Hao [2010] also propose a two-stage algorithm. They are ranked second on the CB-CTT track of the ITC-2007. To construct an initial solution a sequential heuristic with priority rules is used. The second stage combines an intensification phase and a diversification phase based on *Iterated Local Search* (Lourenço et al. [2003]). The former alternately exploits two neighborhoods with *Tabu Search* (Glover and Laguna [1997]). The diversification phase is performed whenever the Tabu Search cannot improve the solution any further. The *depth* of the Tabu Search and the *perturbation strength* are adapted dynamically.

The two-stage approach by Abdullah et al. [2012] for university course timetabling makes use of the construction algorithm by Landa-Silva and Obit [2008] to build a feasible in three phases. First a largest degree heuristic is performed. As long as feasibility is not achieved, neighborhood search and Tabu Search are alternately applied. The improvement phase combines a multi-start Great Deluge algorithm with an electromagnetic-like mechanism (Birbil and Fang [2003]), whereas the latter is used for calculating the decreasing rate of the Great Deluge algorithm's level.

Abdullah and Turabieh [2012] generate a population of solutions with a saturation degree heuristic. The second stage makes use of a tabu-based memetic approach. The selection of parents is based on a roulette wheel principle. Crossover and mutation operators are then applied while maintaining feasibility. Afterwards the new solutions are improved by using a neighborhood structure that is not in the tabu list. The implemented neighborhood structures are adapted from Abdullah et al. [2007c], whereas some of which describe large neighborhood structures. The algorithm has been tested on exam timetabling and CB-CTT

[1]`http://www.cs.qub.ac.uk/itc2007/winner/finalorder.htm`

problems. Other algorithms for university timetabling problems that make use of large neighborhood structures have been proposed by Abdullah et al. [2007b] and Abdullah et al. [2007a].

Petrovic and Burke [2004] criticize that typically metaheuristics incorporate many parameters and the performance often depends on an appropriate setting. In practice, tuning the parameters is likely to be too challenging for a timetabling officer who is not an expert of the particular algorithm. To overcome this problem the authors suggest to use a simple Great Deluge algorithm that makes use of only two parameters, including the *time limit* and an *estimate of the objective value*. Both of them are easy to determine for the timetabling officer. An analysis of the algorithm's performance on course timetabling problems can be found in Burke et al. [2003].

2.3 Integer Programming

Lach and Lübbecke [2012] tackle the CB-CTT problem by a decomposition-based approach. In the first stage lectures are assigned to periods and in the second stage the room assignment is performed. Each stage is solved by integer programming. To ensure feasibility the number of available rooms is taken into account in the first stage. As a consequence of the decomposition the solution is not necessarily a global optimum. A similar integer programming formulation is used by Burke et al. [2011], who propose a branch-and-cut procedure for CB-CTT.

3 Problem Description

The formulation of the curriculum-based course timetabling problem and the benchmark instances the algorithm is tested on are those of the ITC-2007. They have been used by the competition participants and many other authors to build their algorithms and to conduct their computational experiments. Therefore it is reasonable to compare the results with respect to the competition's instances. Consequently, the timetabling competitions, course timetabling formulations and problem instances are presented in this section with a focus on the second timetabling competition.

3.1 Timetabling Competitions

Schaerf [1999a] points out that several variants of the university timetabling problem exist in the literature. This can be explained by distinct requirements of the institutions for which the formulations and solution approaches have been developed. Schaerf criticizes that the results of the solution approaches that have been published in this field are often solely compared with manual solutions and are not surprisingly superior to them. Therefore, he emphasizes the need of a common definition and benchmark instances, so that different algorithms can be compared conclusively.

According to McCollum et al. [2010] the first international timetabling competition (ITC-2002) has successfully generated a common ground for the timetabling field by stating a problem formulation that has been widely accepted by researchers and proposing benchmark instances. The ITC-2002 was organized by the Metaheuristics Network. Information about the ITC-2002 can be found on its website[1].

The timetabling problem of the ITC-2002 designed by Ben Paechter consists of scheduling a set of lectures with certain requirements and attended by a number of students to periods and rooms. A timetable is called feasible if all events have been placed, each student has at most one lecture at a time, each lecture takes place in a room that meets its requirements in terms of capacity and other features and at most one lecture takes place in a room at a time. Soft constraints are violated if a student has either a lecture in the last period of a day, more than two lectures in a row, or only one lecture on a day.

McCollum et al. [2010] describes the differences between the ITC-2002 and the ITC-2007. The second competition was divided into three tracks. Each track focused on different problems of university timetabling, including exam timetabling, post-enrollment course timetabling and curriculum-based course timetabling. The PE-CTT formulation is closely related to the one used for the ITC-2002. In the ITC-2007 constraints were incorporated that better reflect real world problems.

The benchmark instances used for the ITC-2007 CB-CTT are derived from real timetabling problems of the University of Udine. Even though it was one of the objectives to reduce the gap between theory and practice, the formulated

[1] `http://www.idsia.ch/Files/ttcomp2002/`

problems are still simplifications. Di Gaspero et al. [2007] list eight additional features that are used at the University of Udine for their actual formulation, most notably lunch breaks for students, maximum daily course loads for students, room unavailabilities for some periods, room suitabilities for courses and penalties for course assignments to far too big rooms. A reduced number of features has the advantage of preserving the generality of the formulation.

Each track of the ITC-2007 has its own technical report available the competition's website[1]. Other things provided on the website include a detailed description of the competition rules, the list of the five best-performing algorithms of each track and a benchmarking tool for computers to indicate how long an algorithm can be run on a machine within the competition's time limit. Depending on the machine, the time limit might be in the range 300 to 500 seconds for a single core computer that has been modern at the time of the competition.

21 instances were used to benchmark the algorithms for the CB-CTT track. They were classified into *early instances*, *late instances* and *hidden instances*, whereas the late instance set was released two weeks before the competition's deadline and the hidden instances were released after the closure and were used by the organizers to rank the best participants. Therefore these hidden instances could not be used for tuning the algorithms.

As explained in the technical report by McCollum et al. [2007a] the exam timetabling problem of the ITC-2007 consists of scheduling exams into periods of a predefined session while hard constraints have to be satisfied and solution quality is measured as a weighted sum of soft constraint violations. Students that are individually enrolled to exams must not have two exams at the same time. Furthermore, the room capacities and period lengths have to be respected. There might be also additional constraints, such as one exam has to take place before another exam or predefined room requirements. One distinct feature of examination timetabling compared to course timetabling is that there might be several exams in the same room at the same time, as long as the capacity is not exceeded. Soft constraints address the exam distribution for students over the session, the number of exams with different durations scheduled in the same room at the same

[1] `http://www.cs.qub.ac.uk/itc2007/`

time, the number of larger exams scheduled in the later part of the timetable and some other period and room related features.

The details of the PE-CTT problem of the ITC-2007 are stated by Lewis et al. [2007]. It extends the ITC-2002 formulation in a way that courses might not be available at some periods and precedence requirements of courses have to be respected. In contrast to the PE-CTT, where students have to enroll for courses, the conflicts of the CB-CTT are specified by the curricula, in a way that overlaps of courses of the same curriculum are prohibited. A detailed description of the CB-CTT track of the ITC-2007 is given by Di Gaspero et al. [2007] and is explained in the following subsection.

The third international timetabling competition (ITC-2011) had its focus on high school timetabling. Post et al. [2013] describe the details of the competition and how the problem is modeled.

3.2 Formulations

The CB-CTT problem consists of scheduling lectures of courses to periods and rooms, as described in the technical report by Di Gaspero et al. [2007]. The working days of a week are split into periods for which a timetable has to be found. A period may correspond to one hour. However, it could be more natural to interpret a period as two hours, for example, if the university's lectures are always held in blocks of two hours.

For each course the number of lectures and the number of attending students are known in advance, as well as the teacher who holds the course. Furthermore, each course is a member of one or more curricula, that are also fixed. Student conflicts are based on these curricula.

Feasible timetables have to satisfy hard constraints, including all lectures have to be scheduled (*Lectures*), at most one lecture can take place in a room at a time (*RoomOccupancy*), at most one course of the same curriculum or taught by the same teacher can be held at the same time (*Conflicts*) and availabilities of teachers have to be respected (*Availability*). The availability constraint is represented by a set of periods for each course where a lecture of the course must not take place.

Soft constraints represent features that are nice to have. Consequently, the solution quality is measured in terms of a weighted sum of the soft constraint penalties. One soft constraint addresses the room capacity (*RoomCapacity*). Whenever a lecture takes place in a room with a capacity less than the course's number of students, each student above the capacity limit counts as one violation. In addition to the capacity requirements, lectures of the same course should preferably take place in the same room (*RoomStability*). Each additional room used by a course corresponds to one violation. Another soft constraint aims at spreading the lectures of each course over a predefined number of working days (*MinWorking-Days*). Each day less than the required spread is counted as one violation. Finally, curricula should be as compact as possible (*IsolatedLectures*). For each curriculum a lecture that is not adjacent to any other lecture of the same curriculum is counted as one violation.

In addition to the basic formulation of the ITC-2007, Bonutti et al. [2012] describe more sophisticated formulations, that are summarized in Table 1, where UD2 refers to the ITC-2007 formulation. H indicates a hard constraint, while a number represents the penalty of the corresponding soft constraint violation. If a feature is not incorporated it is expressed by a hyphen.

The curriculum compactness can alternatively be formulated as *Windows* soft constraint, that takes the actual waiting time between lectures of the same curriculum into account. For each curriculum each period of waiting time between two lectures on the same day counts as one violation.

It might be favorable if the daily course load of students is within a given range. Therefore, the soft constraint *StudentMinMaxLoad* computes the distance to the desired daily load for each curriculum and each day as the number of lectures less or above the range, whereas each of these lectures counts as one violation.

At some universities it might be the case that lecture rooms are located in different buildings that are geographically spread over a certain area. Consequently, students should not have to attend courses that are scheduled in adjacent periods and located in different buildings. This circumstance is taken into account by *TravelDistance*, such that for each curriculum each occurrence of two temporally adjacent lectures in different buildings counts as one violation.

Formulation	UD1	UD2	UD3	UD4	UD5
Lectures	H	H	H	H	H
Conflicts	H	H	H	H	H
RoomOccupancy	H	H	H	H	H
Availability	H	H	H	H	H
RoomCapacity	1	1	1	1	1
MinWorkingDays	5	5	-	1	5
IsolatedLectures	1	2	-	-	1
Windows	-	-	4	1	2
RoomStability	-	1	-	-	-
StudentMinMaxLoad	-	-	2	1	2
TravelDistance	-	-	-	-	2
RoomSuitability	-	-	3	H	-
DoubleLectures	-	-	-	1	-

Table 1: CB-CTT formulations (Bonutti et al. [2012])

Some course may require rooms that satisfy certain features. The scheduling of courses to suitable rooms (*RoomSuitability*) can be incorporated in terms of a hard constraint or as a soft constraint, in a way that each assignment of a lecture to a unsuitable room is counted as one violation.

In addition to the curriculum compactness it may be required by some courses, that their lectures are scheduled adjacently and in the same room, given that they are held on the same day (*DoubleLectures*). Each lecture that is non-grouped in that sense counts as one violation.

3.3 Instances

The benchmark instances for CB-CTT that were used at the ITC-2007 are available on the competition's website[1] and are called `comp01`,...,`comp21`. Several researchers tested their algorithms on their basis, e.g. Abdullah et al. [2012], Müller [2009] and Lü and Hao [2010].

[1]`http://www.cs.qub.ac.uk/itc2007/Login/SecretPage.php`

In addition to the ITC-2007 instance set, further instances are available on the CB-CTT website of the Timetabling Research Group at the University of Udine[1], whereas some of them are described by Bonutti et al. [2012]. Four instances called `test1`,...,`test4` were used by Di Gaspero and Schaerf [2003]. The `comp` and `test` instance sets are real world problems mainly from the University of Udine. The instances `DDS1`,...,`DDS7` proposed by Bonutti et al. [2012] are derived from real world cases of various institutions. Furthermore, a small `toy` instance is available that is meant to be used for debugging. More recently, four large instances of the University of Erlangen provided by Moritz Mühlenthaler, nine new Udine instances and twelve new problems from other Italian institutions were published.

Besides the instance sets and their characteristics the CB-CTT website also provides some additional features. Researchers can upload their solutions and lower bounds. Therefore the best known solutions can probably be found on the website. Furthermore, tools are provided including a solution validator and an instance generator to create artificial instances for testing proposes.

Bonutti et al. [2012] state a number of characteristics of the `comp` instances, shown in Table 2. Some of those are indicators of the problem size, as the number of courses (C), the number of lectures (L), the number of rooms (R), the number of periods (P) and the number of days (D). Other characteristics describe how constrained a problem is. Cu denotes the number of curricula. Co shows the average percentage of conflicting lectures, TA represents the average availability of a teacher per lecture and RO denotes the room occupation in percent.

Let k_c denote the number of conflicts of a lecture of course c with other lectures. Consequently, each conflicting course counts as much as its number of lectures. Conflicts between courses arise if they have a teacher or curricula in common. Every other lecture of the same course counts as one conflict too. Furthermore, let l_c denote the number of lectures of course c, C the set of all courses and $l := \sum_{c \in C} l_c$ the total number of lectures. The conflict percentage of a single lecture is then computed as $\frac{k_c}{l-1}$ and the average conflict percentage Co as

$$Co = \frac{\sum_{c \in C}(k_c \cdot l_c)}{l \cdot (l-1)}$$

[1]`http://satt.diegm.uniud.it/ctt/`

Instance	C	L	R	P	D	Cu	Co	TA	RO
comp01	30	160	6	6	5	14	14.26%	93.1%	88.9%
comp02	82	283	16	5	5	70	8.26%	76.9%	70.8%
comp03	72	251	16	5	5	68	8.43%	78.4%	62.8%
comp04	79	286	18	5	5	57	5.56%	81.9%	63.6%
comp05	54	152	9	6	6	139	22.15%	59.6%	46.9%
comp06	108	361	18	5	5	70	5.40%	78.3%	80.2%
comp07	131	434	20	5	5	77	4.80%	80.8%	86.8%
comp08	86	324	18	5	5	61	4.67%	81.7%	72.0%
comp09	76	279	18	5	5	75	6.79%	81.0%	62.0%
comp10	115	370	18	5	5	67	5.59%	77.4%	82.2%
comp11	30	162	5	9	5	13	15.18%	94.2%	72.0%
comp12	88	218	11	6	6	150	14.28%	57.0%	55.1%
comp13	82	308	19	5	5	66	5.21%	79.6%	64.8%
comp14	85	275	17	5	5	60	7.28%	75.0%	64.7%
comp15	72	251	16	5	5	68	8.43%	78.4%	62.8%
comp16	108	366	20	5	5	71	5.36%	81.5%	73.2%
comp17	99	339	17	5	5	70	5.81%	79.2%	79.8%
comp18	47	138	9	6	6	52	13.34%	64.6%	42.6%
comp19	74	277	16	5	5	66	7.59%	76.4%	69.3%
comp20	121	390	19	5	5	78	5.31%	78.7%	82.1%
comp21	94	327	18	5	5	78	6.61%	82.4%	72.7%

Table 2: Characteristics of the Instances

For each course c unavailabilities are represented by a set of periods when the course must not take place. Let u_c denote the number of unavailable periods of course c. Given the number of daily periods p and the number of days d, the availability percentage of a single lecture of course c is defined as $a_c := 1 - \frac{u_c}{p \cdot d}$. Consequently, the average availability TA is computed as

$$TA = \frac{\sum_{c \in C}(a_c \cdot l_c)}{l}$$

Finally, let r denote the number of rooms. The room occupation RO is then computed as the total number of lectures divided by the total number of rooms in all periods, i.e.

$$RO = \frac{l}{r \cdot p \cdot d}$$

4 Mathematical Model

The model for the CB-CTT problem presented in this section is based on the integer programming formulation proposed by Lach and Lübbecke [2012]. Since they use a model for a decomposition approach, slight adaptations are made for an integer programming formulation of the whole problem. A very similar model is stated by Burke et al. [2011].

4.1 ITC-2007 Formulation

4.1.1 Notation and Decision Variables

The notation used in the model is summarized in Table 3. D denotes the set of days for the timetable. Each day is divided into periods, where the set of periods is denoted by P. In each period the same set of rooms R is available. The set of courses is denoted by C. Each course c has l_c lectures that need to be scheduled. These lectures should be spread over a minimum number of days specified by m_c. Each course belongs to one or more curricula and is held by one teacher. The set of curricula is denoted by CU and the set of courses belonging to a curriculum cu is denoted by K_{cu}. T denotes the set of teachers and L_t the set of courses being held by teacher t.

For each course c the set of the day-period pairs the course must not be assigned to is denoted by U_c. On the other hand, $A_{(p,d)}$ denotes the set of courses that can be scheduled at period p on day d. Each room's capacity and each course's number of students is known. Consequently one can compute the capacity shortage $s_{c,r}$ for assigning course c to room r. The penalties are denoted by p^{TYPE} with the corresponding superscript.

Symbols	Description
P	set of periods
D	set of days
R	set of rooms
C	set of courses
CU	set of curricula
K_{cu}	set of courses of curriculum cu
T	set of teachers
L_t	set of courses taught by teacher t
U_c	unavailabilities: $U_c = \{(p,d) : p \in P, d \in D, c \text{ unavailable in } (p,d)\}$
$A_{(p,d)}$	set of available courses at (p,d), i.e. $A_{(p,d)} = \{c \in C : (p,d) \notin U_c\}$
$s_{c,r}$	capacity shortage if course c takes place at room r
l_c	number of lectures of course c
m_c	minimum spread over working days of course c
p^{CAP}	penalty for violating the room capacity
p^{STAB}	penalty for violating the room stability
p^{DAYS}	penalty for violating the minimum spread over working days
p^{COMP}	penalty for violating the curriculum compactness

Table 3: Notation of the mathematical model

The timetable is represented by the binary variable $x_{c,d,p,r}$. It takes the value 1 if a lecture of course c is scheduled at period p on day d in room r. $x_{c,d,p,r}$ is defined only for day-period pairs for which course c is available. Therefore, the formulation takes the availability constraint implicitly into account.

Additional binary and integer decision variables are needed in order to formulate the soft constraints. $v_{cu,p,d}$ is used to identify isolated lectures of curriculum cu. $y_{c,r}$ shows whether at least one lecture of course c takes place in room r. Similarly, $z_{c,d}$ indicates if a lecture of course c is held on day d. If the minimum spread over working days is not satisfied, w_c counts the difference to the required number. The decision variables are precisely defined as follows.

$$x_{c,d,p,r} = \begin{cases} 1 & \text{if course } c \text{ is scheduled at period } p \text{ on day } d \text{ in room } r \\ 0 & \text{otherwise} \end{cases}$$

$$\forall d \in D, p \in P, c \in A_{(p,d)}, r \in R$$

$$v_{cu,p,d} = \begin{cases} 1 & \text{if curriculum } cu \text{ has an isolated lecture at period } p \text{ on day } d \\ 0 & \text{otherwise} \end{cases}$$

$$\forall cu \in CU, p \in P, d \in D$$

$$y_{c,r} = \begin{cases} 1 & \text{if course } c \text{ takes place in room } r \text{ at least once} \\ 0 & \text{otherwise} \end{cases} \quad \forall c \in C, r \in R$$

$$z_{c,d} = \begin{cases} 1 & \text{if course } c \text{ has at least one lecture on day } d \\ 0 & \text{otherwise} \end{cases} \quad \forall c \in C, d \in D$$

w_c number of days less than m_c, integer, ≥ 0 $\forall c \in C$

Due to the objective function and the corresponding soft constraints that will be explained later, it would suffice to set the bounds of the decision variables w_c, $z_{c,d}$ and $v_{cu,d,p}$ correctly without enforcing their integer and respectively binary properties explicitly.

4.1.2 Objective Function

According to the UD2 formulation (Bonutti et al. [2012]) a well created timetable should fulfill certain properties, as described in section 3.2. While hard constraints have to be satisfied to achieve feasibility, soft constraints reflect favorable characteristics of the timetable. These convenient features include that students should have a seat in their lectures, waiting times between lectures of the same curriculum should be avoided and lectures of the same course should be spread over a predefined number of days and should possibly take place in the same room. The objective is to minimize the weighted sum of the corresponding penalty terms.

$$\min \underbrace{\sum_{d\in D,p\in P,c\in A_{(p,d)},r\in R} x_{c,d,p,r}\cdot s_{c,r}\cdot p^{\text{CAP}}}_{\text{capacity penalty}} + \underbrace{\sum_{c\in C}\left(\sum_{r\in R} y_{c,r}-1\right)\cdot p^{\text{STAB}}}_{\text{room stability penalty}} +$$
$$+\underbrace{\sum_{c\in C} w_c\cdot p^{\text{DAYS}}}_{\text{min days penalty}} + \underbrace{\sum_{cu\in CU,p\in P,d\in D} v_{cu,p,d}\cdot p^{\text{COMP}}}_{\text{curriculum compactness penalty}} \tag{1}$$

The sum over the schedule variable $x_{c,d,p,r}$ weighted by the penalty p^{CAP} times the respective capacity shortage $s_{c,r}$ for assigning course c to room r yields the capacity penalty term. By employing the decision variable y_{cr} indicating whether a lecture of course c is held in room r, one can easily compute the number of rooms used by the course and hence the corresponding room stability penalty. The penalty term for not satisfying the minimum spread over working days is computed as the number of days less than the requirement times the respective weight. Similarly, for determining the penalty term for the curriculum compactness, one has to calculate the number of isolated lectures weighted by p^{COMP}.

4.1.3 Hard Constraints

The availability constraint is implicitly respected by the definition of $x_{c,d,p,r}$. The other hard constraints can be formulated as follows.

$$\sum_{p\in P,d\in D,(p,d)\notin U_c,r\in R} x_{c,d,p,r} = l_c \quad \forall c\in C \tag{2}$$

$$\sum_{r\in R,c\in L_t\cap A_{(p,d)}} x_{c,d,p,r} \leq 1 \quad \forall p\in R, d\in D, t\in T \tag{3}$$

$$\sum_{r\in R,c\in K_{cu}\cap A_{(p,d)}} x_{c,d,p,r} \leq 1 \quad \forall p\in R, d\in D, cu\in CU \tag{4}$$

$$\sum_{c\in A_{(p,d)}} x_{c,d,p,r} \leq 1 \quad \forall r\in R, p\in P, d\in D \tag{5}$$

Constraint 2 makes sure that all lectures of the courses are scheduled. Due to constraint 3 each teacher holds most one lecture at the same time. Constraint 4 guarantees that two lectures of the same curriculum are not held in parallel. In particular, two lectures of the same course will not take place at the same point in time, since lectures of the same course are also members of the same curriculum. Finally, constraint 5 respects that a room can accommodate at most one lecture at a time.

4.1.4 Soft Constraints

The following constraints link the decision variables representing favorable characteristics of the timetable in the objective function with the schedule variable x_{cdpr}.

$$\underbrace{\sum_{r\in R, p\in P, (p,d)\notin U_c} x_{c,d,p,r}}_{\text{number of lectures of } c \text{ on } d} - z_{c,d} \geq 0 \quad \forall c \in C, d \in D \tag{6}$$

$$\sum_{d\in D} z_{c,d} + w_c \geq m_c \quad \forall c \in C \tag{7}$$

$$\underbrace{\sum_{c\in K_{cu}\cap A_{(p,d)}, r\in R} x_{c,d,p,r}}_{=1 \text{ if } cu \text{ has a lecture at } (p,d)} - \underbrace{\sum_{q\in\{p-1,p+1\}, c\in K_{cu}\cap A_{(q,d)}, r\in R} x_{c,d,q,r}}_{\in\{1,2\} \text{ if } cu \text{ has at least one lecture at adjacent periods}} - v_{cu,p,d} \leq 0$$

$$\forall cu \in CU, p \in P, d \in D \tag{8}$$

$$\underbrace{\sum_{d\in D, p\in P, (p,d)\notin U_c} x_{c,d,p,r}}_{\text{number of lectures of } c \text{ in } r} - M \cdot y_{c,r} \leq 0 \quad \forall c \in C, r \in R \tag{9}$$

Constraint 6 links $z_{c,d}$ with $x_{c,d,p,r}$ in a way that $z_{c,d}$ can indicate, whether a course takes place on a particular day. $z_{c,d}$ can only be set to 1 if there is at least one lecture of course c held on day d. For each course, $z_{c,d}$ and w_c are connected by constraint 7, such that w_c counts the number of days less than the required spread. Note, that due to the objective function and the link with w_c, the variable $z_{c,d}$ will be set to 1 if it is allowed by constraint 6 and if the required spread is not reached yet. On the other hand, if the spread over days is greater than required, there is no guarantee that each respective $z_{c,d}$ is set 1, because w_c can be set to 0 even without considering every day. Consequently, one cannot directly interpret the values assigned to $z_{c,d}$. If this feature is demanded, one can overcome this problem by adding the constraint $\sum_{r \in R, p \in P, (p,d) \notin U_c} x_{c,d,p,r} - M \cdot z_{c,d} \leq 0$ for each $c \in C$ and each $d \in D$ to the model.

Constraint 8 is used to identify isolated lectures, represented by the variable $v_{cu,p,d}$. The first term of the inequality constraint takes the value 1 if a lecture of the corresponding curriculum is held at the particular time and 0 otherwise. The second term represents the schedule of the previous and the subsequent periods. If at least one lecture of the same curriculum takes place in an adjacent period, either 1 or 2 is subtracted and $v_{cu,p,d}$ can take the value 0. Otherwise an isolated lecture with respect to curriculum cu is identified and $v_{cu,p,d}$ is set to 1. Note, that for the first and the last period of each day the second term has to be adapted in a way that the previous and respectively the last period have to be omitted.

Finally, constraint 9 links $x_{c,d,p,r}$ with $y_{c,r}$. M denotes a large number, whereby the total number of lectures of the respective course is sufficiently large. The variable $y_{c,r}$ is set to 1 if at least one lecture of course c is held in room r.

4.2 Extensions

In addition to a model for the UD2 formulation, Lach and Lübbecke [2012] describe how extensions to the problem can be incorporated, including lunch breaks for students, curriculum dependent maximum daily loads of lectures, room unavailabilities and appropriate room sizes. In this subsection it is explained how the constraints corresponding to the UD3-UD5 formulations can be added to the previously described model.

4.2.1 Students' Minimum and Maximum Day-Load

It might be favorable that students have a well balanced timetable, such that they have at least a few lectures on each day. On the other hand, the students' daily load of lectures should not exceed an upper bound. These bounds are denoted by m_{max} and m_{min}, respectively. They could be easily defined in a curriculum-dependent way, if it is required by the actual situation. The students daily load is represented by the number of lectures of the corresponding curriculum on the particular day. If the daily load is outside the preferred range, each lecture below m_{min} or beyond m_{max} is penalized by p^{LOAD}.

Given, that the reasonable assumption $m_{min} \leq m_{max}$ holds and that lectures below m_{min} are penalized the same as lectures beyond m_{max}, one non-negative integer variable $n_{cu,d}$ is sufficient to add this feature to the model, since a daily load less than m_{min} excludes an overload and vice versa. $n_{cu,d}$ counts the number of lectures of curriculum cu on day d less than m_{min} or greater than m_{max} and is defined for each curriculum $cu \in CU$ and each day $d \in D$. The following two constraints link $n_{cu,d}$ with $x_{c,d,p,r}$.

$$\underbrace{\sum_{p \in P, r \in R, c \in K_{cu} \cap A_{(p,d)}} x_{c,d,p,r}}_{\text{number of lectures of } cu \text{ on } d} - n_{cu,d} \leq m_{max} \quad \forall cu \in CU, d \in D \tag{10}$$

$$\underbrace{\sum_{p \in P, r \in R, c \in K_{cu} \cap A_{(p,d)}} x_{c,d,p,r}}_{\text{number of lectures of } cu \text{ on } d} + n_{cu,d} \geq m_{min} \quad \forall cu \in CU, d \in D \tag{11}$$

Finally, the following term has to be added to the objective function

$$\sum_{cu \in CU, d \in D} n_{cu,d} \cdot p^{LOAD}$$

4.2.2 Room Suitability

Courses might have certain requirements regarding the rooms' equipment, such that for each course c the set of suitable rooms $R_c \subseteq R$ is given. This circumstance can be treated either as a soft constraint or a hard constraint. In the former case assigning a course to a room that does not meet its requirements causes a penalty of p^{SUITABLE}. Consequently one has to add the sum over all course assignments to inappropriate rooms weighted by the penalty p^{SUITABLE} to the objective function.

$$\underbrace{\sum_{d \in D, p \in P, c \in A_{(p,d)}, r \in R \setminus R_c} x_{c,d,p,r}}_{\text{assignments to unsuitable rooms}} \cdot p^{\mathrm{SUITABLE}}$$

In the other case, where room suitability is obligatory, the decision variable $x_{c,d,p,r}$ has to be defined only for suitable rooms, as it is done for available periods. Consequently, all constraints have to be adapted accordingly.

4.2.3 Windows

Instead of penalizing isolated lectures as in the curriculum compactness constraint, one could take the actual waiting time between two lectures of the same curriculum into account. Each period the students have to wait between lectures is penalized by p^{WINDOW}. In order to count the waiting time on a particular day, one has to identify the starting time of the first course and the finishing time of the last course on that day by the use of two non-negative integer decision variables. $b_{cu,d}$ denotes the starting time of the first course and $f_{cu,d}$ denotes the finishing time of the last course of curriculum cu on day d. The relation between the new decision variables and $x_{c,d,p,r}$ is specified by the following three constraints.

$$\underbrace{\sum_{c \in K_{cu} \cap A_{(p,d)}, r \in R} x_{c,d,p,r} \cdot (p - M) + M}_{\substack{= \text{starting time, if a } c \text{ of } cu \text{ scheduled at } (p,d) \\ = M\text{, otherwise}}} \geq b_{cu,d} \quad \forall cu \in CU, d \in D, p \in P \tag{12}$$

$$\underbrace{\sum_{c \in K_{cu} \cap A_{(p,d)}, r \in R} x_{c,d,p,r} \cdot (p + 1)}_{\substack{= \text{finishing time, if } c \text{ of } cu \text{ at } (p,d) \\ = 0\text{, otherwise}}} \leq f_{cu,d} \quad \forall cu \in CU, d \in D, p \in P \tag{13}$$

$$\underbrace{\sum_{p \in P, c \in K_{cu} \cap A_{(p,d)}, r \in R} (x_{c,d,p,r} \cdot p)}_{= 0\text{, if no lecture of } cu \text{ on day } d} \geq b_{cu,d} \quad \forall cu \in CU, d \in D \tag{14}$$

The term $\sum_{r \in R} x_{c,d,p,r} \cdot p$ represents the starting time of course c, given that course c is scheduled on day d at period p. Since each lecture lasts one period, the finishing time is represented by $\sum_{r \in R} x_{c,d,p,r} \cdot (p + 1)$. $b_{cu,d}$ and has to be less or equal than the starting time of each course of curriculum cu scheduled on day d, specified by constraint 12. Analogously, $f_{cu,d}$ has to be greater or equal than the finishing time of each course of curriculum cu scheduled on day d, as stated in constraint 13. The adjusted objective function will guarantee that the variables take the values equal to the starting time of the first course and the finishing time of the last course, respectively.

M in constraint 12 denotes a large number, whereby the number of periods on a day is sufficiently large. The formulation with M is needed because if no lecture is scheduled at a period, the term $\sum_{r\in R} x_{c,d,p,r} \cdot p$ is equal to 0. However, $b_{cu,d}$ should not be constrained if no lecture is scheduled at that time. Whenever there is no lecture of curriculum cu scheduled on day d at all, $b_{cu,d}$ and $f_{cu,d}$ are set to 0, due to the constraints 14 and 13, respectively, resulting in a penalty of 0.

The difference between the finishing time of the last lecture and the starting time of the first lecture on that day indicates the time a student has to stay at the university. Subtracting the number of lectures yields the waiting time on that day. Hence, the following penalty term has to be added to the objective function.

$$\sum_{cu\in CU, d\in D} \left(\underbrace{f_{cu,d} - b_{cu,d} - \underbrace{\sum_{p\in P, c\in K_{cu}\cap A_{(p,d)}, r\in R} x_{c,d,p,r}}_{\text{number of lectures on day } d}}_{\text{waiting time on day } d \text{ for students of curriculum } cu} \right) \cdot p^{\text{WINDOW}}$$

4.2.4 Travel Distance

At some universities, lecture rooms are located in different buildings that might be geographically spread. In this case one has to take the travel distance between those buildings into account. Consequently, lectures that are members of the same curriculum and take place in different buildings should not be temporally adjacent. Violations are penalized by p^{TRAVEL}. The set of buildings is denoted by B and for each building $b \in B$ its set of rooms is denoted by R_b.

The binary decision variable $g_{cu,d,p}$ keeps track of adjacent lectures of the same curriculum taking place in different buildings. $g_{cu,d,p}$ has to be defined for all periods but the first on each day. If a lecture of curriculum cu starts at period $p-1$ and another lecture of the same curriculum starts in the subsequent period on the same day, the variable $g_{cu,d,p}$ has to be set to 1. This is guaranteed by the following constraint.

$$\underbrace{\sum_{c \in K_{cu} \cap A_{(p-1,d)}, r \in R_b} x_{c,d,p-1,r}}_{=1 \text{ if } c \text{ of } cu \text{ in } b \text{ at } p-1} + \underbrace{\sum_{c \in K_{cu} \cap A_{(p,d)}, r \in R \setminus R_b} x_{c,d,p,r}}_{=1 \text{ if } c \text{ of } cu \text{ in different building at } p} - g_{cu,d,p} \leq 1$$

$$\forall cu \in CU, d \in D, p \in P \setminus \{1\}, b \in B \quad (15)$$

Finally, the number of violations weighted by the corresponding penalty has to be added to the objective function, described by the following term.

$$\sum_{cu \in CU, d \in D, p \in P \setminus \{1\}} g_{cu,d,p} \cdot p^{\text{TRAVEL}}$$

4.2.5 Double Lectures

In addition to the compactness of a curriculum, it might be favorable that lectures of the same course are blocked if they are held on the same day, i.e. they should be temporally adjacent and located in the same room. This feature may be only required by a subset of courses $N \subseteq C$.

The binary variable $h_{c,d,p}$ identifies lectures of course c on day d at period p without lectures of the same course in adjacent periods in the same room and is defined only for courses that require double lectures. The non-negative integer variable $u_{c,d}$ counts the actual number of non-blocked lectures, given that multiple lectures of course c are scheduled on day d, which in turn is inversely indicated by the binary variable $a_{c,d}$.

The following constraints have to be added to the model.

$$x_{c,d,p,r} - \underbrace{\sum_{q \in \{p-1,p+1\}, (q,d) \notin U_c} x_{c,d,q,r}}_{\geq 1 \text{ if } c \text{ in adjacent period in same room}} - h_{c,d,p} \leq 0 \quad \forall c \in N, d \in D, p \in P, r \in R \quad (16)$$

$$\underbrace{\sum_{d \in D, p \in P, (p,d) \notin U_c, r \in R} x_{c,d,p,r}}_{\text{number of lectures of } c \text{ on day } d} - M \cdot (1 - a_{c,d}) \leq 1 \quad \forall c \in N \quad (17)$$

$$\sum_{p \in P} h_{c,d,p} - M \cdot a_{c,d} \leq u_{c,d} \quad \forall c \in N, d \in D \tag{18}$$

In constraint 16 $x_{c,d,p,r}$ is equal to 1 if a lecture of course c is scheduled at (p, d) in room r. If there is also a lecture of the same course scheduled at an adjacent period in the same room, represented by the second term, the lectures are blocked and $h_{c,d,p}$ can take the value 0. Otherwise a non-blocked lecture of course c on day d at period p is identified and $h_{c,d,p}$ is set to 1. The second term of constraint 16 has to be adapted adequately for the first and the last period of a day.

If course c has two or more lectures scheduled on day d, the variable $a_{c,d}$ is set to 0 due to constraint 17, otherwise it can take the value 1. In the constraints 17 and 18 M denotes a large number, whereby the total number of lectures of the respective course is sufficiently large.

In constraint 18 the first term counts the daily number of non-blocked lectures of the particular course. If only one lecture of course c is scheduled on day d a large number is subtracted. Consequently, single non-blocked lectures on a day will not be counted by $u_{c,d}$. If there are multiple lectures of course c on day d the number of non-blocked lectures will be represented by $u_{c,d}$.

The number of non-blocked lectures weighted by p^{DOUBLE} has to be added to the objective function.

$$\sum_{c \in N, d \in D} u_{c,d} \cdot p^{\text{DOUBLE}}$$

4.3 Results

In this subsection the solutions of the integer programs of the ITC-2007 instances are reported. The programs are solved with CPLEX 12.5. The constraints of the UD2 formulation are used, as described in subsection 4.1. The computational experiments are conducted on a modern Linux PC with a Intel Core i5-3550 CPU running at 3.30GHz and 8 GB memory. The stopping criteria for CPLEX for solving a program is set to 1000 seconds. This is significantly more than the time allowance of the ITC-2007.

The results are presented in Table 4. The column *IP* shows the results of the integer programs. In the column *Best* the best known solutions are given. Bold numbers indicate optimal solutions. The best known solutions are basically those that are reported on the CB-CTT website[1]. The best solutions of the instances `comp05` and `comp12` have been found by the proposed ALNS approach.

Instance	IP	Best
`comp01`	**5**	**5**
`comp02`	174	24
`comp03`	162	66
`comp04`	**35**	**35**
`comp05`	394	284
`comp06`	1353	**27**
`comp07`	2788	**6**
`comp08`	54	**37**
`comp09`	146	**96**
`comp10`	64	**4**
`comp11`	**0**	**0**
`comp12`	468	298
`comp13`	128	**59**
`comp14`	114	**51**
`comp15`	187	66
`comp16`	115	**18**
`comp17`	168	**56**
`comp18`	118	62
`comp19`	83	**57**
`comp20`	4507	**4**
`comp21`	204	**74**

Table 4: Results of the integer programs of the ITC-2007 instances

Only three instances are solved to optimality. For all other instances CPLEX returns the best solution found after reaching the stopping condition. Some of the solutions are far from the best known solutions, even though the time limit is much higher than the one of the ITC-2007. Furthermore, one has to note that the UD2 formulation incorporates only few features. In practice universities might impose additional requirements to the timetable, as it is the case for the actually used formulation at the University of Udine (Di Gaspero et al. [2007]). That would probably make the problem even harder to solve exactly.

One might conclude that an exact approach is inappropriate for large problems whenever a timetable is needed within relatively short time. This is in accordance with Burke et al. [2011] who state that the runtime of the CPLEX MIP solver limits the suitability of their branch-and-cut procedure for large instances.

[1]`http://satt.diegm.uniud.it/ctt/` [accessed: 2013-09-02]

5 Solution Approach

The fact that timetabling problems are typically NP-hard and the observations in the previous section suggest to use heuristic methods to solve the CB-CTT problem, if a good solution has to be found in reasonable time. Therefore, a heuristic approach is applied, in particular an *Adaptive Large Neighborhood Search* (ALNS) based on the papers by Ropke and Pisinger [2006] and Pisinger and Ropke [2007]. Its features are described in this section.

ALNS has been developed by Ropke and Pisinger to solve vehicle routing problems, for which it performs very well. Basically, ALNS repetitively destroys and repairs an incumbent solution and thereby leading to a gradual improvement. It is an extension of *Large Neighborhood Search* proposed by Shaw [1998] and is very similar to the *Ruin and Recreate* method by Schrimpf et al. [2000].

Since it is usually too time-consuming in practice to search NP-hard problems explicitly, some solution approaches explore only subsets of the search space, so-called neighborhoods. In the survey by Ahuja and Orlin [2002] techniques are discussed that make use of neighborhoods that grow exponentially in problem size or are too large to be searched explicitly in reasonable time. The authors call this class of methods *Very Large-Scale Neighborhood Search* (VLSN). In their survey VLNS techniques are categorized into *variable-depth methods*, *network flow based improvement algorithms* and *special cases* that are solvable in polynomial time. Even though LNS does not fit well into either of these categories, Ropke and Pisinger [2006] argue that it belongs to VLNS because its neighborhoods are to large to be searched explicitly.

Ahuja and Orlin [2002] notes that locally optimal solutions are generally of better quality if the considered neighborhood is larger. However, this gain in quality comes at the cost of requiring more computation time to explore the neighborhood. Consequently fewer iterations can be performed within the same amount of time.

In this section term *period* is used instead of *day-period pair* to facilitate readability. More precisely, each distinct pair (d, p), where d denotes a day and p denotes a period in terms of the previously stated model, is simply called *period.*

5.1 Adaptive Large Neighborhood Search

5.1.1 Algorithm

Algorithm 1 describes the LNS method. First, an initial solution has to be created. In each iteration, parts of the incumbent solution are destroyed and subsequently repaired to improve the solution gradually, as shown in line 4. New solutions are accepted according to a certain criterion to become the new incumbent solution (lines 5 and 6). The algorithm keeps track of the best solution and in the end the best solution found is returned.

Algorithm 1 LNS (Pisinger and Ropke [2010], p. 407)

```
1: input: a feasible solution $x$
2: $x^b = x$
3: **repeat**
4:     $x' = r(d(x))$
5:     **if** accept $(x', x)$ **then**
6:         $x = x'$
7:     **end if**
8:     **if** $c(x') < c(x^b)$ **then**
9:         $x^b = x'$
10:    **end if**
11: **until** stop criterion is met
12: **return** $x^b$
```

ALNS extends LNS in a way that the selection of destroy and repair operators is biased towards the best-performing ones, as shown in Algorithm 2. Initially, the weights of all operators are set to 1 (line 3). Hence, each operator has the same selection probability until weights are recomputed. In each iteration, a destroy operator and a repair operator are selected separately by a roulette wheel mechanism (line 5). After applying the operators to the incumbent solution the corresponding weights are updated depending on their performance (line 13).

Algorithm 2 ALNS (Pisinger and Ropke [2010], p. 409)

```
1: input: a feasible solution $x$
2: $x^b = x$
3: $w^d = (1, \dots, 1)$, $w^r = (1, \dots, 1)$
4: **repeat**
5:     select destroy and repair methods
       $d \in D$, $r \in R$, using $w^d$ and $w^r$
6:     $x' = r(d(x))$
7:     **if** accept $(x', x)$ **then**
8:         $x = x'$
9:     **end if**
10:    **if** $c(x') < c(x^b)$ **then**
11:        $x^b = x'$
12:    **end if**
13:    update $w^d$ and $w^r$
14: **until** stop criterion is met
15: **return** $x^b$
```

The algorithm makes use of several destroy and repair operators, where basically the destroy operators are responsible for releasing parts of the search space that are subsequently explored by a repair heuristic. Pisinger and Ropke [2010] point out that combinations of destroy and repair operators correspond to neighborhoods $\mathcal{N}_i$ that might be structurally different, as shown in Figure 2. These neighborhoods do not necessarily overlap, since different repair heuristics might direct the search towards different regions of the solution space.

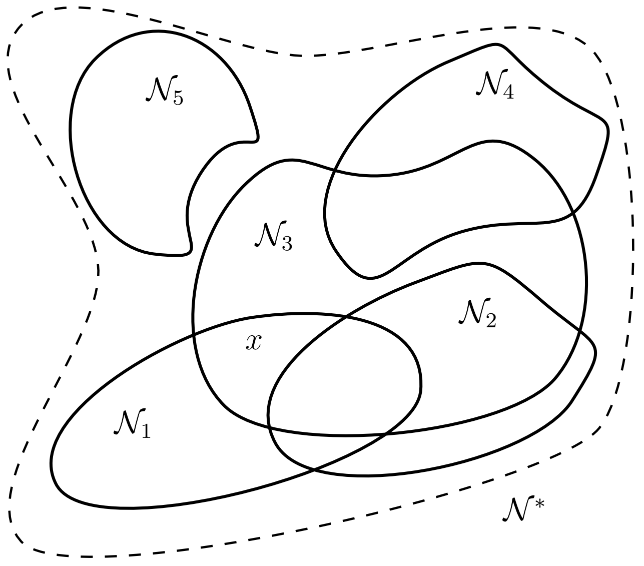


Figure 2: Structurally different neighborhoods (Pisinger and Ropke [2010], p. 412)

Pisinger and Ropke [2010] highlight similarities of ALNS with other approaches that operate on different neighborhoods too. *Variable Depth Neighborhood Search* methods make use of neighborhoods that differ only with respect to their depth, i.e. without loss of generality the k neighborhoods $\mathcal{N}_i$ can be ordered such that $\mathcal{N}_1 \subset \cdots \subset \mathcal{N}_k$. *Variable Neighborhood Search* (VNS) by Mladenović and Hansen [1997] also operates on different neighborhoods that might be structurally different. Given an ordered set of k neighborhoods $\{\mathcal{N}_1, \ldots, \mathcal{N}_k\}$, VNS basically explores one neighborhood $\mathcal{N}_i$ and whenever being trapped in a local optimum the algorithm continues with the next neighborhood $\mathcal{N}_{i+1}$. Each time a better solution is found VNS jumps back to the first neighborhood $\mathcal{N}_1$.

5.1.2 Operator Selection

A very important feature of ALNS is the operator selection mechanism and particularly the adjustment of the operators' selection probabilities according to their performance. Ropke and Pisinger [2006] suggest to select the operators based on a roulette wheel principle. This is done for destroy and repair operators separately. Given that there are k operators with weights $w_i, i \in \{1, \ldots, k\}$, operator j is selected with the probability $\frac{w_j}{\sum_{i=1}^{k} w_i}$.

At the end of each iteration a value σ is added to the score of the destroy and repair operators that have been used. This value depends on the solution quality.

$$\sigma = \begin{cases} \sigma_1 & \text{if the solution is a new global best} \\ \sigma_2 & \text{if the solution is better than the current one and not accepted before} \\ \sigma_3 & \text{if accepted, worse than the current solution and not accepted before} \end{cases}$$

This mechanism encourages operators that find solutions that have not been accepted as a new incumbent solution before. The value added to the score depends on the solution quality. Remarkably, even finding a deteriorating solution is rewarded if the solution is completely new, because the operator might help to direct the algorithm to regions of the search space that have not been visited before.

The search process of the algorithm is divided into segments of size s. Each time the algorithm has performed s iterations, the end of the segment is reached and the weights are recomputed based on the operators' scores achieved during the last segment. Given the old weight w_j^{old} of operator j the formula to compute the new weight w_j^{new} is given by

$$w_j^{new} = w_j^{old} \cdot (1 - r) + r \cdot \frac{\pi_j}{\phi_j}$$

where $r \in [0, 1]$ denotes the reaction factor, π_j denotes the the score achieved during the last segment and ϕ_j denotes the number of times operator j has been called in the last segment.

5.1.3 Destroy Limit

In each iteration n lectures are removed from the current schedule and reinserted by a repair heuristic, whereas the other lectures are fixed on their current positions. n is an integer that is randomly drawn from the interval $[1, n^{max}]$, where n^{max} denotes the destroy limit, i.e. the maximum number of lectures that can be removed. The reference destroy limit n_0^{max} is equal to d percent of the total number of lectures. Pisinger and Ropke [2007] suggest to use an upper bound for the maximum number of destroyed events. Therefore, in case that n_0^{max} exceeds an upper bound u, it is set to u.

The destroy limit is gradually decreased, such that in the last iteration only $\frac{1}{\delta}$ of the reference destroy limit n_0^{max} can be destroyed at most. Therefore, the destroy limit is particularly low when the iteration limit m is close. At this stage it is rather unlikely that destroying very large fractions of the solution will lead to a better solution immediately and deteriorating solutions are unlikely to be accepted by the acceptance scheme. A motivation for decreasing the iteration limit based on computational tests is given in section 6.

In each iteration the destroy limit $n^{max}(t)$ is computed based on the function $N^{max}(t) := n_0^{max} - t^x$ of the current iteration t. Setting $N^{max}(m) = \frac{1}{\delta} \cdot n_0^{max}$ for the last iteration m leads to

$$n^{max}(t) = \left\lfloor N^{max}(t) \right\rfloor = \left\lfloor n_0^{max} - t^{\log_m\left(\frac{\delta-1}{\delta} \cdot n_0^{max}\right)} \right\rfloor = \left\lfloor n_0^{max} - t^{\frac{\ln\left((\delta-1) \cdot n_0^{max}\right) - ln(\delta)}{\ln(m)}} \right\rfloor$$

A plot of the destroy limit as a function of iterations is shown in Figure 3 for an reference destroy limit of 110, an iteration limit of 200,000 and a decrease parameter $\delta = 4$.

The destroy limit might be reset to its initial level whenever the algorithm was not able to improve the globally best solution for a number of iterations, indicating that more diversification is needed. In this case the particularly high destroy limit in the very beginning of the search is useful to escape from the local optimum. This feature will be described in the following subsection in more detail.

The decreasing destroy limit significantly increases the number of iterations within a given time limit, since fixing smaller parts of a solution typically requires less computation time. On the other hand, by destroying less lectures one could

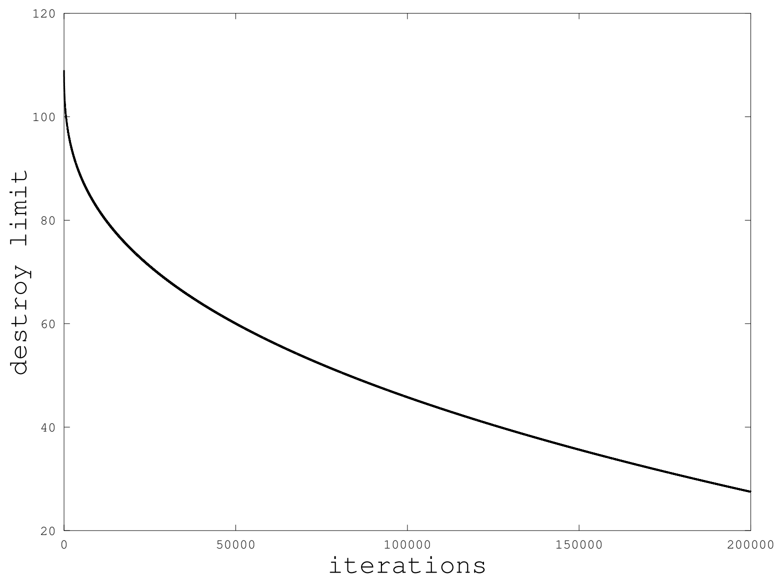


Figure 3: Destroy limit as a function of iterations

potentially loose diversification which in turn could outweigh the gain in performance caused by the larger number of iterations. Consequently, when setting the decreasing parameter δ this trade-off has to be taken into account.

5.1.4 Acceptance Scheme

Ropke and Pisinger [2006] suggest to embed this algorithm in a *Simulated Annealing* framework, developed by Kirkpatrick et al. [1983]. A new solution s' is accepted with the probability $e^{-\frac{f(s')-f(s)}{T}}$, where f denotes an evaluation function, s the current solution and T the temperature. Consequently, all solutions being equal or better than the current solution are accepted, and eventually also worse solutions. The starting temperature is defined implicitly, such that in the beginning a solution that is ψ-percent worse than the initial solution is accepted with a probability of 50%. The respective formula to compute the initial temperature T_{start} is given by

$$T_{start} = -\frac{\psi \cdot f(s_0)}{\ln 0.5}$$

where s_0 denotes the initial solution. In the end of each iteration the temperature is decreased by the factor ρ, representing the cooling rate.

The cooling rate is calculated for each instance individually with regard to the instance's total number of iterations. The reason is that due to the time limit of the ITC-2007, the number of iterations varies significantly between instances. The parameter ρ is set for a reference value of 200,000 iterations. Given an instance-specific iteration limit m the respective cooling rate is calculated as $\rho^{\frac{200000}{m}}$.

The temperature is reheated with regard to the soft penalties of the current solution, whenever h consecutive iterations have not found a new globally best solution. Temperature reheats have already been used by other authors, e.g. Connolly [1992]. This feature helps to escape from local optima.

Even though the search is guided by the current solution, a criterion for reheating on the basis of accepted current solutions seems less appropriate, since for course timetabling problems it is likely that there are many solutions with the same objective value. Moreover, some of these solutions can probably be reached by small changes of the current solutions. Consequently, reheats would be unlikely even if the algorithm was trapped in a particular region.

Additionally, whenever the temperature is reheated the destroy limit is set to its initial level. The decreasing speed of the destroy limit and the cooling rate are adjusted to the number of remaining iterations.

Clearly, there exist more elaborate rules for reheating than simply setting one iteration limit that has to suit all instances and all phases in the search. Since the convergence of the solution is mainly controlled by the cooling rate, which in turn is computed for each instance individually and is recomputed after each reheat, one might argue that the reheat limit should be set on the basis of the remaining iterations after each reheat as well. It turns out that applying this rule would lead either to an undesirable high reheat limit in the beginning of the search or to very short intervals of reheating in the end, though.

Alternatively one could think about a rule on the basis of the number of solutions within an certain range of the best solution. This best solution refers to the best solution found since the last reheat, because in the new region of the search space it might be the case that the solution does not converge to the globally best one. On the one hand, this rule would guarantee that the solution had sufficient time to converge. But on the other hand, even regions would be searched intensively, where the solution would converge to a relatively weak one.

5.1.5 Infeasible Solution Allowance

The algorithm does not prohibit infeasible solutions. Benefits and drawbacks of allowing infeasible solutions are discussed in section 2.2, referring to Lewis [2008]. An obvious advantage is that the algorithm is able to make shortcuts by traversing infeasible regions of the search space. Furthermore, forcing repair heuristics to find feasible solutions would be time consuming, in particular when large portions of the solution have been destroyed. Alternatively, one could reject all infeasible solutions. In this case, however, one might waste good intermediate solutions.

On the other hand, maneuvering the search through infeasible regions involves some risk. For problems with a very constrained search space it might be difficult to find a feasible solution again when the algorithm has entered highly infeasible regions, i.e. when many lectures have been left unscheduled. However, it could be especially important to allow infeasible solutions when facing lots of constraints, since without shortcuts the algorithm might hardly be able to reach different areas.

Consequently, there are conflicting requirements on the penalty function of infeasible solutions. While the former issue would ask for high infeasibility penalties, the latter would urge for low penalties. During the implementation phase it turned out that in case of low penalties the algorithm performs poorly on very constrained instances because occasionally it generated mainly infeasible intermediate solutions for these instances. This indicates that the first issue might be dominant. Furthermore, it is coherent with the intuition that reaching different regions of a very constrained search space might be less problematic when using large neighborhood structures. Therefore, the used penalty function for unassigned lectures is proportional to a measure that indicates how constrained an instance is.

The penalty $p(t)$ for each unscheduled lecture is calculated as a function of the current iteration t as

$$p(t) = \underbrace{\left(\alpha_1 \frac{u}{l} + \alpha_2 \frac{k}{l}\right) \cdot \frac{r}{l} \cdot \frac{f(s_0)}{l}}_{\text{constant}} + p_{max}{}^{\frac{t}{m}}$$

The first part does not depend on the current iteration. $\left(\alpha_1 \frac{u}{l} + \alpha_2 \frac{k}{l}\right) \cdot \frac{r}{l}$ is a measure of how constrained the problem is, where α_1 and α_2 are parameters that can be increased if weights should be set differently or unassigned lectures are less desired in general. $\frac{u}{l}$ indicates the average number of unavailable periods per lecture, where u denotes the number of unavailabilities weighted by the affected number of lectures and l denotes the total number of lectures. $\frac{k}{l}$ represents the average number of conflicts with other lectures. k denotes the sum over each lecture's number of conflicts with other lectures, where both teacher conflicts and curriculum conflicts are taken into account. Using the same notation as for computing the average availability percentage TA and the average conflict percentage Co in section 3.3, the respective formulas are $u = \sum_{c \in C}(u_c \cdot l_c)$ and $k = \sum_{c \in C}(k_c \cdot l_c)$. The terms have individual weights α_1 and α_2 since they affect the search differently. More precisely, unavailable periods of some lectures probably restrict the problem more heavily than an additional curriculum does.

The sum of these terms is multiplied by $\frac{r}{l}$ indicating the room occupation, where r denotes the total number of rooms available in the timetable, computed as the number of rooms times the number of daily periods times the number of days. This incorporates that a large number of conflicts and many unavailabilities might be less problematic if rooms are only sparsely occupied.

$\frac{f(s_0)}{l}$ represents the average penalty per lecture of the initial solution s_0, neglecting penalties for unscheduled lectures. This term is incorporated, since the unassigned penalty should be proportional to the regular penalties that are typically caused by a lecture at the respective instance.

The last term of the function depends on the current iteration. p_{max} denotes the worst case penalty for scheduling a single lecture. It corresponds to the maximum sum over all potential soft penalties, including the room stability penalty, the capacity penalty for assigning the lecture to the smallest room and the curriculum compactness penalty assuming that the lecture is isolated with respect to all of its curricula. The penalty for violating the minimum spread over days is excluded, since scheduling an additional lecture can only improve the solution in terms of the required spread. $\frac{t}{m}$ describes the fraction of the current iteration t over the iteration limit m. Hence, the last term is very low in the beginning, in order to accept infeasible solutions more frequently. However, in the last iteration an unscheduled lecture causes a particularly high penalty, such that all alternative assignments would be preferred.

The globally best solution in the algorithm refers to the feasible solution with the least soft penalties. In case no feasible solution has been found, the solution with the least soft penalties among the solutions with the smallest number of unscheduled lectures is taken. However, the incumbent solution is identified on the basis of the sum over soft penalties and penalties for unassigned lectures.

5.2 Destroy Operators

5.2.1 Related Removal

The *related* removal operator aims to remove similar lectures and is borrowed from Shaw [1998]. The relatedness measure between two courses i and j is defined as

$$R(i,j) := \beta \cdot \frac{\min(o_i, o_j)}{\max(o_i, o_j)} + \frac{k_{ij}}{g_i + 1}$$

where o_i denotes the number of students taking course i, k_{ij} denotes the number of conflicts between the courses i and j taking into account curriculum and teacher conflicts, g_i denotes the number of curricula of course i and β denotes a weight. Consequently, the first term shows the relatedness with respect to the number of students. The rationale behind this is that courses with similar capacity requirements might be easily swapped without causing capacity violations. The second term describes a conflict ratio between the two courses. The reason for adding this term is that moving a course to another period requires the removal of conflicting courses. Due to the conflict term the relatedness matrix might be asymmetric. Also note, that the relatedness measure is the same for all lectures of a course.

The operator is described in Algorithm 3. It starts with randomly selecting a scheduled lecture and adding it to the set B, that represents the set of lectures that will be removed from the timetable. As long as the cardinality of the set is less than the number of lectures to remove, a lecture b is randomly drawn from B. The function $c(b)$ in line 8 maps a lecture b to its course $c(b)$. The list of all scheduled lectures that are not in B and do not belong to the same course of b is denoted by A and is sorted in descending order with respect to the relatedness to b. A lecture is drawn from A by computing its index as $\lfloor |A| \cdot v^{\kappa_1} \rfloor$, where v denotes a random number in $[0, 1)$ and κ_1 denotes a parameter to adjust the selection probabilities of related lectures. If κ_1 is large it is very likely to select the most related lecture. On the other hand, if $\kappa_1 = 1$ each lecture has the same selection probability. The drawn lecture is then added to B and the process is repeated.

Algorithm 3 Related removal (Shaw [1998])

```
1: input: requested removals $n \in \mathbb{N}$, schedule $S$, parameter $\kappa_1 \in \mathbb{R}$
2: set of lectures to remove $B = \emptyset$
3: set $F$ containing all lectures in $S$
4: select random lecture $f \in F$
5: $B = B \cup \{f\}$ $F = F \setminus \{f\}$
6: while $|B| < n$ do
7:    select random lecture $b \in B$
8:    list $A = F \setminus \{f \in F | c(f) = c(b)\}$
9:    sort $A$ in descending order with respect to relatedness to $b$
10:   draw random number $v \in [0, 1)$
11:   $a = A[integer(|A| \cdot v^{\kappa_1})]$
12:   $B = B \cup \{a\}$, $F = F \setminus \{a\}$
13: end while
14: remove all lectures in $B$ from $S$
```

The reason for excluding other scheduled lectures of the same course from being selected is that otherwise this procedure would most likely remove all lectures of the same course before proceeding with another course, since each course is obviously most related to itself. However, the operator is supposed to remove single lectures instead of whole courses.

5.2.2 Random Removal

The *random* destroy operator removes lectures from the schedule at random. Ropke and Pisinger [2006] also employ a random removal heuristic. They note that the random removal can be seen as a special case of the *related* removal operator used by Shaw [1998].

5.2.3 Worst Removal

Ropke and Pisinger [2006] suggest to use a destroy operator that removes highly penalized assignments with a high probability, since reinserting these events may improve the solution. However, it is hard to directly associate single lectures with certain penalties. For example, if a course has two lectures and they take place in different rooms, one does not know which lecture actually caused the penalty for violating the room stability constraint. Consequently, one cannot unambiguously determine which lecture should be removed to make an improvement most likely.

The association of penalties with courses is easier and can be done straightforward for violations of the capacity constraint, the minimum spread and the room stability. The curriculum compactness constraint is interpreted such that lectures that are isolated with respect to a curriculum are rated as being responsible for the corresponding penalty, even though the violation could be removed by scheduling another lecture of the same curriculum at an adjacent period.

The *worst* destroy algorithm is shown in Algorithm 4. It is very similar to the one proposed by Ropke and Pisinger [2006] but with the slight difference that this version operates on the course level for the previously mentioned reason, while the original *worst* destroy algorithm by Ropke and Pisinger selects individual requests for removal.

Algorithm 4 Worst removal (Ropke and Pisinger [2006])

```
1: input: requested removals $n \in \mathbb{N}$, schedule $S$, parameter $\kappa_2 \in \mathbb{R}$
2: list of all courses $A$
3: sort $A$ by descending penalty
4: while $n > 0$ do
5:     draw random number $\upsilon \in [0, 1)$
6:     $a = A[integer(|A| \cdot \upsilon^{\kappa_2})]$
7:     $A = A \setminus \{a\}$
8:     remove all $l_a$ lectures of $a$ from $S$
9:     $n = n - l_a$
10: end while
```

Courses are sorted in descending order with respect to their associated penalties. As long as the requested number of removals is not reached, a course is selected by computing its index in the list (line 6) and all its lecture are removed from the schedule. The selection mechanism is similar to the one proposed by Shaw [1998] for the *related* removal. It incorporates some randomness to avoid the same outcome of different calls of the operator with the same input. In terms of removals each destroyed course counts as much as its number of scheduled lectures. It is likely that the requested number of removals is occasionally exceeded. For example this could happen when the algorithm requests only one removal but a course with two scheduled lectures is selected. In general the limit might be exceeded whenever destroy operators are applied that remove multiple lectures at once.

5.2.4 Random Penalty Removal

The *random penalty* destroy operator randomly selects lectures of courses with a positive penalty value in the sense of the *worst* removal operator and removes them from the schedule. In case that all of these lectures are removed and the requested number of removals is not reached other lectures are removed at random up to the limit.

5.2.5 Random Period Removal

The *random period* destroy operator randomly selects a period, or more precisely a day-period pair, and removes all its scheduled lectures. Each destroyed period counts as much as its number of scheduled lectures in terms of removals.

Rescheduling the lectures within a particular period allows changing their room assignments without affecting the curriculum compactness and the spread over days. Therefore, the operator might be particularly useful to improve the solution with respect to room related constraints, i.e. the room stability constraint and the capacity constraint.

5.2.6 Room Day Removal

The *room day* removal operator selects a day and a room at random. All lectures that are assigned to the room on that day are removed from the schedule. The operator continues with removing lectures from other day-room pairs until the requested number of removals is either reached or exceeded.

Removing all lectures from a particular room on a particular day enables them to get reassigned to different periods on that day while preserving the penalty level of the room-related constraints and the spread over days. Thereby the operator focuses on improving the curriculum compactness.

5.2.7 Isolation & Capacity Removal

The penalties caused by capacity violations and isolated lectures can be associated with individual lectures. The *isolation & capacity* destroy heuristic is very similar to the *worst* removal operator by Ropke and Pisinger [2006] shown in Algorithm 4 but instead of removing whole courses, individual lectures are selected for removal. Furthermore only a subset of all penalties is considered by this operator, such that the list of potentially removable lectures is sorted in descending order with respect to their capacity and compactness penalties. The parameter κ_3 allows to adjust whether high penalty lectures should be selected almost certainly or more randomly.

5.2.8 Spread & Stability Removal

The *spread & stability* removal operator focuses on the penalties that are neglected by the *isolation & capacity* operator, i.e. the penalties for violating the required spread over days and the room stability. For tackling these violations it is reasonable to remove whole courses from the schedule because of the difficulty of making single lectures responsible for penalties of these types. The algorithm is essentially the same as the *worst* removal operator by Ropke and Pisinger [2006] described in Algorithm 4. The ordering of the removable courses is based only on the penalties for violating the day spread and the room stability, though. The selection is controlled by the parameter κ_4.

5.2.9 Curriculum Removal

The *curriculum* destroy algorithm is basically the same as *worst* removal operator by Ropke and Pisinger [2006] given in Algorithm 4, however curricula are selected instead of courses. Destroying a curriculum corresponds to removing all of its lectures from the timetable. The removable curricula are sorted in descending order with respect to their curriculum compactness penalties. All other penalties are ignored. The corresponding selection parameter is called κ_5.

This operator aims to reduce the curriculum penalties. Furthermore, lectures might be moved to periods that have been formerly forbidden due to curriculum conflicts. Consequently the operator might be particularly useful for instances that are very constrained with regard to the curricula.

5.2.10 Teacher Removal

The *teacher* destroy operator is used to ease restrictions due to teacher conflicts. Teachers are randomly selected and all of their lectures are removed from the schedule. Thereby a lecture might be moved to a period that has been previously blocked by another lecture taught by the same teacher.

5.3 Repair Operators

The algorithm makes use of several repair operators. They can be categorized into 2-stage and 1-stage heuristics. The 2-stage heuristics assign lectures to periods first and find a room schedule in the second stage. The 1-stage heuristics evaluate period and room assignments at once and scheduling is performed either in a *greedy* way or on a *regret* basis.

5.3.1 2-Stage Operators

The procedure that assigns lectures to periods is summarized in Algorithm 5. The algorithm receives a vector v as input, where each element corresponds to a course's number of lectures that have to be scheduled. Courses are ordered according to a priority rule and the lectures of the course that is in line are scheduled at their best position with respect to an evaluation criterion. In case no feasible insertion position is left, conflicting lectures can be removed from the schedule that is under construction. In the end the algorithm returns a schedule S_p for each period p and a list of lectures U for which no assignment has been found. Details are described in the following subsections.

5.3.2 Priority Rules

In the beginning of the lecture-period assignment the courses that have to be scheduled are ordered with descending difficulty according to a certain rule, either *saturation degree* (*SD*), *largest degree* (*LD*) or *random.* These priority rules are summarized in Carter et al. [1996]. Even though there are further rules mentioned in the literature, only *LD*, *SD* and *random* are used because the priority rules do not seem to be critical for the algorithm, as shown in section 6.3. The rule selection

Algorithm 5 Lecture-period assignment

```
1: input: vector $v$ of lectures to assign, $v_c$: # lectures to assign of course $c$
2: list of unscheduled lectures $U = \emptyset$
3: list of courses $C = \{c : v_c > 0\}$
4: schedule $S_p = \emptyset \; \forall$ periods $p \in P$
5: sort $C$ according to priority rule
6: compute potential insertion positions $P_c$ for each $c \in C$
7: initialize list of periods from which course $c$ can remove lectures $R_c = P_c$
8: while $C \neq \emptyset$ do
9:   select first course $c_1 = C[1]$
10:   while $v_{c_1} > 0$ do
11:     if $P_{c_1} \neq \emptyset$ then
12:       evaluate all $p \in P_{c_1}$
13:       determine best period $p_{best}$
14:       $S_{p_{best}} = S_{p_{best}} \cup \{l(c_1)\}$
15:       // $l(c_1)$: lecture of $c_1$
16:       $R_{c_1} = R_{c_1} \setminus \{p_{best}\}$
17:       update $P_c$ for each $c \in C$
18:     else if $R_{c_1} \neq \emptyset$ then
19:       evaluate all $p \in R_{c_1}$
20:       determine best period $p_{best}$
21:       conflicting courses $K$ in $p_{best}$
22:       $S_{p_{best}} = (S_{p_{best}} \setminus l(K)) \cup \{l(c_1)\}$
23:       // $l(K)$: conflicting lectures in $p_{best}$ that belong to $K$
24:       $v_k = v_k + 1 \; \forall k \in K$
25:       place $K$ at beginning of $C$
26:       $R_{c_1} = R_{c_1} \setminus \{p_{best}\}$
27:       update $P_c$ for each $c \in C$
28:     else
29:       $U = U \cup \{l(c_1)\}$
30:     end if
31:     $v_{c_1} = v_{c_1} - 1$
32:   end while
33:   $C = C \setminus \{c_1\}$
34:   if saturation degree rule then
35:     reorder $C$ according to rule
36:   end if
37: end while
38: check if any lecture in $U$ can be scheduled due to removed lectures
39: return $U$, $S_p \; \forall$ periods $p \in P$
```

is based on a roulette wheel principle. The selection probabilities are computed by using performance scores in combination with the previously described adaptive mechanism.

The *largest degree* rule used by Broder [1964] prioritizes events with the largest number of conflicts with other lectures. Since this number is the same for all lectures of a course, sorting lectures corresponds to sorting their courses. The *saturation degree* rule, proposed by Brélaz [1979], arranges lectures in ascending order with respect to their number of available periods for scheduling. Again, this number is the same for all lectures of the same course, therefore only the courses have to be sorted. The number of available periods has to be adjusted dynamically during the scheduling process. More precisely, each time all lectures of a course are scheduled the saturation degree of the remaining courses has to be recomputed.

The *random* rule simply orders events randomly. To be consistent with the other rules, the random ordering is again applied to the courses.

Even if a repair operator is called with the same settings and the same requested insertions, the heuristic should produce different schedules. Therefore the saturation degree and the largest degree are perturbed by some random number. More precisely, whenever the saturation degree or the largest degree is computed a random number drawn from $[-\nu, \nu]$ is added, where ν denotes a parameter. Incorporating perturbation is not a new feature for ALNS. In fact Ropke and Pisinger [2006] show that perturbation is important for the algorithm's performance.

5.3.3 Lecture-Period Assignment

As soon as the courses are sorted, all conflict-free insertion positions of the course that is in line are evaluated and one lecture of the course is placed at its best position. Note, that in the first stage all hard constraints can be taken into account, i.e. curriculum conflicts, teacher conflicts and each period's number of available rooms. Basically two operators for resolving the first stage are implemented, i.e. *2-stage best* and *2-stage mean*. When evaluating the periods for insertion one has to bear in mind, that only parts of the schedule are destroyed in each iteration and therefore some lectures remain scheduled and occupy rooms. Moreover, one has to consider that the first stage aims to find a period assignment, consequently it is hard to incorporate room related penalties accurately.

The *2-stage best* heuristic evaluates the periods in the following way. If assigning the lecture to the considered period will lead to isolations with respect to curricula, the curriculum compactness penalty is added as many times as the compactness will be violated by the assignment. On the other hand, the assignment may remove an isolation of an already scheduled lecture, which reduces the insertion cost. Whenever the required spread over days of the corresponding course is not reached and no other lecture of the same course has been scheduled on the considered day, the respective penalty is subtracted, since the solution will be improved. If some lectures of the course have already been scheduled and none of the rooms where these lectures take place are available in the considered period, a new room will be used by the course and consequently the penalty for violating the room stability is added. Finally, the capacity penalty can only be roughly estimated. It is reasonable to assume that if the lecture has the x^{th} most students of all courses that are assigned to the period but do not have a room yet, will get the x^{th} largest available room in the second stage. The resulting capacity penalty is added to the insertion cost. Ties between the lowest cost insertion positions are broken randomly, as it is done also for the other repair operators.

The penalties for violating the minimum spread over days, the room stability and the curriculum compactness are incorporated by the *2-stage mean* heuristic in the same way as before, the capacity penalty is treated differently, though. The idea is that in the first stage it is favorable to achieve an even spread of the courses over periods with respect to the number of students and the capacity of the available rooms. First, a reference utilization of the room capacities u is calculated by dividing the sum of the number of students of all lectures that have to be scheduled Σ_l by the sum of the capacities of all available rooms Σ_r, i.e. $u = \frac{\Sigma_l}{\Sigma_r}$. Then a capacity limit is computed for each period individually as $\eta \cdot u \cdot \Sigma_p$, where Σ_p denotes the sum of the capacities of the available rooms in period p and η denotes a parameter that controls the penalty-free number of students. The capacity penalty added to the insertion cost is proportional to the number of students of the assigned lectures exceeding the capacity limit of the particular period. Even though one might argue, that a good spread over periods does not only depend on the mean but also on other moments, incorporating the variance has not improved the performance of the operator.

Ropke and Pisinger [2006] suggest to perturb the insertion cost by adding a random number. Therefore, two additional operators are implemented, *2-stage best noise* and *2-stage mean noise*, that are based on the previously described heuristics, but each time a period is evaluated a noise value is added to the insertion cost. The noise value is drawn randomly from $[-\mu \cdot p_{max}, \mu \cdot p_{max}]$, where μ denotes a parameter and p_{max} denotes the worst case insertion cost of one lecture as described in subsection 5.1.5. Compared to the original operators, which are only indifferent between equally best insertion positions, the noise feature leads to additional diversification. On the other hand, the noise operators might achieve slightly worse results if suboptimal insertion positions are selected.

5.3.4 Backtracking Procedure

After a lecture is scheduled at its best position, the algorithm continues with the same course if there are still lectures that need to be scheduled. Otherwise it proceeds with the next course according to the order. In case a course is in line that cannot be scheduled conflict-free, a backtracking mechanism is applied. Carter et al. [1996] describe a backtracking procedure and note that such procedures have been used by several authors before. In general backtracking corresponds to removing one or multiple events from the schedule in order to assign an event that could not be scheduled otherwise due to conflicts. These procedures have to incorporate rules to decide about the insertion position and to prohibit cycles.

The implemented backtracking mechanism is similar to the one described by Carter et al. [1996]. Only lectures that do not belong to the fixed part of the current schedule can be removed. Let R_c denote the set of periods from where the considered course c is allowed to remove lectures. Each insertion position $p \in R_c$ is then evaluated in the following way. Let B_p denote the set of all lectures that have to be removed from period p in order to schedule a lecture of course c at that time. Furthermore, let A_p denote the set of lectures that have to be removed from period p but do not have an alternative conflict-free insertion position left.

The first criterion for the insertion period selection is preferring the period with the smallest number of lectures to remove that do not have any alternative conflict-free insertion positions left, i.e. selecting $p : |A_p| \leq |A_q| \; \forall q \in R_c$. The rationale behind this is that removing these lectures will lead to large disruptions. Ties are broken by choosing the period with the smallest number of lectures that have to be removed in total, i.e. $p : |B_p| \leq |B_q| \; \forall q \in R_c$. Each of these lectures has been scheduled according to an evaluation criterion and hence each alternative assignment is probably worse than the current one. In case the procedure is still indifferent, the period with the lowest insertion cost for the considered course is chosen. A more precise calculation of the effects of removing courses in terms of the evaluation criterion is most likely not worth the computational effort, not least because the method is not exact anyway.

The courses of the removed lectures are reinserted in the queue in a way that they are next in line for being scheduled, whereby courses without any potential conflict-free assignment are prioritized the most. However, in case of the *saturation degree* rule the order is still dynamically adjusted. To avoid cycles, a course that has a lecture assigned to a period once must not remove events from the same period at a later time. Note that this mechanism does not necessarily lead to a feasible solution even if such a solution exists.

5.3.5 Lecture-Room Assignment

The first stage heuristic stops when either all lectures are assigned to periods or only lectures remain that cannot be scheduled conflict-free and are not allowed to remove lectures from any period due to the condition that prohibits cycles. In the second stage lectures are assigned to rooms. At this stage only the room stability penalty and the capacity penalty can be affected, whereas the penalties corresponding to the curriculum compactness and the spread over days are predetermined by outcome of the first stage. Furthermore, the final schedule will be free of teacher and curriculum conflicts due to the first stage heuristic.

The room assignment is performed either by the *greatest* heuristic or the *match* heuristic. The operator selection is based on the adaptive mechanism described in subsection 5.1.2. Without interdependencies between periods it would be optimal for each period to assign the lecture with the most students to the largest room, the lecture with the second most students to the room with the second largest capacity and so on. The room stability constraint makes the evaluation more difficult. In particular the assignment is not independent of the order in which the periods are processed anymore. Therefore, both operators find schedules heuristically.

The *greatest* heuristic selects a period randomly. Its lectures are sorted in descending order with respect to their number of students and are scheduled one after another. Each available room in the period is evaluated, taking the capacity penalty and the rooms where other lectures of the same course are scheduled into account. The room stability penalty is added in case no other lecture of the course takes place in the considered room in any other period. The lecture is assigned to the room with the lowest insertion cost. Ties are broken by preferring rooms with the larger capacity. If there are several rooms with the lowest insertion cost and the equally largest capacity, one of these rooms is selected at random. The rationale for selecting the room with the largest capacity is that lectures with many students are processed first. Even though assigning lectures with fewer students to large rooms does not affect the capacity violation, it might be beneficial to use smaller rooms with regard to the room stability. In case all lectures of a period are scheduled the heuristic proceeds with the next randomly selected period. The heuristic stops when all lectures of each period are scheduled.

The *match* heuristic processes one period after another in a random order and for each period also the lectures are processed randomly. The evaluation of the available rooms is performed in the same way as for the *greatest* heuristic, however, ties are broken by selecting the room with the smallest capacity. The reason is that since lectures are scheduled in a random order, there might be lectures left that require large rooms. This is particularly important for the ITC-2007 problem formulation, where a significant mismatch between capacity and the number of students might dominate all other penalty terms.

5.3.6 Example: Evaluation

The following example illustrates the *2-stage best* operator, in particular its period evaluation. Assume that a timetable has to be found for eight courses with their attributes given in Table 5. The column *Teacher* indicates the courses' teachers and *Curricula* shows the curricula to which the courses belong to. *Students* shows each course's number of students. The number of lectures of the courses is given in *Lectures*. *Spread* shows the number of days over which the lectures have to be spread at least, such that the corresponding constraint is not violated. The timetable consists of four days $d_1, \dots, d_4$ with three periods $p_1, \dots, p_3$ each and three rooms $r_1, \dots, r_3$. Room r_1 has a capacity of 100 students, room r_2 can accommodate 50 students and room r_3 has seats for 20 students. Teacher t_4 is not available on day d_4 therefore lectures of course c_4 must not be scheduled then.

The solution shown in Figure 4 is the current solution that shall be improved by the removing and reinserting lectures. Obviously some assignments violate soft constraints, e.g. course c_1 scheduled at (d_1, p_1, r_1) is isolated with respect

Course	Teacher	Curricula	Students	Lectures	Spread
c_1	t_1	cu_1	89	3	2
c_2	t_2	cu_2	32	4	3
c_3	t_3	cu_1, cu_2	57	2	2
c_4	t_4	cu_3	24	3	2
c_5	t_3	cu_3	18	4	3
c_6	t_3	cu_3	16	3	2
c_7	t_5	cu_4	54	3	3
c_8	t_1	cu_4	35	3	3

Table 5: Example: Course attributes

to curriculum cu_1, the room capacity is exceeded by course c_2 at (d_1, p_2, r_3), the required spread over days is not reached by course c_7 and c_2 violates the room stability constraint. These assignments are marked gray in the figure.

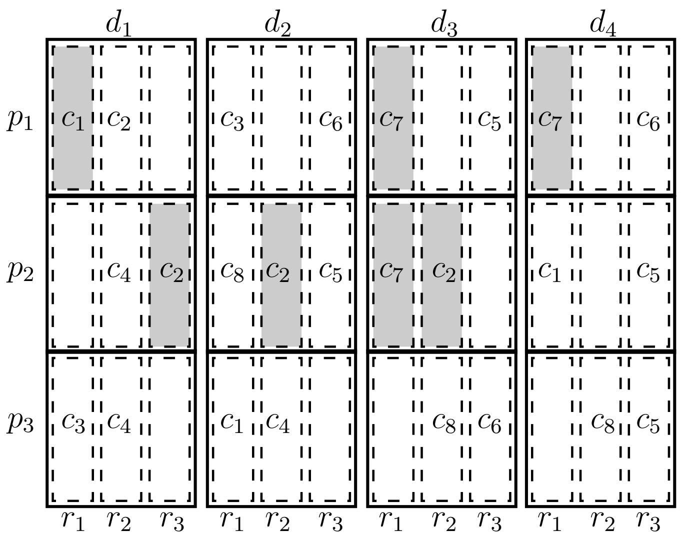


Figure 4: Example: Current schedule

Assume that the *random* destroy operator removes the lectures from (d_1, p_1, r_1), (d_2, p_1, r_1), (d_2, p_3, r_1), (d_3, p_2, r_1) and (d_3, p_3, r_2), as shown in Figure 5a. Next, lectures are assigned to periods by the *2-stage best* operator in the *random* order $c_1 \rightarrow c_8 \rightarrow c_7 \rightarrow c_3$. The room assignment is performed by the *greatest* operator.

Starting with course c_1 the *2-stage best* heuristic evaluates all its potential periods for insertion. Due to curriculum conflicts course c_1 must not be assigned to (d_1, p_3) and (d_4, p_2). The period (d_4, p_3) is blocked since another course held by the same teacher is scheduled then. Finally, all rooms at (d_2, p_2) are already occupied. All other periods are available and are therefore evaluated.

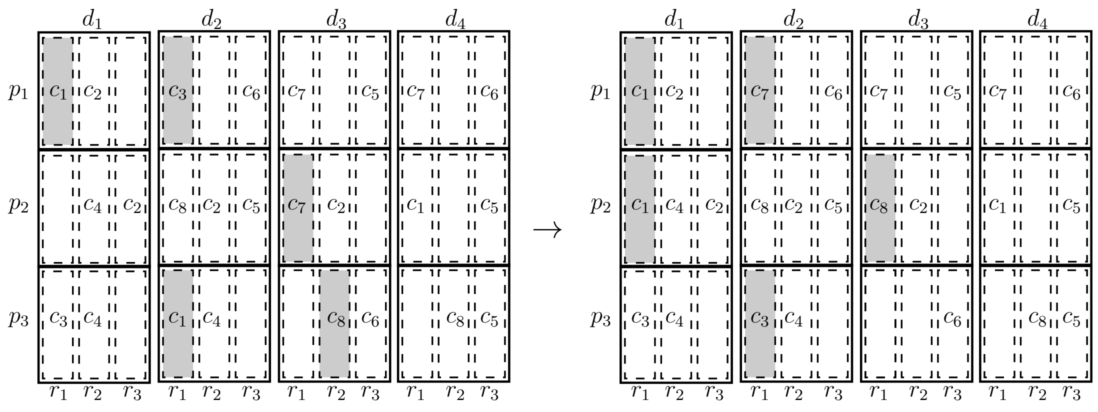
The insertion costs for scheduling c_1 at the potential insertion positions are given in Table 6, where the notation for the penalties of section 4.1 is used. For example, scheduling a lecture of c_1 at (d_1, p_1) leads to an isolation of the lecture with respect to curriculum cu_1. On the other hand, *2-stage best* assumes that c_1 will get the largest available room r_1 and therefore the capacity constraint is satisfied. r_1 is the only room that is already in use by a scheduled lecture of course c_1. Since this room is still available at that time the room stability penalty is not added either. Furthermore, an assignment on that day reduces the penalty for violating the required spread over days.

A negative example for insertion is (d_4, p_1). Scheduling c_1 at that period will lead to an assignment to a too small room in the second stage. Furthermore, the penalty for violating the room stability is added. The isolation of c_1 in the next period is removed, though.

Position	Cost	Position	Cost
(d_1, p_1)	$p^{\text{COMP}} - p^{\text{DAYS}}$	(d_3, p_1)	$39 \cdot p^{\text{CAP}} + p^{\text{STAB}} + p^{\text{COMP}} - p^{\text{DAYS}}$
(d_1, p_2)	$-p^{\text{COMP}} - p^{\text{DAYS}}$	(d_3, p_2)	$p^{\text{COMP}} - p^{\text{DAYS}}$
(d_2, p_1)	$p^{\text{COMP}} - p^{\text{DAYS}}$	(d_3, p_3)	$p^{\text{COMP}} - p^{\text{DAYS}}$
(d_2, p_3)	$p^{\text{COMP}} - p^{\text{DAYS}}$	(d_4, p_1)	$39 \cdot p^{\text{CAP}} + p^{\text{STAB}} - p^{\text{COMP}}$

Table 6: Example: Insertion costs of course c_1

According to the evaluation shown in Table 6, (d_1, p_2) is the best insertion position for course c_1 and therefore one of its lectures is scheduled then. Since there are still lectures of c_1 that need to be scheduled, the algorithm continues with c_1. The insertion position (d_1, p_1) is now the best choice. Due to the lecture of c_1 that has just been scheduled in the next period, the insertion at (d_1, p_1) does not violate the curriculum compactness anymore. The lectures of the other courses are scheduled accordingly. The final timetable after scheduling all lectures is shown in Figure 5b. Compared to the initial timetable the solution has been improved by removing four compactness violations and one spread violation.



(a) Gray: Lectures to remove (b) Gray: Reinserted lectures

Figure 5: Example: Destroy and repair

5.3.7 Initial Solution

The initial solution is generated by applying the *2-stage best* heuristic in combination with the *SD* rule. The room assignment is found by the *greatest* heuristic. The initial solution is not necessarily feasible, however for the ITC-2007 instance set the *2-stage best* heuristic is typically able to find a feasible one.

5.3.8 1-Stage Operators

Ropke and Pisinger [2006] employ a *greedy* heuristic and *regret* heuristics within their ALNS framework. Accordingly, a *greedy* and a *regret* heuristic are used as 1-stage operators for the algorithm of this theses.

For each course $c \in H$, where H denotes the set of courses with lectures that have to be scheduled, the *greedy* heuristic evaluates all its potential insertion positions $G_c = \{(d,p,r) \in D \times P \times R \mid \text{assignment } c \text{ to } (d,p,r) \text{ is feasible}\}$ by the function $f(c,g)$ that maps the assignment of a lecture of course c to position $g \in G_c$ onto its insertion cost. A lecture of the course $c_{best} = \arg\min_{c \in H} \min_{g \in G_c} f(c,g)$, i.e. the course with the lowest insertion cost, is then scheduled at its best position $g_{best} = \arg\min_{g \in G_{c_{best}}} f(c_{best}, g)$.

On the contrary, the *regret* heuristic decides on the basis of regret values, which lecture is scheduled next at its best position. The regret value indicates the opportunity cost for not assigning a lecture to its currently best position. The insertion positions $g_{i,c}$ of course c are sorted in ascending order with respect to their insertion cost, i.e. $i < j \Rightarrow f(c, g_{i,c}) \leq f(c, g_{j,c})$. The regret value $r(c)$ of the k-*regret* heuristic is computed for each course $c \in H$ as the sum of the differences between the course's best insertion position $g_{1,c}$ and the i-th best insertion positions $g_{i,c}$, $i = 2, \ldots, k$, i.e. $r(c) = \sum_{i=2}^{k} |f(c, g_{1,c}) - f(c, g_{i,c})|$. A lecture of the course $c_{best} = \arg\max_{c \in H} r(c)$, i.e. the course with the largest regret value, is then scheduled at its best position $g_{1,c_{best}}$.

Since the insertion cost is the same for all lectures of a course, the evaluation is performed only for courses. The assignment to periods and to rooms is done at once, hence it is easier to calculate the penalties than for two-stage approaches. The penalties regarding the curriculum compactness, the spread over days and the room stability depend on lectures that have already been scheduled, including the lectures that belong to the fixed part of the schedule and those that have been assigned by the heuristic previously.

Each room-period pair is evaluated in the following way. The capacity penalty is computed as the number of students exceeding the room's capacity weighted by the corresponding penalty. For each curriculum of the considered course the adjacent periods are checked whether a lecture of the same curriculum is held. If the assignment isolates the lecture with respect to curricula, the respective penalty is added as many times as there are isolations. On the other hand, in case an isolation of another lecture is removed, the curriculum penalty is subtracted. The room stability penalty is added if lectures of the same course have already been scheduled and none of them takes place in the considered room. Finally, if the required spread over days is not reached yet and no lecture of the same course has been scheduled on the considered day, the penalty for violating the minimum spread is subtracted.

The heuristics' performance can be improved slightly by further encouraging the spread over days. The penalty for violating the spread constraint is added, if another lecture of the same course takes place on the considered day, regardless of the satisfaction of the required spread. The reason for this could be that it is hard for the algorithm to satisfy the minimum spread condition. Penalties resulting from violations of other constraints might be more easily handled. For example, if a destroy operator removes all lectures of the same period these lectures can be rearranged, such that the room related penalties can be reduced without affecting other constraints. Another example would be the removal of all lectures scheduled on the same day in the same room. In this case the repair procedure can focus only on the curriculum compactness without worsening any other penalty. On the contrary, the minimum spread condition cannot be treated separately.

There is no backtracking mechanism implemented in the 1-stage heuristics and the scheduling order does not depend on the number of available insertion positions so far. To avoid too many unscheduled lectures, courses with few periods left for insertion are encouraged by adjusting the evaluation function of the *greedy* heuristic and the computation of the regret value.

In the *greedy* heuristic the penalty for unassigned lectures of the current iteration, computed as stated in subsection 5.1.5, is added to the insertion cost of the considered position if the number of available periods is greater than the number of the course's unscheduled lectures. Therefore, the insertion costs of courses with their number of available periods being less or equal than their number of lectures to schedule tends to be lower than the insertion costs of other courses. Since the penalty for unassigned lectures increases over iterations, this gap gets larger as the algorithm proceeds. Note that the decision of whether the unassigned penalty term should be added depends on the number of available periods instead of room-period pairs. The reason is that lectures of the same course require distinct periods due to conflicts with each other.

In case of the *regret* heuristic an adaption of the evaluation function is not needed since for each decision the k best alternatives are considered. If less than k positions are available the insertion costs of the missing alternatives are represented by the unassigned penalty of the current iteration. Therefore, one has to set k in a way that sufficiently many insertion positions are taken into account to penalize a lack of alternatives appropriately. Here, a 5-regret heuristic is employed.

As suggested by Ropke and Pisinger [2006] two additional noise operators are implemented, denoted by *greedy noise* and *regret noise*. The evaluation of the insertion positions is perturbed in the same way as for the 2-stage operators. Each time an alternative is evaluated a random number drawn from $[-\mu \cdot p_{max}, \mu \cdot p_{max}]$ is added to the insertion cost, where μ denotes a parameter and p_{max} the worst case insertion cost.

It turns out that the reparation phase of the algorithm is typically responsible for approximately two thirds of the total computational effort for most of the ITC-2007 instances. Compared to the other repair heuristics, the regret heuristics require significantly more computation time. Clearly, there is no linear relationship between the problem size and the additional computation time needed by the regret heuristics. For most of the ITC-2007 instances the regret heuristics take on average about three to five times as long as the greedy heuristics to repair a solution and even slightly more compared to 2-stage heuristics.

Two measures are set to overcome this imbalance. First, insertion positions, whose insertion costs are much worse than the actual penalty for unscheduled lectures, are disregarded for the respective courses. In particular, an insertion position is discarded if the capacity penalty of the considered position minus the best possible benefit by scheduling the lecture, i.e. removing compactness penalties of all of its curricula and improving the spread over days, is greater than the penalty for not scheduling the lecture. Consequently, less alternatives have to be sorted and the computation time is reduced. On the other hand, infeasible solutions become more likely.

Pisinger and Ropke [2007] suggest to normalize the operators' scores by their computational effort in case of strongly varying computation times of the employed heuristics, in order to achieve a good trade-off between quality and time. Hence, the second measure is to update the repair heuristics' selection probabilities with respect to their computation times. More precisely, the average computation time of *2-stage best* is treated as a reference computation time. Each time the repair heuristics' weights are updated, the respective score is multiplied by the average computation time $t_{2stagebest}$ of *2-stage best* and divided by the average computation time t_j of the considered repair heuristic j. Using the same notation as in subsection 5.1.2 the formula to compute the weights w_j^{new} is modified to

$$w_j^{new} = w_j^{old} \cdot (1 - r) + r \cdot \frac{\pi_j}{\phi_j} \cdot \frac{t_{2stagebest}}{t_j}$$

Each time a segment's end is reached, the average computation times are updated by computing

$$t_j^{new} = t_j^{old} \cdot (1 - r) + r \cdot \frac{\tau_j}{\phi_j}$$

where r denotes the same reaction factor as before, t_j^{new} denotes the new average computation time of heuristic j, t_j^{old} denotes the previous average computation time of operator j and τ_j denotes the sum of the computation times of operator j in the last segment. As before, ϕ_j denotes the number of calls of operator j in the last segment. As a consequence, the selection probabilities of the regret heuristics are reduced.

6 Computational Experiments

In this section computational results are presented. The algorithm is tested on the ITC-2007 CB-CTT instances that are described in section 3 and compared with other algorithms. Further experiments include the analysis of the importance of each operator with regard to the solution quality and the effects of disabling certain features of the algorithm. The plots are generated with Octave v.3.6.2.

6.1 Parameter Tuning

The algorithm incorporates several parameters. The parameter tuning as well as the selection of operators for the final algorithm are based on the average solution quality of the instances `comp01`,...,`comp14`. It seems reasonable to make decisions on the basis of these instances, since they have been available for the participants of the ITC-2007 for tuning their algorithms. This allows a fair comparison in particular with Müller [2009], the winner of the CB-CTT track of the ITC-2007.

Initial parameter values have been either found during the implementation phase or are borrowed from Ropke and Pisinger [2006]. For setting the parameters appropriately, the algorithm's change in performance is evaluated when altering one value at a time and keeping the others fixed. This is done for all parameters in parallel, though. Typically a slightly greater and a slightly lower value is checked for each parameter. The average penalty over 5 runs on the instances `comp01`,...,`comp14` is computed. The iteration limit basically corresponds to the time limit of the ITC-2007 adjusted to the computational power of the machine. After each parameter value has been evaluated, the parameters are set to the values that performed best. This new setting is the basis for the next round. After the second round there are no significant differences observable, therefore the tuning is stopped at this point. The final parameter setting is given in Table 7.

Parameter	Value	Description
ψ	6%	SA: Initially accept ψ-percent worse solution with 50%
ρ	0.999959	SA: Cooling rate for reference of 200,000 iterations
h	50000	SA: Reheat after h iterations
σ_1	30	ALNS: Score for new global best
σ_2	15	ALNS: Score for new, accepted, better than current
σ_3	18	ALNS: Score for new, accepted, worse than current
s	50	ALNS: Segment size
r	0.1	ALNS: Reaction factor for weight adjustment
α_1	0.35	Infeasibility penalty: Unavailability weight
α_2	0.05	Infeasibility penalty: Conflict weight
d	30%	Destroy limit: Maximum destroy percentage
u	110	Destroy limit: Upper bound destroy events
δ	4	Destroy limit: Decrease parameter
β	1	Relatedness measure: Number of students weight
κ_1	5	Related removal: Selection probability
κ_2	8	Worst removal: Selection probability
κ_3	10	Isolation & capacity removal: Selection probability
κ_4	5	Spread & stability removal: Selection probability
κ_5	3	Curriculum removal: Selection probability
η	1.3	2-stage mean: Factor for penalty-free extra capacity
ν	6	Noise: Priority rule, noise $\in [-\nu, \nu]$
μ	0.04	Noise: Insertion cost, noise $\in [-\mu \cdot p_{max}, \mu \cdot p_{max}]$

Table 7: Parameter setting

It might be more precise to base decisions on the average of 10 runs. Furthermore combinations of different values of several parameters could be evaluated, instead of altering just one at a time. This would take interdependencies between parameters into account, which might be particularly important for parameters that are obviously connected, e.g. the scores for adapting the selection probabilities. However, such an exhaustive tuning would be very time consuming for that many parameters. Moreover, for most of the parameters the algorithm does not react very sensitive on slight alterations of their values. A significant gain in performance is particularly possible by setting the cooling rate carefully, though.

6.2 Results

In this subsection the algorithm's performance on the ITC-2007 CBB-CTT instances is presented. In order to generate comparable results, the algorithm's time limit has to be set according to the benchmarking tool provided on the competition's website[1]. However, since several parameters and functions of the algorithm make use of the iteration limit, the benchmarking tool is used to determine the iteration limit for each instance, which in turn is used as stopping condition. Due to the fact that ALNS incorporates some randomization, the actual computation time of a single run might slightly deviate from the requested time limit.

To set the iteration limits adequately a computer with an AMD Turion X2 Ultra Dual-Core Mobile TM-82x2 processor, 4 GB memory and an Ubuntu 13.10 64-bit operating system is used. Since the benchmarking tool and the algorithm are supposed to run on a single processor machine, one core is switched off, i.e. starting with the boot option `maxcpus=1`. This leads to a time limit of 480 seconds. For determining the iteration limit the 5 runs' average is computed and rounded. Setting the iteration limit according to the average over 5 runs is also done by Clark et al. [2008] who rank fifth on the competition's CB-CTT track. Bellio et al. [2013] use an iteration limit as stopping criterion as well.

Even when using a time limit instead of an iteration limit as stopping criterion, an iteration limit has to be set anyway. Among others, the iteration limit is used to compute the destroy limit in each iteration, which has a significant influence on

[1]`http://www.cs.qub.ac.uk/itc2007/index_files/benchmarking.htm`

the runtime. Consequently, whenever the iteration limit has been determined, it has to be checked whether it coincides with the employed iteration limit. As long as they deviate from each other, the employed iteration limit has to be adjusted and one has to rerun the process.

The final results are generated on a more modern computer with a Intel Core i5-3550 CPU running at 3.30GHz, 8 GB memory and a Linux Mint release 14 64-bit operating system by making use of the previously generated iteration limits. This PC is clearly able to generate results faster. Its new hardware makes it probably inappropriate for determining the iteration limits, though.

The final results on the instances `comp01`,...,`comp21` are shown in Table 8, where ALNS is compared with the algorithms by Abdullah and Turabieh [2012], Bellio et al. [2013], Abdullah et al. [2012] and the two best algorithms of the ITC-2007, i.e. the algorithms by Müller and Lü and Hao. The results are either those of the competition or borrowed from the respective papers, as stated in the table's footnotes.

In column *ALNS avg.* the algorithm's average results over 10 runs with random seeds are presented, while *ALNS best* shows the respective best outcome of these runs. In this context one has to note, that these best results do not necessarily correspond to the overall best results found by ALNS. In particular, according to the CB-CTT website[1] the algorithm found new best solutions for the instances `comp05` and `comp12` during the tuning phase with slightly different run times, though.

The results that correspond to the competition's algorithms and the one by Abdullah et al. [2012] are also averages over 10 runs, while Abdullah and Turabieh [2012] apply 11 runs and Bellio et al. [2013] use 31 runs. Cells that are marked gray indicate that the corresponding algorithm performs best compared to the other ones on the respective instance. The column *Best* refers to the best known solutions, whereas bold numbers indicate proven optimality.

ALNS is superior to the others on six instances and clearly outperforms the best algorithms of the ITC-2007. The algorithm by Abdullah and Turabieh [2012] performs best overall, though. The algorithm is typically able to find the optimal solution for the instances `comp01` and `comp11` and occasionally also for `comp04`.

[1]`http://satt.diegm.uniud.it/ctt/` [accessed: 2013-09-02]

Inst.	Abd.[1]	ALNS avg.	ALNS best	Bellio[2]	Müller[3]	Abd.[4]	LüHao[3]	Best[5]
comp01	5.00	5.00	5	5.16	5.00	5.00	5.00	**5**
comp02	36.36	47.10	41	55.93	61.30	53.90	61.20	24
comp03	74.36	75.80	69	80.87	94.80	84.20	84.50	66
comp04	38.45	36.20	35	39.48	42.80	51.90	46.90	**35**
comp05	314.45	311.40	297	340.87	343.50	339.50	326.00	284*
comp06	45.27	54.00	48	55.64	56.80	64.40	69.40	**27**
comp07	12.00	18.10	9	28.68	33.90	20.20	41.50	**6**
comp08	40.82	43.10	40	45.03	46.50	47.90	52.60	**37**
comp09	108.36	105.10	100	106.96	113.10	113.90	116.50	**96**
comp10	8.36	17.10	12	23.26	21.30	24.10	34.80	**4**
comp11	0.00	0.00	0	0.00	0.00	0.00	0.00	**0**
comp12	320.27	326.50	316	337.80	351.60	355.90	360.10	298*
comp13	64.27	67.60	62	74.70	73.90	72.40	79.20	**59**
comp14	64.36	57.00	53	58.51	61.80	63.30	65.90	**51**
comp15	72.73	76.90	69	79.93	94.80	88.00	84.50	66
comp16	23.73	36.50	30	39.54	41.20	51.70	49.10	**18**
comp17	76.36	78.70	72	79.29	86.60	86.20	100.70	**56**
comp18	75.64	70.90	65	80.90	91.70	85.80	80.70	62
comp19	66.82	66.70	60	67.80	68.80	78.10	69.50	**57**
comp20	13.45	38.90	24	47.74	34.30	42.90	60.90	**4**
comp21	100.73	103.40	93	104.19	108.00	121.50	124.70	**74**
Avg.	74.37	77.90	71.43	83.44	87.22	88.13	91.13	63.29

[1] Tabu-based memetic approach, Abdullah and Turabieh [2012]

[2] Simulated annealing, Bellio et al. [2013]

[3] `http://www.cs.qub.ac.uk/itc2007/winner/finalorder.htm`

[4] Multi-start Great Deluge, Abdullah et al. [2012]

[5] `http://satt.diegm.uniud.it/ctt/` [accessed: 2013-09-02]

* New best solution found by ALNS during the tuning phase

Table 8: Average results for the CB-CTT ITC-2007 instances

Statistics about the generated results are presented in Table 9. All numbers correspond to the average over the 10 runs. The column *Limit* refers to the iteration limits of the instances. *Found* shows the iteration when the best solution was found as percentage of the iteration limit. In this context one has to note that the relatively small numbers for `comp01` and `comp11` can be explained by the early findings of the respective optimal solution.

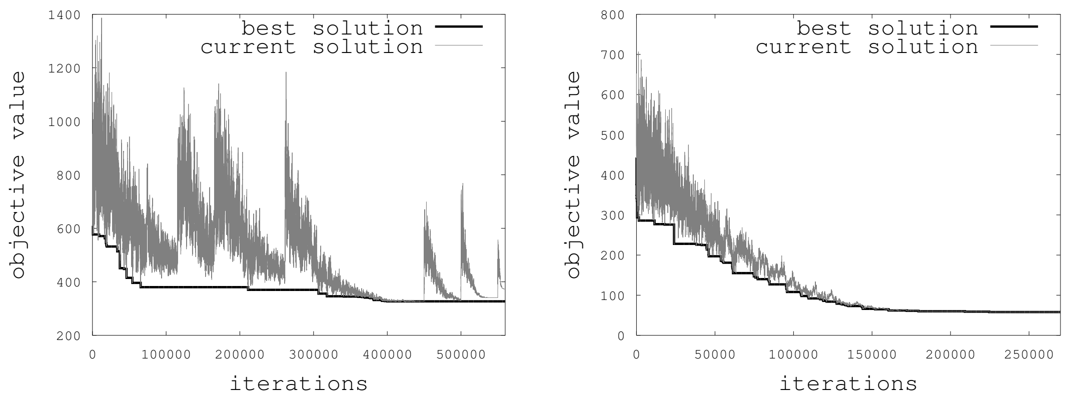
Inst.	Limit	Found	Inf.	Accept	2-stage	Greedy	Regret	Reh.
`comp01`	710k	44.34%	10.01%	47.08%	0.88%	3.71%	34.22%	8.4
`comp02`	310k	77.22%	10.05%	41.89%	11.23%	8.00%	12.83%	1.9
`comp03`	350k	78.04%	5.30%	45.79%	4.90%	4.84%	7.62%	2.3
`comp04`	300k	63.22%	0.64%	45.97%	0.50%	0.67%	0.96%	1.8
`comp05`	560k	67.05%	6.01%	50.00%	5.98%	5.62%	6.90%	5.0
`comp06`	270k	73.05%	3.76%	35.39%	2.59%	3.43%	6.63%	0.9
`comp07`	220k	77.61%	3.96%	43.44%	3.65%	3.12%	5.91%	0.5
`comp08`	260k	65.36%	2.81%	42.64%	2.15%	2.60%	5.03%	1.8
`comp09`	300k	78.33%	2.90%	42.22%	2.15%	2.81%	5.60%	0.6
`comp10`	220k	90.69%	3.95%	42.22%	4.46%	3.40%	4.54%	0.6
`comp11`	190k	18.28%	0.25%	32.81%	0.00%	0.00%	1.11%	2.7
`comp12`	370k	65.87%	2.20%	33.00%	1.09%	1.63%	6.49%	2.2
`comp13`	290k	63.89%	0.43%	43.81%	0.25%	0.47%	0.93%	1.7
`comp14`	280k	60.62%	2.16%	48.16%	2.29%	1.60%	3.42%	1.8
`comp15`	360k	69.18%	5.23%	47.37%	4.93%	4.82%	7.15%	2.4
`comp16`	250k	78.08%	4.13%	43.15%	3.78%	3.87%	5.72%	1.0
`comp17`	230k	70.52%	3.41%	44.75%	3.44%	2.86%	4.61%	0.8
`comp18`	520k	63.05%	0.12%	9.26%	0.01%	0.03%	0.72%	5.2
`comp19`	360k	69.68%	5.04%	42.94%	4.05%	4.78%	8.39%	1.9
`comp20`	210k	83.86%	11.12%	37.79%	12.02%	6.01%	19.14%	0.5
`comp21`	250k	74.38%	6.07%	43.53%	6.09%	5.02%	8.73%	0.9

Table 9: Statistics of the intermediate solutions

Inf. indicates the percentage of produced infeasible solutions. In *Accept* the number of accepted infeasible solutions is given as a percentage of all infeasible solutions. *2-stage*, *Greedy* and *Regret* correspond to the percentage of infeasible solutions generated by 2-stage repair operators, greedy heuristics and regret heuristics, respectively. It is interesting to note that neither the 2-stage repair operators nor the greedy operators are generally superior with respect to the number of generated infeasible solutions. This is particularly remarkable because only the 2-stage repair operators have a backtracking mechanism implemented. Consequently, prioritizing lectures by means of the evaluation function proves to be sufficient to generate mainly feasible solutions for these instances. Unsurprisingly, the regret operators generate more infeasible solutions because it discards highly penalized insertion positions.

Finally, the column *Reh.* shows how often the temperature was reheated. Since the reheating parameter is set independently of the actual iteration limit, only some instances have enough iterations in total to reheat the temperature several times. Still, the feature of reheating the temperature is very useful. Disabling this feature would lead to an deterioration of 3.04% of the solution quality. This comparison is based on the average over all instances with 10 runs each. The iteration limits were adjusted for the algorithm without reheating, however the parameter settings were kept.

Figure 6 shows how reheating affects the developments of the current solution and the best solution. The left figure refers to a run of `comp05`. The peaks of the series indicate reheats and hence the acceptance of worse solutions. The right figure shows a run of `comp06` without reheats.



(a) `comp05` (b) `comp06`

Figure 6: Solution convergence

6.3 Additional Experiments

The tests of this subsection are based on the instances `comp01`,. . . ,`comp21` with 10 runs on each instance. Furthermore, the parameters are always set to the values that are stated in subsection 6.1, even though slight adjustments might be required to achieve the best results. Unless specified otherwise, the same iteration limits are used as in the previous subsection.

In Table 10 operator statistics are listed indicating which operators are essential for a good performance. The column *Selection* presents the average selection frequencies of the operators in percent. *Deter.* indicates the average deterioration of the solution quality, given that the respective operator is removed while keeping all other operators. One has to note that 10 runs per instance might be insufficient to draw conclusions from very small deviations.

Operator	Selection	Deter.
Random	14.91%	0.72%
Rand. penalty	11.91%	1.74%
Rand. period	13.58%	2.90%
Curriculum	2.66%	0.34%
Teacher	5.98%	0.82%
Worst	6.23%	0.92%
Related	10.03%	1.48%
Iso. & cap.	15.21%	2.00%
Spread & stab.	7.06%	2.03%
Room day	12.42%	1.97%
Greatest	19.49%	1.11%
Match	19.85%	1.25%

Operator	Selection	Deter.
2-stage best	26.18%	0.54%
2-stage mean	3.42%	1.41%
Greedy	38.31%	2.36%
Regret	16.19%	1.94%
2-stage best n.	7.47%	1.63%
2-stage mean n.	2.27%	0.90%
Greedy noise	4.31%	1.58%
Regret noise	1.85%	1.25%
SD	14.04%	0.08%
LD	14.48%	1.40%
Random order	10.82%	0.85%

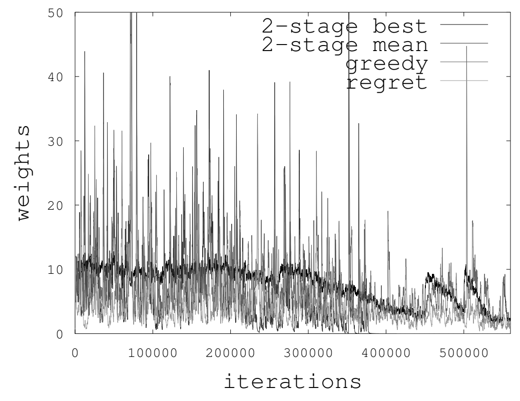
Table 10: Operator statistics

With regard to the destroy operators there is a tendency observable that operators which remove single lectures (e.g. *random penalty*, *isolation & capacity*) are more valuable than the ones that remove whole courses (e.g. *curriculum*, *worst*). Furthermore the operators *random period* and *room day* prove to perform well, as these operators are able to improve the solution with respect to particular soft constraints, while preserving the solution structure regarding other constraints.

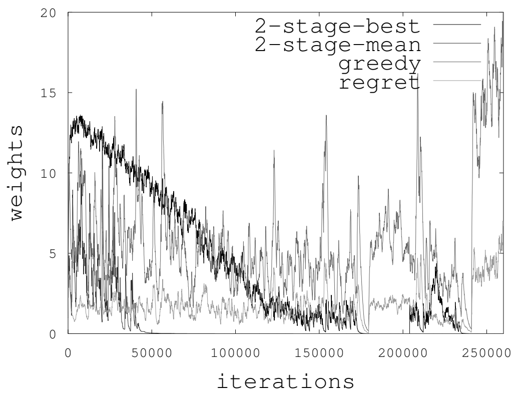
Greedy is the most important repair operator. However, *regret* might generate solutions of even better quality, its selection rate is reduced due to its high computation time, though. On the contrary to the findings of Ropke and Pisinger [2006], adding noise to the evaluation function does not seem to be very critical for the algorithm's performance. Even without the use of any noise operator the deterioration amounts to only 0.67%. The reason for this might be that the different operators lead to a sufficient diversification even without employing additional perturbation. The room assignment operators perform equally well. The priority rule *LD* tends to be more important than the other ones. The smaller selection rate of the *random* rule may be explained by the fact that *SD* and *LD* already incorporate a noise term, which makes a completely random ordering less relevant.

Table 11 compares the average operator selection rates of the instances `comp05` and `comp08`, where the former is characterized by a large number of curricula and many unavailabilities. The results indicate that for problems with different characteristics the selection rates deviate significantly for some operators. For example, the *curriculum* operator is applied more regularly for `comp05`, since resolving curriculum conflicts is probably critical for this highly constrained instance. However, the more evenly distributed selection rates of `comp05` might be partly explained by its large number of reheats, leading to higher acceptance rates and thus to larger scores for all operators.

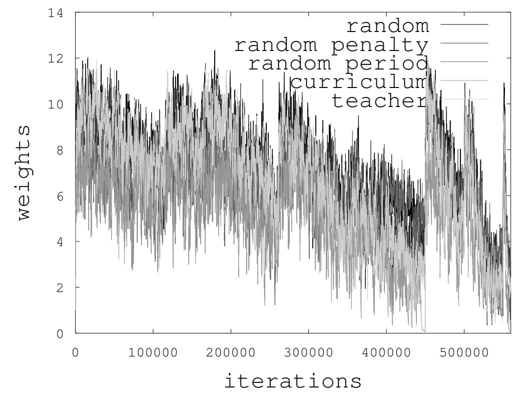
The development of the operators' weights over iterations is shown in Figure 7 for single runs of the instances `comp05` and `comp08`. The series of all operators but the noise repair operators are represented. The peaks of the repair operators' series may be explained by short computation times of the respective operators in some segments, which in turn leads to higher weights. The increase in the destroy operators' weights of `comp05` at some points correspond to reheats, which leads to higher acceptance rates and hence to larger scores. The series of `comp08` show that the weights of some operators converge to zero relatively fast. This is in accordance with the small selection rates of the corresponding operators, shown in Table 11.



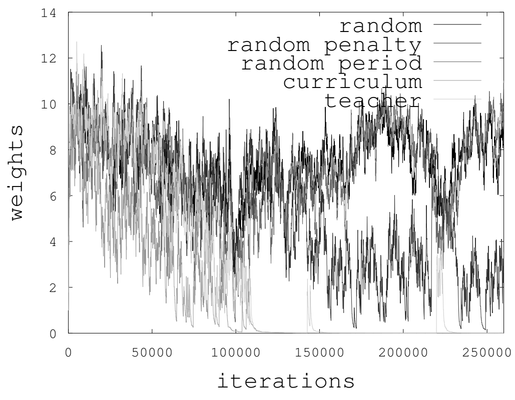
(a) Destroy operators: `comp05`



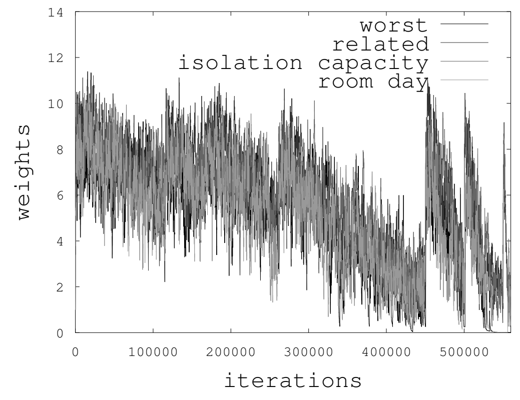
(d) Destroy operators: `comp08`



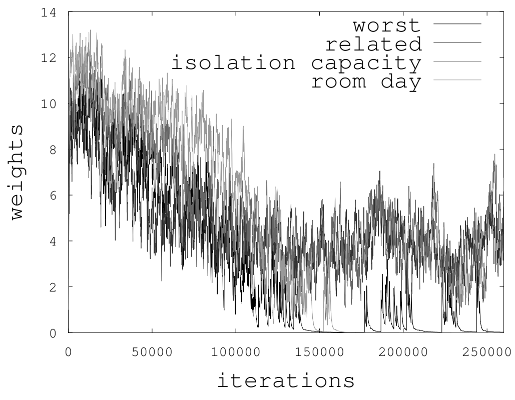
(b) Repair operators: `comp05`



(e) Repair operators: `comp08`



(c) Repair operators: `comp05`



(f) Repair operators: `comp08`

Figure 7: Progression of the operators' weights

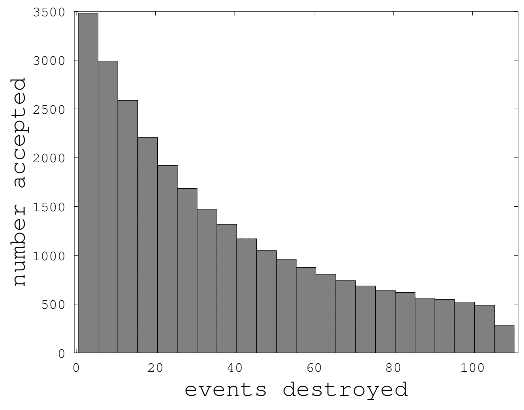
Destroy Operator	`comp05`	`comp08`
Random	12.75%	18.48%
Random penalty	11.74%	10.45%
Random period	11.46%	17.44%
Curriculum	7.46%	2.15%
Teacher	10.98%	4.37%
Worst	7.80%	4.49%
Related	9.81%	10.80%
Iso. & cap.	10.66%	10.41%
Spread & stab.	7.28%	7.38%
Room Day	10.06%	14.03%

Repair Operator	`comp05`	`comp08`
2-stage best	25.64%	25.95%
2-stage mean	8.66%	1.90%
Greedy	23.78%	45.99%
Regret	12.82%	14.23%
2-stage best n.	10.38%	6.38%
2-stage mean n.	5.66%	1.41%
Greedy noise	9.42%	3.23%
Regret noise	3.63%	0.91%

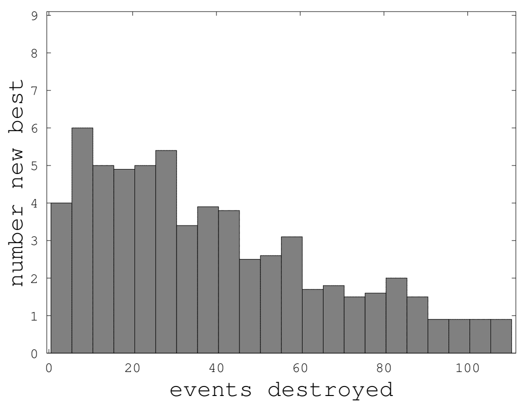
Table 11: Operator selection rates for `comp05` and `comp08`

An ordinary LNS with an uniformly distributed operator selection and adjusted iteration limits performs 1.78% worse than ALNS. However, by removing the noise operators LNS can be improved such that ALNS and LNS perform approximately equally well. This improvement does not come as a surprise, since discarding the noise operators barely affects the average performance of ALNS, as already noted. Furthermore, the noise operators are among those with the smallest selection rate in ALNS and are thus overrepresented in LNS.

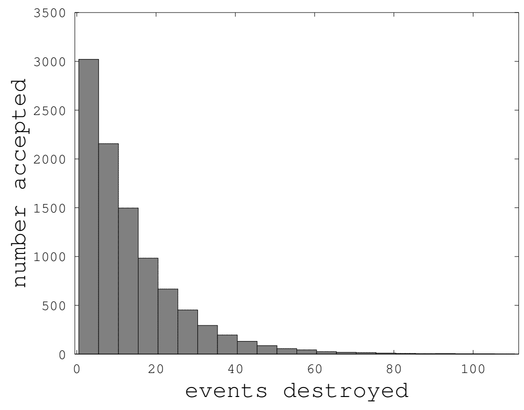
Figure 8 shows the effects of destroying different numbers of lectures with regard to accepted solutions and new best solutions. The histograms are based on the average over 10 runs of `comp06` without reheating and without decreasing the destroy limit. Similar patterns can be observed for other instances. The x-axis of each plot refers to the value that is passed to the destroy operator as the requested number of removals. This value might slightly deviate from the actual number of destroyed lectures, as noted previously. The y-axis indicates either the number of accepted solutions or the number of new best solutions resulting from repairing a partial solution with the respective number of removals. The search is split into segments, each corresponding to one third of the total number of iterations. Histograms are plotted for each segment.



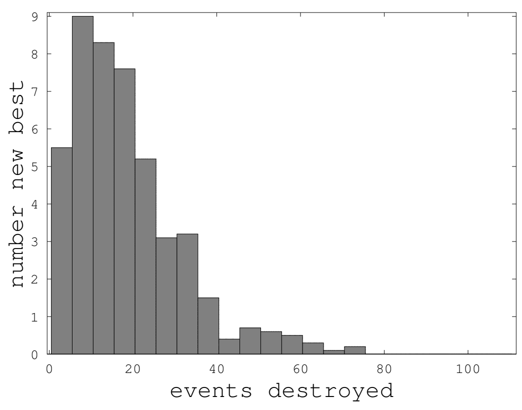
(a) Accepted solutions in $[0, 90000]$



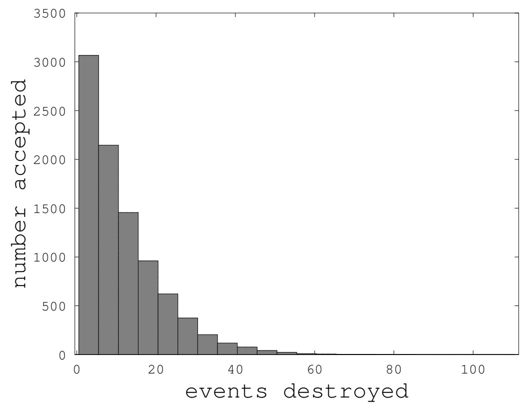
(d) New best solutions in $[0, 90000]$



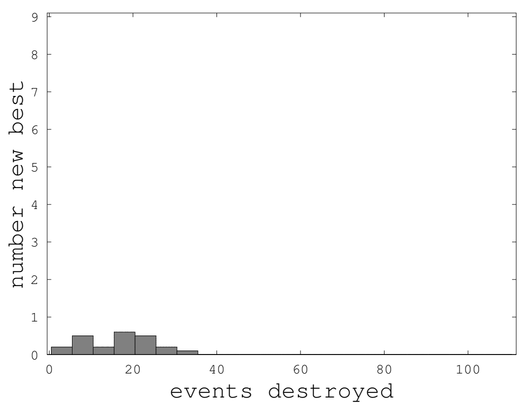
(b) Accepted solutions in $(90000, 180000]$



(e) New best solutions in $(90000, 180000]$



(c) Accepted solutions in $(180000, 270000]$



(f) New best solutions in $(180000, 270000]$

Figure 8: Benefit of different destroy limits, `comp06`

Unsurprisingly, the figures indicate that it is comparatively unlikely that removing a large number of lectures will lead to a new best solution immediately. However, these solutions are also barely accepted in later stages of the search. Therefore, destroying a large number of lectures cannot contribute much to the solution quality as the search proceeds. On the other hand, repairing partial solutions with many unscheduled lectures is relatively costly in terms of computational effort. Consequently the destroy limit is reduced as a function of the iterations.

Discarding the feature of reducing the destroy limit over iterations leads to a deterioration of 7.54%, given that the iteration limits are adjusted and the parameter setting is kept. However, one has to note that in this context it is likely that the algorithm requires a retuning of some parameters. In particular, the parameter d that specifies the maximum destroy percentage might have to be reduced. Furthermore, the reheat parameter h has to be adjusted to the new iteration limits.

The evaluation functions of the *greedy* and *regret* heuristics incorporate an extra penalty for assigning lectures to days with scheduled lectures of the same course. Disabling this feature leads to a deterioration of 0.44%.

Without the measure to improve the *regret* heuristics' computation times, i.e. discarding very weak alternatives, and without taking the computation time for calculating the repair operators' weights into account, the solution quality is on average 1.85% worse, given that the iteration limits are adjusted appropriately.

7 Conclusion

The presented method for solving the *Curriculum-Based Course Timetabling* problem is based on the *Adaptive Large Neighborhood Search* by Ropke and Pisinger [2006]. Additional features are implemented, including a reduction of the destroy limit over iterations, reheating the temperature for simulated annealing, allowing infeasible solutions and taking the repair operators' computation times into account when adjusting their weights. Most notably it turns out that destroying large portions of the solution is less beneficial as the search proceeds. On the contrary to the insights of Ropke and Pisinger, adding perturbation to the evaluation of possible insertion positions does not improve the performance significantly.

The algorithm incorporates several destroy and repair operators. Some of which are adapted versions of the ones that have been used by Ropke and Pisinger [2006], while additional operators tackle the structure of timetabling problems. The destroy operators that have proven to be most effective are those that remove single lectures instead of whole courses and operators that focus on certain constraints while preserving solution characteristics with respect to other constraints.

The function for evaluating lectures' insertion positions performs slightly better in case of an additional encouragement of the spread over days. Perhaps the algorithm's performance can be improved further by identifying favorable attributes of intermediate solutions and adjusting the evaluation function accordingly.

The proposed approach is able to generate competitive results for the benchmark instances of the *second international timetabling competition*. Moreover it outperforms the competition's best algorithms. New best solutions for two instances have been found during the execution of computational tests.

Despite of the algorithm's good performance, it might have shortcomings when it comes to the practical implementation. In case a university specifies desired characteristics of a timetable differently than suggested by the competition's formulation, incorporating additional constraints can be done by manipulating evaluation functions and keeping track of the new solution characteristics. Perhaps additional operators are required to tackle the modified structure in order to exploit the algorithm's full potential, though. Moreover, some parameters might have to be adjusted. All that requires a profound knowledge of the method and possibly overcharges the timetable officer.

References

S. Abdullah and H. Turabieh. On the use of multi neighbourhood structures within a tabu-based memetic approach to university timetabling problems. *Information Sciences*, 191:146–168, 2012.

S. Abdullah, S. Ahmadi, E. Burke, and M. Dror. Investigating Ahuja-Orlin's large neighbourhood search approach for examination timetabling. *OR Spectrum*, 29 (2):351–372, 2007a.

S. Abdullah, S. Ahmadi, E. K. Burke, M. Dror, and B. McCollum. A tabu based large neighbourhood search methodology for the capacitated examination timetabling problem. *Journal of the Operational Research Society*, 58(11): 1494–1502, 2007b.

S. Abdullah, E. K. Burke, and B. McCollum. Using a randomised iterative improvement algorithm with composite neighbourhood structures for the university course timetabling problem. In K. F. Doerner, M. Gendreau, P. Greistorfer, W. Gutjahr, R. F. Hartl, and M. Reimann, editors, *Metaheuristics*, volume 39 of *Operations Research/Computer Science Interfaces Series*, pages 153–169. Springer US, 2007c.

S. Abdullah, H. Turabieh, B. McCollum, and P. McMullan. A hybrid metaheuristic approach to the university course timetabling problem. *Journal of Heuristics*, 18(1):1–23, 2012.

K. Ahuja and J. B. Orlin. A survey of very large-scale neighborhood search techniques. *Discrete Applied Mathematics*, 123(1-3):75–102, 2002.

R. Bellio, L. Di Gaspero, and A. Schaerf. Design and statistical analysis of a hybrid local search algorithm for course timetabling. *Journal of Scheduling*, 15 (1):49–61, 2012.

R. Bellio, S. Ceschia, L. Di Gaspero, A. Schaerf, and T. Urli. Feature-based tuning of simulated annealing applied to the curriculum-based course timetabling problem. Unpublished manuscript, 2013.

Ş. İ. Birbil and S.-C. Fang. An electromagnetism-like mechanism for global optimization. *Journal of Global Optimization*, 25(3):263–282, 2003.

A. Bonutti, F. De Cesco, L. Di Gaspero, and A. Schaerf. Benchmarking curriculum-based course timetabling: Formulations, data formats, instances, validation and results. *Annals of Operations Research*, 194(1):59–70, 2012.

D. Brélaz. New methods to color the vertices of a graph. *Communications of the ACM*, 22(4):251–256, 1979.

S. Broder. Final examination scheduling. *Communications of the ACM*, 7(8): 494–498, 1964.

E. K. Burke, Y. Bykov, J. Newall, and S. Petrovic. A time-predefined approach to course timetabling. *Yugoslav Journal of Operations Research*, 13(2):139–151, 2003.

E. K. Burke, J. Mareček, A. J. Parkes, and H. Rudová. A branch-and-cut procedure for the Udine course timetabling problem. *Annals of Operations Research*, 194 (1):71–87, 2011.

M. W. Carter, G. Laporte, and S. Y. Lee. Examination timetabling: Algorithmic strategies and applications. *Journal of The Operational Research Society*, 47(3): 373–383, 1996.

S. Ceschia, L. Di Gaspero, and A. Schaerf. Design, engineering, and experimental analysis of a simulated annealing approach to the post-enrolment course timetabling problem. *Computers & Operations Research*, 39(7):1615–1624, 2012.

M. Clark, M. Henz, and B. Love. QuikFix— A repair-based timetable solver. In *Proceedings of the Seventh International Conference on the Practice and Theory of Automated Timetabling*, Montreal, Canada, 2008.

D. Connolly. General purpose simulated annealing. *Journal of the Operational Research Society*, 43(5):495–505, 1992.

T. B. Cooper and J. H. Kingston. The complexity of timetable construction problems. In E. Burke and P. Ross, editors, *Practice and Theory of Automated Timetabling*, volume 1153 of *Lecture Notes in Computer Science*, pages 281–295. Springer Berlin Heidelberg, 1996.

D. De Werra. An introduction to timetabling. *European Journal of Operational Research*, 19(2):151–162, 1985.

L. Di Gaspero and A. Schaerf. Multi-neighbourhood local search with application to course timetabling. In E. K. Burke and P. De Causmaecker, editors, *Practice and Theory of Automated Timetabling IV*, volume 2740 of *Lecture Notes in Computer Science*, pages 262–275. Springer Berlin Heidelberg, 2003.

L. Di Gaspero, B. McCollum, and A. Schaerf. The second international timetabling competition (ITC-2007): Curriculum-based course timetabling (track 3). Technical Report QUB/IEEE/Tech/ITC2007/CurriculumCTT/v1.0, Queen's University, Belfast, United Kingdom, 2007.

G. Dueck. New optimization heuristics the great deluge algorithm and the record-to-record travel. *Journal of Computational Physics*, 104(1):86–92, 1993.

S. Even, A. Itai, and A. Shamir. On the complexity of timetable and multicommodity flow problems. *SIAM Journal on Computing*, 5(4):691–703, 1976.

F. Glover and M. Laguna. *Tabu Search*. Kluwer Academic Publishers, Norwell, MA, USA, 1997.

S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

G. Lach and M. E. Lübbecke. Curriculum based course timetabling: New solutions to Udine benchmark instances. *Annals of Operations Research*, 194(1):255–272, 2012.

D. Landa-Silva and J. H. Obit. Great deluge with nonlinear decay rate for solving course timetabling problems. In *Proceedings of the 2008 IEEE Conference on Intelligent Systems (IS 2008)*, pages 8.11–8.18. IEEE press, 2008.

R. Lewis. A survey of metaheuristic-based techniques for university timetabling problems. *OR Spectrum*, 30(1):167–190, 2008.

R. Lewis, B. Paechter, and B. McCollum. Post enrolment based course timetabling: A description of the problem model used for track two of the second international timetabling competition. Cardiff Working Papers in Accounting and Finance A2007-3, Cardiff Business School, Cardiff University, 2007.

H. Lourenço, O. Martin, and T. Stützle. Iterated local search. In *Handbook of Metaheuristics*, pages 321–353. Springer New York, 2003.

Z. Lü and J.-K. Hao. Adaptive tabu search for course timetabling. *European Journal of Operational Research*, 200(1):235–244, 2010.

B. McCollum, P. McMullan, E. K. Burke, A. J. Parkes, and R. Qu. The second international timetabling competition: Examination timetabling track. Technical Report QUB/IEEE/Tech/ITC2007/Exam/v4.0/17, Queen's University, Belfast, 2007a.

B. McCollum, P. McMullan, B. Paechter, R. Lewis, A. Schaerf, L. Di Gaspero, A. Parkes, R. Qu, and E. Burke. Second international timetabling competition, 2007b. URL `http://www.cs.qub.ac.uk/itc2007/`. [accessed 2013-09-21].

B. McCollum, A. Schaerf, B. Paechter, P. McMullan, R. Lewis, A. J. Parkes, L. Di Gaspero, R. Qu, and E. K. Burke. Setting the research agenda in automated timetabling: The second international timetabling competition. *INFORMS Journal on Computing*, 22(1):120–130, 2010.

N. Mladenović and P. Hansen. Variable neighborhood search. *Computers & Operations Research*, 24(11):1097–1100, 1997.

T. Müller. ITC-2007 solver description: A hybrid approach. *Annals of Operations Research*, 172(1):429–446, 2009.

B. Paechter, L. M. Gambardella, and O. Rossi-Doria. First internationl timetabling competition, 2002. URL `http://www.idsia.ch/Files/ttcomp2002/`. [accessed 2013-09-21].

S. Petrovic and E. Burke. University timetabling. In J. Y.-T. Leung, editor, *Handbook of Scheduling: Algorithms, Models, and Performance Analysis, chapter 45*. Chapman Hall/CRC Press, 2004.

D. Pisinger and S. Ropke. A general heuristic for vehicle routing problems. *Computers & Operations Research*, 34(8):2403–2435, 2007.

D. Pisinger and S. Ropke. Large neighborhood search. In M. Gendreau and J.-Y. Potvin, editors, *Handbook of Metaheuristics*, volume 146 of *International Series in Operations Research & Management Science*, pages 399–419. Springer US, 2010.

G. Post, L. Di Gaspero, J. H. Kingston, B. McCollum, and A. Schaerf. The third international timetabling competition. *Annals of Operations Research*, pages 1–7, 2013.

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

S. Ropke and D. Pisinger. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation Science*, 40(4): 455–472, 2006.

A. Schaerf. A survey of automated timetabling. *Artificial Intelligence Review*, 13 (2):87–127, 1999a.

A. Schaerf. Local search techniques for large high-school timetabling problems. *IEEE Transactions on Systems, Man, and Cybernetics— Part A: Systems and Humans*, 29(4):368–377, 1999b.

A. Schaerf, L. Di Gaspero, S. Ceschia, and T. Urli. Timetabling research group at the University of Udine, Italy, 2004. URL `http://satt.diegm.uniud.it/`. [accessed 2013-09-02].

G. Schrimpf, J. Schneider, H. Stamm-Wilbrandt, and G. Dueck. Record breaking optimization results using the ruin and recreate principle. *Journal of Computational Physics*, 159(2):139–171, 2000.

P. Shaw. Using constraint programming and local search methods to solve vehicle routing problems. In M. Maher and J.-F. Puget, editors, *Principles and Practice of Constraint Programming - CP98*, volume 1520 of *Lecture Notes in Computer Science*, pages 417–431. Springer Berlin Heidelberg, 1998.

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

D. C. Wood. A system for computing university examination timetables. *The Computer Journal*, 11(1):41–47, 1968.

Abstract

The task of generating timetables for universities consists of assigning courses involving teachers and students to periods and rooms. Additionally certain constraints have to be satisfied depending on the particular problem. *Curriculum-based Course Timetabling* (CB-CTT) is a variant of university course timetabling, which in turn is a category of educational timetabling. The main characteristic of CB-CTT is, that courses of the same curriculum have students in common. Consequently these courses must not be scheduled at the same time.

Typically, universities define their own requirements on a timetable. As a result, algorithms were designed for specific problems of single universities. Consequently the algorithms' performances were hard to compare. Therefore, the *international timetabling competitions* (ITC) in 2002 and 2007 tried to build a common ground for comparison by defining simplified problem formulations and releasing benchmark instances.

Generating university timetables is typically NP-hard. Therefore, it requires heuristic approaches to solve large problems in reasonable time. The solution method presented in this theses is based on *Adaptive Large Neighborhood Search* (ALNS). In each iteration a relatively large fraction of the solution is destroyed and subsequently repaired. The algorithm makes use of several destroy operators. The resulting neighborhoods can be explored by different repair operators. The selection of the destroy and repair operators is based on their performance in previous iterations. As a result, the algorithm adapts itself to the particular problem instance. New solutions are accepted according to *Simulated Annealing*.

ALNS proves to be very effective for CB-CTT. The algorithm generates competitive results for the benchmark instances. In particular, ALNS outperforms the best algorithms of the ITC-2007.

Zusammenfassung

Bei der Erstellung der Stundenpläne für Universitäten besteht die Aufgabe darin, Kurse, die Vortragende und Studenten betreffen, Perioden und Räume zuzuweisen. Zusätzlich müssen problemabhängige Nebenbedingungen berücksichtigt werden. *Curriculum-based Course Timetabling* (CB-CCT) kann als Variante des *University Course Timetabling* verstanden werden, das wiederum *Educational Timetabling* zuzuordnen ist. Die besondere Charakteristik von CB-CCT ist jene, dass Kurse desselben Curriculums teilweise von denselben Studenten besucht werden und deshalb nicht zeitgleich stattfinden dürfen.

Üblicherweise hat jede Universität ihre eigenen Anforderungen an deren Stundenpläne. Aus diesem Grund wurden ursprünglich Algorithmen speziell für das vorliegende Problem der jeweiligen Universität entwickelt. Entsprechend war ein Vergleich der Algorithmen im Bezug auf deren Leistung kaum möglich. Die *International Timetabling Competitions* (ITC) von 2002 und 2007 versuchten deshalb mittels vereinfachter Problemformulierungen und Vergleichsinstanzen eine allgemeine Basis zu begründen.

Stundenpläne für Universitäten zu erstellen ist typischerweise NP-schwer. Deshalb werden heuristische Verfahren benötigt, um große Probleme in vernünftiger Zeit lösen zu können. Die hier vorgestellte Lösungsmethode basiert auf *Adaptive Large Neighborhood Search* (ALNS). Hierbei wird in jeder Iteration ein relativ großer Teil der Lösung zerstört und anschließend wieder repariert. Der Algorithmus verwendet mehrere Zerstörungsoperatoren. Die resultierende Umgebung der Teillösung kann durch verschiedene Reparaturoperatoren untersucht werden. Die Auswahl der Zerstörungs- und Reparaturoperatoren basiert auf dem Erfolg der Operatoren in früheren Iterationen. Der Algorithmus kann sich dadurch an die jeweilige Probleminstanz anpassen. Neue Lösungen werden anhand von *Simulated Annealing* akzeptiert.

ALNS erweist sich als leistungsstark für CB-CCT. Der hier beschriebene Algorithmus führt zu konkurrenzfähigen Ergebnissen bei den Vergleichsinstanzen. Insbesondere übertreffen die Resultate jene der besten Algorithmen der ITC-2007.

Curriculum Vitae

Personal data

Name	Alexander Kiefer
Date of birth	January 11, 1986
Citizenship	Austria

Education

since 10/2011	University of Vienna, Austria Master's program: Business Administration majoring in Production and Logistics Thesis: *Adaptive Large Neighborhood Search for the Curriculum-Based Course Timetabling Problem*
10/2006 - 06/2011	University of Vienna, Austria Bachelor's program: Economics Thesis: *Should Monetary Policy Respond to Asset Prices and Asset Bubbles?* Thesis: *Social Security: Systems and Reforms* Degree: *Bakkalaureus der Sozial- und Wirtschaftswissenschaften*
since 10/2005	Vienna University of Technology, Austria Diploma program Mathematics majoring in Economics and Business
09/2000 - 06/2004	Upper secondary education, BRG Laa/Thaya, Austria graduation with honors general qualification for university entrance

International experience

11/2010	Mines Paris Tech., Paris, France Athens student exchange program
08/2009 - 02/2010	University of Groningen, The Netherlands Erasmus student exchange program

Work experience

10/2011 - 09/2013	Study assistant at University of Vienna, Austria, at chair of Production and Operations Management
08/2010 - 10/2010	Internship at DTU Aqua, National Institute of Aquatic Resources, Copenhagen, Denmark

Additional skills

German	native
English	fluent
Dutch	basic
Russian	basic
Programming	C++, MATLAB
IT skills	LaTeX, MS Office, IBM CPLEX, Xpress, SAP