Ann Oper Res (2017) 252:255–282
DOI 10.1007/s10479-016-2151-2

S.I. : PATAT 2014

Adaptive large neighborhood search for the curriculum-based course timetabling problem

Alexander Kiefer[1] · **Richard F. Hartl**[2] · **Alexander Schnell**[2]

Published online: 4 March 2016

Abstract In curriculum-based course timetabling, lectures have to be assigned to periods and rooms, while avoiding overlaps between courses of the same curriculum. Taking into account the inherent complexity of the problem, a metaheuristic solution approach is proposed, more precisely an adaptive large neighborhood search, which is based on repetitively destroying and subsequently repairing relatively large parts of the solution. Several problem-specific operators are introduced. The proposed algorithm proves to be very effective for the curriculum-based course timetabling problem. In particular, it outperforms the best algorithms of the international timetabling competition in 2007 and finds five new best known solutions for benchmark instances of the competition.

Keywords University courses · Timetabling · Metaheuristics · Adaptive large neighborhood search

1 Introduction

Timetabling problems are eminently relevant in practice. Petrovic and Burke (2004) state that these problems can be found in various fields, including nurse rostering, timetabling of public transport systems, timetabling of sport events and educational timetabling. Schaerf (1999) subdivides educational timetabling into *school timetabling*, *examination timetabling* and *course timetabling* and generally defines the problem as scheduling lectures that involve teachers and students in a prefixed period of time, while taking different constraints into account. Two variants of course timetabling have been formulated for the *second international*

✉ Alexander Kiefer
alexander.kiefer@univie.ac.at

1 Christian Doppler Laboratory for Efficient Intermodal Transport Operations, Department of Business Administration, University of Vienna, Oskar-Morgenstern-Platz 1, 1090 Vienna, Austria

2 Department of Business Administration, University of Vienna, Oskar-Morgenstern-Platz 1, 1090 Vienna, Austria

timetabling competition (ITC-2007), including *Post Enrollment Course Timetabling* (PE-CTT) and *Curriculum-based Course Timetabling* (CB-CTT). The focus of this study is on the latter one. In PE-CTT, conflicts between courses are specified by the enrollment data, while in the case of CB-CTT courses of the same curriculum must not be scheduled at the same time.

Typically, educational timetabling problems are NP-complete, as demonstrated by Cooper and Kingston (1996) in five different ways. Finding feasible course and exam timetables is NP-complete due to a reduction of graph coloring, as described by several authors, e.g., Werra (1985). In this context, lectures are represented by nodes, edges between nodes are introduced if the corresponding lectures belong to the same curriculum and the colors represent different periods in the timetable.

Educational timetabling has been intensively studied. For an overview of solution techniques applied in this field we refer to the surveys by Schaerf (1999) and Qu et al. (2009). Recent developments in educational timetabling are analyzed by Kristiansen and Stidsen (2013). A survey dealing solely with CB-CTT has recently been conducted by Bettinelli et al. (2015).

With regard to metaheuristcs, Lewis (2008) classifies approaches into *one-stage optimization algorithms*, *two-stage optimization algorithms* and *algorithms that allow relaxations* and explains their strengths and weaknesses. According to the author, neither principle is generally superior. One-stage algorithms tackle hard and soft constraints at once, typically by making use of a weighted sum objective function with sufficiently high penalties for hard constraint violations. Two-stage techniques decompose 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. Algorithms that allow relaxations refer to approaches where some features of the problem are relaxed. Violations of the hard constraints are however prohibited. These algorithms are thereby able to improve the solution with regard to the soft constraints. Additionally, the relaxations need to be removed during the search. This class of algorithms generally refers to two types of relaxations: Events that cannot be scheduled in a feasible way are either left temporarily unscheduled or scheduled at artificial extra slots.

Bellio et al. (2012, 2016) proposed one-stage *simulated annealing* approaches. Müller (2009) used a two-stage hybrid approach. His algorithm is the winner of the ITC-2007 tracks about CB-CTT and exam timetabling and rank fifth on the PE-CTT track, as stated on the website of the competition.[1] In its improvement phase a *hill climbing* algorithm, a *great deluge* algorithm and a *simulated annealing* approach alternate. The second-ranking algorithm of the CB-CTT track proposed by Lü and Hao (2010) is a two-stage algorithm that combines an intensification phase and a diversification phase based on *iterated local search* in the second stage. The two-stage approach by Abdullah et al. (2012) employs a multi-start great deluge algorithm with an electromagnetic-like mechanism. Abdullah and Turabieh (2012) applied a tabu-based memetic approach.

A metaheuristic based on *adaptive large neighborhood search* (ALNS) is presented in this paper. ALNS, originally proposed by Ropke and Pisinger (2006) for tackling vehicle routing problems, is based on iteratively destroying and subsequently repairing relatively large fractions of an incumbent solution. It is an extension of *large neighborhood search* (LNS) proposed by Shaw (1998) and closely related to the *ruin and recreate* method by Schrimpf et al. (2000). 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

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

searched explicitly in reasonable time, whereas the latter characteristic also holds for LNS. Other authors have also employed large neighborhood structures for university timetabling problems, e.g., Abdullah et al. (2007). In the field of high school timetabling, approaches based on ALNS have been proposed by Sørensen et al. (2012), Sørensen and Stidsen (2012) and Kristiansen et al. (2013).

The contributions of this paper are the following: a state-of-the-art metaheuristic based on ALNS is developed for the ITC-2007 formulation of CB-CTT. The proposed variant of ALNS has the distinctive feature of decreasing the upper bound of destruction gradually over time. It is documented that this feature is favorable in ALNS. The algorithm is able to generate competitive results for the ITC-2007 CB-CTT benchmark instances. Moreover, new best known solutions have been found for five instances. The outline is as follows. Section 2 provides a brief description of the ITC-2007 formulation of CB-CTT, a mixed integer programming model, and results for the ITC-2007 benchmark instances generated by CPLEX. The proposed solution approach is presented in Sect. 3. Computational results for the ITC-2007 benchmark instances are shown in Sect. 4. In Sect. 5 the contribution of the different operators and the effects of the features of the algorithm are discussed, while Sect. 6 concludes.

2 Problem description

2.1 Timetabling competitions

The first international timetabling competition (ITC-2002) had its emphasis on course timetabling. Within the competition, a problem formulation was stated and benchmark instances were released. Since then, the formulation has been used by several other researchers, as noted by McCollum et al. (2010). One of the objectives of the second competition was to reduce the gap between theory and practice by providing more realistic formulations and benchmark instances that were derived from real world problems. Each of the three tracks of the competition focused on different problems of university timetabling, including exam timetabling, PE-CTT and CB-CTT. The third international timetabling competition (ITC-2011) was about high school timetabling. The focus of this study is on CB-CTT. In particular, the formulation of the ITC-2007 is used and the proposed algorithm is tested on the ITC-2007 benchmark instances.

The CB-CTT problem consists of scheduling lectures of courses to periods and rooms, as described by Di Gaspero et al. (2007). The working days of a week are split into periods for which a timetable has to be found. For each course the number of lectures and the number of attending students are known, as well as the teacher holding the course. Furthermore, each course might be part of several curricula. Timetables that satisfy a set of hard constraints are called feasible. In the following, the constraints of the problem are briefly described and the respective abbreviation is given in brackets. The hard constraints include that 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*). Any unscheduled lecture counts as one violation, as well as each excessive lecture per room and period. Each pair of conflicting lectures counts as one violation too. Finally, whenever a lecture is scheduled at an unavailable period, i.e., when its teacher is not available, one violation is counted.

The quality of a timetable is measured in terms of soft constraint violations. Whenever a lecture takes place in a room with a capacity less than the number of students of the course, each student exceeding the capacity limit counts as one violation (*RoomCapacity*). Moreover, lectures of the same course should preferably take place in the same room (*RoomStability*). Each additional room corresponds to one violation. Another soft constraint aims at spreading the lectures of each course over a predefined number of working days (*MinWorkingDays*). 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.

2.2 Mathematical model

The presented model for the CB-CTT problem 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 a formulation of the whole problem. A similar model is stated by Burke et al. (2010). The notation used in the model is summarized in Table 1.

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 multiple curricula and is taught 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 taught 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

Table 1 Notation of the mathematical model

Symbol	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

other hand, $A_{(p,d)}$ denotes the set of courses that can be scheduled at period p on day d. The capacity of the rooms and the number of students of the courses are 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 respective superscript.

The timetable is represented by the binary variable $x_{c,d,p,r}$, which takes the value 1 if a lecture of course c is scheduled at period p on day d in room r. It 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. The decision variables are defined as follows.

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

$$v_{cu,p,d} = \begin{cases} 1 & \text{if curriculum } cu \text{ has an isolated lecture at time } (p,d) \\ 0 & \text{otherwise} \end{cases}$$
$$\forall\, cu \in CU,\ p \in P,\ d \in D$$

$$y_{c,r} = \begin{cases} 1 & \text{if course } c \text{ has at least one lecture in room } r \\ 0 & \text{otherwise} \end{cases} \qquad \forall\, c \in C,\ r \in R$$

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

$$w_c \quad \text{number of days less than } m_c, \text{ integer}, \geq 0 \qquad \forall\, c \in C$$

The weighted sum of the soft constraint penalties is minimized by the objective function (1). The sum of the variables $x_{c,d,p,r}$ 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 $y_{c,r}$, indicating whether a lecture of course c is held in room r, one can easily compute the number of rooms used by the course. The deviation of this number from 1 times the penalty coefficient p^{STAB} yields the penalty for violations of the constraint *RoomStability*. The penalty term for an insufficient 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 compute the number of isolated lectures weighted by p^{COMP}.

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

The availability constraint is implicitly taken into account. The other hard constraints are formulated as follows.

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

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

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

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

Constraints (2) make sure that all lectures of the courses are scheduled. Due to Constraints (3) each teacher holds at most one lecture at a time. Constraints (4) guarantee that two lectures of the same curriculum are not held in parallel. Finally, Constraints (5) respect that a room can accommodate at most one lecture at a time. The soft constraints are modeled in the following way.

$$\sum_{r\in R, p\in P, (p,d)\notin U_c} x_{c,d,p,r} - z_{c,d} \geq 0 \qquad \forall\, c \in C,\ d \in D \tag{6}$$

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

$$\sum_{\substack{c\in K_{cu}\cap A_{(p,d)}\\ r\in R}} x_{c,d,p,r} - \sum_{\substack{q\in\{p-1,p+1\}, r\in R\\ c\in K_{cu}\cap A_{(q,d)}}} x_{c,d,q,r} - v_{cu,p,d} \leq 0 \qquad \forall\, cu \in CU,\ p \in P,\ d \in D \tag{8}$$

$$\sum_{d\in D, p\in P, (p,d)\notin U_c} x_{c,d,p,r} - M \cdot y_{c,r} \leq 0 \qquad \forall\, c \in C,\ r \in R \tag{9}$$

Constraints (6) link $z_{c,d}$ with $x_{c,d,p,r}$ in a way that $z_{c,d}$ 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 Constraints (7), such that w_c counts the number of days less than the required spread. Constraints (8) are 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. For the first and the last period of each day the second term has to be adapted in a way that the previous and the last period have to be omitted, respectively. Finally, Constraints (9) link $x_{c,d,p,r}$ with $y_{c,r}$, where M denotes a large number. Variable $y_{c,r}$ is set to 1 if at least one lecture of course c is held in room r.

2.3 Results

We applied CPLEX 12.5 to the mixed integer programs of the ITC-2007 CB-CTT benchmark instances on a Linux PC with an Intel Core i7-4770 CPU running at 3.4 GHz and 16 GB memory. The results are shown in Table 2. Either the optimal solution or the best solution found after 24 hours is reported in column *IP*. Note, that this time limit clearly exceeds the one of the ITC-2007. The column *Best* refers to the best known solutions, available on the CB-CTT website.[2] Bold numbers indicate optimal solutions. Besides the best known solutions, also the solution techniques used for generating them as well as the best known lower bounds for the instances are stated on the website. Hence, the optimality of the bold values is either

[2] http://satt.diegm.uniud.it/ctt/ (accessed: 2015-07-01).

Table 2 Results of the integer programs of the ITC-2007 instances

Instance	IP	Best
comp01	**5**	**5**
comp02	94	24
comp03	103	64
comp04	**35**	**35**
comp05	355	284
comp06	89	**27**
comp07	64	**6**
comp08	**37**	**37**
comp09	147	**96**
comp10	26	**4**
comp11	**0**	**0**
comp12	426	294
comp13	78	**59**
comp14	64	**51**
comp15	97	62
comp16	44	**18**
comp17	103	**56**
comp18	85	**61**
comp19	70	**57**
comp20	54	**4**
comp21	144	**74**

proven by the (exact) solution technique itself, or by the match of the lower and the upper bound. CPLEX found optimal solutions for 4 out of 21 instances. In particular, the optimal solutions for the instances comp01, comp04, comp08 and comp11 were found within 35, 2077, 6742 and 8 s, respectively. For some instances the solution found deviates significantly from the best known solution, though.

3 Solution approach

In case of NP-hard problems, where timetabling problems typically belong to, one usually has to resort to (meta-)heuristic methods, particularly when good solutions have to be found in reasonable time. The previously shown results of the exact method underline this statement for the considered problem class. Therefore, an metaheuristic approach for the CB-CTT problem is presented in this section. More precisely, an ALSN approach based on the papers by Ropke and Pisinger (2006) and Pisinger and Ropke (2007) is developed.

3.1 Adaptive large neighborhood search

3.1.1 General description

In LNS an initial solution has to be created first. At each iteration, parts of the incumbent solution are destroyed and subsequently repaired. New solutions are accepted according to a

certain criterion to become the new incumbent solution. The algorithm keeps track of the best solution, i.e., the solution with the least soft penalties among the solutions with the smallest number of unscheduled lectures. In the end the best solution found is returned. ALNS extends LNS in a way that several destroy and repair operators are used and their selection probability is biased towards the best-performing ones. The approach is sketched in Algorithm 1, while essential steps are described in detail in the following paragraphs.

Algorithm 1 Adaptive large neighborhood search

```
input: solution x,
  evaluation function f, segment size s,
  maximum number of removals n^max
operators: weights w = 1,
  computation times t, scores π = 0,
  computation times in last segment τ = 0
best solution x^b = x, iteration i = 0
while time limit not reached do
  roulette wheel selection of destroy and repair
  operators d and r employing w
  draw # lectures to remove n ∈ [1, n^max]
  x' = r(d(x, n))
  if x' accepted then
    x = x'
  end if
  if f(x') < f(x^b) then
    x^b = x'
  end if
  update π_d, π_r, τ_d, τ_r
  i = i + 1
  if i mod s = 0 then
    update t w.r.t. τ
    update w w.r.t. π and t
    π = 0, τ = 0
  end if
  decrease n^max
  if no new x^b for h iterations then
    increase probability of accepting
    worsening solutions, increase n^max
  end if
end while
return x^b
```

The *adaptive* element of ALNS refers to the dynamic weight adjustment mechanism. Initially, each operator has the same selection probability until weights are recomputed. At each iteration, a destroy operator and a repair operator are selected separately by a roulette wheel mechanism. Given that there are k operators with weights $w_i, i \in \{1, \ldots, k\}$, operator j is selected with the probability $w_j / \sum_{i=1}^{k} w_i$. After applying the operators to the incumbent solution the corresponding scores are updated by adding a value σ depending on the solution quality.

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

This scheme encourages operators that find solutions having not been accepted before, in order to direct the algorithm towards unvisited regions of the search space.

The search process of the algorithm is divided into segments of $s = 100$ iterations. Each time the algorithm has performed s iterations, the end of the segment is reached and the weights are recomputed based on the scores achieved in the last segment. In general, the weights of the heuristics are recomputed as a convex combination of the old weight and the average score achieved in the last segment.

It turns out that the reparation phase is the most time-consuming part of the algorithm. Moreover, the computation times vary significantly between heuristics. Consequently, it appears to be particularly important to normalize the scores of the repair operators by their computational effort in order to achieve a good trade-off between quality and time, as sug-

gested by Pisinger and Ropke (2007). Therefore, whenever the weight of repair heuristic j is computed, its average computation time t_j and the average computation time $t_{2stagebest}$ of the reference repair operator *2-stage best* are taken into account. Given the old weight w_j^{old} of repair operator j, the formula to compute its new weight w_j^{new} is given by

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

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. The last fraction is omitted for non-repair operators.

Incorporating the average computation time of a reference operator is motivated by the fact, that the average computation time of an operator may change during the search. Omitting the reference computation time in the formula would consequently result in an imbalanced acquisition of scores over the different stages of the search. The selection of the reference operator itself is generally arbitrary, as this choice barely affects the final operator selection probabilities. We decided to pick the *2-stage best* operator, as this operator is regularly called in any stage of the search and proves to be essential for the performance of the algorithm, as it will be shown in Sect. 5. The operator *2-stage best* itself will be described in Sect. 3.3.1.

At the end of each segment, the average computation times of the repair operators, including *2-stage best*, are updated by computing

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

where τ_j denotes the sum of the computation times of operator j in the last segment, ϕ_j denotes the number of calls of operator j in the last segment and $r \in [0, 1]$ denotes the reaction factor.

3.1.2 Acceptance scheme

Ropke and Pisinger (2006) suggest to embed this algorithm in a *simulated annealing* (SA) framework, developed by Kirkpatrick et al. (1983). A new solution x' is accepted with the probability $\min\left\{1, e^{-(f(x')-f(x))/T_i}\right\}$, where f denotes an evaluation function, x the incumbent solution and T_i the temperature in iteration i. Consequently, all solutions with an objective value less than the one of the incumbent solution are accepted, and partly also solutions with a greater objective value. The starting temperature is defined implicitly, such that in the beginning a solution being ψ-percent worse than the initial solution is accepted with a probability of 50 %. At the end of each iteration the temperature is decreased by a cooling rate, in a way that the temperature reaches a target T_{end} in the last iteration, where T_{end} is passed as a parameter. Similarly to Lewis and Thompson (2015), the cooling rate is calculated for each iteration individually, since a time limit is used as termination criterion. More precisely, the cooling rate ρ_i of iteration i is computed as $\rho_i = (T_{\text{end}}/T_i)^{1/\mu_i}$, where μ_i denotes the expected number of iterations before the search terminates. T_{i+1} is then computed as $T_{i+1} = T_i \cdot \rho_i$.

In order to predict the expected number of remaining iterations, the average total number of iterations ν for a given time limit t_{end} is pre-computed as an average over five runs. The expected number of remaining iterations μ_i is then computed as $\mu_i = \nu \cdot \left(1 - \frac{t_i}{t_{\text{end}}}\right)$, where t_i denotes the time consumed until iteration i. The pre-computed average iteration limit is used, since for the proposed algorithm one can hardly make reliable predictions about the number

of remaining iterations solely on the basis of already performed iterations. One has to note, that in contrast to our approach, Lewis and Thompson (2015) do not rely on a pre-computed average iteration limit when deciding upon the expected number of remaining iterations.

3.1.3 Destroy limit

At each iteration, n lectures are removed from the current schedule to become reinserted, whereas the remaining lectures are fixed. The integer n 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 set to d percent of the total number of lectures. It turns out that the usage of different percentages depending on the instance size is beneficial, i.e., d_s is used for small instances with less than 280 lectures and d_l for larger instances.

The destroy limit is then gradually decreased, in a way that in the last iteration m at most $\frac{1}{\delta}$ of the reference destroy limit n_0^{max} can be destroyed. The function $n^{max}(i)$ that gives the destroy limit for any iteration i has the form $n^{max}(i) = \lfloor n_0^{max} - i^y \rfloor$, resulting in a steep decrease of the destroy limit in the beginning and a flatter decrease in the end of the search. Since $n^{max}(m) = \lfloor \frac{1}{\delta} \cdot n_0^{max} \rfloor$, the value of y can be computed by setting $n_0^{max} - m^y = \frac{1}{\delta} \cdot n_0^{max}$. This leads to the formula

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

where $m = \mu_i + i$ denotes the (expected) iteration limit. The expected number of remaining iterations μ_i is computed as in the previous subsection.

Decreasing the destroy limit considerably increases the number of iterations within a given time limit, since repairing smaller parts of a solution typically requires less computation time. On the other hand, by destroying less lectures one could potentially lose diversification which in turn could outweigh the gain in performance resulting from the larger number of iterations. Consequently, when setting the decreasing parameter δ, this trade-off has to be taken into account. A motivation for decreasing the destroy limit based on computational tests is given in Sect. 5.2.

In the fields of resource-constrained project scheduling and lot-sizing with setup times, respectively, Muller (2009) and Muller et al. (2012) suggested to gradually reduce the parameter that controls the degree of destruction. However, in the terminology of this paper, their respective parameters correspond to the actual number of requested removals rather than the destroy limit.

3.1.4 Temperature reheats

The temperature is reheated, whenever h consecutive iterations have not found a new best solution. For that purpose, the temperature is set in the same way as described in Sect. 3.1.2, but with respect to the solution quality of the incumbent solution. More precisely, the new temperature is again defined implicitly, such that a solution that is ψ-percent worse than the incumbent solution is accepted with 50 % probability. Temperature reheats have been employed by several authors, e.g., Connolly (1992). In addition to temperature reheats the destroy limit is set to its initial level. The combination of the larger destroy limit and the higher acceptance probability of worse solutions helps to escape from local optima. The decreasing speed of the destroy limit and the cooling rate are adjusted to the expected number of remaining iterations.

3.1.5 Infeasible solutions

The algorithm generally allows infeasible solutions and thereby being able to take shortcuts by traversing infeasible regions of the search space. Benefits and drawbacks of permitting infeasible solutions are discussed by Lewis (2008). The repair heuristics try to schedule lectures even if their insertion costs are high. However, when the algorithm has to decide whether an infeasible solution is accepted, the penalty p is added to the objective value for each unassigned lecture. Initially, p is set to $\max(1, p_{avg})$, where p_{avg} denotes the average penalty per lecture of the initial solution. The penalty p is then dynamically adjusted during the search. Whenever a feasible solution is accepted, the penalty p is set to $\max(1, p^{old}/\alpha)$, where α denotes a parameter and p^{old} the previous value of p. Conversely, whenever a solution with unassigned lectures is accepted, p is set to $\min(p^{old} \cdot \alpha, p_{max})$, where p_{max} denotes the worst case penalty for scheduling a lecture. p_{max} is computed as

$$p_{max} = p^{STAB} + \max_{c \in C} \left(\max_{r \in R} \left(p^{\text{CAP}} \cdot s_{cr} \right) + p^{\text{COMPACT}} \cdot |\{cu \in CU : c \in K_{cu}\}| \right)$$

where the same notation is used as described in Table 1. The penalty that corresponds to *MinWorkingDays* is omitted, since scheduling a lecture cannot worsen the objective value with respect to this constraint.

Bellio et al. (2012) use the same adjustment scheme for the penalty that corresponds to violations of the *Conflicts* constraint with different bounds, though. This method has been proposed by Gendreau et al. (1994). In general, the penalty is updated if a number of consecutive solutions is feasible or infeasible, respectively.

3.2 Destroy operators

3.2.1 Related removal

The *related* operator is inspired by Shaw (1998) and aims to remove similar lectures. The relatedness measure between lectures of 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} the number of curriculum and teacher conflicts between the courses i and j, g_i 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 ones.

The operator is described in Algorithm 2, where the function $c(b)$ in line 8 maps a lecture b to its course $c(b)$. The set of lectures that are going to be removed B is initialized with a random scheduled lecture. As long as the cardinality of this set is less than the requested number of removals, a lecture b is randomly drawn from 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 $\lfloor |A| \cdot \upsilon^{\kappa} \rfloor$, where υ denotes a random number in $[0, 1)$. The probability of selecting very related lectures is controlled by the parameter κ. If κ is large it is likely to select the most

related lecture. On the other hand, if $\kappa = 1$ each lecture has the same selection probability. The drawn lecture is then added to B and the process is repeated.

Algorithm 2 Related removal

input: requested number of removals n, schedule S, parameter κ
set F containing all lectures in S
select random lecture $f \in F$
set of lectures to remove $B = \{f\}$
$F = F \setminus \{f\}$
while $|B| < n$ **do**
  select random lecture $b \in B$
  list $A = F \setminus \{f \in F | c(f) = c(b)\}$
  sort A in descending order w.r.t. relatedness to b
  draw random number $\upsilon \in [0, 1)$
  $a = A[\lfloor |A| \cdot \upsilon^{\kappa} \rfloor]$
  $B = B \cup \{a\}$, $F = F \setminus \{a\}$
end while
remove all lectures in B from S

3.2.2 Random removal

The *random* destroy operator removes lectures from the schedule at random. Ropke and Pisinger (2006) employ a random removal heuristic as well.

3.2.3 Worst removal

As suggested by Ropke and Pisinger (2006) the *worst* destroy operator, shown in Algorithm 3, aims at removing highly penalized assignments, since reinserting these events may be beneficial. Due to interdependencies between lectures, it might be hard to associate penalties with single lectures. Consequently, the heuristic operates at the level of courses. The association of penalties with courses can be done straightforward for violations of *RoomCapacity*, *MinWorkingDays* and *RoomStability*. The penalty for *IsolatedLectures* is assigned to the lecture that is isolated.

Algorithm 3 Worst removal

input: requested number of removals n, schedule S, parameter κ
list of all courses A
sort A by descending penalty
while $n > 0$ **do**
  draw random number $\upsilon \in [0, 1)$
  $a = A[\lfloor |A| \cdot \upsilon^{\kappa} \rfloor]$
  $A = A \setminus \{a\}$
  remove all lectures of a from S, l_a: number of removed lectures
  $n = n - l_a$
end while

The probability of selecting the most penalty-causing course for removal is again controlled by parameter κ. The same parameter will also be used for similar destroy operators described in the following subsections. In terms of removals each destroyed course counts as much as its number of scheduled lectures. Since whole courses are destroyed, it is likely that the requested number of removals is occasionally exceeded. In general, the limit might be exceeded whenever destroy operators are applied that remove multiple lectures at once.

3.2.4 Random penalty removal

The *random penalty* destroy operator randomly selects lectures of courses that cause penalties and removes them from the schedule. In case all of these lectures are removed and the

requested number of removals is not reached, other lectures are removed up to the limit at random.

3.2.5 Random period removal

The *random period* destroy operator repetitively selects a day-period pair at random and removes all its scheduled lectures. 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.

3.2.6 Room day removal

The *room day* destroy operator repetitively removes all lectures that are assigned to a randomly selected room on a randomly selected day. Removing all lectures from a particular room-day allows 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.

3.2.7 Isolation and capacity removal

The *isolation and capacity* destroy heuristic is very similar to the *worst* removal operator. Instead of removing whole courses, individual lectures are selected for being removed, where only their capacity and compactness penalties are considered.

3.2.8 Spread and stability removal

The *spread and stability* operator is essentially the same as the *worst* removal operator. The ordering of the removable courses is based only on the penalties for violating the spread over days and the room stability, though.

3.2.9 Curriculum removal

The *curriculum* operator is similar to the *worst* removal operator, however curricula are selected instead of courses. The removable curricula are sorted in descending order with respect to their curriculum compactness penalties. This operator aims at reducing the compactness penalties. Furthermore, the operator might help to move lectures to periods that have been previously forbidden due to curriculum conflicts.

3.2.10 Teacher removal

The *teacher* operator is used to ease restrictions regarding teacher conflicts. Teachers are randomly selected and all of their lectures are removed from the schedule.

3.3 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, while the 1-stage heuristics perform period and room assignments at once. In this subsection and the subsequent one, the term *period* is used instead of *day-period pair* to facilitate readability.

3.3.1 2-Stage repair operators

The procedure that assigns lectures to periods is summarized in Algorithm 4. The algorithm receives a vector as input, where each element corresponds to a course indicating its number of lectures that have to be scheduled. Apart from the initial state where all lectures are unscheduled, these unscheduled lectures have either been removed by the priorly called destroy operator or remained unscheduled in the previous repair phase. Courses with unscheduled lectures are put in a list which is ordered according to a priority rule. Iteratively, the lectures of the course with the highest priority 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. Consequently, the courses where these lectures belong to have unscheduled lectures again and are therefore reinserted into the list of courses with unscheduled lectures. In order to prohibit cycles, lectures causing the removal of other lectures from a particular day-period pair once, must not remove lectures from the same pair at a later point. The algorithm generally continues as long as there are unscheduled lectures, but terminates in case the remaining lectures cannot be feasibly scheduled and cannot remove lectures from the schedule any more. In the end, the algorithm returns a schedule, i.e., an assignment of lectures to day-period pairs. In case there are still unscheduled lectures, a list of these lectures is also returned. The schedule is then passed to the room assignment operator of the second stage. Details are described in the following paragraphs.

Algorithm 4 2-Stage repair operators

1: input: vector $\boldsymbol{v}$ of lectures to assign
2: vector of unscheduled lectures $\boldsymbol{u} = \mathbf{0}$
3: list of courses $C = \{c : v_c > 0\}$
4: schedule S,
entries $S_p = \emptyset \; \forall$ periods $p \in P$
5: sort C according to priority rule
6: $\forall c \in C$: compute potential insertion positions (periods) P_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: $S_{p_{best}} = S_{p_{best}} \cup \{lecture(c_1)\}$
15: $R_{c_1} = R_{c_1} \setminus \{p_{best}\}$
16: update $P_c \; \forall c \in C$
17: **else if** $R_{c_1} \neq \emptyset$ **then**
18: evaluate all $p \in R_{c_1}$
19: determine best period p_{best}
20: conflicting courses K in p_{best}
21: $S_{p_{best}} = (S_{p_{best}} \setminus lectures(K)) \cup \{lecture(c_1)\}$
22: $v_k = v_k + 1 \; \forall k \in K$
23: append K at beginning of C
24: $R_{c_1} = R_{c_1} \setminus \{p_{best}\}$
25: update $P_c \; \forall c \in C$
26: **else**
27: $u_{c_1} = u_{c_1} + 1$
28: **end if**
29: $v_{c_1} = v_{c_1} - 1$
30: **end while**
31: $C = C \setminus \{c_1\}$
32: **if** saturation degree rule **then**
33: reorder C according to rule
34: **end if**
35: **end while**
36: check if any unscheduled lecture can be scheduled due to removed lectures
37: **return** $\boldsymbol{u}, S$

Priority Rules The priority rules employed in Algorithm 4 are either *saturation degree* (*SD*), *largest degree* (*LD*) or *random*. The rule selection is based on a roulette wheel principle by using adaptively adjusted weights, as previously described. The *LD* rule employed by Broder (1964) prioritizes courses with the largest number of conflicts with other courses. The *SD* rule proposed by Brélaz (1979) arranges courses in ascending order with respect to their number

of available periods for scheduling. The *random* rule mentioned by Carter et al. (1996) orders courses randomly. In order to diversify the outcome, a random number drawn from $[-\nu, \nu]$, where ν denotes a parameter, is added to the priority value whenever *SD* or *LD* are used.

Lecture-period assignment Two mechanisms for evaluating insertion positions are implemented. The *2-stage best* heuristic evaluates all conflict-free insertion periods for a lecture of the course that is next in line for being scheduled in the following way. If assigning the lecture to the considered period will add or remove isolations with respect to curricula, the curriculum compactness penalty is added or subtracted accordingly. Whenever the required spread over days of the 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, the penalty for violating the room stability is added. Finally, the capacity penalty is roughly estimated by assuming that if the lecture has the xth most students of all courses that are assigned to the period in the part of the schedule that is under construction, it will get the xth largest available room in the second stage. The capacity penalty then corresponds to the excess number of students. Ties between the lowest cost insertion positions are broken randomly, as it is done for the other repair operators.

The *2-stage mean* heuristic differs only in the treatment of the capacity penalty. A reference utilization of the room capacities u is computed 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 corresponds to the number of students of the assigned lectures exceeding the capacity limit of the considered period.

For diversification purposes, 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. Each time a period is evaluated a noise value is added to the insertion cost that 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.

Backtracking procedure In case a course is next in line that cannot be scheduled in a conflict-free way, a backtracking mechanism is applied. The implemented backtracking mechanism is similar to the one described by Carter et al. (1996). The aim is to induce the least distortions to the current schedule when removing conflicting lectures. Only lectures that do not belong to the fixed part of the current schedule can be removed. Each insertion position where the considered course is allowed to remove lectures from is evaluated. Positions with the smallest number of conflicting lectures that do not have any alternative conflict-free position left are prioritized. Ties are broken by selecting positions with the smallest number of conflicting lectures in total. In case the procedure is still indifferent, the period with the lowest insertion cost for the considered course is chosen.

The courses of the removed lectures are reinserted in the queue in a way that they are next in line for being scheduled, where courses without any potential conflict-free position are prioritized. However, in case of the *SD* rule the order is dynamically adjusted. To avoid cycles, a course that has a lecture assigned to a period must not remove events from the same period at a later time. In general, this mechanism does not guarantee finding a feasible

solution, though. Hence, there might be unscheduled lectures that have to be passed to the repair operator called next.

Lecture-room assignment In the second stage lectures are assigned to rooms either by the *greatest* or the *match* heuristic. The operator selection is based on the previously described adaptive mechanism. 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 with respect to the room-related penalties. The lecture is then assigned to the room with the lowest insertion cost. Ties are broken by preferring rooms with the larger capacity. The rationale behind this is to keep the smaller rooms for courses with less students, which might be beneficial with regard to the room stability.

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 that are processed later and require large rooms. The second-stage heuristics stop when all lectures scheduled by the first-stage operator have been assigned to rooms.

3.3.2 1-Stage repair operators

A *greedy* and a *regret* heuristic are employed as 1-stage operators, as done by Ropke and Pisinger (2006). For each course, the *greedy* heuristic evaluates all its potential insertion positions. A lecture of the course with the lowest insertion cost, is then scheduled at its best position. On the contrary, the *regret* heuristic decides on the basis of regret values, which lecture is scheduled next. The regret value indicates the opportunity cost for not assigning a lecture to its currently best position. The regret value of a *k-regret* heuristic is computed as the sum of the differences between the best insertion position and the i^{th} best insertion position, $i = 2, \ldots, k$. A lecture of the course with the largest regret value is then scheduled at its best position.

Algorithm 5 1-Stage repair operators

```
1: input: vector v of lectures to assign
2: vector of unscheduled lectures u = 0
3: list of courses C = {c : v_c > 0}
4: schedule S,
   entries S_q for period-room-pairs q
5: ∀c ∈ C: compute potential insertion positions
   (period-room-pairs) Q_c
6: while C ≠ ∅ do
7:    for all c ∈ C do
8:       if Q_c = ∅ then
9:          u_c = v_c
10:         C = C \ {c}
11:      else
12:         evaluate all q ∈ Q_c
13:      end if
14:   end for
15:   determine best course c_best
16:   determine best position q_best
17:   S_{q_best} = S_{q_best} ∪ {lecture(c_best)}
18:   v_{c_best} = v_{c_best} − 1
19:   if v_{c_best} = 0 then
20:      C = C \ {c_best}
21:   end if
22:   update Q_c ∀c ∈ C
23: end while
24: return u, S
```

Each room-period pair is evaluated in the following way. The capacity penalty corresponds to the number of students exceeding the capacity of the room. If inserting a lecture of the con-

sidered course would cause or remove compactness violations, the curriculum compactness penalty is added or subtracted accordingly. 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 has not been 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 performance of the heuristics can be slightly improved by further encouraging the spread over days. The penalty for violating *MinWorkingDays* is added, if another lecture of the same course takes place on the considered day, regardless of the satisfaction of the required spread.

In the *greedy* heuristic the penalty for unassigned lectures of the current iteration, computed as in Sect. 3.1.5, is added to the insertion cost of the considered position if the number of available periods is greater than the number of lectures to assign of the course. Thereby, courses with less or equal potential insertion periods than lectures to schedule are prioritized. In case of the *regret* heuristic, the insertion costs of missing alternatives is set to the unassigned penalty of the current iteration, if less than k positions are available. Consequently, setting k sufficiently large takes the lack of alternatives into account. A 5-regret heuristic is used for this study.

As suggested by Ropke and Pisinger (2006) two additional noise operators are implemented, i.e., *greedy noise* and *regret noise*. 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.

3.4 Initial solution

The initial solution is generated by applying the *2-stage best* heuristic in combination with the *SD* rule and the *greatest* heuristic. The initial solution is not necessarily feasible, however for the ITC-2007 instances a feasible one is typically found.

4 Computational experiments

4.1 Instances

Within the ITC-2007, 21 instances have been proposed for the CB-CTT track, classified into *early*, *late*, and *hidden* ones. The *late* instance set was released two weeks before the deadline of the competition, while the *hidden* instances were released after the closure and were used to rank the best participants. The benchmark instances are available on the website of the competition and are called `comp01`,…,`comp21`. Their characteristics are described by Bonutti et al. (2012) in detail. Best known solutions can be found on the website of the Timetabling Research Group at the University of Udine,[3] where researchers can upload their solutions and lower bounds. Additionally, we apply our algorithm to recently proposed instances being available on the same website, i.e., the sets `DDS`, `EasyAcademy` and `Udine`. Bellio et al. (2016) consider them as candidate benchmark sets for future comparisons.

4.2 Parameter tuning

The algorithm incorporates several parameters, whose setting is given in Table 3. Some of the initial parameter values are taken from Ropke and Pisinger (2006), including the control

[3] http://satt.diegm.uniud.it/.

Table 3 Parameter setting

Parameter	Value	Description
ψ	4 %	SA: initially accept ψ-percent worse solution with 50 %
T_{end}	0.02	SA: target temperature
h	60,000	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 solution
σ_3	18	ALNS: score for new, accepted, worse than current solution
r	0.16	ALNS: reaction factor for weight/time adjustment
α	1.001	Infeasibility penalty: penalty adjustment
d_s	30 %	Destroy limit: small instances
d_l	25 %	Destroy limit: large instances
δ	3	Destroy limit: decrease parameter
β	1	Relatedness measure: number-of-students weight
κ	5	Destroy operators: control selection probability
η	1.3	2-stage mean: control penalty-free excess number of students
ν	6	Noise: priority rule, noise $\in [-\nu, \nu]$
μ	0.04	Noise: insertion cost, noise $\in [-\mu \cdot p_{max}, \mu \cdot p_{max}]$

parameters for the weight adjustment mechanism, i.e., σ_1, σ_2, σ_3, r, and the parameter ψ controlling the start temperature. The remaining initial parameter values have been found during the experiments in the implementation phase. Similarly to Ropke and Pisinger (2006), for setting the parameters appropriately, the 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 larger and a slightly smaller value are checked for each parameter. The average penalty over five runs on the instances `comp01`,…,`comp21` is computed. After each parameter value has been evaluated, the parameters are set to the best-performing values. This new setting is the basis for the next round. The procedure is repeated until no significant differences are observable. This basic tuning method has been chosen, as many parameters have to be tuned. In addition, the algorithm does not react very sensitive on changes of the parameter values. One has to note, that the participants of the ITC-2007 were not able to tune their algorithms on the instances `comp15`,…,`comp21`. Moreover, Bellio et al. (2016) do not use any of the `comp` instances for tuning, but rather a large set of artificial instances.

4.3 Results

The final results presented in this section are generated on a computer with a hardware that has been actual at the time of the competition, i.e., an AMD Turion X2 Ultra Dual-Core Mobile ZM-82 running at 2.2 GHz, 4 GB memory and an Ubuntu 14.04 64-bit operating system. As specified by the competition rules, only one CPU is used. In order to generate comparable results, the time limit is set according to the benchmarking tool provided on the website of the competition. For the stated equipment, the benchmarking tool suggests a time limit of 480 s.

ALNS is used to generate solutions for the `comp` instance set, with ten runs on each instance. Moreover, the algorithm keeps track of the selection rates of each operator. The average selection rates over all runs and each instance are then used as an input for LNS. By

Table 4 ALNS versus LNS with predefined selection probabilities

Instance	LNS predefined prob.		ALNS		Best known[a]
	Average	Best	Average	Best	
comp01	5.0	5	5.0	5	**5**
comp02	41.5	34	44.1	33	24
comp03	71.7	68	75.0	71	64[b]
comp04	35.1	35	35.2	35	**35**
comp05	305.2	294	307.8	292	284[b]
comp06	47.8	41	48.4	39	**27**
comp07	14.5	10	19.3	12	**6**
comp08	41.0	39	41.4	39	**37**
comp09	102.8	100	103.0	100	**96**
comp10	14.3	7	14.9	11	**4**
comp11	0.0	0	0.0	0	**0**
comp12	319.4	306	324.8	310	294[b]
comp13	60.7	59	63.4	60	**59**
comp14	54.1	51	54.9	52	**51**
comp15	72.1	66	74.7	67	62[b]
comp16	33.8	26	36.1	29	**18**
comp17	75.7	67	75.6	63	**56**
comp18	66.9	64	67.5	65	**61**[b]
comp19	62.6	59	66.0	61	**57**
comp20	27.2	19	26.2	21	**4**
comp21	97.0	93	98.5	92	**74**
Average	73.73	68.71	75.32	69.38	62.76

[a] http://satt.diegm.uniud.it/ctt/ (accessed: 2015-07-01)
[b] New best known solution found by the proposed algorithm

employing LNS with predefined operator selection probabilities one can omit the adaptive procedure and thus extra iterations may be executed within the same time. Moreover, the algorithm makes use of the adjusted selection probabilities from the very beginning. While for some instances, separate tuning (yielding instance-specific selection probabilities) could give slightly better results, it will be seen that this robust tuning over all instances will provide excellent results on average. The reheat limit is set to 80,000 for LNS, while the other parameter values are those stated in Table 3.

LNS with predefined selection probabilities proves to be superior compared to ALNS, as shown in Table 4. *Average* refers to the average penalty over ten runs and *best* presents the best result out of these runs. The column *Best known* shows the best known solutions, as stated on the website of the Timetabling Research Group at the University of Udine, whereas bold numbers indicate proven optimality. The proposed algorithm found new best solutions for the instances comp03, comp05, comp12, comp15 and comp18 during the tuning and experimental phases.

The results for the instances comp01,…,comp21 are shown in Table 5, where our approach is compared with the algorithms by Abdullah and Turabieh (2012), Bellio et al. (2016), Abdullah et al. (2012) and the two best algorithms of the ITC-2007, i.e., the algo-

Table 5 Average results for the CB-CTT ITC-2007 instances

Instance	LNS[a]	Abd.&T.[b]	Bellio[c]	Müller[d]	Abd.et al[e]	Lü&Hao[d]
comp01	5.00	5.00	5.23	5.00	5.00	5.00
comp02	41.50	*36.36*	52.94	61.30	53.90	61.20
comp03	*71.70*	74.36	79.16	94.80	84.20	84.50
comp04	*35.10*	38.45	39.39	42.80	51.90	46.90
comp05	*305.20*	314.45	335.13	343.50	339.50	326.00
comp06	47.80	*45.27*	51.77	56.80	64.40	69.40
comp07	14.50	*12.00*	26.39	33.90	20.20	41.50
comp08	41.00	*40.82*	43.32	46.50	47.90	52.60
comp09	*102.80*	108.36	106.10	113.10	113.90	116.50
comp10	14.30	*8.36*	21.39	21.30	24.10	34.80
comp11	0.00	0.00	0.00	0.00	0.00	0.00
comp12	*319.40*	320.27	336.84	351.60	355.90	360.10
comp13	*60.70*	64.27	73.39	73.90	72.40	79.20
comp14	*54.10*	64.36	58.16	61.80	63.30	65.90
comp15	*72.10*	72.73	78.19	94.80	88.00	84.50
comp16	33.80	*23.73*	38.06	41.20	51.70	49.10
comp17	*75.70*	76.36	77.61	86.60	86.20	100.70
comp18	*66.90*	75.64	81.10	91.70	85.80	80.70
comp19	*62.60*	66.82	66.77	68.80	78.10	69.50
comp20	27.20	*13.45*	46.13	34.30	42.90	60.90
comp21	*97.00*	100.73	103.32	108.00	121.50	124.70
Average	73.73	74.37	81.92	87.22	88.13	91.13

[a] Our approach, LNS with predefined operator selection probabilities
[b] Tabu-based memetic approach, Abdullah and Turabieh (2012)
[c] Simulated annealing, Bellio et al. (2016)
[d] http://www.cs.qub.ac.uk/itc2007/winner/finalorder.htm
[e] Multi-start Great Deluge, Abdullah et al. (2012)

rithms by Müller (2009) and Lü and Hao (2010). Their results are either those reported on the website of the competition or taken from the respective papers, as described in the footnotes. One has to note, that Bellio et al. (2016) use an iteration limit being roughly equivalent to the time limit, instead of the actual time limit.

In column *LNS* the average results of LNS over 10 runs with random seeds are presented. The results of the algorithms of the competition and those by Abdullah et al. (2012) are averages over 10 runs too, while Abdullah and Turabieh (2012) apply 11 runs and Bellio et al. (2016) use 31 runs. Italicized values indicate that the corresponding algorithm performs best compared to the other ones on the respective instance. LNS with predefined selection probabilities is superior to the other approaches on twelve instances and clearly outperforms the best algorithms of the ITC-2007.

Our results for the instance sets DDS, EasyAcademy and Udine are presented in Table 6 and are compared solely with the results by Bellio et al. (2016), since few authors have published solutions for these instance sets yet. For this purpose, we retained the parameter setting and the operator selection probabilities resulting from the tuning on the comp instances. The column *LNS avg.* refers to the average outcome over ten runs, while *LNS best* presents the best

Table 6 Results for the new instances

Instance	LNS avg.[a]	Bellio[b]	Best known[c]	LNS best[d]
DDS1	120.60	110.26	**48**	106
DDS2	0.00	0.00	**0**	0
DDS3	0.00	0.00	**0**	0
DDS4	36.00	21.13	**17**	27
DDS5	0.00	0.00	**0**	0
DDS6	7.10	9.87	**0**	4
DDS7	0.00	0.00	**0**	0
EasyAcademy01	65.00	65.26	65	65
EasyAcademy02	0.00	0.06	**0**	0
EasyAcademy03	3.00	2.06	2	2
EasyAcademy04	0.00	0.35	**0**	0
EasyAcademy05	0.00	0.00	**0**	0
EasyAcademy06	5.00	5.13	5	5
EasyAcademy07	0.60	0.48	**0**	0
EasyAcademy08	0.00	0.00	**0**	0
EasyAcademy09	6.30	5.16	4	4
EasyAcademy10	0.00	0.03	**0**	0
EasyAcademy11	0.00	2.90	**0**	0
EasyAcademy12	4.00	4.03	4	4
Udine1	11.10	13.29	**0**	7
Udine2	19.60	19.26	8	16
Udine3	7.70	8.52	**0**	4
Udine4	65.40	66.16	64	64
Udine5	1.60	3.13	**0**	0
Udine6	0.20	0.35	**0**	0
Udine7	1.40	2.03	**0**	0
Udine8	38.60	39.26	31	34
Udine9	26.90	29.84	21	23

[a] Our approach, LNS with predefined operator selection probabilities
[b] Simulated annealing, Bellio et al. (2016)
[c] http://satt.diegm.uniud.it/ctt/ (accessed: 2015-07-01)
[d] Our approach, best result out of ten runs

result out of these runs. In column *Best known* the best known solutions for these instances are shown, where bold numbers indicate proven optimality.

Bellio et al. (2016) propose another set of instances, called Erlangen, which differ significantly from the other instance sets, particularly in their huge problem size. These instances cannot be solved well by our ALNS without retuning.

4.4 Statistics

Statistics about the intermediate solutions for the comp instances generated by ALNS are presented in Table 7. The numbers correspond to averages over 10 runs. The column *It.* refers

Table 7 Statistics about the intermediate solutions

Inst.	It.	Found (%)	Inf. (%)	Accept (%)	2-stage (%)	Greedy (%)	Regret (%)	Reh.
comp01	834,756	52.51	6.00	19.54	1.87	11.53	7.53	7.6
comp02	464,612	79.31	29.92	45.00	26.99	33.34	27.72	2.0
comp03	490,198	69.04	25.17	41.04	22.61	28.49	22.95	2.5
comp04	419,385	73.06	1.69	42.27	1.36	2.25	1.02	1.7
comp05	607,746	63.19	52.59	47.17	48.13	60.21	49.38	4.9
comp06	385,357	76.33	9.54	36.51	8.80	11.42	7.42	0.9
comp07	288,897	75.42	11.24	36.80	9.58	13.88	8.73	0.7
comp08	351,494	74.37	1.69	44.99	1.69	1.78	1.52	1.3
comp09	419,126	66.99	18.03	42.62	13.41	21.90	16.59	2.1
comp10	313,288	78.71	9.46	42.98	11.88	10.08	7.12	0.7
comp11	222,627	21.01	0.00	0.00	0.00	0.00	0.00	2.1
comp12	425,798	69.81	38.63	43.17	34.09	44.20	38.11	1.9
comp13	400,009	72.77	2.35	45.32	1.85	2.83	1.94	1.5
comp14	393,265	74.40	13.11	45.74	12.80	14.48	11.06	1.5
comp15	485,042	74.32	25.17	40.39	22.35	28.66	22.83	2.4
comp16	334,376	85.09	14.32	42.36	15.03	16.14	10.86	0.8
comp17	382,248	71.13	14.82	47.02	14.77	15.91	13.22	1.1
comp18	620,209	52.33	2.60	43.65	1.40	3.89	2.45	5.2
comp19	491,659	72.35	23.93	37.02	18.66	29.41	22.58	2.3
comp20	301,414	77.41	15.84	37.88	18.92	16.47	12.87	0.6
comp21	354,564	79.27	20.56	44.76	21.79	21.44	18.20	0.9

to the number of iterations executed within the time limit. *Found* shows the iteration, when the best solution was found as percentage of the total number of iterations. *Inf.* indicates the percentage of generated 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, with regard to the total number of solutions produced by the respective group of operators. It is interesting to see that apparently the greedy repair operators tend to generate infeasible solutions more frequently than the other operators. Finally, *Reh.* shows the number of temperature reheats.

4.5 Statistical tests

In order to test the statistical significance of the obtained solutions for the `comp` instances, we conducted Friedman's test, where we compared our approach with the other algorithms mentioned in Table 5. The test uses the average results for each instance and yields a p value of 6.9×10^{-9} indicating that the results differ significantly for a critical level of $\alpha = 0.05$. Following Derrac et al. (2011), we then performed a post-hoc analysis by computing adjusted p-values for the Friedman test with Holm's and Hochberg's procedures. LNS is considered as control method being compared to all other algorithms. The average ranks and adjusted p values (Friedman) are stated in Tables 8 and 9, respectively, whereas the same abbreviations

Table 8 Average ranks

Algorithm	Avg. ranks
LNS	1.5476
Abd.&T.	2.0238
Bellio	3.3571
Müler	4.4048
Abd.et al	4.5476
Lü&Hao	5.1190

Table 9 p values

Algorithm	Unadjusted	Holm	Hochberg
Lü&Hao	6.18×10^{-10}	3.09×10^{-9}	3.09×10^{-9}
Abd.et al	2.03×10^{-7}	8.14×10^{-7}	8.14×10^{-7}
Müller	7.47×10^{-7}	2.24×10^{-6}	2.24×10^{-6}
Bellio	0.00172	0.00345	0.00345
Abd.&T.	0.4095	0.4095	0.4095

for the authors are used as in Table 5. The results indicate that there is a significant difference between LNS and most of the other algorithms ($\alpha = 0.05$), except for the one by Abdullah and Turabieh (2012).

5 Sensitivity analysis

Computational results underlying the analysis presented in this section have been computed on the Vienna Scientific Cluster. Therefore, an iteration limit has been used as termination criterion, computed for each instance as an average over five runs with the given time limit. Since ALNS incorporates randomization, the actual computation time of a single run might deviate from the requested time limit. The results are based on the `comp` instances with ten runs per instance. Unless stated otherwise, the same parameter values have been used as for the original version.

5.1 Contribution of the different operators

Operator statistics are listed in Table 10 indicating which operators are essential for a good performance of ALNS. The column *Selection* presents the average selection frequencies of the operators in percent. *Deter.* shows the average deterioration of the solution quality, given that the respective operator is removed while keeping all other operators and using the same iteration limit as for the original algorithm.

Apparently, all operators are useful as omitting them gives worse results, on average. With regard to the destroy operators *isolation and capacity* and *random period* contribute the most. *2-stage best* and *regret* are the most important repair operators. The selection rate of *regret* may be reduced due to its high computational effort. The room assignment procedures and the priority rules perform similarly.

Table 10 Operator statistics

Operator	Selection %	Deter. %
Random	15.19	0.40
Rand. penalty	12.21	0.60
Rand. period	13.33	1.30
Curriculum	2.75	0.87
Teacher	5.75	0.19
Worst	6.13	0.27
Related	9.86	0.77
Iso. and cap.	15.56	2.23
Spread and stab.	7.09	1.02
Room day	12.14	0.47
2-Stage best	22.44	4.68
2-Stage mean	1.53	1.08
Greedy	39.87	1.84
Regret	25.32	2.55
2-Stage best n.	4.69	0.54
2-Stage mean n.	0.84	0.78
Greedy noise	3.18	0.90
Regret noise	2.13	0.23
Greatest	14.64	0.64
Match	14.86	0.86
SD	10.65	0.73
LD	10.82	1.05
Random order	8.03	0.86

5.2 Effect of decreasing the destroy limit

Figure 1 shows the effects of destroying different numbers of lectures with regard to accepted solutions and new best solutions. The histograms are based on ALNS without reheating and without decreasing the destroy limit applied to `comp06`. Similar patterns can be observed for other instances, though. The x-axis of each plot refers to the value that is passed to the destroy operator as the requested number of removals. The y-axis indicates either the average number of accepted solutions or the average number of new best solutions resulting from repairing a partial solution with the respective number of removals. The search is split into segments of one third of the total number of iterations. Histograms are plotted for each segment.

While ALNS approaches in the literature typically keep the destroy limit constant, Fig. 1 gives a clear indication that this is not the best choice. Indeed, the figures show that it is rather unlikely that removing a large number of lectures will lead to a new best solution immediately. Moreover, these solutions are 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. In addition, repairing partial solutions with many unscheduled lectures is relatively costly in terms of computational effort. Consequently, the destroy limit is reduced over iterations.

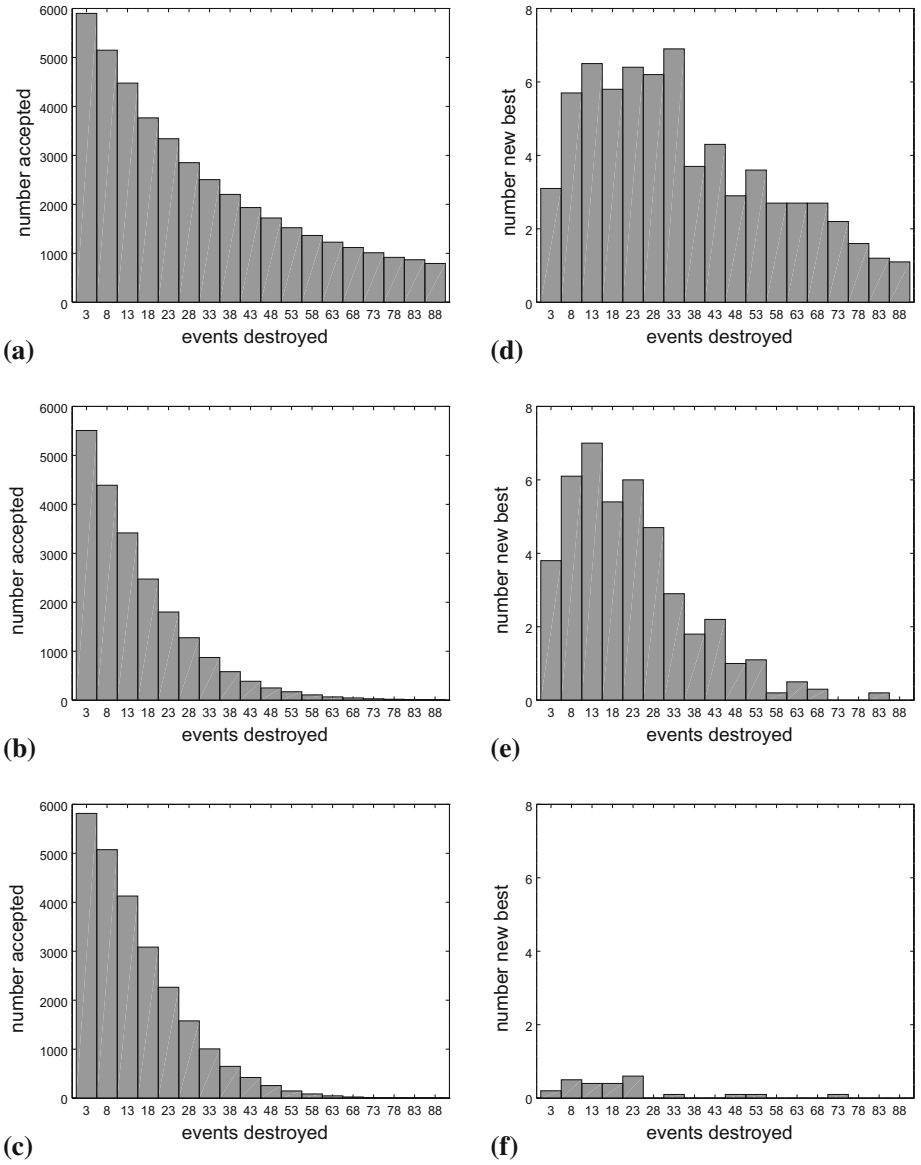

Fig. 1 Benefit of different destroy limits, `comp06`. **a** accepted solutions in first third, **b** accepted solutions in second third, **c** accepted solutions in last third, **d** new best solutions in first third, **e** new best solutions in first third, **f** new best solutions in first third

Table 11 presents the effect of omitting features on the performance. Column *Deterioration* shows the deviation of the modified algorithm from ALNS. Since these modifications typically affect the computational effort, the iteration limits have been adjusted. The last two lines of the table refer to the algorithm with uniformly distributed operator selection probabilities, i.e., without the adjustment scheme.

The feature of reheating the temperature turns out to be extremely useful. Contrary to Ropke and Pisinger (2006), we find that adding noise to the evaluation function does not seem

Table 11 Sensitivity analysis w.r.t. particular features, compared to ALNS

Modification	Deterioration (%)
ALNS without noise operators	0.22
ALNS without extra penalty for *MinWorkingDays*	0.68
ALNS without decreasing the destroy limit	2.24
ALNS without decrease, adjusted destroy limit	0.30
ALNS without temperature reheats	2.28
ALNS without accepting infeasible solutions	0.65
Uniform operator selection	2.42
Uniform operator selection, without noise operators	−0.26

to be very crucial for the performance of ALNS. Even without the use of any noise operator, there is only a slight deterioration observable. This indicates that the different operators lead to a sufficient diversification even without employing additional perturbation. It is interesting to see, that by discarding the weak performing noise operators, the version with uniformly distributed operator selection probabilities performs even slightly better than ALNS. This can be explained by the gain resulting from the extra iterations due to the removal of the adaptive mechanism. In case all parameter values are kept, decreasing the destroy limit has a strong effect on the performance. Fixing the destroy limit requires a reduced destroy limit compared to the original setting, though. ALNS with a fixed and adjusted destroy limit still performs worse than the original version. Apparently, omitting any feature of the algorithm will lead to a deterioration. However, when it comes to implementation in practice, some operators or features may be dropped without a significant loss in quality, in order to make the approach less complex.

6 Conclusion

This paper presented an adaptive large neighborhood search approach for solving the curriculum-based course timetabling problem. Implemented features include a reduction of the destroy limit over iterations, reheating the temperature of the simulated annealing acceptance scheme, allowing infeasible solutions, and taking the computation times of the repair operators into account when adjusting their selection probabilities. The algorithm incorporates several destroy and repair operators, including problem-specific operators that tackle the structure of timetabling problems. The performance of the algorithm was slightly enhanced by encouraging the spread over days in the evaluation function of the potential insertion positions of the lectures. Surprisingly, adding perturbation to the evaluation of potential insertion positions did not improve the performance significantly. The proposed approach generated competitive results for the benchmark instances of the second international timetabling competition. In particular, it outperformed the best algorithms of the competition. New best known solutions were found for five instances.

Acknowledgements The financial support by the Austrian Federal Ministry of Science, Research and Economy and the National Foundation for Research, Technology and Development is gratefully acknowledged. The computational results presented have been achieved in part using the Vienna Scientific Cluster (VSC). We acknowledge the constructive input by the anonymous reviewers.

References

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

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

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

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

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

Bellio, R., Ceschia, S., Di Gaspero, L., Schaerf, A., & Urli, T. (2016). Feature-based tuning of simulated annealing applied to the curriculum-based course timetabling problem. *Computers and Operations Research, 65*, 83–92.

Bettinelli, A., Cacchiani, V., Roberti, R., & Toth, P. (2015). An overview of curriculum-based course timetabling. *TOP*, *23*(2), 313–349.

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

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

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

Burke, E. K., Mareček, J., Parkes, A. J., & Rudová, H. (2010). Decomposition, reformulation, and diving in university course timetabling. *Computers and Operations Research*, *37*(3), 582–597.

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

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

Cooper, T. B., & Kingston, J. H. (1996). The complexity of timetable construction problems. In E. Burke & P. Ross (Eds.), *Practice and theory of automated timetabling. Lecture notes in computer science* (Vol. 1153, pp. 281–295). Berlin: Springer.

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

Derrac, J., García, S., Molina, D., & Herrera, F. (2011). A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. *Swarm and Evolutionary Computation*, *1*(1), 3–18.

Di Gaspero, L., McCollum, B., & Schaerf, A. (2007). 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, UK.

Gendreau, M., Hertz, A., & Laporte, G. (1994). A tabu search heuristic for the vehicle routing problem. *Management Science*, *40*(10), 1276–1290.

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

Kristiansen, S., & Stidsen, T. (2013). A comprehensive study of educational timetabling—A survey. DTU management engineering report, Department of Management Engineering, Technical University of Denmark.

Kristiansen, S., Sørensen, M., Herold, M., & Stidsen, T. (2013). The consultation timetabling problem at danish high schools. *Journal of Heuristics*, *19*(3), 465–495.

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

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

Lewis, R., & Thompson, J. (2015). Analysing the effects of solution space connectivity with an effective metaheuristic for the course timetabling problem. *European Journal of Operational Research*, *240*(3), 637–648.

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

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

Muller, L. (2009). An adaptive large neighborhood search algorithm for the resource-constrained project scheduling problem. In *MIC 2009: The VIII Metaheuristics international conference*.

Muller, L. F., Spoorendonk, S., & Pisinger, D. (2012). A hybrid adaptive large neighborhood search heuristic for lot-sizing with setup times. *European Journal of Operational Research*, *218*(3), 614–623.

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

Petrovic, S., & Burke, E. (2004). University timetabling. In J. Y. T. Leung (Ed.), *Handbook of scheduling: Algorithms, models, and performance analysis, chapter 45*. Boca Raton: Chapman Hall/CRC Press.

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

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

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

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

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

Shaw, P. (1998). Using constraint programming and local search methods to solve vehicle routing problems. In M. Maher & J. F. Puget (Eds.), *Principles and practice of constraint programming—CP98. Lecture notes in computer science* (Vol. 1520, pp. 417–431). Berlin: Springer.

Sørensen, M., & Stidsen, T. (2012). High school timetabling: Modeling and solving a large number of cases in denmark. In *Proceedings of the ninth international conference on the practice and theory of automated timetabling (PATAT 2012)*, pp. 359–364.

Sørensen, M., Kristiansen, S., & Stidsen, T. (2012). International timetabling competition 2011: An adaptive large neighborhood search algorithm. In *Proceedings of the ninth international conference on the practice and theory of automated timetabling (PATAT 2012)*, pp. 489–492.