Computers & Operations Research 195 (2026) 107621

Contents lists available at ScienceDirect

Computers and Operations Research

journal homepage: www.elsevier.com/locate/cor

Revisiting local search for Curriculum-Based Course Timetabling

Sara Ceschia [a], Francesca Da Ros [a,b], Luca Di Gaspero [a], Andrea Schaerf [a,*]

[a] Intelligent Optimization Laboratory, DPIA, Università degli Studi di Udine, Via delle Scienze, 206, Udine, 33100, Italy
[b] Department of Medical Oncology, CRO Aviano, National Cancer Institute, IRCCS, Via Franco Gallini, 2, Aviano, 33081, Italy

ARTICLE INFO

Keywords:
Educational timetabling
Simulated Annealing
Neighborhood relations

ABSTRACT

Curriculum-Based Course Timetabling (CB-CTT) is a well-established benchmark problem in educational timetabling, with numerous real and synthetic datasets. Despite its long history of algorithmic developments, designing search methods that work effectively across diverse datasets under time limits remains challenging.

In this paper, we focus on local search and, after a thorough analysis of all neighborhood structures reported in the literature, we propose an improved Simulated Annealing (SA) algorithm driven by an extended version of the classic LectureMove neighborhood, called LectureKick, that allows flexible lecture relocation while preserving feasibility. The neighborhood is further enriched with several sampling biases specifically designed to address the soft constraints of CB-CTT, while the overall behavior of the SA is regulated by a cut-off mechanism embedded in the cooling schedule.

The method involves numerous parameters, both for the SA schedule and for the neighborhood biases, which are rigorously tuned on a large artificial dataset. The final algorithm is subsequently evaluated on the available benchmarks, including the "standard" ITC 2007 *comp* dataset, under the original competition timeout conditions.

The results show that our approach clearly outperforms all previously proposed local search methods and is competitive with the best methods reported for CB-CTT overall, while improving upon them on several instances.

1. Introduction

Educational timetabling is a representative domain in combinatorial optimization, with practical applications in universities, schools, and other institutions. Among the many variants that have been proposed, Curriculum-Based Course Timetabling (CB-CTT) is a central benchmark problem, largely due to its adoption as the Third Track of the Second International Timetabling Competition (ITC 2007) and the subsequent availability of standardized datasets and results.

In CB-CTT, lectures of university courses must be assigned to periods and rooms over a weekly horizon, satisfying hard constraints while minimizing violations of soft constraints that capture institutional and student preferences. The formulation introduced by Di Gaspero and Schaerf (2003) in 2003 and later refined within ITC 2007 (McCollum et al., 2010) has stimulated extensive research on both modeling and algorithmic aspects.

A wide range of datasets is now available, including the original *comp* instances, additional real-world instances, and synthetic ones. From an algorithmic perspective, CB-CTT has been addressed using metaheuristics, mathematical programming, and hybrid methods, resulting in a rich literature.

Local search methods have played a prominent role since the earliest work on CB-CTT, including the winning ITC 2007 approach (Müller, 2009), which relied on the combined use of multiple neighborhoods. Subsequent studies have largely followed the same multi-neighborhood direction, exploring extensive neighborhood families and identifying the most effective neighborhoods as well as the most promising combinations among them. In parallel, exact and hybrid approaches have progressively improved best-known results, especially when larger computational budgets are allowed. As a consequence, different classes of methods tend to dominate under different time limits.

In this paper, we revisit selected neighborhood contributions proposed for CB-CTT within a unified local search framework designed to compete with state-of-the-art methods. The proposed approach is a probabilistically biased Simulated Annealing procedure built around LectureKick, an extension of the classic LectureMove neighborhood, enhanced with targeted biases toward the specific soft constraints.

The method is evaluated under ITC 2007 time limits on the *comp* benchmark and additional datasets, and compared against both strong local search baselines and current state-of-the-art methods. The experimental results show that it clearly outperforms all previously proposed

* Corresponding author.
E-mail address: andrea.schaerf@uniud.it (A. Schaerf).

https://doi.org/10.1016/j.cor.2026.107621
Received 26 March 2026; Received in revised form 26 June 2026; Accepted 21 July 2026
Available online 7 August 2026
0305-0548/© 2026 The Authors. Published by Elsevier Ltd. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

local search approaches and achieves performance comparable to, and in several cases better than, the best reported results overall.

All instances and best solutions are available on our website OptHub for inspection and future comparison. The source code is publicly available as a GitHub project (https://github.com/iolab-uniud/cb-ctt).

The main contributions of this work are:

- **Revision and extension of neighborhood operators:** We revise and adapt neighborhood operators proposed in the literature within a Simulated Annealing framework, enabling a consistent and controlled assessment of their contribution. In particular, we extend the classic LectureMove neighborhood by equipping it with soft-constraint–aware sampling biases.
- **Systematic component screening:** We conduct a systematic algorithm design analysis to identify the neighborhood components and bias mechanisms that yield substantial performance gains, while discarding those that prove unnecessary or ineffective.
- **Extensive empirical evaluation:** We benchmark the resulting method against leading local search approaches and current state-of-the-art techniques on the standard *comp* instances and additional real-world datasets. To support future comparisons, we also collect and select a set of particularly challenging instances that can serve as an additional benchmark.

The remainder of the paper is organized as follows. Section 2 reviews the main contributions in the literature. Section 3 defines the CB-CTT problem. Section 4 describes the proposed solution method. Section 5 presents the experimental setup and discusses the comparative results. Finally, Section 6 summarizes the findings and outlines directions for future research.

2. Related work

The literature on educational timetabling is extensive, covering a wide range of problems and formulations (Ceschia et al., 2023). Within this area, university course timetabling has attracted the most attention (Abdipoor et al., 2023), and CB-CTT has emerged as one of its most prominent formulations (Bettinelli et al., 2015).

CB-CTT was originally introduced by Di Gaspero and Schaerf (2003) and later refined for Track 3 of ITC 2007 (McCollum et al., 2010). Bonutti et al. (2012) subsequently proposed additional formulations that extend the problem by incorporating features such as maximum student load and travel distance between sites. In their classification, the original formulation is denoted as UD1, the ITC 2007 version as UD2, and UD3, UD4, and UD5 are introduced as three further variants. Despite these extensions, the ITC 2007 version (i.e., UD2), which is considered in this work, remains by far the most extensively studied, largely due to its role in the competition and the continued use of its benchmark instances.

Several datasets are available for CB-CTT. The earliest real-world instances were published in the original work by Di Gaspero and Schaerf (2003). The benchmark most widely used in the literature, however, is the ITC 2007 dataset, commonly referred to as *comp*, which consists of 21 instances from the University of Udine and other Italian institutions. Additional datasets were later provided by Bonutti et al. (2012) and, more recently, by De Coster et al. (2022), who introduced several new instances collected from various Italian institutions.

The literature also includes instance generators aimed at producing artificial datasets that resemble real-world ones, primarily for algorithm tuning and comparative studies. The first such generator was proposed by Burke et al. (2010) in the context of their graph-coloring-based formulation, but it was later criticized by Lopes and Smith-Miles (2010) for generating instances that were not sufficiently realistic. For this reason, Lopes and Smith-Miles (2010) developed a revised generator calibrated to better reflect the structural properties of the *comp* dataset, which was subsequently extended by De Coster et al. (2022) to span a wider range of feature values. Alongside this line of work, an Instance Space Analysis (Smith-Miles and Muñoz, 2023) for CB-CTT was first presented by Smith-Miles and Lopes (2011) and later revised and extended by De Coster et al. (2022), who introduced a set of 256 features and compared four search methods on a large collection of instances.

Most of the original *comp* instances are now solved to optimality, although often at the cost of several hours of computation. Ceschia et al. (2023, Table 13) report the optimal values and, for the remaining instances, the best-known lower bounds together with the corresponding references.

From an algorithmic perspective, CB-CTT has been tackled by local search, hybrid methods, and exact approaches; for a broad overview of these lines of research and of the current best results, we refer the reader to the recent survey by Ceschia et al. (2023). Here, we focus on local search methods, which are most closely related to our contribution.

Local search has played a central role in CB-CTT since the earliest contributions. The original work by Di Gaspero and Schaerf (2003) already explored both Tabu Search and Hill Climbing over a combination of two neighborhoods. This line of research was further consolidated by the winning approach of ITC 2007 (Müller, 2009), which orchestrated five neighborhoods within a cyclic scheme combining Hill Climbing, Great Deluge, and Simulated Annealing until the allotted time limit was reached.

Subsequent studies increasingly shifted attention to the analysis, selection, and combination of neighborhood structures. Lü et al. (2011) carried out an in-depth study of different combinations of basic and complex neighborhoods under a Steepest Descent framework, while Bellio et al. (2016) adopted the two basic neighborhoods identified by Lü, Hao, and Glover within a Simulated Annealing approach, complemented by feature-based parameter tuning. Extending this line of work, Kampke et al. (2017) introduced and experimentally evaluated nine neighborhoods, enlarging the set previously considered in the literature. Their study identified LectureMove, originally proposed by Müller, as the most effective operator. This move combines lecture relocation with a conditional swap mechanism: one lecture is moved to a new slot and, if that slot is occupied, the conflicting lecture is reassigned to the original slot of the first one. Building on this result, Kampke et al. (2019) employed LectureMove as the sole neighborhood within a hybrid framework, while Song et al. (2021) further elaborated and orchestrated neighborhood-based moves in a Simulated Annealing approach.

More recently, Feutrier (2024) shifted the attention from algorithm design to search-space characterization by conducting a landscape analysis of Müller's method. Feutrier's work introduced a novel landscape representation together with a new set of features, which were then used to predict performance and assess the relevance of the extracted descriptors.

The relative strength of different approaches depends significantly on the available computational budget. Under the original ITC 2007 time limits, the best-performing methods are the memetic approach of Abdullah and Turabieh (2012), which combines a Genetic Algorithm with Tabu Search, and the Adaptive Large Neighborhood Search of Kiefer et al. (2017), which employs an exact solver (CPLEX) in the repair phase. When the strict ITC 2007 time limits are relaxed, several mathematical programming approaches have reported improved results on the competition instances (Burke et al., 2010; Asín-Achá and Nieuwenhuis, 2014; Bagger et al., 2019; N.-C.F. Bagger et al., 2019; Cacchiani et al., 2013; Lach and Lübbecke, 2012; Phillips, 2015); for a detailed and systematic account of these developments, we again refer to Ceschia et al. (2023).

3. Problem definition

We now briefly recall the main elements of the CB-CTT formulation (called UD2) adopted in Track 3 of ITC 2007. For full details, the reader is referred to Di Gaspero et al. (2007).

3.1. Basic entities

CB-CTT is defined over four types of entities, each capturing a distinct dimension of the problem: *days*, *timeslots*, and *periods*, which define the temporal structure of the timetable; *courses* and *teachers*, which represent the teaching demand and its assignment to instructors; *rooms*, which model the available physical resources; and *curricula*, which group related courses so as to induce student-based scheduling constraints.

Days, Timeslots, and Periods. A teaching week consists of a fixed number of *days*, each divided into a fixed number of *timeslots*. A *period* is a scheduling unit identified by a $\langle day, timeslot \rangle$ pair.

Courses and Teachers. A *course* is a teaching activity delivered by a single *teacher* and composed of a fixed number of *lectures*. Each course is characterized by the number of enrolled students, a minimum number of distinct teaching days, and possibly a set of forbidden periods due to teacher unavailability.

Curricula. A *curriculum* is a set of courses attended by overlapping groups of students. It therefore induces conflicts among its courses, which cannot be scheduled in the same period.

Rooms. A *room* is a physical resource with a fixed seating capacity.

A solution to CB-CTT assigns every lecture of every course to a *slot*, defined as a $\langle period, room \rangle$ pair. A feasible solution must satisfy all *hard* constraints, while the objective is to minimize violations of the *soft* constraints.

3.2. Hard constraints

The formulation includes three types of hard constraints:

RoomOccupancy: No two lectures can be assigned to the same room in the same period.

Conflicts: Lectures of courses belonging to the same curriculum, or taught by the same teacher, cannot be assigned to the same period.

Availabilities: Lectures of a course cannot be scheduled in periods in which the teacher is unavailable.

3.3. Soft constraints

The ITC 2007 formulation includes four types of soft constraints:

RoomCapacity: A violation occurs when the number of enrolled students exceeds the capacity of the assigned room; the penalty equals the number of excess students.

RoomStability: A violation occurs when the lectures of a course are assigned to more than one room; the penalty is equal to the number of rooms used in addition to the first.

MinWorkingDays: A violation occurs when the lectures of a course are scheduled on fewer days than the required minimum; the penalty is equal to the number of days below that minimum.

IsolatedLectures: A violation occurs when, on a given day, a lecture of a curriculum is not adjacent to any other lecture of the same curriculum; the penalty is equal to the number of isolated lectures.

The objective function is defined as the weighted sum of the violations of these soft constraints. The weights are fixed as follows: 1 for RoomCapacity, 1 for RoomStability, 5 for MinWorkingDays, and 2 for IsolatedLectures.

4. Solution method

In this section, we describe the main components of the local search approach: the search space and the initial solution procedure (Section 4.1), the neighborhood relation (Section 4.2), the neighborhood sampling strategy (Section 4.3), and the Simulated Annealing procedure that guides the search (Section 4.4).

4.1. Search space and initial solution procedure

The search space is represented by a vector of size equal to the total number of lectures. Each element of the vector is the slot, i.e., a pair composed of the period and the room, assigned to the lecture.

The initial solution is constructed through a three-stage procedure, composed of a random construction and two local search procedures.

It starts by assigning lectures one at a time to randomly selected slots, ensuring that teacher availability is respected and that the chosen slot is not already occupied. In practice, this procedure almost always terminates with a complete solution, indicating that it is easy to construct assignments that satisfy all hard constraints besides Conflicts. In the few cases where some lectures cannot be assigned to any admissible slot, the procedure is restarted from scratch and eventually succeeds on all real-world and artificial instances considered.

The first local search stage starts from this randomly generated solution and continues until a first feasible solution is found. Specifically, its cost function only accounts for violations of the Conflicts constraints, given that all other hard constraints are satisfied by construction. For this stage, we adopt the neighborhood proposed by Bellio et al. (2016), namely the union of the following two basic neighborhoods: MoveLecture (ML) that changes the period and the room (i.e., the slot) of one lecture and SwapLectures (SL) that swaps both the period and the room of two lectures of distinct courses. Since feasibility is easy to achieve on the available instances, this phase does not demand sophisticated optimization and usually lasts only a few milliseconds. Accordingly, we employ a simple Hill Climbing strategy that samples moves uniformly at random and accepts them whenever they are improving or sideways.

While the first local search is aimed at finding any feasible solution, the second is designed to quickly produce a reasonably good one. It includes all four soft constraints in the cost function and explores only feasible solutions. Since this stage serves only to obtain a rough initial solution, it again relies on the neighborhoods of Bellio et al. (2016) and on a Hill Climbing strategy. Its running time is typically a few seconds (3.67 ± 0.65 s on the *comp* instances).

The complete initial solution procedure is shown in Algorithm 1. In the pseudocode, the symbol Σ_{Conf} denotes the search space that satisfies all hard constraints except Conflicts, and Σ_{feas} denotes the set of all feasible solutions. F_{Conf} represents the cost function considering only violations of Conflicts, and F_{soft} is the complete objective function, considering all soft constraints.

Algorithm 1 Initial solution procedure

```
1: procedure InitialSolutionProcedure(InputData: I, Solution: S)
2:     repeat
3:         S ← RandomSolution(I, Σ_Conf)                ▷ First stage
4:     until S is complete
5:     S ← HillClimbing(I, S, Σ_Conf, ML ∪ SL, F_Conf)  ▷ Second stage
6:     S ← HillClimbing(I, S, Σ_feas, ML ∪ SL, F_soft)  ▷ Third stage
7:     return S                  ▷ Return a complete and feasible solution
8: end procedure
```

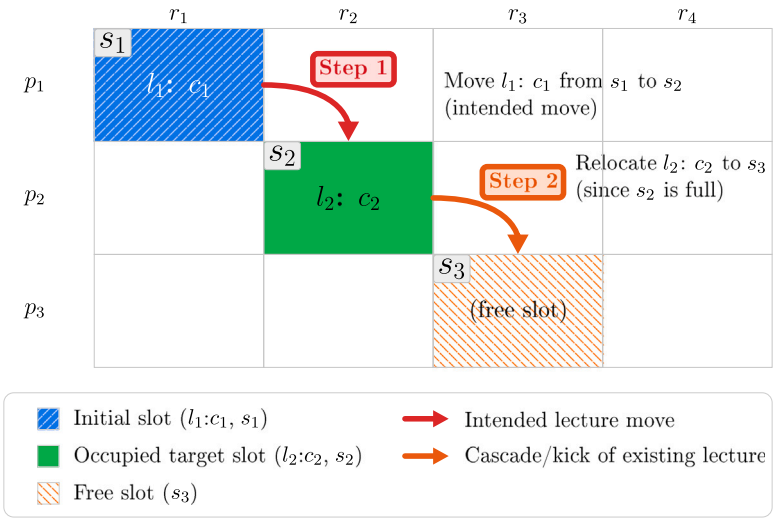


Fig. 1. The LectureKick move: l_1 is moved from s_1 to the target slot s_2 (Step 1). If s_2 is already occupied by l_2, the latter is relocated to a free slot s_3 (Step 2).

4.2. Neighborhood relation

The neighborhood operator used in our main procedure is an extension of LectureMove, originally introduced by Müller (2009) and later used by Kampke et al. (2019). In LectureMove, one lecture is moved to a new slot and, if that slot is occupied, the conflicting lecture is reassigned to the slot released by the first one. Our extension relaxes this mechanism by allowing the displaced lecture to be moved to any slot, i.e., to any period–room pair.

We refer to the resulting neighborhood as LectureKick, since the second lecture is effectively *"kicked out"* of its original position and relocated elsewhere. The LectureKick neighborhood is defined as follows.

LectureKick (LK) Given a lecture l_1 of course c_1 currently assigned to slot $s_1 = \langle p_1, r_1 \rangle$, and a target slot $s_2 = \langle p_2, r_2 \rangle$, the move relocates l_1 from s_1 to s_2. If s_2 is free, the move consists only of this relocation. Otherwise, let l_2 be the lecture of course c_2 currently assigned to s_2; in this case, l_2 is in turn relocated to a slot $s_3 = \langle p_3, r_3 \rangle$.

Preconditions: *(i)* $s_1 \neq s_2$, *(ii)* course c_1 has no lectures in period p_2 (unless $p_1 = p_2$), *(iii)* period p_2 is available for c_1, *(iv)* course c_1 has no conflicts in period p_2 (except possibly with c_2 if s_2 is occupied), *(v)* if s_2 is occupied, then $s_2 \neq s_3$, *(vi)* if s_2 is occupied, then course c_2 has no lectures in period p_3 (unless $p_2 = p_3$), *(vii)* if s_2 is occupied, then period p_3 is available for c_2, *(viii)* if s_2 is occupied, then course c_2 has no conflicts in period p_3 (except possibly with c_1 if $p_1 = p_3$), *(ix)* if s_2 is occupied, then slot s_3 is free (unless $s_1 = s_3$).

The preconditions ensure that the move is both feasible, in the sense that it preserves all hard constraints, and non-null, i.e., it does not leave the current state unchanged. At each iteration, the move is generated incrementally by selecting one attribute at a time in the sequence $\langle l_1, p_2, r_2, p_3, r_3 \rangle$, with the last two elements considered only when required. Whenever a precondition is violated, the partial move is immediately discarded and the generation process is restarted from scratch. This approach is substantially more efficient than generating the entire move first and checking its preconditions only afterwards. The move is illustrated in Fig. 1.

To guide move generation toward choices satisfying the preconditions and to accelerate move evaluation, we rely on extensive auxiliary data structures. For example, for each course we maintain the list of available periods and sample directly from it. Similarly, for every course–period pair, we store the current number of conflicts, so that conflicting periods can be discarded immediately during generation and the effect of candidate moves can be assessed efficiently.

4.3. Neighborhood sampling

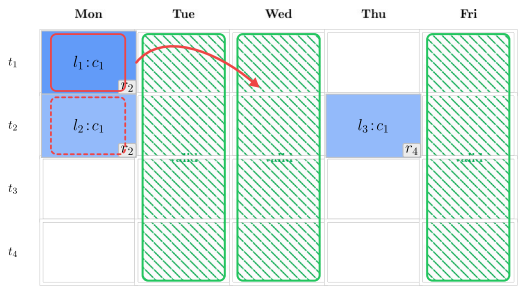
We now describe the strategy used to sample moves from the neighborhood. In standard Simulated Annealing, each iteration samples a move uniformly at random. In our case, however, the neighborhood is not sampled uniformly. Instead, to guide the search toward higher-quality solutions, we introduce a set of *sampling biases* that, when activated, increase the likelihood of selecting certain classes of moves. More precisely, we define six such biases. At each iteration, an active bias restricts the selection to lectures with specific characteristics; when no bias is active, moves are sampled uniformly from the whole neighborhood. The biases are illustrated in Fig. 2 and described in the following.

The first four biases target the four soft constraints and are mutually exclusive, so that at most one of them can be active at any iteration. Accordingly, a single random number in the interval $[0, 1]$ is drawn, and roulette-wheel selection is used to choose among the five alternatives: the four biases and the uniform distribution.

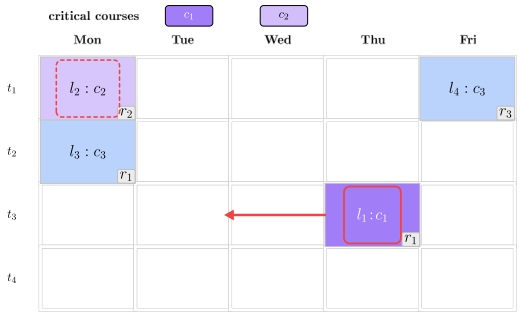
The first bias (Fig. 2(a)) targets MinWorkingDays. When it is active, lecture l_1 is selected from a course currently violating MinWorkingDays and, more specifically, from a day on which the course has more than one lecture. The new period p_2 is then selected on a day on which the course has no lecture. If a second lecture l_2 is involved in the move, no additional restriction is imposed on it. Managing this bias requires two redundant auxiliary data structures: a list of courses currently violating MinWorkingDays, and a matrix storing the number of lectures of each course on each day.

The second bias (Fig. 2(b)) targets IsolatedLectures. In this case, restricting the selection to isolated lectures would be too narrow, since moving a non-isolated lecture may also remove a violation by placing it adjacent to an isolated one. For this reason, we adopt a broader criterion and identify a set of *critical* courses, namely those belonging to a number of curricula above a given threshold. When this bias is active, l_1 is selected from one of these critical courses. As before, no additional restriction is imposed on l_2, if involved in the move. The rationale is that, because isolation is defined at the curriculum level, courses belonging to many curricula are more likely to contribute to IsolatedLectures violations. Critical courses are identified from the input data, and therefore no additional state-dependent data structure is required.

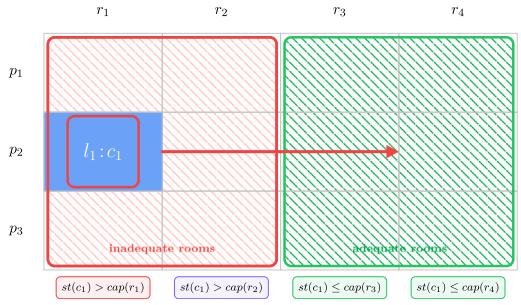
The next two biases (Figs. 2(c) and 2(d)) target RoomCapacity and RoomStability, respectively. When one of them is active, lecture l_1 is selected among those currently involved in a violation of the corresponding type, and the new room r_2 is chosen so as not to incur



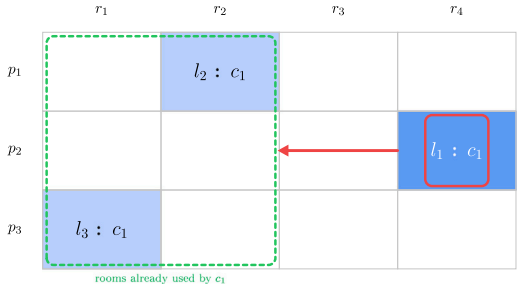
(a) MinWorkingDays: select l_1 from a course violating the constraint, on a day with multiple lectures, and move it to a day where the same course has no lectures. The figure shows a timeslot/day view, not the full periods/rooms timetable.



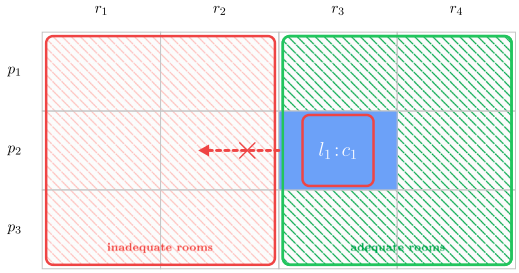
(b) IsolatedLectures: select l_1 from a *critical* course (e.g., one belonging to many curricula). No restriction is imposed on the destination. The figure shows a timeslot/day view, not the full periods/rooms timetable.



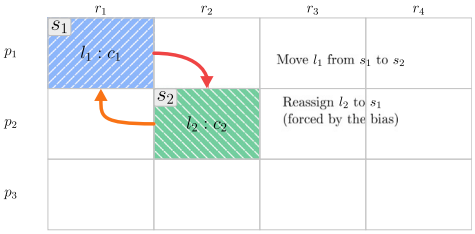
(c) Violation-driven RoomCapacity: choose l_1 among lectures violating the soft constraint (i.e., the number of students $st(c)$ exceeds the room capacity), and select r_2 to avoid the same violation.



(d) Violation-driven RoomStability: select l_1 among lectures contributing to the cost, and choose r_2 among rooms already used by course c_1.



(e) Preventive RoomCapacity: the move is not chosen because of an existing violation; it is only filtered to avoid introducing a new one.



(f) Swap bias: same structure as the original LectureKick move, except that when a second lecture is involved its destination is the slot vacated by l_1.

Fig. 2. Visual summaries of the six neighborhood-selection biases. The first four illustrate the mutually exclusive biases associated with the soft constraints, whereas the last two represent the independent preventive bias for RoomCapacity and the bias enforcing the swap structure.

that same violation, namely a sufficiently large room in the case of RoomCapacity, or a room already used by course c_1 in the case of RoomStability.

While the biases described so far are mutually exclusive, the remaining two are independent and may be activated simultaneously, as well as in combination with any of the previous ones meaning that a separate roulette wheel selection with just two options is applied for each of them.

The fifth bias (Fig. 2(e)) is again related to RoomCapacity, but does not directly target existing violations; rather, it prevents the introduction of new ones. When this bias is active, the new rooms r_2 and r_3 must be large enough to accommodate the lectures l_1 and l_2, respectively.

The sixth and final bias (Fig. 2(f)) restricts the move to the swap variant proposed in the literature. When this bias is active, if a second lecture l_2 is involved, it is reassigned to the slot s_1 vacated by l_1, i.e., $s_3 = s_1$; otherwise, s_3 is selected at random.

Finally, the biases targeting MinWorkingDays, RoomCapacity, and RoomStability are automatically disabled whenever no violation of the corresponding type is present.

Similar mechanisms are also used by Müller (2009, Section 5.3) although in a different form. First, what we refer to as a bias in our terminology corresponds to what Müller calls a separate neighborhood. More specifically, Müller implemented three such neighborhoods, each designed to reduce a specific soft-constraint penalty. The move related to MinWorkingDays follows the same mechanism as our bias, whereas the other two are implemented differently. The one targeted to RoomStability draws a course and a room at random and tries to assign all lectures in the course to the selected room; if the target room is occupied by another lecture, that lecture is reassigned to the original room of the lecture being moved. Finally, for IsolatedLectures, Müller tries to move an isolated lecture to a conflict-free period adjacent to another lecture in the same curriculum, possibly changing its room if needed.

Table 1
General SA parameters and biased neighborhood parameters.

Parameter	Description
General SA parameters	
T_0	Start temperature
T_f	Final temperature
α	Cooling rate
ρ	Cut-off ratio
$\mathcal{I}$	Total number of iterations
Parameters for biased neighborhood selection	
b_{mwd}	Bias toward MinWorkingDays violations
b_{cc}	Bias toward lectures of critical courses
ϑ	Threshold on curricula for defining a course as critical
b_c	Bias toward RoomCapacity violations
b_s	Bias toward RoomStability violations
b_{sp}	Bias toward swap moves
b_{ar}	Bias toward moves to rooms of adequate size

4.4. Simulated annealing

The search is guided by Simulated Annealing (SA). We adopt the original scheme proposed by Kirkpatrick et al. (1983), enhanced with a cut-off mechanism that accelerates the temperature decrease in the early stages of the search. For a recent overview of SA components and variants, we refer the interested reader to Franzin and Stützle (2019).

Algorithm 2 outlines the pseudocode of the SA procedure. The algorithm starts from an initial solution and, at each iteration, samples a random move m from the neighborhood of the current solution S as explained in Section 4.3.

The acceptance of m depends on the current temperature T, which is initially set at a high value T_0 and progressively decreased until a final value T_f is reached. Specifically, according to the *Metropolis criterion*, a move that does not worsen the objective value is always accepted; otherwise, it is accepted with probability $e^{-\Delta F/T}$, where ΔF denotes the increase of the objective value.

The temperature is updated according to a *geometric cooling schedule*; specifically, the temperature is multiplied by a cooling factor α, with $0 < \alpha < 1$. The update can occur in two ways: either after a fixed number N_s of iterations, or via a *cut-off mechanism*, in which the temperature is decreased once N_a moves have been accepted, with $N_a < N_s$. When the cut-off mechanism is triggered, the spared iterations are redistributed across the future temperatures by increasing N_s. We define N_a as a proportion of N_s through a parameter ρ, with $0 < \rho < 1$, that is, $N_a = \rho N_s$.

The stopping criterion is mapped to the maximum number of iterations $\mathcal{I}$, corresponding to the total number of solution evaluations performed. According to this choice, the value of N_s is computed from $\mathcal{I}$, by distributing the available iterations uniformly across the temperature levels. In particular, N_s is defined as $N_s = \mathcal{I} / \log_\alpha(T_f/T_0)$

Table 1 summarizes the parameters of our solution method, dividing those strictly linked with the SA procedure from those related to the neighborhoods.

Algorithm 2 Simulated Annealing with a cut-off mechanism.

procedure SA With Cutoff(SearchSpace Σ, Neighborhood $\mathcal{N}$, CostFunction F, Parameters T_0, T_f, α, $\mathcal{I}$, ρ)
  $T \leftarrow T_0$ ▷ *Initialize temperature*
  $S \leftarrow$ InitialSolution(Σ) ▷ *Generate an initial solution*
  $S^* \leftarrow S$ ▷ *Set the best solution to the initial one*
  $N_s = \mathcal{I} / \log_\alpha(T_f/T_0)$ ▷ *Compute the sampled moves per temperature*
  $N_a = \rho \cdot N_s$ ▷ *Compute the accepted moves per temperature*
  ▽ *Iterate until the final temperature is reached*
  while $T \geq T_f$ **do**
    $n_s \leftarrow 0$ ▷ *Reset the counters of sampled moves*
    $n_a \leftarrow 0$ ▷ *Reset the counters of accepted moves*
    ▽ *Sampling and acceptance limits*
    while $n_s < N_s$ and $n_a < N_a$ **do**
      $m \leftarrow$ RandomMove($S, \mathcal{N}$) ▷ *Move sampling*
      $\Delta F \leftarrow F(S \oplus m) - F(S)$ ▷ *Cost difference computation*
      ▽ *Metropolis acceptance criterion*
      if $\Delta F \leq 0$ **then**
        $S \leftarrow S \oplus m$ ▷ *Apply the move to the current solution*
        $n_a \leftarrow n_a + 1$ ▷ *Increment the count of accepted moves*
        if $F(S) < F(S^*)$ **then**
          $S^* \leftarrow S$ ▷ *Update the best solution if improved*
        end if
      else if RandomReal$(0,1) < e^{-\Delta F/T}$ **then**
        $S \leftarrow S \oplus m$ ▷ *Apply the move to the current solution*
        $n_a \leftarrow n_a + 1$ ▷ *Increment the count of accepted moves*
      end if
      $n_s \leftarrow n_s + 1$ ▷ *Increment the count of sampled moves*
    end while
    $T \leftarrow T \cdot \alpha$ ▷ *Decrease the temperature*
    ▽ *Redistribute the spared iterations to future temperatures*
    $N_s \leftarrow N_s + (N_s - n_s) / \log_\alpha (T_f/T)$
  end while
  return S^* ▷ *Return the best solution found*
end procedure

5. Experimental analysis

The code was implemented in C++ using the EasyLocal++ library (v. 3), compiled with GNU C++ (v. 13.3), and is available on GitHub. All experiments were run under Ubuntu Linux 24.04 on a machine equipped with an Intel® Core™ i7-7700 CPU at 3.60 GHz, featuring four physical cores and eight virtual cores enabled by hyper-threading. Each experiment was executed on a single virtual core.

For the comparative experiments, we adopt the timeout defined in the competition and calibrate it on our computational platform using the original benchmarking tool,[1] resulting in a time limit of 192 s per run. It is worth mentioning that this is much shorter than what the competitors were granted, due to hardware and software advances; for example, Abdullah and Turabieh (2012) and Kiefer et al. (2017) granted 600 and 480 s to their methods, respectively.

To ensure machine independence, we estimate on the reference machine how many iterations the algorithm performs within this time budget and use that number as the corresponding iteration limit. This also enables us to parallelize the experiments on other machines without violating the competition time-limit rule.

The analysis is organized as follows. In Section 5.1, we introduce the tuning and validation instances adopted in the study, both drawn from the literature. Given the number of components and parameters involved in the proposed method, we first perform a preliminary screening phase to identify non-influential design choices (Section 5.2). We then tune the remaining parameters in Section 5.3 using irace (López-Ibáñez et al., 2016, v. 4.2.0). The validation phase includes a comparison against results from the literature under the same experimental conditions as the ITC 2007 competition (Section 5.4). Finally, additional experiments under extended time budgets are conducted to assess the long-term behavior of the proposed approach (Section 5.5).

5.1. Instances

For validation, we consider the *comp* dataset originally introduced for ITC 2007. It comprises 21 instances; however, comp11 and comp15 are excluded. The former is omitted because it is trivial, with an easily

[1] Available from the ITC 2007 website (McCollum et al., 2010).

Table 2
Overview of the datasets considered in the computational study, with the number of instances and their main features.

Dataset	Ref.	Instances		Features				
		Total	Used	Days	Periods per day	Rooms	Courses	Lectures
comp	Di Gaspero et al. (2007)	21	19	5–6	5–6	5–20	30–131	138–434
test	Di Gaspero and Schaerf (2003)	4	2	5	4–5	10–13	46–56	207–252
DDS	Bonutti et al. (2012)	7	2	5–6	5–15	8–31	49–217	146–972
Udine	Bellio et al. (2016)	9	8	5	5	16–23	62–152	201–400
EasyAcademy	Bellio et al. (2016) and De Coster et al. (2022)	23	3	5–6	5–24	8–131	50–545	139–3,827
U. Erlangen	Mühlenthaler and Wanka (2014)	6	6	5	6	110–176	705–850	788–930
U. Prishtina	Sylejman et al. (2019)	8	0	5–6	12	10–27	125–366	302–628
Overall/min–max		*78*	*40*	*5–6*	*4–24*	*5–176*	*30–850*	*138–3,827*

attainable optimal cost of 0, whereas the latter is excluded because, in the ITC 2007 formulation, it coincides with comp03.

Many additional datasets have been proposed in the literature (Di Gaspero and Schaerf, 2003; Bonutti et al., 2012; Bellio et al., 2016; De Coster et al., 2022; Mühlenthaler and Wanka, 2014; Sylejman et al., 2019). However, most of them originate from richer and more complex formulations, and have been translated, i.e., simplified, into the CB-CTT setting. As a consequence, many of the resulting instances are easy to solve under the CB-CTT formulation and are therefore of limited interest as benchmark instances.

To build a benchmark that remains meaningful for future comparisons, we retained only the challenging instances. In particular, we discarded those for which a *perfect* (i.e., zero-cost) solution could be found within a few seconds. We also classified as easy, and therefore excluded, those instances for which our method consistently produced the same value for every soft-constraint component.

This selection may be regarded as conservative; however, focusing on genuinely challenging instances leads to a more informative and robust comparative analysis. For completeness, all instances and their best solutions remain publicly available on OptHub.

Table 2 reports, for each dataset, the total number of instances and the number actually retained for the experimental analysis, together with the main instance features (reported as minimum and maximum values over the dataset). Overall, only 40 out of 78 instances were retained. In particular, none of the instances from the University of Prishtina was included in the validation set.

For the training set, we employ 996 instances among those generated by De Coster et al. (2022), choosing those whose characteristics are similar to those of the *comp* dataset.

5.2. Parameter screening

Before carrying out full parameter tuning, we performed a parameter screening phase to reduce the effective configuration space of the proposed method. Given the large number of parameters and design choices involved, the goal of this phase was to identify those with negligible impact on solution quality and safely discard them from the final algorithmic design.

As a preliminary step, we tuned the full algorithm on the training instances using irace, obtaining a reference configuration for the screening phase. The resulting parameter values are reported in the Screening value column of Table 3.

The automated tuning phase was complemented with a targeted sensitivity analysis. In this analysis, each bias was evaluated under extreme settings while keeping all remaining parameters fixed at their reference values. Each configuration was run for $\mathcal{I} = 10^8$ iterations and repeated five times with different random seeds; the median solution quality was then used for comparison. Statistical significance was assessed at the 0.05 level using Friedman tests followed, when appropriate, by pairwise Wilcoxon signed-rank tests with Holm correction. When only two configurations were compared, only the Wilcoxon signed-rank test was applied.

A full factorial design was impractical given the size of the parameter space. We therefore adopted a hierarchical screening strategy, progressively removing parameters that exhibited little or no sensitivity and iteratively simplifying the algorithmic design. As a result, biases showing limited or negligible sensitivity to their assigned values were excluded from the final configuration, yielding a simpler method with lower computational overhead.

We first examined the bias favoring the selection of rooms of adequate size (b_{ar}). Three configurations were considered: $b_{ar} = 0$, corresponding to no bias; $b_{ar} = 1$, which always selects adequate rooms; and $b_{ar} = 0.792$, as suggested by irace. The Friedman test revealed significant differences among the configurations ($p < 0.001$). Pairwise comparisons showed that $b_{ar} = 0.792$ and $b_{ar} = 1$ did not differ significantly (Holm-adjusted $p = 0.725$), whereas both significantly outperformed $b_{ar} = 0$ ($p < 0.001$). The bias could therefore be safely fixed at $b_{ar} = 1$, thereby favoring the selection of rooms of adequate size whenever possible.

Following the previous adjustment, we next examined the room-capacity bias (b_c), comparing $b_c = 0$ with the tuned value $b_c = 0.223$. Despite the relatively high tuned setting, $b_c = 0$ (average rank 1.468) performed significantly better than $b_c = 0.223$ (average rank 1.532), with $p < 0.01$. A plausible explanation is that room capacity is already effectively handled by the previously introduced bias favoring adequate rooms.

Subsequently, we analyzed the room-stability bias (b_s), the swap bias (b_{sp}), the MinWorkingDays bias (b_{mwd}), and the critical-courses bias (b_{cc} and ϑ). In all cases, disabling the bias led to a deterioration in performance. All of them were therefore retained in the final configuration.

In summary, the biases that exhibited little or no sensitivity to their assigned values were b_{ar} and b_c, which were therefore fixed to 1 and 0, respectively. This led to a simpler implementation and lower computational overhead.

5.3. Parameter tuning

After the preliminary screening phase, the remaining parameters were tuned on the training instances using irace. The tuning was carried out in two sequential stages: the SA parameters were tuned first, followed by the neighborhood-related ones. Each stage was assigned a budget of 30,000 experiments. The parameters not being tuned were fixed to specific values rather than explored jointly: the SA-tuning stage used the screening values (column Screening value in Table 3) for the neighborhood parameters, while the neighborhood-tuning stage used the SA values obtained in the previous stage. We adopted this two-stage strategy to keep the joint configuration space tractable while still allocating a substantial per-stage budget to irace. No initial configurations were seeded in any of the irace runs; in all cases the procedure sampled from the parameter ranges reported in Table 3. Floating-point parameters were represented with three decimal digits of precision, while all other irace settings were kept at their default values. SA was allowed a maximum of $\mathcal{I} = 2 \times 10^8$ iterations. Table 3

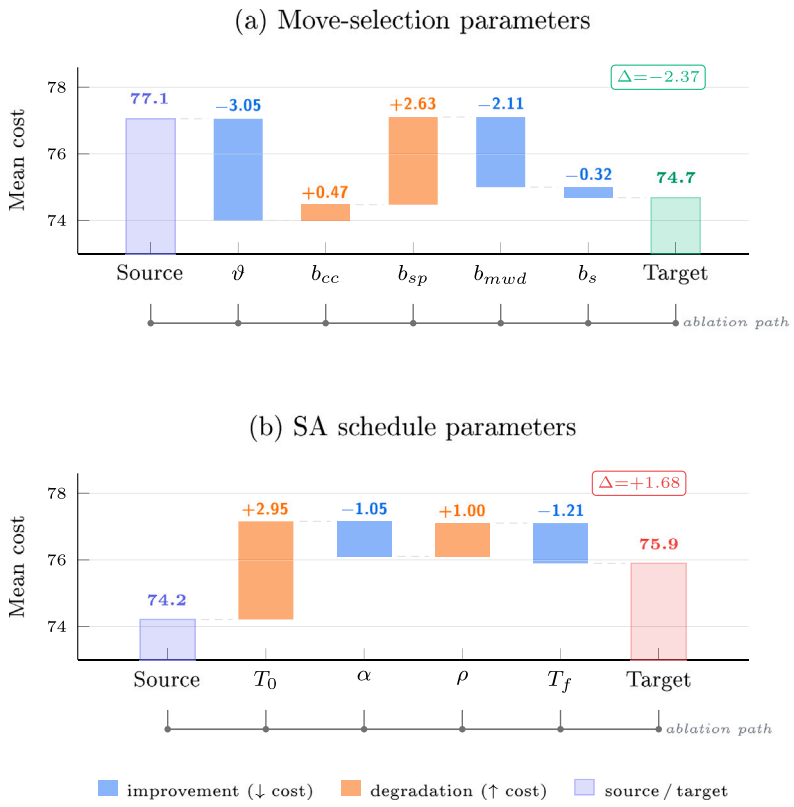


Fig. 3. Ablation analysis on comp instances (full mode). Each bar shows the marginal change in mean cost when flipping one parameter from the source to the target value. Blue: improvement (↓ cost); orange: degradation (↑ cost).

Table 3
Parameter tuning ranges and final values selected by irace.

Param.	Range	Screening value	Final value
General SA parameters			
T_0	2.0–10.0	6.568	5.394
T_f	0.01–0.10	0.085	0.069
α	0.980–0.995	0.981	0.988
ρ	0.05–0.5	0.229	0.133
Parameters for biased neighborhood selection			
b_{mwd}	0.0–0.25	0.208	0.202
b_{cc}	0.0–0.25	0.183	0.122
ϑ	3–10	4	3
b_c	0.0–0.25	0.223	0.000[a]
b_s	0.0–0.25	0.087	0.146
b_{sp}	0.0–1.0	0.810	0.803
b_{ar}	0.0–1.0	0.792	1.000[a]

[a] Final values were fixed after a sensitivity analysis rather than set with irace.

also reports the final tuned values of the parameters. Some parameters are close to their screening values, whereas others change substantially.

To better understand which parameters drive the performance of the tuned configuration, we performed an ablation analysis using the ablation function provided by irace. This function traces a greedy path between two configurations, i.e., a source (here, the first configuration randomly drawn by irace) and a target (the elite configuration returned by irace), by flipping one parameter at a time: at each step it changes the single differing parameter that yields the best mean cost, until the source has been fully transformed into the target. Each intermediate configuration along the path is therefore cumulative, and the order in which parameters are introduced reflects how much each contributes to the transition. We ran the ablation in full mode, evaluating every candidate configuration on the complete instance set rather than discarding candidates early through racing, and carried out the analysis on the comp instances so as to assess out-of-sample rather than tuning-set behavior. The resulting ablation paths are shown in Fig. 3.

The analysis reveals a clear asymmetry between the two parameter families. For the move-selection parameters (Fig. 3a), the improvement from source to target ($\Delta = -2.37$) is almost entirely attributable to ϑ, which alone accounts for a drop in mean cost of −3.05 (from 77.1 to 74.0); the remaining parameters have a marginal net effect, and the introduction of b_{sp} even produces a substantial temporary increase (+2.63) that is then offset by b_{mwd} (−2.11) before b_s brings the cost to the target value. For the SA schedule parameters (Fig. 3b), in contrast, the source corresponds to the lowest-cost configuration (74.2) and the fully transformed target performs worse (75.9, $\Delta = +1.68$). The individual steps oscillate: T_0 causes the largest degradation (+2.95), partially recovered by α (−1.05) and T_f (−1.21), while ρ adds a further increase (+1.00). However, no intermediate configuration improves upon the source. This suggests that the generalizable advantage of the elite configuration is concentrated in the move-selection component, whereas the tuning of the SA schedule appears specific to the tuning dataset.

This interpretation of these results should be treated with some caution. The schedule parameters are inherently coupled, since they jointly define a single cooling regime; therefore, a one-at-a-time ablation may pass through intermediate configurations that combine parameter values not intended to be used together. Moreover, since the analysis is based on a limited validation set, the reported differences are best viewed as indicative.

5.4. Comparative results

As noted above, all comparative experiments are carried out under the original ITC 2007 time limit. To account for the stochastic nature of SA, each validation instance is solved through 30 independent runs. We first present the results on the *comp* instances (Section 5.4.1) and then those on the other datasets (Section 5.4.2).

Table 4
Comparison with other state-of-the-art local search methods on the ITC 2007 *comp* dataset.

	Müller (2009)	Abdullah et al. (2012)	Bellio et al. (2012)	Lü and Hao (2009)	Bellio et al. (2016)		Our SA	
					FBT	F-Race	Avg	Gap %
comp01	**5.00**	**5.00**	**5.00**	**5.00**	5.23	5.16	**5.00**	0.00 %
comp02	61.30	53.90	51.60	60.60	52.94	53.42	**38.23**	−25.90 %
comp03	94.80	84.20	82.70	86.60	79.16	80.48	**73.33**	−7.36 %
comp04	42.80	51.90	47.90	47.90	39.39	39.29	**36.93**	−6.00 %
comp05	343.50	339.50	333.40	328.50	335.13	329.06	**324.57**	−1.20 %
comp06	56.80	64.40	55.90	69.90	51.77	53.35	**40.13**	−22.48 %
comp07	33.90	20.20	31.50	28.20	26.39	28.45	**9.80**	−51.49 %
comp08	46.50	47.90	44.90	51.40	43.32	43.06	**40.27**	−6.49 %
comp09	113.10	113.90	108.30	113.20	106.10	106.10	**102.27**	−3.61 %
comp10	21.30	24.10	23.80	38.00	21.39	21.71	**11.27**	−47.10 %
comp12	351.60	355.90	346.90	365.00	336.84	338.39	**323.63**	−3.92 %
comp13	73.90	72.40	73.60	76.20	73.39	73.65	**64.67**	−10.68 %
comp14	61.80	63.30	60.70	62.90	58.16	59.71	**55.00**	−5.43 %
comp16	41.20	51.70	43.30	53.70	38.06	39.19	**27.70**	−27.22 %
comp17	86.60	86.20	83.10	100.50	77.61	78.84	**69.40**	−10.58 %
comp18	91.70	85.80	84.30	82.60	81.10	83.29	**68.07**	−16.07 %
comp19	68.80	78.10	71.20	75.00	66.77	67.13	**61.63**	−7.69 %
comp20	34.30	42.90	50.60	58.20	46.13	45.94	**16.67**	−51.41 %
comp21	108.00	121.50	106.90	125.30	103.32	102.19	**92.80**	−9.19 %
Avg rank	5.05	5.32	4.34	5.82	2.97	3.39	1.11	–

5.4.1. Results on the comp dataset

Table 4 compares our SA with representative local search methods from the literature (Müller, 2009; Bellio et al., 2016; Abdullah et al., 2012; Bellio et al., 2012; Lü and Hao, 2009). For each algorithm, we report the average objective value, with the best values highlighted in bold. To quantify the relative performance of our method, we compute the percentage gap with respect to the best average value achieved by the competing approaches:

$$\text{Gap } \% = \frac{F - F^*}{F^*} \cdot 100\%,$$

where F denotes the average result obtained by our method and F^* the best average value among the competing approaches. Since this is a minimization problem, a negative gap indicates that our approach improves upon the current state-of-the-art. We also report the average rank over all instances to summarize the overall comparative performance independently of the absolute magnitude of the objective differences.

Our SA achieves the best average performance on all instances, with improvements over competing local search methods ranging from about −1% to −50%. The only exception is instance comp01, where its performance is essentially on par with that of the best alternative approaches.

Following the experimental protocol described in Section 5.2, we first assess overall differences among the methods using the Friedman test, and then perform post-hoc pairwise comparisons with the Wilcoxon signed-rank test and Holm correction. The Friedman test yields a p-value below 0.001, showing that the compared algorithms differ significantly overall. The subsequent Holm-adjusted Wilcoxon tests confirm that our method significantly outperforms the competing approaches.

We now broaden the comparison to include all methods that achieve state-of-the-art average performance in the literature under the ITC 2007 time limit (Abdullah and Turabieh, 2012; Kiefer et al., 2017). Table 5 presents the average results of the competitors as in Table 4. For our method, in addition to the average objective value, we also report the standard deviation, the minimum value, and the total number of iterations (in millions).

Our SA achieves the best average performance on 7 instances; Kiefer et al. (2017) also attains 7 best results, while Abdullah and Turabieh (2012) prevails on 4. On instance comp01, all methods obtain the same result, consistent with what was already observed for local search methods. Overall, our method attains the best average rank (1.79). Nevertheless, the Friedman test does not reject the null hypothesis at the 0.05 significance level, meaning that the observed differences among the methods are not statistically significant. Thus, from a statistical standpoint, the algorithms cannot be distinguished, likely due to the limited number of instances (19), which weakens the power of the analysis.

The total number of iterations varies substantially across instances for two main reasons. First, although the auxiliary data structures allow us to evaluate the cost difference of a candidate move without recomputing the resulting state from scratch (also known as *delta evaluation*), some parts of this evaluation still scale linearly with instance size. Second, the generation of feasible moves depends on the constraintness of the instance itself. This variation is even more evident in the results on the additional datasets reported in Table 6.

Fig. 4 summarizes the results of Tables 4 and 5 in a single instance-wise comparison. For each instance, it reports the best of average results among local-search methods, the best of average results available in the literature, and the average and best-run values obtained by our SA.

5.4.2. Results on other datasets

We now compare our results with those reported in previous studies on datasets other than the original *comp* instances (Lü and Hao, 2009; Abdullah et al., 2012; Bellio et al., 2016; N.-C. Bagger et al., 2019; Banbara et al., 2019; Bagger et al., 2018; Kiefer et al., 2017; Lach and Lübbecke, 2012; Asín-Achá and Nieuwenhuis, 2014). Some of these instances were also considered by Mühlenthaler and Wanka (2016), although their focus was on the fairness properties of existing solutions rather than on comparative algorithmic performance.

Table 6 reports the results of the comparison. We begin by analyzing average performance under the same time limit as in the competition setting, considering the methods of Lü and Hao (2009), Bellio et al. (2016) and Kiefer et al. (2017). For each instance, we compare the average result of our method over multiple runs with the corresponding average values reported in the literature. The table also reports the relative gap between our average and the best average among the considered methods (column Gap).

Not all the referenced works report results on the same set of instances. In particular, two instances (EA13 and EA19) are considered only in our study. On all remaining instances, our SA generally improves upon the average results reported in the literature, with only

Table 5
Comparison with state-of-the-art methods on the ITC 2007 *comp* dataset.

	Abdullah and Turabieh (2012)	Kiefer et al. (2017)	Our SA				
			Avg	Std	Min	I (M)	Gap%
comp01	**5.00**	**5.00**	**5.00**	0.00	5	246.13	0.00%
comp02	**36.36**	41.50	38.23	4.78	30	225.92	5.15%
comp03	74.36	**71.70**	73.33	3.48	68	248.03	2.28%
comp04	38.45	**35.10**	36.93	1.86	35	359.20	5.22%
comp05	314.45	**305.20**	324.57	12.04	303	143.52	6.35%
comp06	45.27	47.80	**40.13**	3.84	33	278.51	−11.35%
comp07	12.00	14.50	**9.80**	2.25	6	256.04	−18.33%
comp08	40.82	41.00	**40.27**	1.62	37	351.90	−1.36%
comp09	108.36	102.80	**102.27**	2.08	98	278.94	−0.52%
comp10	**8.36**	14.30	11.27	3.17	6	285.78	34.77%
comp12	320.27	**319.40**	323.63	8.80	307	186.20	1.33%
comp13	64.27	**60.70**	64.67	3.00	61	333.18	6.53%
comp14	64.36	**54.10**	55.00	1.29	53	337.26	1.66%
comp16	**23.73**	33.80	27.70	2.68	22	310.82	16.73%
comp17	76.36	75.70	**69.40**	3.60	64	295.44	−8.32%
comp18	75.64	**66.90**	68.07	2.80	63	361.28	1.74%
comp19	66.82	62.60	**61.63**	2.40	57	227.84	−1.54%
comp20	**13.45**	27.20	16.67	4.04	10	231.14	23.92%
comp21	100.73	97.00	**92.80**	3.87	86	255.40	−4.33%
Avg rank	2.21	2.00	1.79	–	–	–	–

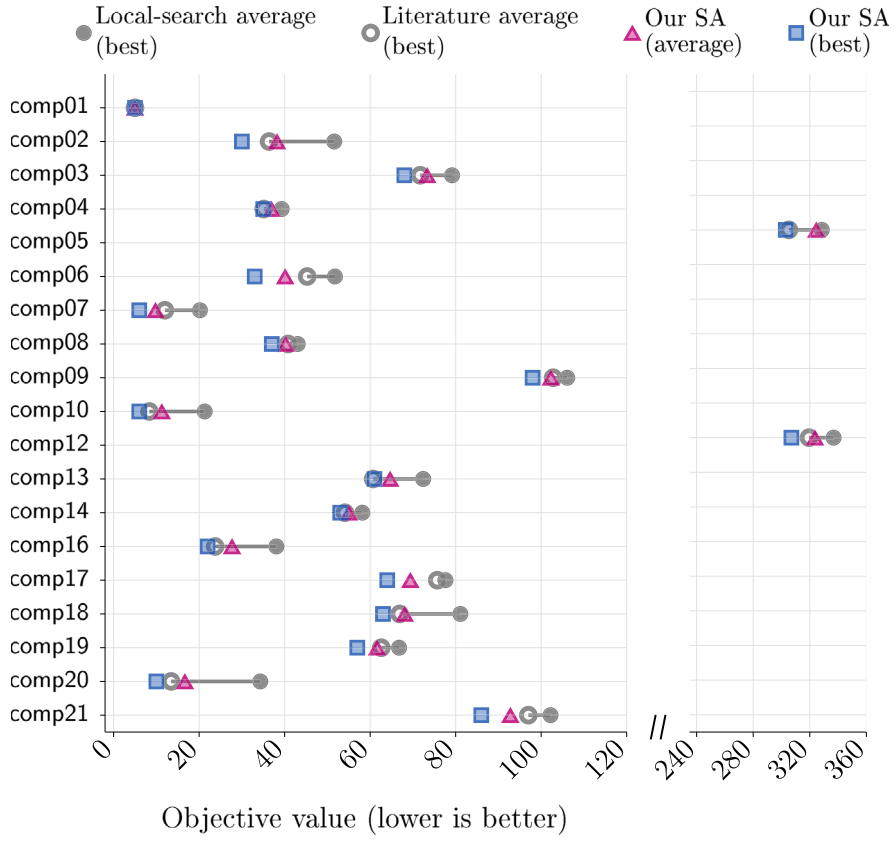


Fig. 4. Graphical comparison of our SA results for each ITC 2007 *comp* instance against the best of average results reported for local-search methods (Table 4) and in the literature (Table 5), along with our average result over repeated runs and our best result.

Table 6
Comparison with previously reported results on the other datasets. Negative gap values indicate an improvement over the corresponding reference result.

Instance	Lü and Hao (2009)	Bellio et al. (2016)		Kiefer et al. (2017)	Our SA				Bounds		Our best	
		FBT	F-race		Avg	Std	$\bar{t}$ (M)	Gap %	LB	UB	Min	Gap_b %
test												
test3	82.9	–	–	–	67.50	3.00	204.20	−18.58 %	59[a]	67[b]	61	−8.96 %
test4	89.4	–	–	–	69.47	4.07	181.54	−22.30 %	49[c]	73[g]	61	−17.57 %
DDS												
DDS1	–	110.26	106.55	120.60	76.73	5.10	143.17	−27.98 %	48[d]	48[e]	66	37.50 %
DDS6	–	9.87	10.35	7.10	4.87	1.59	307.21	−31.46 %	0	0[a]	2	–
U. Erlangen												
Erlangen-2011-2	–	6,568.42	–	–	5,233.87	200.31	155.94	−20.32 %	–	4,680[f]	4,928	5.30 %
Erlangen-2012-1	–	9,130.87	8,604.77	–	8,242.40	312.04	116.99	−4.21 %	–	7,519[f]	7,682	2.17 %
Erlangen-2012-2	–	16,183.74	–	–	12,162.93	660.86	108.30	−24.84 %	–	8,813[h]	10,734	11.96 %
Erlangen-2013-1	–	9,299.94	8,174.33	–	8,438.97	238.74	110.74	3.24 %	–	7,159[f]	8,011	11.90 %
Erlangen-2013-2	–	11,273.97	–	–	9,762.53	431.28	111.50	−13.41 %	–	8,150[h]	8,950	7.46 %
Erlangen-2014-1	–	8,251.13	7,080.75	–	7,263.73	447.62	110.64	2.58 %	–	5,981[h]	6,545	6.53 %
Udine												
Udine1	–	13.29	12.45	11.10	2.80	2.41	214.65	−74.77 %	0	0[e]	0	–
Udine2	–	19.26	21.42	19.60	15.43	2.80	324.37	−19.87 %	8[e]	8[e]	10	25.00 %
Udine3	–	8.52	10.23	7.70	3.40	2.50	290.18	−55.84 %	0	0[e]	0	–
Udine4	–	66.16	66.32	65.40	65.53	0.90	364.41	0.20 %	–	64[a]	64	0.00 %
Udine5	–	3.13	2.97	1.60	0.33	0.76	199.98	−79.17 %	0	0[a]	0	–
Udine7	–	2.03	2.32	1.40	0.53	0.90	217.88	−61.90 %	0	0[a]	0	–
Udine8	–	39.26	39.10	38.60	35.70	2.64	318.48	−7.51 %	29[e]	29[e]	29	0.00 %
Udine9	–	29.84	29.90	26.90	25.87	2.45	268.50	−3.84 %	21[e]	21[e]	24	14.29 %
EasyAcademy												
EA11	–	2.90	3.23	0.00	1.67	3.79	59.34	–	0	0[a]	0	–
EA13	–	–	–	–	251.13	0.35	275.24	–	–	–	251	–
EA19	–	–	–	–	1,999.13	2.56	258.36	–	–	–	1,996	–

[a] Same result reported by multiple works.
[b] Abdullah et al. (2012), Intel Pentium 4, 2.33 GHz; 600 s.
[c] Bagger et al. (2018), Intel Core 3.40 GHz, 8 GB RAM, Gurobi 6.5.2; 8320 s.
[d] Lach and Lübbecke (2012), Linux PC 3.4 GHz, 1 GB RAM, CPLEX 11.0.1; 3800 s.
[e] Banbara et al. (2019), Xeon E5520 quad-core, 2.26 GHz, 48 GB RAM, clingo 5; 3 h.
[f] Bellio et al. (2016), Intel Xeon E5-2660, 2.20 GHz; 600 s, ad hoc tuning.
[g] Zhipeng Lü, Université d'Angers, uploaded on OptHub.
[h] Alexander Kiefer, University of Vienna, uploaded on OptHub.

two exceptions: EA11 and Udine4, where our average performance is slightly worse than the best reported one. For Udine4, however, the gap is below 0.05%.

We next consider best-per-instance performance. For each instance, we compare our best result with the best upper bound reported in the literature, regardless of whether it was obtained under the competition time limit or with an extended computational budget. In this case, we report the relative gap Gap_b with respect to the best-known value.

In terms of absolute performance, we improve upon the best previously reported upper bounds for two instances (test3 and test4). On the *Udine* dataset, our method matches the best-known results on 6 out of 8 instances.

The *U. Erlangen* benchmark is more challenging. There, some of the best-known upper bounds were obtained by Bellio et al. (2016) through an ad hoc tuning procedure in which the method was specifically calibrated to this subset of instances. To ensure a fair comparison, we likewise performed a dedicated tuning of our method on the same dataset. This tuning determined a higher initial temperature (T_0 = 97.622), a higher cut-off ratio (ρ = 0.405), and a lower bias for swap moves (b_{sp} = 0.274). The remaining parameters are T_f = 0.09, α = 0.991, b_{mwd} = 0.165, b_{cc} = 0.026, ϑ = 3, and b_s = 0.105. Table 7 reports the results over 30 runs, with the gap computed with respect to the average value. Under this dedicated tuning, our approach outperforms the ad hoc configuration of Bellio et al. (2016) on all instances.

Comparing our results in Tables 6 and 7 we observe that the dedicated tuning reduces the average cost over the six *U. Erlangen* instances by about 14.5% relative to the general configuration. This highlights that our approach, as is customary for SA in general, is rather sensitive to parameter values.

5.5. Results on longer runs

To analyze the long-term behavior of the proposed approach, we conducted additional experiments on the *comp* instances using extended computational budgets. Specifically, we considered time limits equal to 10× and 100× the original competition time limit (i.e., 32 min and 5 h 20 min, respectively).

Table 8 reports the results over 30 repetitions. We denote by SA the configuration using the original time limit, and by SA_{10} and SA_{100} the configurations whose computational budget is increased by a factor of 10 and 100, respectively. For each setting, we report the average objective value together with the percentage gap relative to SA, which serves as the baseline.

The last three columns of the table report the reference values used for comparison and the best result obtained by our method across the three configurations. More precisely, the first of these columns (i.e., the third-to-last) reports the optimum whenever it is known, and otherwise the best lower and upper bounds (LB and UB) available in the literature. The second (i.e., the second-to-last) reports the best solution found across the three configurations, denoted by SA_*, while the last reports the corresponding gap with respect to the optimum, when available, or to the best known upper bound. These bounds originate from different sources and were typically obtained under extended computational budgets; for details, see Ceschia et al. (2023). Whenever our method reaches the known optimum, the corresponding result is underlined.

Increasing the computational budget yields, as expected, consistent performance improvements. In particular, the extended runs achieve gains of up to nearly –30%, with average reductions of –6% for SA_{10} and –10% for SA_{100}. Relative to the best-known results, our method matches the current best value on 9 instances. In one further case, the gap remains below 1% (comp05), while in another a new best solution is obtained (comp12), highlighted in bold.

Table 7
Comparison with the results using the *ad hoc* tuning procedure (Bellio et al., 2016) on the U. Erlangen dataset.

	Bellio et al. (2016)		Our SA			
	Avg	Min	Avg	Std	Min	Gap %
Erlangen-2011-2	4,971.80	4,680	4,647.87	98.86	4,441	−6.52 %
Erlangen-2012-1	7,856.85	7,519	7,346.57	119.82	7,128	−6.49 %
Erlangen-2012-2	10,130.33	9,587	9,742.07	442.14	8,849	−3.83 %
Erlangen-2013-1	7,825.55	7,159	7,376.60	170.19	7,004	−5.74 %
Erlangen-2013-2	8,844.34	8,329	8,233.33	156.87	7,872	−6.91 %
Erlangen-2014-1	6,525.05	6,144	6,026.20	139.36	5,749	−7.65 %

Table 8
Results obtained by the simulated annealing configurations SA, SA_{10}, and SA_{100}, using the original time limit and computational budgets increased by a factor of 10 and 100, respectively, and comparison with bounds from the literature.

	SA	SA_{10}		SA_{100}			SA_*	
	Avg	Avg	Gap %	Avg	Gap %	LB/UB	Min	Gap %
comp01	5.00	5.00	0.00 %	5.00	0.00 %	5	5	0.00 %
comp02	38.23	33.30	−12.90 %	30.90	−19.18 %	24	26	8.33 %
comp03	73.33	70.03	−4.50 %	68.27	−6.91 %	58/64	64	0.00 %
comp04	36.93	35.60	−3.61 %	35.40	−4.15 %	35	35	0.00 %
comp05	324.57	308.83	−4.85 %	300.87	−7.30 %	247/284	286	0.70 %
comp06	40.13	36.27	−9.63 %	33.50	−16.53 %	27	28	3.70 %
comp07	9.80	8.00	−18.37 %	7.00	−28.57 %	6	6	0.00 %
comp08	40.27	39.40	−2.15 %	39.30	−2.40 %	37	37	0.00 %
comp09	102.27	100.30	−1.92 %	99.20	−3.00 %	96	96	0.00 %
comp10	11.27	9.77	−13.31 %	8.20	−27.22 %	4	6	50.00 %
comp12	323.63	313.23	−3.21 %	305.33	−5.65 %	248/294	**289**	−1.70 %
comp13	64.67	60.90	−5.82 %	59.60	−7.84 %	59	59	0.00 %
comp14	55.00	53.33	−3.03 %	53.40	−2.91 %	51	51	0.00 %
comp16	27.70	25.70	−7.22 %	24.43	−11.79 %	18	22	22.22 %
comp17	69.40	66.53	−4.13 %	63.53	−8.45 %	56	60	7.14 %
comp18	68.07	66.80	−1.86 %	65.07	−4.41 %	61	63	3.28 %
comp19	61.63	59.37	−3.68 %	58.70	−4.76 %	57	57	0.00 %
comp20	16.67	15.43	−7.40 %	11.83	−29.00 %	4	6	50.00 %
comp21	92.80	87.10	−6.14 %	84.53	−8.91 %	74	81	9.46 %
Avg Gap	–	–	−5.99 %	–	−10.47 %	–	–	8.06 %

As already mentioned, all the best solutions are validated and made available on OptHub.

6. Conclusions

Although local search for CB-CTT has been extensively studied, our results show that substantial further improvements can still be achieved. These results stem from a systematic re-examination of existing neighborhood structures and move-selection mechanisms, together with a careful design process based on extensive screening and parameter tuning.

The experimental analysis shows that, under the ITC 2007 time limit, our method consistently outperforms all previously proposed local search methods and retains a slight edge over the current state-of-the-art approaches (without statistical significance, though). Nevertheless, a small number of instances remain challenging, and for these, our method is not yet able to attain the best-known results.

An important factor behind the success of the proposed method was the systematic investigation of all possible biases, avoiding "premature commitment" to specific design decisions (Hoos, 2012). Its efficiency also benefited from a careful implementation and from the effective and sharp maintenance of several auxiliary data structures, which kept the computational cost of move selection and cost-difference evaluation low. A further contribution of this work lies in the consolidation and clarification of the available benchmarks and corresponding results, thereby providing a more transparent and rigorous basis for future comparisons.

For future work, we plan to investigate more deeply, in the spirit of De Coster et al. (2022), the relationship between instance features and algorithmic performance, with the goal of understanding where the more challenging instances lie in the instance space. We also intend to revisit alternative, larger neighborhoods, such as Kempe chains, to assess whether they can further improve performance, possibly in combination with LectureKick. Finally, given the sensitivity of Simulated Annealing to parameter settings (Franzin and Stützle, 2019), another promising research direction is the development of adaptive or self-configuring strategies, for instance based on reinforcement learning, to reduce the reliance on extensive offline parameter tuning.

CRediT authorship contribution statement

Sara Ceschia: Writing – original draft, Writing review & editing, Software, Methodology, Investigation, Conceptualization. **Francesca Da Ros:** Writing – original draft, Writing review & editing, Validation, Software, Methodology, Investigation. **Luca Di Gaspero:** Writing – original draft, Writing review & editing, Visualization, Software, Investigation, Data curation, Conceptualization. **Andrea Schaerf:** Writing – original draft, Writing review & editing, Validation, Supervision, Software, Methodology, Investigation, Conceptualization.

Declaration on the Use of Generative Artificial Intelligence

The authors acknowledge the use of ChatGPT to support language editing, only for grammar correction and rephrasing, and for the visual refinement of the figures.

Acknowledgments

This research has been funded by the European Union-NextGenerationEU under the project FAIR MaPSART (PE00000013, CUPC63C22000770006).

The computational experiments were conducted using the LABIC research infrastructure, supported by funding from the Regione Autonoma Friuli Venezia Giulia, the Italian Ministry of Foreign Affairs, and the Italian Ministry of University and Research.

Data availability

Research Link Provided.

References

Abdipoor, S., Yaakob, R., Goh, S.L., Abdullah, S., 2023. Meta-heuristic approaches for the university course timetabling problem. Intell. Syst. Appl. 19, 200253.

Abdullah, S., Turabieh, H., 2012. On the use of multi neighbourhood structures within a Tabu-based memetic approach to university timetabling problems. Inform. Sci. 191, 146–168.

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

Asín-Achá, R., Nieuwenhuis, R., 2014. Curriculum-based course timetabling with SAT and MaxSAT. Ann. Oper. Res. 218, 71–91.

Bagger, N.-C.F., Desaulniers, G., Desrosiers, J., 2019. Daily course pattern formulation and valid inequalities for the curriculum-based course timetabling problem. J. Sched. 22 (2), 155–172.

Bagger, N.-C., Kristiansen, S., Sørensen, M., Stidsen, T., 2019. Flow formulations for curriculum-based course timetabling. Ann. Oper. Res. 280 (1–2), 121–150.

Bagger, N.-C.F., Sørensen, M., Stidsen, T.R., 2018. Benders' decomposition for curriculum-based course timetabling. Comput. Oper. Res. 91, 178–189.

Bagger, N.-C., Sørensen, M., Stidsen, T., 2019. Dantzig–Wolfe decomposition of the daily course pattern formulation for curriculum-based course timetabling. Eur. J. Oper. Res. 272 (2), 430–446.

Banbara, M., Inoue, K., Kaufmann, B., Okimoto, T., Schaub, T., Soh, T., Tamura, N., Wanko, P., 2019. Teaspoon: Solving the curriculum-based course timetabling problems with answer set programming. Ann. Oper. Res. 275 (1), 3–37.

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. Comput. Oper. Res. 65, 83–92.

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

Bettinelli, A., Cacchiani, V., Roberti, R., Toth, P., 2015. An overview of curriculum-based course timetabling. TOP 1–37.

Bonutti, A., De Cesco, F., Di Gaspero, L., Schaerf, A., 2012. Benchmarking curriculum-based course timetabling: formulations, data formats, instances, validation, visualization, and results. Ann. Oper. Res. 194 (1), 59–70.

Burke, E., Mareček, J., Parkes, A., Rudová, H., 2010. A supernodal formulation of vertex colouring with applications in course timetabling. Ann. Oper. Res. 179 (1), 105–130.

Cacchiani, V., Caprara, A., Roberti, R., Toth, P., 2013. A new lower bound for curriculum-based course timetabling. Comput. Oper. Res. 40 (10), 2466–2477.

Ceschia, S., Di Gaspero, L., Schaerf, A., 2023. Educational timetabling: Problems, benchmarks, and state-of-the-art results. European J. Oper. Res. 308 (1), 1–18.

De Coster, A., Musliu, N., Schaerf, A., Schoisswohl, J., Smith-Miles, K., 2022. Algorithm selection and instance space analysis for curriculum-based course timetabling. J. Sched. 25, 35–58.

Di Gaspero, L., McCollum, B., Schaerf, A., 2007. The Second International Timetabling Competition (ITC-2007): Curriculum-based Course Timetabling (Track 3). Technical Report, Queen's University, Belfast (UK).

Di Gaspero, L., Schaerf, A., 2003. Multi-neighbourhood local search with application to course timetabling. In: Proceedings of the 4th International Conference on the Practice and Theory of Automated Timetabling (PATAT-2002), Selected Papers. In: Lecture Notes in Computer Science, vol. 2740, Springer, pp. 262–275.

Feutrier, T., 2024. Landscape Analysis and Solver Reconfiguration for the Curriculum-Based Course Timetabling (Ph.D. thesis). Université de Lille.

Franzin, A., Stützle, T., 2019. Revisiting simulated annealing: A component-based analysis. Comput. Oper. Res. 104, 191–206.

Hoos, H.H., 2012. Programming by optimization. Commun. ACM 55 (2), 70–80.

Kampke, E.H., Scheideger, L.M., Mauri, G.R., Boeres, M.C.S., 2019. A network flow based construction for a grasp sa algorithm to solve the university timetabling problem. In: International Conference on Computational Science and Its Applications. Springer, pp. 215–231.

Kampke, E.H., Segatto, E.A., Boeres, M.C.S., Rangel, M.C., Mauri, G.R., 2017. Neighborhood analysis on the university timetabling problem. In: International Conference on Computational Science and Its Applications. Springer, pp. 148–164.

Kiefer, A., Hartl, R.F., Schnell, A., 2017. Adaptive large neighborhood search for the curriculum-based course timetabling problem. Ann. Oper. Res. 252 (2), 255–282.

Kirkpatrick, S., Gelatt, D., Vecchi, M., 1983. Optimization by simulated annealing. Science 220, 671–680.

Lach, G., Lübbecke, M., 2012. Curriculum based course timetabling: new solutions to Udine benchmark instances. Ann. Oper. Res. 194 (1), 255–272.

Lopes, L., Smith-Miles, K., 2010. Pitfalls in instance generation for udine timetabling. In: Learning and Intelligent Optimization. LION4, Springer, pp. 299–302.

López-Ibáñez, M., Dubois-Lacoste, J., Pérez Cáceres, L., Birattari, M., Stützle, T., 2016. The irace package: Iterated racing for automatic algorithm configuration. Oper. Res. Perspect. 3, 43–58.

Lü, Z., Hao, J.-K., 2009. Adaptive tabu search for course timetabling. Eur. J. Oper. Res. 200 (1), 235–244.

Lü, Z., Hao, J.-K., Glover, F., 2011. Neighborhood analysis: a case study on curriculum-based course timetabling. J. Heuristics 17 (2), 97–118.

McCollum, B., Schaerf, A., Paechter, B., McMullan, P., Lewis, R., Parkes, A.J., Di Gaspero, L., Qu, R., Burke, E.K., 2010. Setting the research agenda in automated timetabling: The second international timetabling competition. INFORMS J. Comput. 22 (1), 120–130, URL: https://www.eeecs.qub.ac.uk/itc2007/.

Mühlenthaler, M., Wanka, R., 2014. On the connectedness of clash-free timetables. In: Proceedings of the 10th International Conference on the Practice and Theory of Automated Timetabling. PATAT-2014, pp. 330–346.

Mühlenthaler, M., Wanka, R., 2016. Fairness in academic course timetabling. Ann. Oper. Res. 239 (1), 171–188.

Müller, T., 2009. ITC2007 solver description: a hybrid approach. Ann. Oper. Res. 172 (1), 429–446.

Phillips, A., 2015. Mathematical Programming-Based Models and Methods for University Course Timetabling (Ph.D. thesis). University of Auckland.

Smith-Miles, K., Lopes, L., 2011. Generalising algorithm performance in instance space: A timetabling case study. In: Coello Coello, C.A. (Ed.), Learning and Intelligent Optimization. LION 2011, In: Lecture Notes in Computer Science, Springer, pp. 524–538, number 6683.

Smith-Miles, K., Muñoz, M.A., 2023. Instance space analysis for algorithm testing: Methodology and software tools. ACM Comput. Surv. 55 (12), 1–31.

Song, T., Chen, M., Xu, Y., Wang, D., Song, X., Tang, X., 2021. Competition-guided multi-neighborhood local search algorithm for the university course timetabling problem. Appl. Soft Comput. 110, 107624.

Sylejman, K., Rexha, B., Gashi, E., 2019. Automated curriculum-based course timetabling in the University of Prishtina. URL: https://course-timetabling.uni-pr.edu/. (Accessed 18 February 2026).