MASARYK UNIVERSITY

FACULTY OF INFORMATICS

Large Neighborhood Search for High School Timetabling

Master's Thesis

BC. ONDŘEJ CHLUBNA

Brno, Spring 2026

MASARYK UNIVERSITY

FACULTY OF INFORMATICS

Large Neighborhood Search for High School Timetabling

Master's Thesis

BC. ONDŘEJ CHLUBNA

Advisor: doc. Mgr. Hana Rudová, Ph.D.

Department of Machine Learning and Data Processing

Brno, Spring 2026


FACULTAS ARTIS INFORMATICÆ
UNIVERSITAS MASARYKIANA

Declaration

Hereby I declare that this paper is my original authorial work, which I have worked out on my own. All sources, references, and literature used or excerpted during elaboration of this work are properly cited and listed in complete reference to the due source.

During the preparation of this thesis, I used Grammarly for grammar checking and ChatGPT for code review and assessment of writing style. After using these tools, I reviewed the content and take full responsibility for it, in accordance with the principles of academic integrity.

Bc. Ondřej Chlubna

Advisor: doc. Mgr. Hana Rudová, Ph.D.

Acknowledgements

First and foremost, I thank my supervisor, doc. Hana Rudová, Ph.D., for her time, valuable advice, thoughtful guidance, and supportive attitude. I would also like to thank RNDr. Václav Sobotka for his practical insights regarding the methods used in this thesis.

Furthermore, I would like to thank the school administrators of the High School of ICT, Postal Services, and Finance in Brno, especially Mgr. Lenka Skřivanová, who helped me understand the challenges and constraints of creating a usable high school timetable.

Last but not least, I would like to thank my wonderful girlfriend for all her support and patience. I thank coffee for its general existence and God for making the program work (if Satan or any other deity was responsible for making the program work, I thank them instead and entrust my soul to their possession).

Computational resources were provided by the e-INFRA CZ project (ID:90254), supported by the Ministry of Education, Youth and Sports of the Czech Republic.

Abstract

This thesis addresses the high school timetabling problem, a complex NP-hard scheduling task involving the assignment of educational events to timeslots under numerous constraints involving students, teachers, and classrooms. The proposed solver combines large neighborhood search with simulated annealing in a two-phase optimization framework that separates feasibility-driven search and soft constraint refinement. The approach is tailored to real-world requirements of the Czech High School of ICT, Postal Services, and Finance in Brno, incorporating domain-specific constraints and custom destroy and repair operators designed to exploit timetable structure. The solver is implemented in `C#` and evaluated on large-scale real-world datasets derived from manually constructed timetables. It can handle the majority of the constraints in the real-world problem. Experimental results show that the proposed method consistently produces near-feasible timetables with significantly fewer hard constraint violations than the manual solutions, while achieving comparable soft constraint quality. Overall, the implemented solution can effectively support practical timetable construction with manual refinement.

Keywords

Educational timetabling, high school timetabling, large neighborhood search, metaheuristics, real-world data, scheduling

Contents

1 Introduction

Creating high-quality high school timetables is a complex and time-consuming task that directly affects the daily operation of educational institutions. A timetable must coordinate students, teachers, and rooms while respecting a wide range of organizational and pedagogical requirements. Even small changes in the schedule can propagate through the system, leading to conflicts or undesirable patterns. As a result, manual timetable construction requires significant effort, knowledge, expertise, and, above all, a lot of time.

These tasks are studied in the field of educational timetabling, which includes high school timetabling (HST) [1]. The search spaces of HST are known to be very large, often involving thousands of events. Exact optimization methods often struggle to scale to such instances, making heuristic and metaheuristic approaches the dominant solution techniques in practice. A key challenge in real-world HST is not only finding feasible solutions but also balancing conflicting requirements. For example, ensuring continuous student schedules, respecting teacher availability, and maintaining a desirable distribution of lessons over time often compete with one another. A further complexity is the frequent constraint to schedule each day as a continuous, back-to-back series of events, with the only possible timeslot with no events being the lunch break. Additionally, practical timetables frequently include implicit conventions and exceptions that are difficult to formalize, further complicating automated approaches.

This thesis addresses a large-scale high school timetabling problem for a real-world instance derived from the Czech High School of ICT, Postal Services, and Finance in Brno (SŠIPF). This school is one of the largest high schools in the Czech Republic[1], with around 1,300 pupils, 130 teachers, and 50 classes. Since the school houses many different specializations with various educational demands, the entities and constraint spaces of the timetable are also very diverse. The goal is to design and implement a solver capable of generating high-quality

1. Based on internal data and data from the Registry of schools and educational facilities. In terms of student capacity, SŠIPF is the 50th largest high school in the Czech Republic and the 7th largest in Brno.

timetables under realistic constraints, while remaining flexible enough to adapt to domain-specific requirements.

To achieve this, the thesis proposes a solver based on large neighborhood search (LNS) combined with simulated annealing (SA). The approach follows a two-phase optimization framework. The first phase focuses on reducing hard constraint violations to approach feasibility. In contrast, the second phase refines the solution by optimizing soft constraints. The solver employs tailored destroy-and-repair operators that exploit the structural properties of timetables, enabling effective exploration of the solution space while maintaining computational efficiency.

The solver is implemented in `C#` and evaluated on real-world datasets derived from manually constructed timetables. The solver implements the majority of constraints considered during the real SŠIPF timetable construction. The experimental evaluation focuses on solution quality, measured by constraint violations, and on the convergence behavior of the search process. The results show that the proposed method produces near-feasible timetables with significantly fewer hard constraint violations than manual solutions, while achieving comparable soft constraint quality. These findings indicate that the solver can serve as a practical tool to support timetable construction, reducing manual effort while preserving flexibility for final adjustments.

The remainder of this thesis is structured as follows. Chapter 2 introduces the high school timetabling problem and reviews relevant solution methods. Chapter 3 describes the problem instance, including the entities and, most importantly, the constraints. Chapter 4 presents the internal representation of the problem within the solver. Chapter 5 details the proposed solution approach, including the large neighborhood search and its destroy-and-repair operators. Chapter 6 discusses implementation and system architecture. Chapter 7 evaluates the performance of the solver on real-world datasets and analyzes its behavior. Finally, Chapter 8 summarizes the contents of the thesis and outlines directions for future work.

2 High school timetabling

This chapter introduces the *high school timetabling* (HST) problem, its main characteristics, and the classes of solution methods relevant to the proposed solver.

2.1 Educational timetabling

Educational timetabling [1] is a widely studied subclass of scheduling problems. Its goal is to assign events involving students, teachers, and rooms into timeslots. Although this may appear straightforward, in practice, it is highly complex. This complexity is evident from the sheer number of conventions across various academic institutions, the variety of potentially conflicting requirements that a good timetable should fulfill, and the number of papers published on this topic.

There are three widely used variants of the educational timetabling problem: high school timetabling, university course timetabling, and university examination timetabling. This work will focus on high school timetabling [2] applicable to the High School of ICT, Postal Services, and Finance (SŠIPF) in Brno.

There is a variety of automatic timetabling applications that are frequently used in similar institutions. Among the most popular in the context of Czech high schools are aSc TimeTables [3], Bakaláři [4], or Skolaris [5]. These applications can create timetables after all resources and relevant requirements are imported. Bakaláři additionally offers direct integration with the Škola OnLine information system and supports manual timetable creation with a resource checker. However, all of these applications are proprietary and impose limitations on requirement specifications. SŠIPF has not decided to incorporate any existing automatic solution into the timetable creation.

2.2 Problem overview

The unique feature of HST is the organization of students into classes with a class timetable of events. Each event is typically assigned to a teacher and a classroom. Together, these form the three resources

of HST: students (forming classes with groups), teachers, and rooms. Classes may be partitioned into multiple groups, with each group possibly attending different events in parallel.

Educational models vary significantly across countries and institutions [1, 6], making it difficult to describe their common characteristics. A characteristic feature of HST in Czechia and other countries in continental Europe is the prohibition on student idle time: the daily timetable of each student should form a continuous sequence of events, with the lunch break usually being the only allowed interruption.

2.2.1 Formulation of HST model

A general-purpose XML-based input format for representing HST instances was proposed by Post et al. [2], known as XHSTT (XML High School Timetabling) [7]. The standard was formulated after years of international discussion and was intentionally designed to be rich and flexible enough to capture the requirements of as many educational systems as possible.

In XHSTT, the scheduled units called *events* are assigned to *timeslots*. An event is linked to a *group* of students as part of a *lesson*, reflecting the taught curriculum. Each event has a duration and requires some resources (either preassigned or to be assigned). Resources usually include a *room* and a *teacher*. Each resource assigned during timetabling can belong to a specific resource type subgroup (such as laboratories, computer rooms, chemistry teachers, or gym teachers). The time allocation of teachers and rooms usually cannot overlap; a teacher cannot teach two events at the same time.

Other common constraints included in XHSTT are specific time windows for any given resources or events. Time windows can specify which time slots are allowed for timetabling a given resource or event. Other common constraints include limiting the daily workload of students and teachers, reducing idle time for resources, or scheduling some events at the same time. The specific requirements relevant to the considered HST variant are detailed in Chapter 3.

As in most scheduling problems, constraints in HST can be categorized as hard (mandatory) or soft (preferences) [8]. In XHSTT, both of these types generate a cost: an *infeasibility value* for hard constraints and an *objective value* for soft constraints. In our case, these costs form

the objective function, which the solver minimizes. Each violation has a weight that determines its contribution to the overall cost, with hard constraint violations given higher weights to prioritize feasibility. The goal of an HST solver is, therefore, to:

1. reduce the infeasibility value to zero,
2. minimize the objective value.

2.3 Heuristic and metaheuristic methods

HST, in its general form, is an NP-hard problem [9] with a large assignment search space. Exact methods, such as integer linear programming or constraint programming, can yield promising results; however, on their own, they can quickly become computationally impractical due to high runtime and limited scalability. As a consequence, most solvers employ heuristic and metaheuristic techniques due to their ability to explore large, complex search spaces efficiently [9, 10]. Since even medium-sized real-world HST problems often result in prohibitively long computation times, a fast, feasible solution is often preferable to a slow, optimal solution. In practice, many state-of-the-art HST solvers adopt hybrid approaches, combining multiple heuristic and metaheuristic components with mathematical programming techniques [11, 12].

Heuristic and metaheuristic methods encompass large families of algorithms [13], within which we will discuss hill climbing, simulated annealing, and large neighborhood search, since they are directly implemented in the proposed solution. Other common methods used in HST include tabu search [11], iterated local search, evolutionary algorithms (including genetic algorithms) [14], and swarm intelligence methods [15].

Hill climbing (HC) [13] is a base local search heuristic that iteratively improves a single solution by applying modifications that reduce the objective value. As worsening moves are never accepted, hill climbing is computationally efficient and easy to implement, yet it is highly susceptible to becoming trapped in local optima.

Simulated annealing (SA) [13] is a single-solution-based probabilistic metaheuristic inspired by the physical annealing process. SA modifies a single solution, which can be accepted even if the objective value

worsens, with a probability controlled by a temperature parameter. This mechanism enables the search to escape local optima and explore a broader portion of the search space. As the temperature gradually decreases, the algorithm becomes increasingly selective, favoring improving or neutral moves and eventually converging towards a locally optimal solution.

Large neighborhood search (LNS) [13] is an approach that focuses on the structure of the neighborhood itself. Instead of applying small local modifications, LNS iteratively destroys a substantial part of the current timetable by removing a larger subset of assignments, then repairs it using a constructive heuristic. This scope allows the algorithm to move between distant regions of the search space while preserving parts of the existing solution.

Another approach that served as direct inspiration is the *Slack induction by string removals* (SISRs) algorithm [16]. SISRs is a relatively new ruin-and-recreate strategy for the vehicle routing problem, introduced by Christiaens and Vanden Berghe in 2020. The core idea of the approach is to temporarily increase the flexibility of a solution by removing contiguous sequences of assignments (*strings*), thereby inducing *slack* that can be exploited during reconstruction. This slack can be both structural and temporal. Unlike many heuristic frameworks that rely on a large number of specialized neighborhood operators, SISRs is built around a small and conceptually simple set of operators. The modification of the solution is performed through a single string-based ruin operator, while solution improvement is achieved through the repeated application of a greedy repair heuristic. This design reduces implementation complexity and limits the number of algorithmic parameters that require tuning. In the context of HST, where constraints and resource interactions are highly interdependent, a limited operator set helps avoid problem-specific moves and instead encourages systematic exploration driven by the objective function. Even though SISRs is a major inspiration for the implemented solution, it is important to mention that SISRs was proposed in the context of vehicle routing problems, where routes naturally form ordered sequences of visits.

3 Problem description

Timetabling problems are typically described by three main components: modeled entities, constraints, and optimization objectives. Mirroring this structure, this chapter will examine the given HST problem variant through its entities and the requirements placed on solutions, with optimization objectives addressed through soft constraints. The outlined problem is tailored for SŠIPF, a large high school in the Czech Republic offering multiple fields of specialization, with around 1,300 pupils, 130 teachers, and 50 classes.

3.1 Informal overview

Before providing a close description of the modeled entities, this section outlines the problem framework. The timetable is usually seen from the perspective of classes. Each class can be partitioned into groups (e.g., a class `IT2B` can be partitioned into a `Spanish` language group and a `French` language group). Groups of a class attend events. Events are taught by teachers and take place in a room. There are different types of rooms desired during timetable creation, such as `regular classroom`, `computer lab`, or `gymnasium`. Each event also has a lesson that reflects what is being taught (e.g., `Math`, `English`, `IT`).

The timetable consists of events spanning a two-week time window. Each week consists of five days, with events taking place in periods indexed from 0 to 9. A timeslot is defined as the atomic unit of time corresponding to a taught period. The solver aims to schedule all events into timeslots and assign them to specific rooms.

Most hard constraints are placed on the day structure for a class and its groups. For each student in the groups of a class and day, the timetable must consist of a continuous sequence of events, beginning in the early timeslots of the day. The only exception to the continuous sequence of events is the lunch break. A student should have timetable of at most 9 periods a day. The constraint also incorporate a variety of soft preferences. For example, school days should preferably not end too late, Fridays should be shorter when possible, and lessons should not repeat in a single day. An example timetable for a single week is given in Figure 3.1.

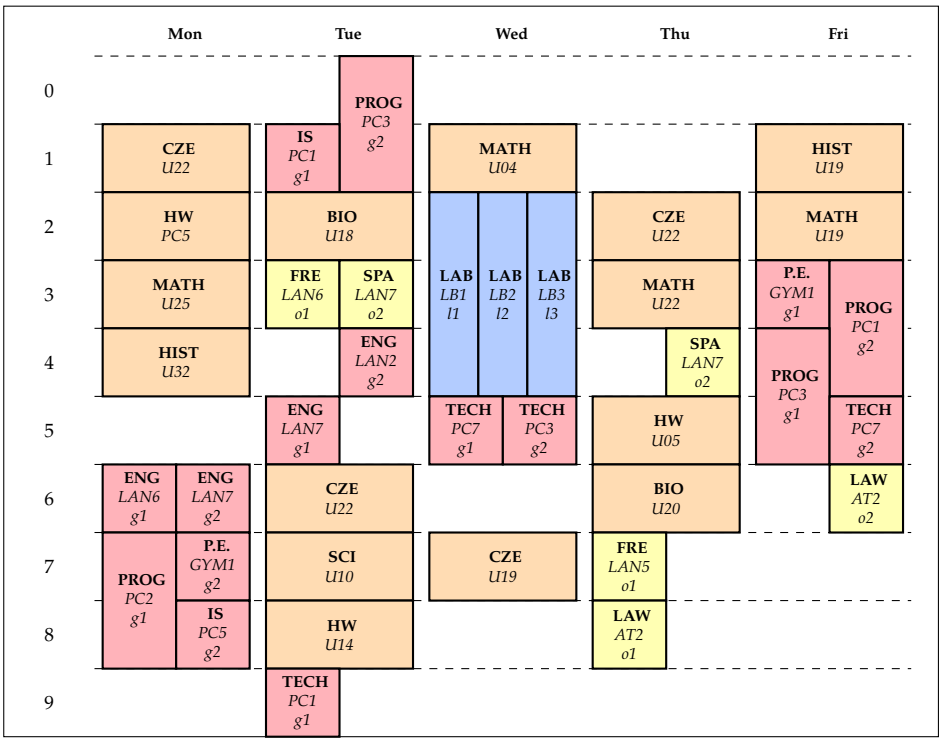

Figure 3.1: An example one week timetable for a single class. Each event shows the lesson, room, and group when the group is not whole class. Different colors correspond to different partitions of the class.

3.2 Entities

This section introduces the core timetable entities.

Classes, partitions and groups A class $c \in C$ serves as the highest-level student grouping. Each class can be partitioned into disjoint groups. Groups represent a subset of a class attending an event. A partition is a set of groups of a class that together form the whole class. A class can have many distinct partitions $\Pi_{c,i}$, where $i \in \mathbb{N}$ specifies its partition index, with the *trivial partition* $\Pi_{c,0}$ being the group of the whole class. Every group belongs to exactly one partition $g \in \Pi_{c,i}$.

Teachers A teacher $t \in T$ is a member of staff assigned to an event prior to the timetable construction. Teachers are a unary resource. Each event is associated with at most one teacher.

Rooms and room types A room $r \in R$ denotes a physical location in which an event takes place. Each room is characterized by exactly one room type $rt \in RT$, which describes its functional properties required by events.

Lessons A lesson $l \in L$ denotes the curricular content taught during an event.

Events Each event $e \in E$ is defined by a single[1] fixed group g, teacher t, lesson l, duration $d \in \mathbb{N}^+$, and a set of acceptable room types RT_e. The duration specifies how many consecutive timeslots the event occupies. Events constitute the central objects of the timetabling problem. Each event can be described by a tuple:

$$e = (g, t, l, d, RT_e), \qquad RT_e \subseteq RT$$

Timeslots A timeslot $k = 0, \ldots, K$ is the atomic unit of duration, corresponding to school periods. Timeslots are organized into days and weeks, forming a finite scheduling horizon. In the considered setting, the horizon spans two weeks, each consisting of five teaching days with ten timeslots per day.

Decision variables The decision variables determine the placement of events in the timetable. For each event $e \in E$, the solver decides the starting timeslot k_{start} and assigns a specific room r whose room type is contained in the event's set of acceptable room types. For events with a duration longer than one timeslot, the assignment implicitly occupies a sequence of consecutive timeslots. The solution can therefore be seen as a set of tuples:

$$\Sigma = \{ (e, k_{\text{start}}(e), r(e)) \mid e \in E \}$$

However, for simplicity, the solver omits the explicit room assignment $r(e)$. Instead, additional constraints ensure that the number of available rooms of each room type is sufficient for feasibility.

---

1. In real-world instances, some events may involve multiple groups of different classes attending jointly. Such events are relatively rare and were therefore excluded from the datasets considered to simplify the problem formulation. In the implemented solver, this situation can be represented using fixed events, described in Chapter 6.

3.3 Hard constraints

Hard constraints define conditions that must be satisfied by every feasible timetable. These constraints reflect fundamental organizational and physical limitations of the timetable. Hard constraints that may be violated during the timetabling process are denoted as $\mathcal{H}_i$, with unassigned events denoted as $\mathcal{H}_1$.

Resource exclusivity At any given timeslot, each unary resource (namely, student groups, teachers, and rooms) may participate in at most one event. A partition of a class is treated as an alternative resource; at any timeslot, a class may use at most one of its partitions. Consequently, each group belonging to the selected partition can attend at most one event in that timeslot. Likewise, each teacher may teach at most one event per timeslot, and each room may host at most one event per timeslot.

Daily structure of a class timetable For each class and each teaching day, the timetable must satisfy structural constraints reflecting the organization of the school day. The daily structure constraints enforce contiguity, early starts, and a bounded daily span of events.

The events attended by the groups of a class on a given day must form a *continuous back-to-back sequence* of timeslots ($\mathcal{H}_2$); no idle periods of students are allowed (the only exception being lunch breaks, which are going to be addressed later). As such, partitions with unassigned groups are allowed only at the start or end of the day; otherwise, students in unassigned groups in those partitions would have an idle period while others are attending events. Therefore, for all timeslots with partitions of a class and day, either all groups in a partition are assigned, or the partition is preceded/followed only by events of the same partition, with the same unassigned groups (with the exception of lunch breaks).

The timeslots of each day will be denoted as $k_0, k_1, \ldots, k_9$. The timetable of each student must *begin in the morning*. Therefore, for each class and day, there must exist at least one partition with all groups assigned at k_0, k_1, or k_2; otherwise, a late-start constraint ($\mathcal{H}_3$) is violated.

For all students and each day, the *temporal span* of timetable events must not exceed 9 timeslots ($\mathcal{H}_4$). Since a day consists of 10 timeslots,

either k_0 or k_9 must be free of events, or the same partition must be used at both k_0 and k_9, with no group assigned to both the first and the last timeslot.

Student lunch breaks Within each teaching day, each student must have exactly one lunch break ($\mathcal{H}_5$), defined as a timeslot without any assigned events. Lunch breaks may occur only in timeslots k_4, k_5, k_6, or k_7. For each class and day, the lunch break is assigned at the level of a single partition: there must exist one partition in which all groups have a free timeslot in at least one of the periods k_4–k_7. This ensures that students do not split into different lunch times within the same class. If a group has assigned events both before and after the lunch break on a given day, the lunch break occupies exactly one timeslot. Otherwise, the lunch break coincides with the end of the group's daily timetable. In the model, lunch breaks can be represented as dummy events assigned to all groups in the selected partition $\Pi_{c,i}$ of class c, with no teacher or room requirement, and a duration of $d = 1$.

All-day events Some events are timetabled for the whole day. We can model all-day events as any other event with the duration set to the number of timeslots in a day, which takes precedence over all other constraints; lunch breaks and maximal daily span constraints do not apply to all-day events.

Teacher availability, daily span and teacher lunch breaks Teachers may have predefined unavailable timeslots during which they cannot be assigned. These unavailability constraints reflect fixed obligations such as other duties, part-time contracts, or regular meetings. Any event assigned to a teacher must be assigned only within the teacher's available timeslots. Each teacher's daily timetable may span at most 9 timeslots ($\mathcal{H}_6$), measured from the first to the last assigned period. Each teacher must also have a lunch break, corresponding to at least one unassigned timeslot in the interval k_4–k_7 on each day ($\mathcal{H}_7$).

3.4 Soft constraints

The following is a list of soft constraints considered in the model, denoted as $\mathcal{S}_i$.

Daily lesson uniqueness per group This constraint penalizes multiple occurrences of the same lesson within a single day for a given group. Each additional occurrence of a lesson in a day is counted as one constraint violation ($\mathcal{S}_1$). Distributing lessons more evenly across the week supports gradual learning and reduces monotony in the daily timetable.

Odd and even week similarity Since the timetable spans a two-week period, the weeks can be distinguished as odd and even. This preference promotes structural similarity between the timetables of odd and even weeks. In particular, if a class is assigned to attend a lesson in a given timeslot during the odd week, it is preferred that the same lesson be assigned on the corresponding day and period in the even week. This can be done by creating pairs of events with the same weekly placement. A constraint violation occurs when a pair appears in different weekly timeslots ($\mathcal{S}_2$). Not all events need to be paired. Maintaining similar daily patterns across both weeks simplifies the overall timetable structure and reduces cognitive load for both students and teachers.

Preferred event placement This group of soft constraints captures desirable characteristics of event placement within the timetable.

Shorter school days are preferred, encouraging assignments in which classes finish earlier and avoiding unnecessary late events. Any event taught at k_8 or k_9 is penalized for each occupied timeslot ($\mathcal{S}_3$). *Shorter Fridays* should also be especially encouraged, so any event taught at k_7, k_8, or k_9 on a Friday is additionally penalized for each occupied timeslot ($\mathcal{S}_4$). An *event starting at* k_0 is a constraint violation, as early morning should be discouraged ($\mathcal{S}_5$). Similarly, *events occurring at* k_9 should be discouraged, as they occur late in the afternoon ($\mathcal{S}_6$).

Non-trivial partition of a class at k_3 (class being split into multiple groups) is penalized since it is difficult to find a substitute teacher during this timeslot. Each event with a non-trivial partition at k_3 counts as a constraint violation ($\mathcal{S}_7$).

The school day should preferably start at k_1 with any class day which does not have all groups of a partition present at k_1 counted as a constraint violation ($\mathcal{S}_8$).

Teacher-specific time preferences are respected when possible, allowing events to be assigned to timeslots preferred by the event teacher. Each non-preferred teacher timeslot in which an event is scheduled counts as a constraint violation ($\mathcal{S}_9$).

3.5 Summary of constraints

This section summarizes all constraint violations that may occur during the search process. The objective function $f(\Sigma)$ is defined as the weighted sum of all constraint violation counts. The weights used by the solver are specified in Appendix C.

Tables 3.1 and 3.2 provide a consolidated overview of all the constraints that form the objective function components. Detailed definitions of individual constraints are given in the preceding sections. Certain constraints require additional interpretation in the context of the implementation: in particular, the back-to-back sequence constraint is discussed in more detail in Chapter 4 as a closed hole ($\mathcal{H}_2$). Furthermore, the late start of class day constraint ($\mathcal{H}_3$) incorporates a soft preference that rewards shifts of the first fully assigned period toward k_2.

Constraint	Name
$\mathcal{H}_1$	Unassigned event
$\mathcal{H}_2$	Closed holes
$\mathcal{H}_3$	Late start of class day
$\mathcal{H}_4$	Long day of class
$\mathcal{H}_5$	Multiple lunches for group day
$\mathcal{H}_6$	Long day of teacher
$\mathcal{H}_7$	Missing teacher lunch break

Table 3.1: Hard constraints.

Constraint	Name
$\mathcal{S}_1$	Non-unique lessons
$\mathcal{S}_2$	Odd-even mismatch
$\mathcal{S}_3$	Events after k_7
$\mathcal{S}_4$	Events after k_6 on Friday
$\mathcal{S}_5$	Events at k_0
$\mathcal{S}_6$	Events at k_9
$\mathcal{S}_7$	Non-trivial partition at k_3
$\mathcal{S}_8$	Not all groups of partition present at k_1
$\mathcal{S}_9$	Non-preferred teacher time

Table 3.2: Soft constraints.
The notation k_i represents the i-th timeslot of any day.

4 Problem representation

This chapter describes the design decisions regarding problem representation within the solver, along with a few notes on solver performance optimization through the choice of data structures.

At a high level, the problem representation can be described as a three-layer structure. The first layer corresponds to the static description of the problem instance, including entities such as events, teachers, groups, and room types, which will be the focus of Section 4.1. This layer is immutable during the search process. The second layer represents the current solution, with the assignment of events to timeslots. The *solution* is represented by a single array, where each index corresponds to an event, and the stored value denotes its assigned starting timeslot. The third layer, described in Section 4.2, contains derived data that is maintained incrementally and accelerates feasibility checks and objective evaluation.

4.1 Entities

Each entity type is represented as a finite indexed set, allowing all relationships between entities to be expressed using integer indices rather than object references, enabling the solver to access them in constant time. This includes classes, partitions, groups, teachers, room types, lessons, events, and timeslots. Since the events are assigned over a timespan of 10 days consisting of 10 periods, timeslots can also be represented as a finite indexed set of values $\{0, \ldots, 99\}$, allowing easy access to each day or period by division or modulo of 10.

The central scheduled entity is the event. As such, events are used as a tool to address many requirements outlined in the problem description. Several problem-specific requirements are encoded directly within events. *All-day events* are encoded as standard events with a duration equal to the full length of the day. Similarly, *lunch breaks* are modeled as regular events without an associated teacher or room and can only be scheduled within the previously described time window. As such, lunch essentially serves as an additional lesson and cannot be assigned multiple times within a single group day, which counts as multiple lunches for group day constraint violation ($\mathcal{H}_5$). These lunch

events are generated for selected groups, corresponding to the most frequently used non-whole-class division groups, so that each student must attend exactly one lunch break event each day. The decision to create lunch break events for non-whole groups, rather than a single lunch break event for the whole class, was made to relax the model. The representation of *even and odd week similarity* was done by pairing the events. In particular, events that share the same group, teacher, lesson, and duration are split into pairs, as events that should preferably occur at the same week timeslot. The specific properties of events, such as all-day, lunch, and pairing, are marked by boolean flags.

Teachers and groups represent the primary resources for the problem. In addition, teachers may have limited availability and preferences over timeslots. These are represented using bitsets over the timeline. Teacher availability is inherited from event availability, ensuring that events can only be assigned in timeslots where the corresponding teacher is available. Rooms are modeled at the level of room types rather than individual physical rooms.

4.2 Derived state

In addition to the static problem description and the solution representation, the solver maintains a set of derived data structures. These structures are computed and updated incrementally during the search process. Their primary purpose is to enable efficient feasibility checks and fast evaluation of the objective function without the need to recompute all values multiple times during each iteration of the optimization process.

Resource occupancy For each teacher, group, and class, the occupied timeslots are tracked using bitset maps, enabling constant-time resource availability checks. Similarly, the number of free rooms of each room type in a given timeslot is maintained.

Structural information Additional data are maintained to capture the structural properties of the timetable and event placement. This includes information about the first and last occupied timeslots within a day, as well as whether the same lesson appears multiple times within a group day.

Open and closed holes We will define a hole as a timeslot in which at least one event of a partition is present, with other groups of the same partition not being assigned. Intuitively, this corresponds to the situation where one group of a partition is attending an event while the other groups are absent. In particular, we will distinguish between open and closed holes. An *open hole* is located at the boundary of a partially constructed timetable (at the beginning or end of a day, with only the same partition before or after it). A *closed hole* represents a missing group within the used timeslot partition, compared to other partitions of the class or assigned events of the same group. As such, the only constraint violations of daily class timetable continuity ($\mathcal{H}_2$) are closed holes. Closed holes are allowed during the scheduling process only to decrease the rigidity of the search. Each closed hole for a group is counted as an individual constraint violation. An example of this principle is given in Figure 4.1.

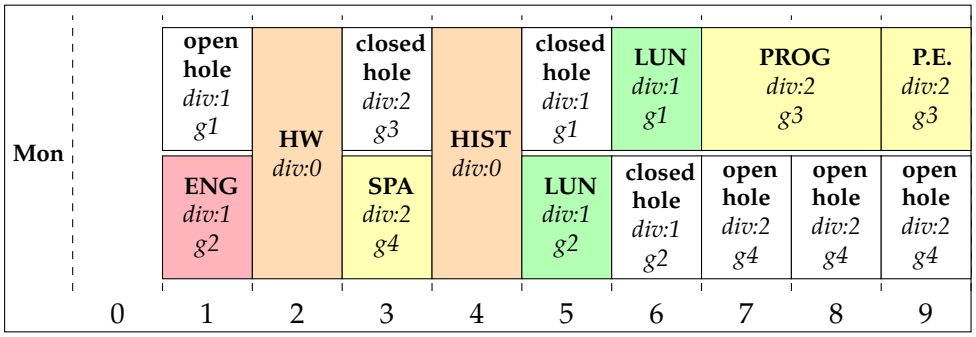

Figure 4.1: Example of a partial timetable for a single day of a single class. Holes and lunch events visualized. The *div* label symbolizes partition of the groups.

5 Solution approach

This chapter describes the proposed solution approach to the high school timetabling problem (HST). Given the combinatorial nature of the problem and the interplay of hard and soft constraints, the method is grounded in metaheuristic optimization techniques commonly used in timetabling literature [1]. The approach is based on iterative improvements that alternate between the partial destruction of a solution and its reconstruction. The following sections progressively introduce the individual components of this method and their roles within the overall search process.

5.1 Large neighborhood search

The solver is based on *large neighborhood search* (LNS) as described in Algorithm 1[1]. The algorithm starts by generating a greedy initial solution (Line 1) and setting the simulated annealing temperature T to the initial temperature T_0 (Line 4). At each iteration, the algorithm performs two main steps: a *destroy* phase and a *repair* phase. Destroy and repair operators are chosen from a set of defined operators (Lines 6 and 7). During the destroy phase, a subset of events is removed from the current solution (Line 8). The repair phase then attempts to reinsert the removed events to minimize the objective function (Line 9), with the change in the objective function $f(\Sigma)$ being denoted as Δ. If the repaired solution is accepted by the Metropolis criterion [17] (Line 11), it is used in the next iteration of the algorithm (Line 12), and the best overall solution is stored (Line 14). Each iteration lowers the temperature T by the cooling rate α (Line 17). The search is repeated until the *stopping criterion*, such as a predefined number of iterations used in our case, is reached (Line 18).

The proposed approach is closely related to the class of ruin-and-recreate heuristics, inspired by the *slack induction by string removals* (SISRs) algorithm [16, 18]. Similar to SISRs, the search alternates between removing parts of the current solution and reconstructing them

1. The presented formulation of the algorithm is simplified for clarity. The implemented solution additionally employs stagnation-triggered reheating and a return to the best previously found solution after prolonged stagnation.

Algorithm 1 generalLNS()

Output: $\Sigma^\star$

generate initial solution Σ_0
$\Sigma \leftarrow \Sigma_0$
$\Sigma^\star \leftarrow \Sigma_0$
initialize temperature $T \leftarrow T_0$
repeat
  Select destroy operator $d \in \mathcal{D}$
  Select repair operator $r \in \mathcal{R}$
  $\Sigma' \leftarrow d(\Sigma)$ (*partial solution*)
  $\Sigma' \leftarrow r(\Sigma')$ (*repaired solution*)
  $\Delta \leftarrow f(\Sigma') - f(\Sigma)$
  if $\text{rand}(0,1) < e^{-\Delta/T}$ **then**
    $\Sigma \leftarrow \Sigma'$
    if $f(\Sigma) < f(\Sigma^\star)$ **then**
      $\Sigma^\star \leftarrow \Sigma$
    end if
  end if
  $T \leftarrow \alpha \cdot T$
until stopping criterion is satisfied

to explore large neighborhoods with a small set of operators. The destruction phase aims to increase the *slack* of the solution, i.e., the flexibility available for reinserting events during repair. Slack can be either structural (removal of larger coherent units) or temporal (freeing consecutive timeslots). Slack is typically created by removing contiguous sequences of events, referred to as *strings*. The proposed approach builds on these principles, with the destroy operators purposefully creating both structural and temporal slack. While SISRs typically employs simple greedy reconstruction, the proposed method uses a more structured repair procedure that explicitly evaluates feasible placements, allowing for finer control over constraint satisfaction.

The LNS is performed in two distinct phases [19], followed by a short final refinement phase. A brief visualization of the solution pipeline can be seen in Figure 5.1. For the first and second phases of LNS, *simulated annealing* (SA) is used as an acceptance criterion. In the

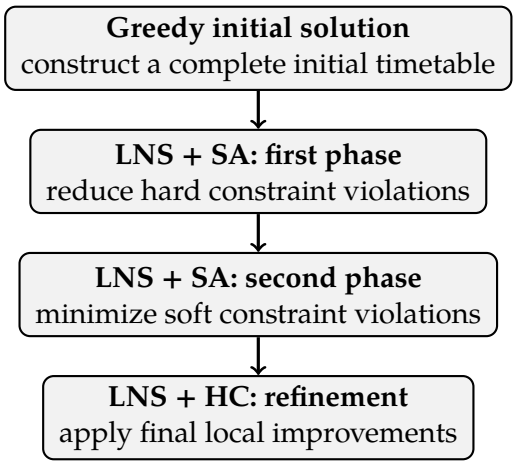

Figure 5.1: Overview of the solver pipeline

first phase, the objective function is dominated by hard constraints, guiding the search towards a feasible solution. Once hard constraints are minimized, the second phase focuses on refining soft constraints. This separation is done to simplify the search process, which requires different hyperparameter choices for both phases. In particular, the setting of the neighborhood size is crucial. There is a large disparity in scale and severity between hard and soft constraints, necessitating distinct SA temperature and SA cooling rate configurations in both phases. Separating the phases allows the search to first locate a feasible region of the solution space and subsequently perform a more localized optimization within it.

The second phase of the optimization introduces reheating since the temperature is already low enough to prevent the acceptance of solutions with additional hard constraints and to escape from local minima [20]. Since the second phase should focus on the minimization of soft constraints and cannot allow for too permissive acceptance criterion, the reheating is performed in very short thermal shocks, which introduce a temperature jump not only for the increase in temperature but also for the decrease in temperature. The cooling following reheating is performed instantly, rather than being a slight continuous decrease. The reheating duration thus corresponds to the number of iterations during which the temperature is increased by a fixed constant. Upon prolonged stagnation, the search is restarted from the

best solution found so far, a strategy commonly used in restart-based simulated annealing [21].

After the second phase, for a brief final refinement, simulated annealing is replaced by *hill climbing* (HC), and a more relaxed event placement strategy is applied. As we will see in Algorithm 2, the solver usually prioritizes filling open holes to produce continuous days. In this final stage, the event placement procedure deprioritizes placing multiple groups of the same partition together (in practice, this is done by setting the *Poor*Δ*Limit* of the following Algorithm 2 to zero). This configuration allows for a larger number of open holes while enabling events to be assigned to a more beneficial placement.

5.2 Greedy initial solution construction

The initial solution construction aims to produce a trivial feasible solution, which LNS will improve upon. The initial solution is created by placing events in an empty solution one by one. The solver will use a predefined greedy ordering, prioritizing events that are expected to be more difficult to place. Consequently, all-day events receive strictly higher scores than all other events, and lunch events are prioritized over regular events. Since lunch events are initially placed before regular events, they function as an anchor from which the sequence of events for each day is generated.

The ordering of events in the greedy construction phase is determined by a composite score that reflects the relative difficulty of placing each event. The score combines several factors related to temporal flexibility and resource scarcity. Formally, for each event e, the score is defined as

$$\begin{aligned} score(e) \;=\; & \frac{len_e^2}{3} \cdot (1 + \beta_1\, unavailable_e)^2 \cdot (1 + \beta_2\, roomType_e) \\ & \cdot (1 + \beta_3\, weekOrder_e) \\ & + \beta_4 * isLunch_e + \beta_5 * isAllDay_e \end{aligned}$$

where:

- len_e denotes the length of event e.
- $unavailable_e$ represents the proportion of timeslots that are unavailable for event e, based on teacher unavailability.

- $roomType_e$ represents the room type scarcity index, based on the number of rooms and events with the same room type.
- $weekOrder_e$ denotes the order of event e among the same (lesson, group, teacher) triplet in the input, starting from 1.
- $isLunch_e$ is an indicator of whether the event is flagged as lunch.
- $isAllDay_e$ is an indicator of whether the event is flagged as all-day.
- β_1 to β_5 denote weighting constants reflecting the relative importance of the individual components.

The importance of most parameters is self-explanatory, with the only component requiring further clarification being $weekOrder_e$. This index is introduced to break the symmetry among recurring events sharing the same (lesson, group, teacher) triple. Without it, these events would receive identical scores, making their simultaneous placement difficult. The specific values of the β weighting constants are provided in Appendix C.

After an event with the highest score is chosen, it is placed according to Algorithm 2.

Algorithm 2 eventPlacer(e, Σ)

Input: event e, partial solution Σ
Output: placement success; updated solution Σ

$\mathcal{C}_e \leftarrow$ feasible candidates in closed holes of the group
if $\mathcal{C}_e = \emptyset \ \vee \ isLunch(e) \ \vee \ \min_{c \in \mathcal{C}_e} \Delta(e, c) > Poor\Delta Limit$ **then**
  $\mathcal{C}_e \leftarrow \mathcal{C}_e \cup$ feasible candidates in open holes of the group
  if $\mathcal{C}_e = \emptyset \ \vee \ \min_{c \in \mathcal{C}_e} \Delta(e, c) > Poor\Delta Limit$ **then**
    $\mathcal{C}_e \leftarrow \mathcal{C}_e \cup$ all feasible candidates
  end if
end if
if $\mathcal{C}_e = \emptyset$ **then**
  return false
end if
Assign e to $\text{argmin}_{c \in \mathcal{C}_e} \Delta(e, c)$
return true

At first, a set of candidate placements at the closed holes of the given group is found and evaluated for feasibility (Line 1). A candidate is infeasible if the event does not fit, the teacher is unavailable, or no rooms of the required room types are available. If no feasible closed hole candidates are found, the event is lunch, or the best candidate score is poor (Line 2). The open holes of the given group are added to the set of possible candidates (Line 3) and evaluated for feasibility. A score is considered poor if it exceeds the *Poor*Δ*Limit* constant. The change in the objective function for event e placed at c is marked as $\Delta(e, c)$. If no candidates are found, or the best candidate has a poor score (Line 4), all other feasible placements are added to the candidate set (Line 5), namely, the starts of group days, the ends of group days, and empty days. The criterion for continuing the search after open hole candidates omits lunch events as a sufficient condition for preventing hole explosions. If no candidate is found, then e is left unassigned. Otherwise, the solver assigns e to a timeslot with the largest decrease in objective value (Line 11).

Algorithm 3 greedyInitialization()

Input: events E
Output: initial solution Σ; remaining unassigned events $\mathcal{U}$

1: Create empty solution Σ
2: Calculate $score(e)$ for each $e \in E$
3: $\mathcal{U} \leftarrow$ events E ordered by decreasing value of $score(e)$
4: $previousUnassigned \leftarrow 0$
5: **while** $previousUnassigned \neq |\mathcal{U}|$
6: $\quad \mathcal{U}' \leftarrow \emptyset$
7: $\quad previousUnassigned \leftarrow |\mathcal{U}|$
8: $\quad$ **for each** $e \in \mathcal{U}$ **do**
9: $\quad\quad success \leftarrow$ eventPlacer(e, Σ)
10: $\quad\quad$ **if** $\neg success$ **then**
11: $\quad\quad\quad$ Add e to $\mathcal{U}'$
12: $\quad\quad$ **end if**
13: $\quad$ **end for**
14: $\quad \mathcal{U} \leftarrow \mathcal{U}'$
15: **end while**

The procedure for the complete initial solution generation is given in Algorithm 3. The initial solution is constructed by repeatedly attempting to place events in a predefined order $score(e)$ (Line 3). For each event e of $\mathcal{U}$, the procedure eventPlacer(e, Σ) is invoked (Line 9). If the placement does not succeed, the event remains unassigned and is carried over to the next iteration (Line 11). Once no further progress is made (Line 5), the algorithm terminates.

If an event is paired (has another event marked for odd/even week similarity), then candidates for the feasible placement of both events are generated. They then behave as any other candidate, with $\Delta(e, c)$ representing the mean change in the objective function if both events are placed. If a pair of candidates is chosen, both events are placed.

5.3 Destroy operators

The destroy phase is characterized by the use of destroy operators and the type of slack they create. The solver uses two distinct destroy operators: *random events* and *targeted class*. The two operators complement each other. While random destruction promotes reconstruction on the borders of days, targeted destruction enables structured reconstruction within sequences of events.

Random events The operator removes events from the borders of class days at random, aiming to create temporal slack by creating empty timeslots that free their resources at the borders of days. Although the removed events may not correspond to problematic regions, the resulting randomness helps the search escape local optima and explore diverse areas of the solution space. This approach also significantly reduces the creation of holes, as events of the same class partition are removed and placed into holes. The fraction of destroyed events is varied to introduce further flexibility, drawn from a uniform distribution over the parameters *FractionMin* and *FractionMax*.

Targeted class The targeted class destroy operator focuses on removing whole days of selected classes. This operator aims to remove coherent parts of the timetable that are likely to contribute to structural inefficiencies. While the main idea of class days deletion is simple, the operator itself is relatively complex, with a high variety of parameters

Algorithm 4 targetedClassDestroy(Σ)

Input: solution Σ
Output: partial solution Σ' with destroyed events $\mathcal{D}$

1: $\Sigma' \leftarrow \Sigma$
2: $\mathcal{D} \leftarrow \emptyset$
3: **if** rand() $<$ *TargetRate* **then**
4: $\quad c_{seed} \leftarrow$ random class from constraint violation buckets
5: **else**
6: $\quad c_{seed} \leftarrow$ random class
7: **end if**
8: $\mathcal{C} \leftarrow \{c_{seed}\}$
9: $k \leftarrow \text{uniform}(ClassesMin, ClassesMax)$
10: **while** $|\mathcal{C}| < k$ **do**
11: $\quad$ Select $c' \notin \mathcal{C}$ using teacher similarity to c_{seed}
12: $\quad \mathcal{C} \leftarrow \mathcal{C} \cup \{c'\}$
13: **end while**
14: **for all** $c \in \mathcal{C}$ **do**
15: $\quad \mathcal{D} \leftarrow \mathcal{D} \cup$ destroyClassDays(c, Σ')
16: **end for**

needed to specify its concrete destroy selection, as described in Algorithm 4. This operator induces both temporal and structural slack by creation of large continuous free timeslots while removing entire day-level sequences of events.

The algorithm begins by choosing a seed class c_{seed} from which the solution will be destroyed. The *TargetRate* parameter specifies the probability of selecting classes associated with constraint violations, meaning a class is randomly selected from a bucket of all hard constraint violations of classes (Line 4). Otherwise, a class is selected at random (Line 6). The number of classes selected for destruction is denoted by k and chosen uniformly from the *ClassesMin* and *ClassesMax* parameters (Line 9). From the seed class, other classes are selected based on the teacher similarity to the seed class across all events of both classes. The similarity is computed as the dot product of the class teacher-load vectors, in which each component is the squared proportion of a given teacher in the corresponding class. This metric was chosen because

Algorithm 5 destroyClassDays(c, Σ')

Input: class c, partial solution Σ'
Output: destroyed events $\mathcal{D}'$, updated Σ'

1: $\mathcal{D}' \leftarrow \emptyset$
2: $\mathcal{B} \leftarrow$ set of all days of c
3: Sort $\mathcal{B}$ by decreasing span
4: **repeat**
5: **if** rand() $<$ *TargetedLongDays* **then**
6: Choose b from top-3 elements of $\mathcal{B}$
7: **else**
8: Choose b uniformly from $\mathcal{B}$
9: **end if**
10: Remove events of b from Σ'
11: Remove b from $\mathcal{B}$
12: Add removed events to $\mathcal{D}'$
13: **until** rand() $>$ *ChanceToTakeAnotherSlice*
14: **return** $\mathcal{D}'$

teachers are a limited resource that is not otherwise taken into account during destruction, and teacher similarity between two classes is easy to precompute.

Destruction is decomposed into a separate subroutine destroyClassDay in Algorithm 5. For each selected class, candidate strings for deletion are identified as complete event sequences of days. Restricting destruction to day boundaries avoids unnecessary fragmentation of timetables. The *TargetedLongDays* parameter specifies the probability of targeting within the longest day spans (Line 6). Otherwise, the day for deletion is chosen with uniform probability (Line 8). Events of the chosen day are removed from Σ' (Line 10), and other days are subsequently selected until the *ChanceToTakeAnotherSlice* condition is hit (Line 13).

As a guiding mechanism, the selection of day sequences to destroy is based on indicators of undesirable timetable structures, such as constraint violations and long day spans. The operator preferentially removes parts of the timetable that are more likely to benefit from reconstruction while allowing the targeting behavior to be controlled

via parameters. This targeted removal allows the repair phase to reorganize entire daily patterns while keeping a high level of variability. The selection of days is done independently for each class rather than being selected for all classes at once. Independent day selection allows for greater variability, with a high chance that the same day will be destroyed due to the number of chosen classes and days.

5.4 Repair operators

Each repair operator is defined by a strategy for selecting the order in which events are reinserted. Each event is placed using the greedy placement procedure described in Algorithm 2, selecting the position that minimizes the increase of the objective function. When selecting a repair operator to be employed after the destroy process, a transactional multiplier can be given, which decreases or increases the chance of a specific repair operator being chosen after a given destroy operator. The proposed solution employs two repair operators: *random ordering*, and *feasible starts count*.

Random ordering The random repair operator places events in a random order, keeping only the strict prioritization of all-day and lunch events. By creating less-structured placements, random ordering helps explore alternative configurations and prevents the search from becoming overly greedy. From the perspective of SISRs, this mechanism serves a similar purpose to the randomized reconstruction inherent in string-based removal and reinsertion.

Feasible starts count This operator prioritizes events based on the number of currently feasible placements. For each unassigned event e, the number of feasible starting timeslots is calculated. The ordering is iteratively updated after each placement. This approach follows the "fail-first" principle, focusing on the most constrained decisions first. As a result, it effectively utilizes the slack created during the destroy phase and reduces the likelihood of leaving difficult events unassigned. This dynamic reordering represents a departure from the original SISRs principle, which relies on static, computationally inexpensive orderings.

6 Implementation

The solver itself is implemented in the `C#` programming language, targeting the .NET runtime, without the use of external dependencies. The choice of `C#` was primarily motivated by its balance between performance, safety, and agility for future modifications. Even though lower-level languages such as `C` or `C++` may offer higher raw performance, the highly optimized execution of the modern .NET runtime makes `C#` a suitable choice. The implementation especially makes use of efficient array handling and bit-level operations provided by the .NET runtime. The following sections describe the key aspects of the implementation in more detail. First, the overall system architecture is presented, outlining the main components and their interactions. Subsequently, the challenges encountered during development are presented.

6.1 System architecture

The system architecture is outlined in the high-level architecture diagram shown in Figure 6.1. The solver is composed of several highly interdependent components, each with a unique set of responsibilities. The central component is `Core`, which encapsulates the state of the solver, including both the current solution and its associated objective function components, as well as all derived data. As a state management component, `Core` is accessed by most other components of the solver.

User interaction and solver configuration are handled by the `Application` component. During initialization, the `Scoring` component precomputes static scores for individual events. In addition, `Scoring` will be used to calculate the ordering of events during the feasible starts repair strategy. The initial solution construction is created as per Algorithm 3. The `Constructor` component is used solely for individual event placement. It uses the `Feasibility` component to determine whether placing an event into a given timeslot is feasible. If a placement is feasible, the `PolicyEvaluator` computes the change in the objective function for the candidate event placement and updates the constraint counts accordingly. Transactional changes to the solver

state during event assignment and unassignment are managed by the `Txn` (transactions) component. Together, these components define how the initial solution is generated.

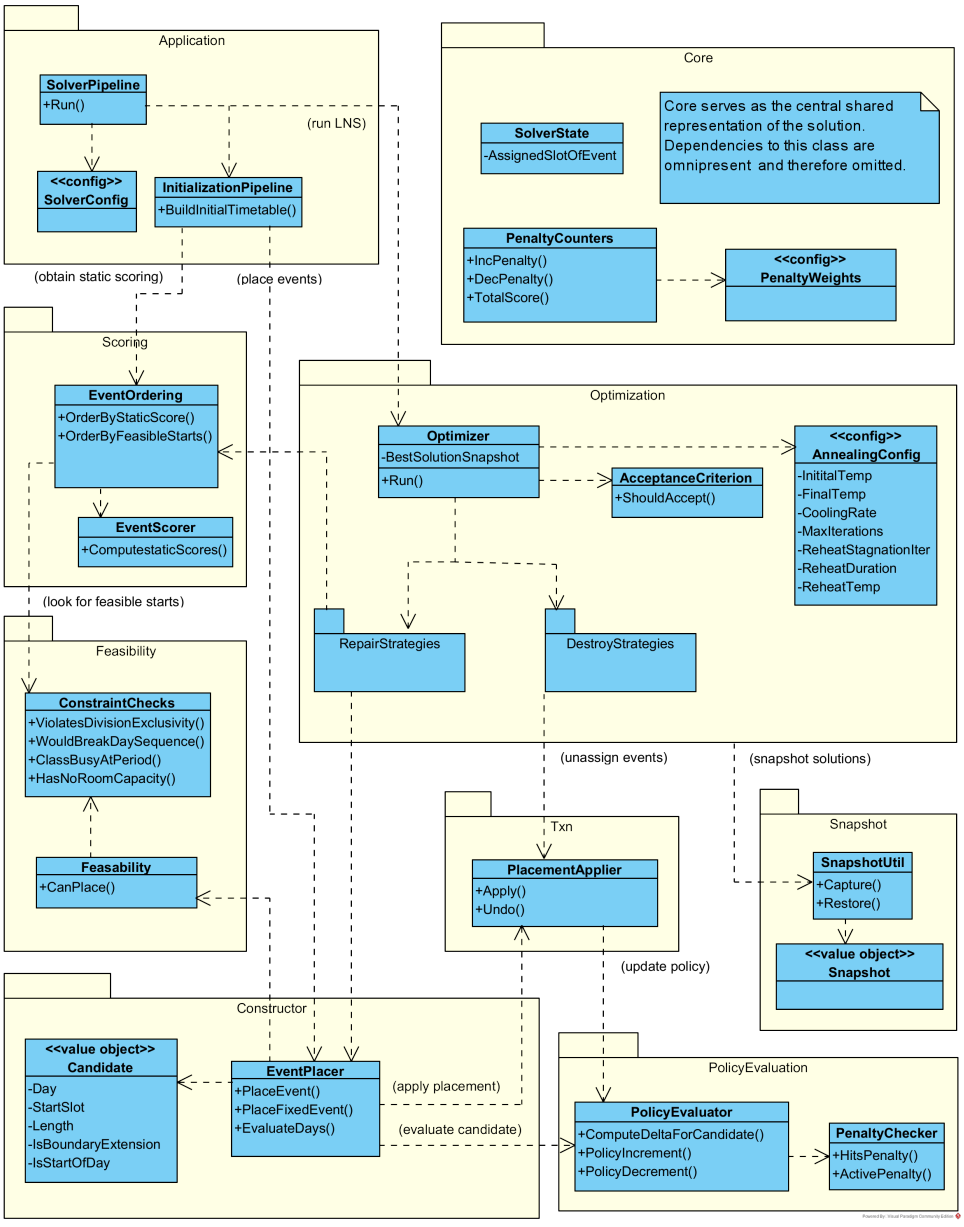

Figure 6.1: High-level architecture diagram of the solution.

The LNS is implemented within the `Optimization` component, which includes the definitions of the operators in `DestroyStrategies` and `RepairStrategies`. Destroy strategies remove events using the `Txn` component. In contrast, repair strategies operate similarly to the initial timetable construction, with events ordered in `Scoring` and placed with `Constructor`. During the LNS, solutions are snapshotted and restored using the `Snapshot` component. This component is not only responsible for managing solutions during the LNS but also for saving persistent solutions between program runs.

6.2 Implementation challenges

The implementation of the solver posed several challenges, primarily related to its practical use, maintaining consistency of the solution state with reversible modifications, ensuring efficient evaluation, and persistent snapshotting.

The solver was deliberately designed to be relatively independent of any specific input format while incorporating an XHSTT data loader [2, 7]. Specifications of the XHSTT input format can be seen in Appendix B. However, if other input formats were to be implemented, their only requirement is that the problem data be loaded into the `Core` state management component and that the parameters of the large neighborhood search be properly configured. An additional feature, not explicitly specified during the initial requirements analysis but found to be highly beneficial in practice, is the support for *fixed events*. Fixed events are events with immutable assigned timeslots, either provided directly in the input or determined by a simple preprocessing step. These events are excluded from modification during the search process. The use of fixed events is advantageous for highly constrained events, where manual placement by school administrators is preferred, as well as for all-day events, whose repeated removal and reconstruction would otherwise incur a significant computational cost.

As described in Chapter 4, the solver state manages numerous derived data structures in addition to static entities and the primary assignment of events to timeslots. These structures are tightly coupled, requiring every modification of the solution to be consistently

propagated across all of them. Consequently, the `Txn` component is responsible for managing most state transitions. The requirement for reversibility further increases this complexity. The solver adopts a transactional approach, in which each modification defines both its forward and inverse effects. Reversibility also introduces additional challenges in constraint evaluation, as changes in constraint violation counts must be correctly handled for both candidate placement and event unassignment. The transactional approach, combined with the use of numerous derived data structures, proves beneficial for performance. In particular, it eliminates the need for repeated global recomputation over all entities, allowing most operations to be handled locally. Further performance gains were achieved by using efficient, low-level data structures and by employing early termination in computationally intensive functions.

In addition to in-memory snapshots used during the search process, the solver supports persistent storage of solutions across program runs. This enables intermediate or final solutions to be saved in a JSON format and later restored without recomputation, which is especially useful for solution manipulation within the two phases of LNS. The persistence mechanism enables the entire solver state to be serialized and reconstructed consistently across all components, including derived data structures and objective function components. Internally, the solution is first captured into a snapshot and then transformed into a portable representation.

7 Experimental evaluation

The implemented solver has been tested on two real-world large-scale problem instances provided by SŠIPF. The goal of the experimental evaluation is to assess the ability of the proposed solver to generate feasible and high-quality timetables under realistic constraints. The evaluation focuses on three aspects: (i) reduction of hard constraint violations, which determines feasibility, (ii) optimization of soft constraints, reflecting timetable quality, and (iii) convergence behavior of the search process. We will first discuss how the problem instances were obtained, followed by a summary of the entities used. Then we will focus on experimental parameter setting, which was performed to determine suitable values for the simulated annealing temperature, the weights of the destroy and repair operators, and the reheating duration for both phases. Finally, we will discuss the results of the solver, focusing on possible improvements and shortcomings.

The experiments were conducted using the MetaCentrum [22] e-INFRA CZ distributed computing infrastructure on a high-performance computing node (AMD EPYC 7513, 3 GB RAM requested, Debian 12, 22 CPU, no GPU).

7.1 Problem instances

The experiments were conducted on two real-world datasets obtained from SŠIPF, corresponding to the academic years 2025/26 and 2024/25. These datasets were derived from manually created timetables used in practice, with additional data provided by the school administrators. Some elements of the required datasets, such as the allowed event room types and the complete list of room types, were not directly available and were extrapolated from the given data with the help of school administrators. Since teacher availability data were unavailable, availability constraints were synthetically generated within a scope similar to real timetabling demands. The preferred times of teachers ($\mathcal{S}_9$) were omitted during the evaluation since they could not be reasonably estimated or compared with the $\mathcal{S}_9$ of the used timetables. Both datasets exhibit comparable structural properties, with similar

numbers of classes, teachers, and rooms. The scope of the problem instances is described in Table 7.1.

Entity	2025/26	2024/25
Classes	52	53
mean occupancy rate	0.78	0.76
Total partitions	145	150
Total groups	272	269
Teachers	132	129
mean occupancy rate	0.37	0.38
Rooms	64	64
mean occupancy rate	0.66	0.67
Room types	9	9
Events	4928	4984
all-day events	8	22
lunch events	1126	1032
fixed events	48	49
mean event duration	1.24	1.23
mean allowed room types	1.63	1.57

Table 7.1: Entities and selected aspects of the used datasets.

7.2 Parameter setting

Parameter setting was performed on the 2025/26 problem instance, with the 2024/25 problem instance remaining untouched until the evaluation of the final results. A limitation of this setup is the availability of only two problem instances, which simplifies the setting process and may limit the generalization of the tuned parameters. It is important to keep the limited number of datasets in mind during all the experiments.

The parameters being discussed are the following. The initial temperature of simulated annealing T_0 (a). Simulated annealing is performed over a set number of iterations with a predefined final temperature. The cooling rate is adjusted to ensure the solver reaches the final temperature within the specified iteration count. The weight of the random events destroy operator w_d (b), with the weight of the

targeted class operator being $1 - w_d$. The weight of the random ordering repair operator w_r (c), with the weight of the feasible starts count operator being $1 - w_r$. The weight of an operator corresponds to the probability of its being chosen during the destroy or repair phase. The aforementioned parameters (a), (b), and (c) were tuned for both phases, with the second phase introducing one additional parameter: (d) Reheating duration R_d. The solver uses stagnation-based reheating with temperature jumps.

The scope of the destroy operators, i.e., the percentage of events removed during destruction, is not included in the parameter setting, as these values were fixed based on preliminary experiments and later readjusted in the alternative parameter setting (see Appendix D). Another parameter fixed in the preliminary experiments is the transitional multiplier for a given combination of destroy and repair operators, which multiplies the repair weight if a certain destroy operator is selected. The most important result is the multiplier applied at the transition from the random event destroy operator to the random ordering repair operator, which has been set to 0.001, since this combination of operators severely worsens the solution. The complete list of parameters provided by preliminary experiments is given in Appendix C. Due to the number of interacting parameters, full joint parameter setting would be computationally expensive. Therefore, the parameter setting process is performed sequentially, isolating the influence of individual parameters while keeping the remaining parameters fixed to reasonable baseline values determined from the preliminary experiments.

All of the following experiments are performed over 10 independent runs (2 repetitions on 5 separate initial timetables), with the results being the mean of all outputs in accordance with multi-run stochastic evaluation [23]. If not mentioned otherwise, the solver optimizes the objective function as a whole, often referred to as the score. In subsequent experiments, previously tuned parameters are fixed to their best-performing values, while the currently examined parameter is varied.

7.2.1 First phase

We begin by analyzing the effect of the initial temperature T_0 of the first phase, as it has the most significant impact on the exploration and exploitation balance of the simulated annealing process. The initial timetables on which HST is performed start with a mean score of $6,377,259 \pm 118,329$ with 512 hard constraint violations on average, out of which 435 are closed holes ($\mathcal{H}_2$). This means that the main optimization must be done by diminishing the number of closed holes. The initial temperature range was chosen from 4,000 to 18,000. As expected, for all of the chosen temperatures, the score is decreasing in exponential decay, as can be observed in Figure 7.1. The left-side plots show the overall convergence behavior, while the right-side plot presents the gap between the score and the best solution found for $T_0 = 16{,}000$. The results for all temperatures can be seen in Table 7.2. All of the timetables were successful in reducing hard constraints, with the mean hard constraint count being denoted by $\sum \mathcal{H}_i$ and its standard deviation being denoted by $\sigma_{\sum \mathcal{H}_i}$. The choice of $T_0 = 4{,}000$ is too slow, while higher temperatures provide comparable results, indicating that sufficient exploration is already achieved and further increases do not meaningfully improve solution quality. The choice of $T_0 = 16{,}000$ was made because it yielded the lowest total score.

For both weights of the random operators w_d and w_r, the values of 0.00, 0.25, 0.33, 0.50, 0.66, 0.75, and 1.00 are evaluated. The worst scores are expected for the values of 0 and 1, since these weights indicate the usage of just one operator with limited functionality. The results

T_0	Final score	$\sum \mathcal{H}_i$	$\sigma_{\sum \mathcal{H}_i}$	$\mathcal{H}_1$	$\mathcal{H}_2$	$\mathcal{H}_3$	$\mathcal{H}_5$	Rest of $\mathcal{H}_i$
4,000	615,500	29.5	5.2	0.1	25.8	1.4	2.2	0
6,000	516,659	17.2	4.4	0.0	14.4	0.5	2.3	0
8,000	520,187	20.0	5.6	0.0	17.7	0.4	1.9	0
10,000	497,181	17.0	3.2	0.0	14.3	0.4	2.3	0
12,000	487,331	16.1	2.9	0.0	13.3	1.1	1.7	0
14,000	491,911	16.4	3.4	0.1	13.4	0.7	2.2	0
16,000	469,967	13.7	3.3	0.0	11.5	0.3	1.9	0
18,000	471,611	14.7	5.4	0.1	12.2	0.3	2.1	0

Table 7.2: Impact of temperature parameter T_0 on solution quality in the first phase.

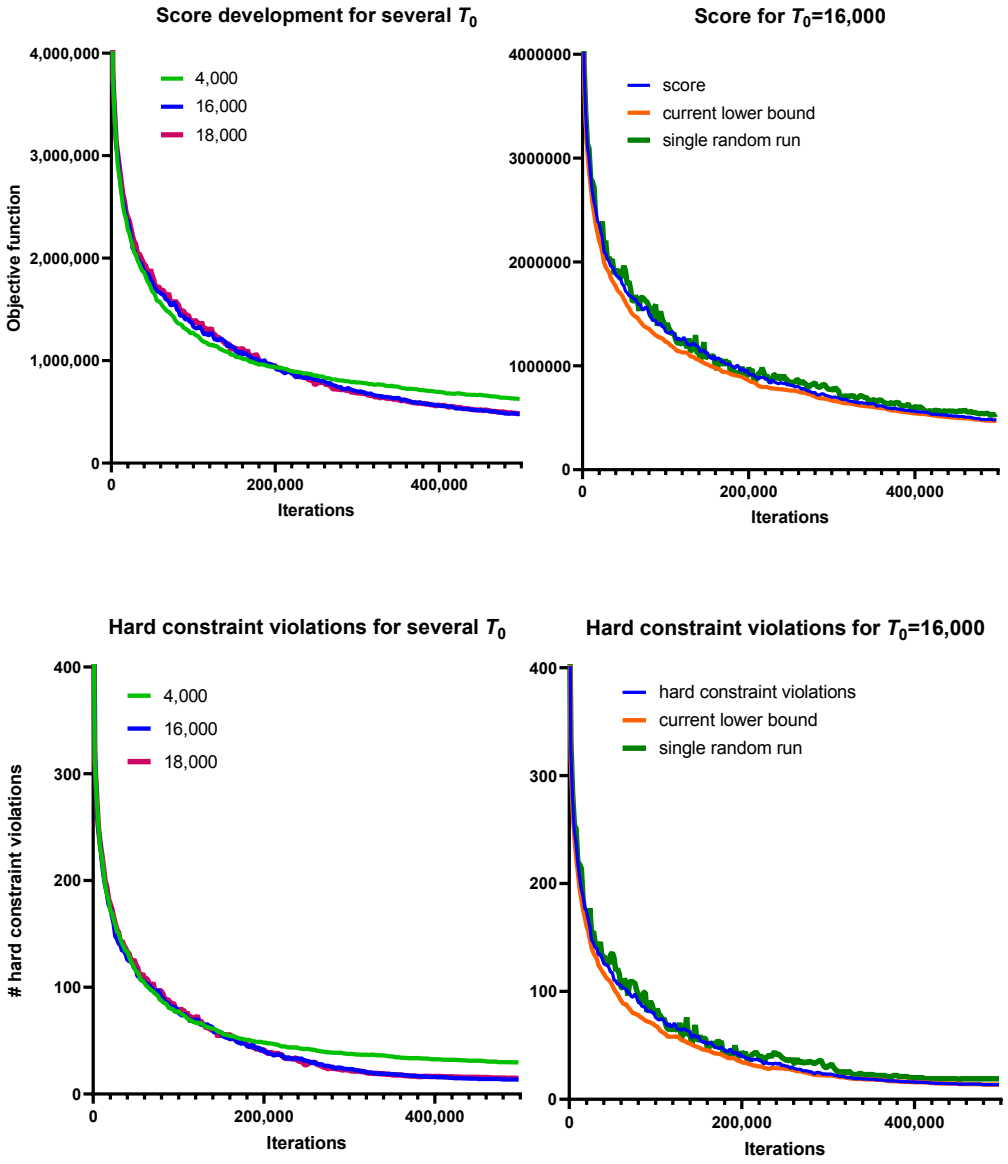

Figure 7.1: Objective function and hard constraint violations development for different choices of initial temperatures T_0 in the first phase.

for the destroy operators are shown in Table 7.3, and those for the repair operators are shown in Table 7.4. As expected, the usage of all the operators is justified since the weights of 0.00 and 1.00 provide by far the worst results. The final parameter values $w_d = 0.66$ and $w_r = 0.25$ were chosen because they yielded the largest reduction in hard constraints while achieving one of the lowest soft constraints scores. These findings indicate that the first phase benefits from a moderate level of randomness to escape poor regions of the search space, while still relying on structured repair to enforce feasibility. Since reheating will not be performed in the first phase, this concludes all the necessary experiments.

w_d	Final score	$\sum \mathcal{H}_i$	$\mathcal{H}_1$	$\mathcal{H}_2$	$\mathcal{H}_3$	$\mathcal{H}_5$	weighted $\sum \mathcal{S}_i$
0.00	1,045,384	67.0	0.6	64.4	1.1	0.1	375,384
0.25	556,780	23.4	0.0	21.6	0.3	1.5	322,780
0.33	523,584	19.5	0.1	17.5	0.3	1.6	328,584
0.50	474,773	15.6	0.0	13.6	0.3	1.7	318,773
0.66	438,924	11.4	0.0	8.2	0.4	2.8	324,924
0.75	471,946	13.2	0.2	10.7	0.4	1.9	339,946
1.00	603,418	16.6	1.0	9.5	1.5	4.6	437,418

Table 7.3: Impact of random events destroy weight w_d on solution quality in the first phase.

w_r	Final score	$\sum \mathcal{H}_i$	$\mathcal{H}_1$	$\mathcal{H}_2$	$\mathcal{H}_3$	$\mathcal{H}_5$	weighted $\sum \mathcal{S}_i$
0.00	483,921	15.4	0.2	11.0	0.5	3.7	329,921
0.25	453,597	12.3	0.2	9.8	0.3	2.0	330,597
0.33	455,344	13.1	0.0	9.7	0.6	2.8	324,344
0.50	454,540	12.4	0.0	9.4	0.4	2.6	330,540
0.66	470,446	12.9	0.0	10.2	0.6	2.1	341,446
0.75	460,736	12.8	0.0	10.6	0.5	1.7	332,736
1.00	525,577	17.0	0.0	13.6	0.7	2.7	355,577

Table 7.4: Impact of random ordering repair weight w_r on solution quality in the first phase.

7.2.2 Second phase

For the initial timetables in the second phase, five outputs from the previously chosen first-phase parameters on different initial timetables were sampled at random. This reflects the real solver pipeline, where solutions of the first phase serve as input to the second phase. The experimental protocol matched that of the first phase: each instance was run twice on each initial timetable, and results were reported as the mean across 10 repetitions. The second phase is more characterized by the punishing weight of the non-unique lesson constraint violations $\mathcal{S}_1$, which has weight in the multitudes of all the other soft constraints. As such, we will compare the resulting timetable based on three main values: the mean of hard constraints $\sum \mathcal{H}_i$, the mean of non-unique lessons $\mathcal{S}_1$, and the mean weighted sum of all other soft constraints $\sum_{i>1} w_i S_i$. In a more constrained environment, the search operates in a much narrower neighborhood, where large structural changes are less beneficial and local adjustments dominate.

As before, we will begin with the initial temperature T_0. Candidate values have been determined based on preliminary tests as 200, 300, 500, 700, 900, 1,100. The decrease of the objective value for selected temperatures is shown in Figure 7.2. The decrease is not as severe as in the first phase since the values of both soft and hard constraints have already been optimized, while still following a curve with a pronounced elbow, as expected. The mean initial score of 432,077 is reduced below 330,000 for all temperatures. As shown in Figure 7.2, higher temperatures, such as 1,100, lead to an overly lenient acceptance criterion, disrupting the found solution in the early stages of explo-

T_0	Final score	$\sum \mathcal{H}_i$	$\mathcal{S}_1$	$\sum_{i>1} w_i S_i$	$\mathcal{S}_3$	$\mathcal{S}_4$	$\mathcal{S}_5$	$\mathcal{S}_6$
200	323,105	8.9	21	191,305	623	54	50	101
300	317,949	8.8	20	189,949	619	53	52	97
500	303,091	7.9	19	185,691	618	50	49	97
700	305,815	8.0	19	187,815	626	49	51	99
900	304,765	7.7	19	190,165	626	48	53	102
1,100	309,435	8.4	18	189,635	623	52	52	99

Table 7.5: Impact of temperature parameter T_0 on solution quality of the second phase.

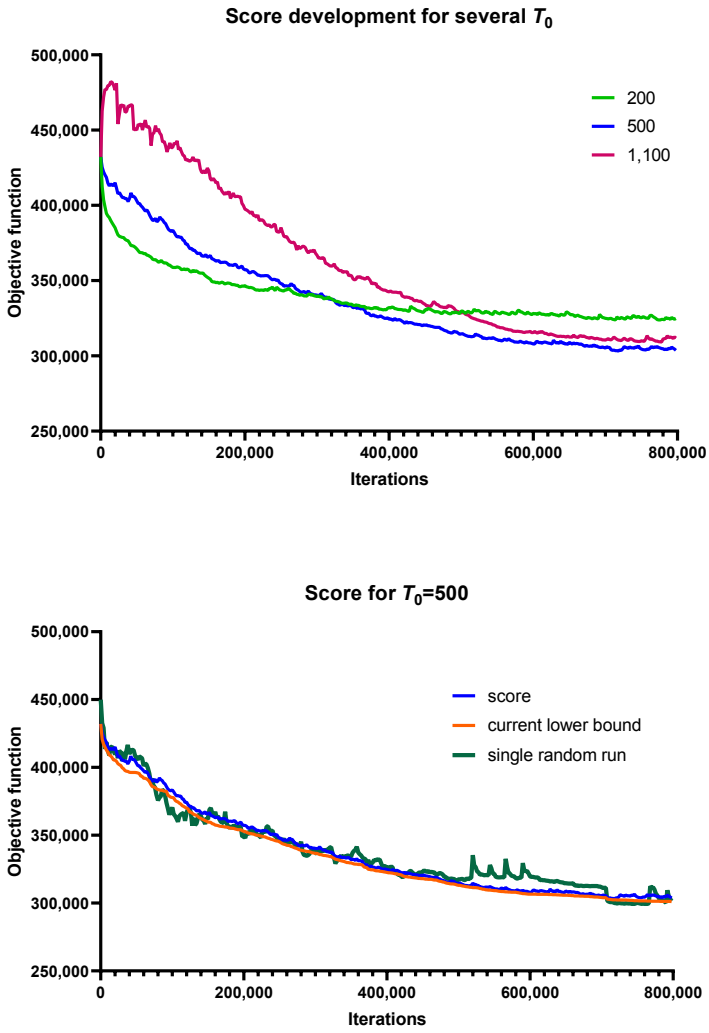

Figure 7.2: Objective function development for different choices of initial temperatures T_0 of the second phase.

ration. A smaller temperature, such as 200, has the opposite issue of not allowing sufficient exploration of the search space. An interesting feature of the figures is the sudden spikes in the total score at higher iterations caused by the temperature jumps. Based on the data from Table 7.5, an initial temperature of $T_0 = 500$ was selected.

As before, for both weights of the random operators w_d and w_r, the values of 0.00, 0.25, 0.33, 0.50, 0.66, 0.75, and 1.00 are evaluated. The results of both experiments are recorded in Table 7.6 and Table 7.7. The interpretation of the results is less clear than in the first phase, as the differences between the weights are very small, to the point of being comparable. The destroy weights of $w_d = 0.00$ and $w_d = 1.00$ yield the worst results, indicating that both destroy operators are needed. The setting of $w_d = 0.50$ yields the biggest decrease in the objective function. The solver is less sensitive to w_d and w_r in the second phase, indicating that operator choice plays a secondary role once feasibility is achieved. We can safely conclude that the exact parameter values of the operators are not central, and we set them to $w_d = 0.50$ and $w_r = 0.50$.

w_d	Final score	$\sum \mathcal{H}_i$	$\mathcal{S}_1$	$\sum_{i>1} w_i S_i$	$\mathcal{S}_3$	$\mathcal{S}_4$	$\mathcal{S}_5$	$\mathcal{S}_6$
0.00	316,163	8.5	20	191,563	612	51	58	101
0.25	307,643	8.6	18	186,243	625	50	49	101
0.33	306,943	8.3	19	185,343	617	51	49	99
0.50	306,677	8.2	19	186,677	613	53	52	95
0.66	314,599	8.6	19	189,799	624	55	51	99
0.75	314,676	8.6	19	189,876	629	53	50	102
1.00	352,719	8.2	26	219,720	670	81	48	125

Table 7.6: Impact of random events destroy weight w_d on solution quality of the second phase.

w_r	Final score	$\sum \mathcal{H}_i$	$\mathcal{S}_1$	$\sum_{i>1} w_i S_i$	$\mathcal{S}_3$	$\mathcal{S}_4$	$\mathcal{S}_5$	$\mathcal{S}_6$
0.00	321,474	8.6	21	192,874	642	54	51	104
0.25	302,817	8.0	18	187,217	620	52	51	98
0.33	302,299	7.9	18	186,499	619	50	50	101
0.50	301,706	8.0	19	184,507	615	50	49	100
0.66	305,348	8.3	18	186,949	611	51	52	100
0.75	306,072	8.6	17	185,672	612	50	50	102
1.00	302,388	8.2	17	187,389	600	50	55	100

Table 7.7: Impact of random ordering repair weight w_r on solution quality of the second phase.

The only remaining experimental parameter is the duration of the reheating jumps. The preliminary values for the experiment were chosen as 0, 30, 60, 100, and 200. The results can be seen in Table 7.8. Based on the experiments, it is unclear how significant the impact of reheating on solution quality is. The results indicate that the biggest impact could be the higher likelihood of eliminating hard constraint violations by returning to higher temperatures capable of enabling local-minima escape. This hypothesis is confirmed by $R_d = 30$ and $R_d = 200$; however, for the other $R_d > 0$, the hard constraints are not decreasing. The impact on soft constraints seems minimal, if any. The table also shows the standard deviation of the objective value, which is not affected by small R_d but steadily increases with $R_d > 100$, indicating that with higher duration, the results are less predictable. The impact of reheating on solution quality is inconsistent across configurations, suggesting that reheating primarily acts as a safeguard against stagnation rather than a systematic improvement mechanism. The value of R_d was set to 30, as reheating could help eliminate additional hard constraints, even if it is a negligible feature of the proposed solution.

R_d	Final score	σ score	$\sum \mathcal{H}_i$	$\mathcal{S}_1$	$\sum_{i>1} w_i S_i$	$\mathcal{S}_3$	$\mathcal{S}_4$	$\mathcal{S}_5$	$\mathcal{S}_6$
0	311,685	11,652	8.9	18	186,485	618	50	50	100
30	303,013	10,258	8.2	19	185,013	618	48	50	101
60	312,175	11,025	8.8	19	185,775	614	50	51	99
100	316,164	12,370	8.9	20	187,764	615	53	51	99
200	307,840	15,491	8.2	19	188,240	622	50	51	103

Table 7.8: Impact of the reheating duration R_d on solution quality of the second phase.

7.3 Final results

Since the final parameters were determined by the aforementioned experiments, this chapter aims to examine more closely the final solution, along with its resulting timetables and constraint violations. The results will be examined for both datasets: 2025/26, on which all the preliminary experiments and parameter setting were performed, and the previously unseen 2024/25. As was the case previously, all

evaluations were performed on 5 initial timetables. The number of repetitions for each initial timetable was increased to 10, for a total of 50 runs, from which the mean values were calculated. For the initial timetables of the second phase, the results of the first phase timetables (each with a different initial timetable before the first phase) were sampled at random.

As mentioned in Chapter 5, after the second phase, a short LNS with hill climbing (HC) with a relaxed event placement strategy is applied. The HC deprioritizes the placement of events into open holes so that a more suitable candidate is selected for groups of the same partition sharing timeslots. The HC is performed with only the random event destroy operator (as it is used to disrupt groups of the same partition at the same timeslots, which are located at the starts and ends of days). The repair operator is chosen uniformly. The most meaningful difference in HC is setting the $Poor\Delta Limit$ (as seen in Algorithm 2) to zero, with the rest of the parameters being inherited from the second phase. This change allows for splitting of events of multiple groups at the boundaries of days. The results of this phase can be observed when comparing the last two columns of Table 7.9. We can see that the final hill climbing provides a small decrease in score, which is mostly visible in the structure of the timetables themselves. Preliminary experiments show that the impact of this phase could be amplified by modifying the size of the destroy neighborhood, which needs to be the focus of further research.

7.3.1 Convergence behavior and solution quality

Figure 7.3 displays the development of the objective function during the search. As expected, based on parameter setting, the solver minimizes the total objective function as well as the soft constraints. The solver behaves similarly across both data instances. The interesting difference in performance during the search can be seen in the soft constraint reduction (without $\mathcal{S}_1$) in the first phase, where 2024/25 has far lower values during the feasibility optimization. This difference cannot be easily explained, as both datasets have similar scopes and configurations, with the only meaningful differences being slightly tighter environmental constraints in 2025/26 and a higher number of

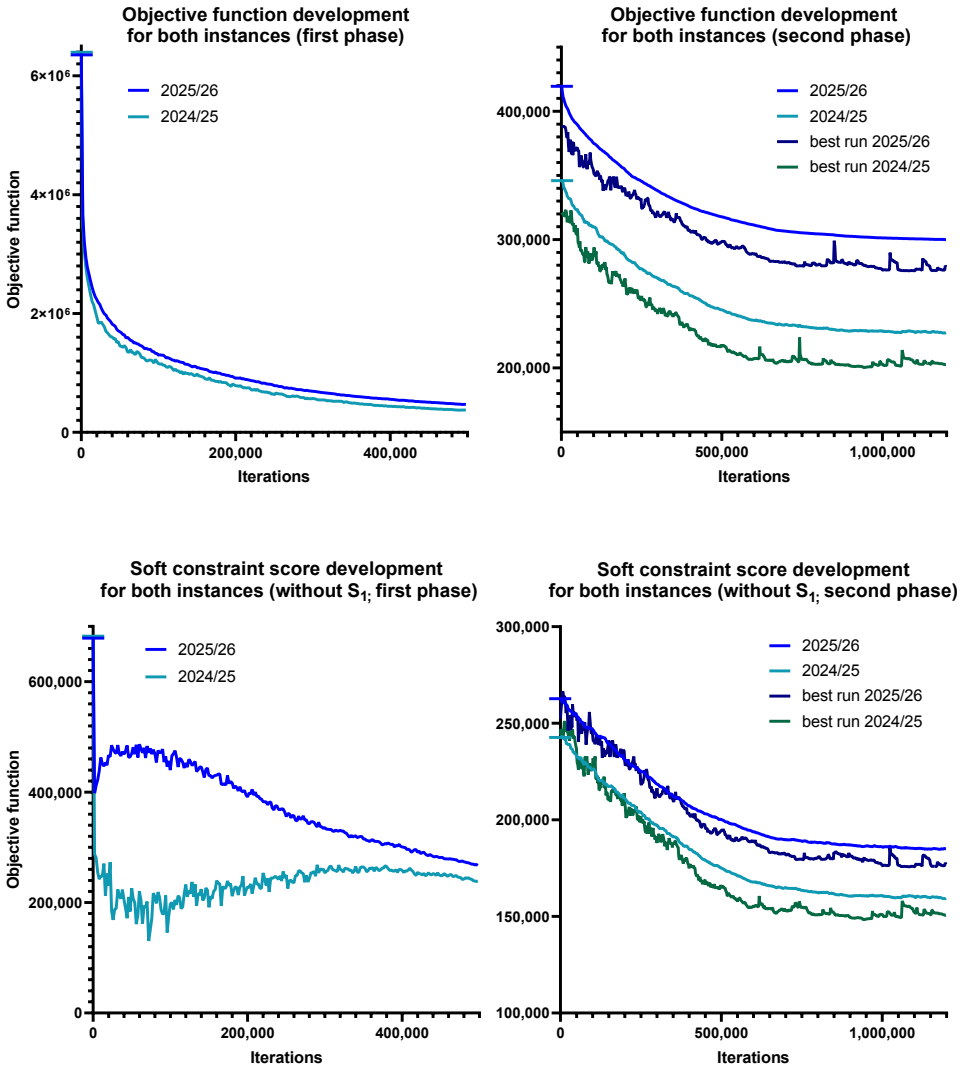

Figure 7.3: Objective function development during the final results evaluation. The top figures showcase the objective function; the bottom figures showcase soft constraint components of the objective function.

all-day events in 2024/25. Selected examples of the 2025/26 timetable output can be seen in Appendix F.

As we can see in Table 7.9, all phases were needed to further optimize the solution and lower the objective function. The final score results can be seen in Table 7.10, with the expanded results presented in Appendix E. The comparison for the evaluation is given by the score of the actual timetable used, which was created manually by the school administrators. When comparing only the total objective function, the solver is significantly better, achieving a score of about 86% of the manual solution for the 2025/26 and 67% for the 2024/25 instance. The solver consistently outperforms the manual timetable in terms of the objective function, primarily due to a substantial reduction in hard constraint violations. The concrete mean constraint violation counters can be seen in Figure 7.4 and Figure 7.5. While the solver produces far fewer hard constraint violations, they are of different types than those in the manual solution. The manual solution usually has hard constraints involving teachers ($\mathcal{H}_6$, $\mathcal{H}_7$), yet the solver usually has troubles with closed holes ($\mathcal{H}_2$) and lunches ($\mathcal{H}_5$). The non-uniqueness of the lessons ($\mathcal{S}_1$) is problematic for the solver, as in the 2025/26 instance, it rarely reaches the constraint counts of the manual solution. The solver mostly has problems with lessons that have high repetition for non-trivial partitions of the class; if the group

Instance	Initial score	First phase	Second phase	Final score
2025/26	6,377,259	458,157	301,913	299,448
2024/25	6,266,416	360,574	219,765	217,098

Table 7.9: Objective function development after individual phases.

Instance		First phase	Final score	$\sum \mathcal{H}_i$	$\mathcal{S}_1$	$\sum_{i>1} w_i S_i$
2025/26	Mean	458,157	299,448	8.3	17.1	182,368
	Best run	429,260	285,085	8.0	16.0	173,085
	Manual	—	350,185	14.0	11.0	188,185
2024/25	Mean	360,574	217,098	3.7	10.0	159,658
	Best run	309,255	196,915	4.0	5.0	146,915
	Manual	—	321,625	14.0	11.0	159,625

Table 7.10: Final evaluation results on both problem instances.

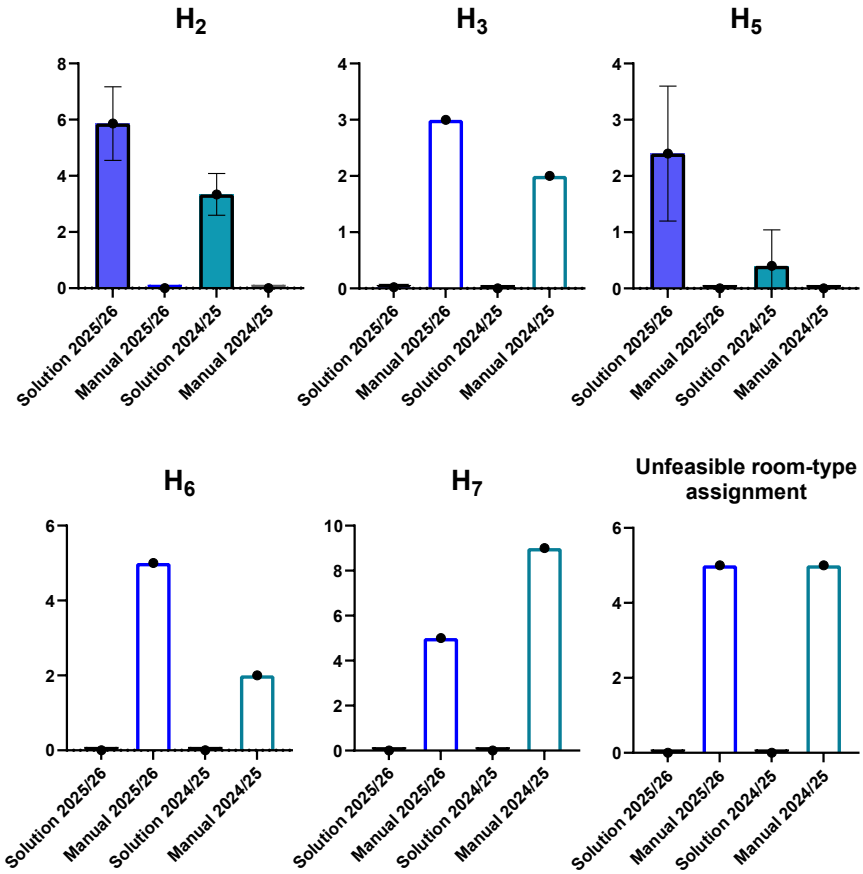

Figure 7.4: The count of hard constraint violations in the final solutions.

has 5 lessons of `English` a week, it is hard to abide by both $\mathcal{H}_2$ and $\mathcal{S}_1$. For the remaining soft constraints, the solver achieves slightly better yet comparable results to the manual timetable. The best of the 50 test runs is able to eliminate soft constraints to about 92% of the manual solution, with the mean level being about 96% for the 2025/26 instance and the same as the manual timetable for the 2024/25 instance. The results also show the generated timetables are relatively stable, with a low deviation of individual soft constraint violations.

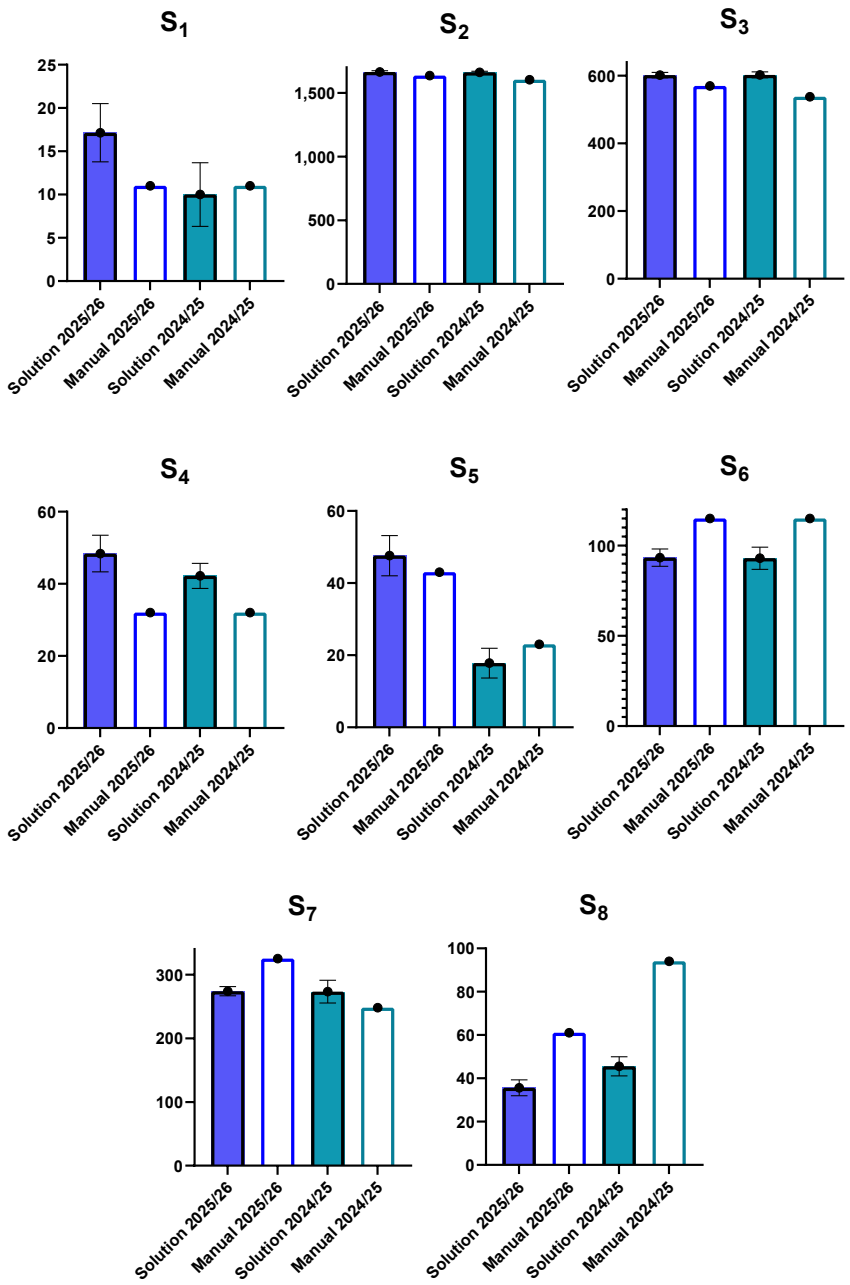

Figure 7.5: The count of soft constraint violations in the final solutions.

7.3.2 Further notes

When focusing on the performance of individual destroy-and-repair operators, we can observe a change in their purpose during the search process. In the first phase, the random operators mostly serve to promote more exploration of the search space, while not lowering the objective function itself; this allows for more structured, targeted class operators to reduce it. In the second phase, the random events destroy operator becomes more essential, as soft constraints often affect the shape of day boundaries and, as such, lower the objective value. Targeted class operator takes on the role of search space exploration in the second phase, since the core structure of days is already mostly optimal from the first phase. This role reversal reflects the transition from feasibility-driven search to fine-grained optimization. The concrete performance of the operators in both phases is shown in Table 7.11.

	First phase		Second phase	
Operator	Acceptance rate	Δf	Acceptance rate	Δf
Random events	35.3 %	13	20.9 %	-2.7
Targeted class	59.0 %	-73	29.0 %	1.2
Random ordering	50.8 %	51	25.6 %	1.6
Feasible start count	42.7 %	-36	24.6 %	-1.3

Table 7.11: Performance of operators across the two phases. Δf denotes the mean change in the objective function after acceptance.

An important aspect of the experimental evaluation that has not yet been discussed is solver performance. This aspect was deliberately deprioritized, as overall runtime is not the primary focus of the solver. When executed in batch on a high-performance compute node (AMD EPYC 7513, 3 GB RAM, Debian 12, 22 CPU, no GPU), the first phase with 500,000 iterations required 21.5 ± 2.4 minutes on average, while the second phase with 1,200,000 iterations required 38.1 ± 3.2 minutes. The relatively short runtime is beneficial for the actual usage of the solver, as it allows to run the solver several times and using the best results. Since the first phase and the second phase are performed independently, this pipeline could be further utilized by performing the second phase only on a few best results of the first phase.

7.4 Alternative parameter setting

In addition to the primary parameter configuration obtained through sequential setting, an alternative parameter setting was evaluated to assess the robustness of the solver with respect to changes in (i) the scope of destroy operators and (ii) the weighting of selected soft constraints. This experiment was partly motivated by the perceived differences between the solutions and manually created timetables. The goal of this experiment is therefore to test the robustness of the solver by assessing whether it produces correct timetables under slightly changed requirements.

As the current solver has trouble with closed holes ($\mathcal{H}_2$) and student lunches ($\mathcal{H}_5$), their weight has been slightly increased. The weight of soft constraints was also changed to better target the soft requirements at which the solver is lagging behind manual timetables, with the exact weights being available in Table C.2. The reweighting did not require any changes to the simulated annealing parameters. The scopes of destroy operators were changed in accordance with the experiments outlined in Appendix D, motivated by the differences in constraint weight space and the subsequent need for neighborhood adjustment.

The objective function decrement across the solver phases can be observed in Table 7.12. The results of the retuning using the same methodology as in the final results (50 repetitions with 5 initial timetables) can be seen in Table 7.13, with the expanded results presented in Appendix E. Compared with the original results in Table 7.10, we see that the increase in the weights of H_2 and H_5 had a significant impact on the count in hard constraint violations on the 2025/26 instance, with no impact on the 2024/25. This could be explained by 2025/26 having a higher hard violations count in the original parameter setting, indicating more room for improvements.

Instance	Initial score	First phase	Second phase	Final score
2025/26	7,184,216	427,684	302,485	297,684
2024/25	7,073,720	362,034	235,316	233,189

Table 7.12: Objective function development after individual solver phases for alternative parameter setting.

Instance		First phase	Final score	$\sum \mathcal{H}_i$	$\mathcal{S}_1$	$\sum_{i>1} w_i S_i$
2025/26	Mean	427,299	297,684	5.3	19.6	195,524
	Best run	385,325	255,205	2.0	22.0	187,205
	Manual	—	350,515	14.0	11.0	188,515
2024/25	Mean	362,034	233,189	3.7	10.3	168,029
	Best run	329,605	207,235	3.0	6.0	159,235
	Manual	—	332,135	14.0	11.0	170,135

Table 7.13: Final evaluation results on both problem instances for alternative parameter setting.

The changes in selected constraint violation counts can be seen in Figure 7.6. The solver readjusted to the new weights, as evidenced by shifts in several constraint counts. The impact on the overall timetable quality is mixed, with the solver successfully readjusting, indicating trade-offs between competing soft constraints. The mean soft score of the 2025/26 instance is at 104% of the manual timetable, with the soft score of the 2024/25 instance being at the 99% of the manual timetable. We can therefore see a behavior shift between the datasets; while both results worsened compared to the original parameter settings, the 2025/26 instance had more troubles readjusting to the new constraint weight space. This could be partly explained by the elimination of more hard penalties in the 2025/26 instance, further constraining the search space. However, the best results for both instances achieve far better solutions than the manual timetable score.

7.5 Discussion

Even though the solver succeeded in surpassing the manual solution, there are several important real-world insights that need to be taken into account when evaluating the timetable. The number of hard constraints is the highest improvement. Yet, remaining closed holes ($\mathcal{H}_2$) still make the timetable infeasible. These constraints need to be addressed through manual corrections; some could be allowed or resolved by changing the allowed event room types, similarly to the manual solution. The distribution of soft constraints between the solver and manual solutions is also different, with the solver exceeding events at k_9 elimination ($\mathcal{S}_6$), while having complications with elimination

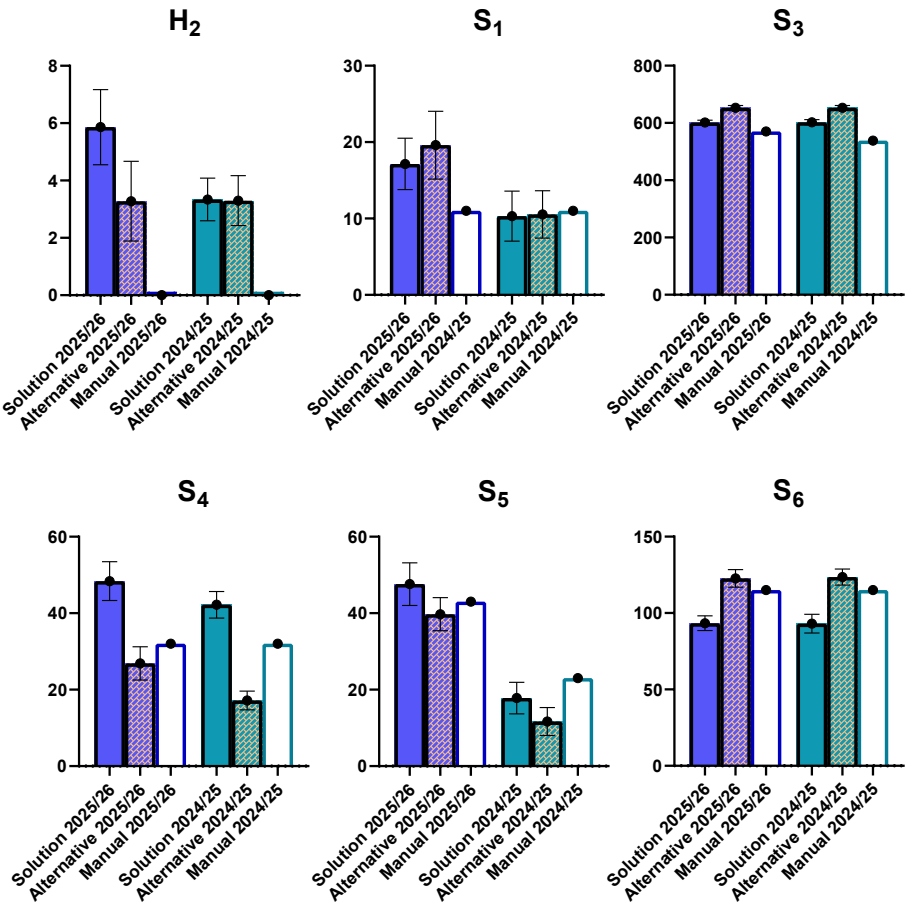

Figure 7.6: The count of selected soft constraint violations in the final and alternative solutions.

events on Friday afternoons ($\mathcal{S}_4$). However, all of the soft constraints remain within acceptable and comparable boundaries. The higher non-uniqueness of lessons ($\mathcal{S}_1$) is not as problematic, as it mostly involves language lessons with high repetition over the biweekly period, which could appear multiple times in a day. On rare occasions, non-uniqueness involves other types of lessons, such as `P.E.`, which are more problematic and would require manual refinement.

7.5.1 Additional exceptions of manual timetables

The manual solutions were used as the main comparison for the generated timetables. While the hard constraint violations were outlined earlier, it is important to mention that the manual solutions have additional exceptions, which the solver registers as infeasible. One of these are room types of events. Both datasets contain at least 5 infeasible room type assignments, which were done only to assign events with all feasible rooms being occupied. The other technique the manual timetable employs regarding rooms is splitting the event across multiple rooms, with each timeslot taking place in a different room of a different room type.

To fit the longer events while providing lunches for all students, the manual timetable also occasionally splits the events into multiple parts, with lunch break being in the middle. This technique is employed rather sparingly, with only a handful of examples.

7.5.2 Missing features

One feature of the timetable that was not addressed through the constraints is the teacher span. When compared with the manual solution, the total teacher span (the sum of the differences between the starts and ends of days for teachers) increased by 12% while the total number of days when teachers have at least one event increased by 7%. These downsides are predictable and could be mitigated by introducing additional soft constraints in the future or by being more strict about teacher availability.

Another feature that might significantly improve the quality of the generated timetables is additional soft constraint focused on the distribution of lessons within the biweekly timetable of classes. While $\mathcal{S}_1$ and $\mathcal{S}_2$ target the desired lesson distribution, they are both very local and very crude. The solver should try to equalize the number of lessons between weeks, ideally balancing each gap between two instances of the same event to make the schedule as stable as possible. For example, it should be preferable to schedule `Math` on Monday of the odd week and Thursday of the even week rather than on two consecutive days followed by an eight-day gap.

Since the solver models lunches as any other events, it might also be useful to create additional constraints on the placement of lunches. For example, in a real world timetable it is convenient for students if a lunch is their last event, since this means they do not have to wait in the facility for additional events. A future version of the solver might thus prefer lunches at the ends of days for groups.

The explicit assignment of physical rooms was intentionally omitted from the proposed model, as it is a trivial problem compared to the timeslot assignment. The feasibility of rooms is currently handled at the level of room types, while detailed room allocation should be delegated to a separate solve. This solver is not a part of this thesis, however, a prototype version is in development with positive results.

The last feature omitted on purpose was the inclusion of a small number of events with multiple groups of different classes. While these events can be created using the fixed events, their timeslots must be fixed in advance, and they are not part of the optimization process.

7.5.3 Summary

The proposed solver demonstrates an improvement over manually constructed timetables in terms of feasibility, primarily through a substantial reduction in hard constraint violations. While manual timetables can rely on implicit exceptions, the solver operates strictly within the defined constraint model, resulting in solutions that are structurally more consistent and closer to modeled feasibility. At the same time, this improvement comes with a trade-off. Rather than uniformly reducing all soft constraints, the solver redistributes them, prioritizing reduction in hard constraints and certain structural properties at the expense of others. In particular lesson distribution ($\mathcal{S}_1$) remains the most challenging soft constraint. Despite these limitations, the solver produces stable and consistently high-quality solutions across both problem instances with acceptable constraint violation counts, indicating robustness of the proposed approach. The remaining infeasibilities, such as closed holes ($\mathcal{H}_2$), are limited in scope and can be resolved with relatively minor manual adjustments, mostly involving hard constraint violation forbidden by the system.

Overall, the solver is not a fully autonomous replacement for manual timetable construction, but it serves as a highly effective decision-

support tool. An additional advantage of the proposed solver is its computational efficiency. Although runtime is not the primary objective in this domain, the solver is sufficiently fast to generate a large number of candidate timetables, out of which the best performing ones can be selected for manual refinement. By generating near-feasible timetables with significantly reduced manual effort, the solver provides a practical foundation for semi-automated timetable generation in real-world settings.

8 Conclusion and future work

This thesis addressed the high school timetabling problem for real-world instances derived from the Czech High School of ICT, Postal Services, and Finance in Brno (SŠIPF). The problem was formulated as a constrained optimization task involving the assignment of events to timeslots while satisfying organizational and pedagogical requirements. Particular attention was given to the specific characteristics of Czech high school timetables, especially the requirement for continuous student timetables, with lunch breaks as the only allowed interruption.

To solve the problem, a timetabling solver based on large neighborhood search combined with simulated annealing was proposed, inspired by the slack induction by string removals algorithm. The approach employed a two-phase optimization framework: the first phase focused on reducing hard constraint violations, while the second phase optimized soft constraints. After the second phase, a short refinement using hill climbing was applied. The solver used specialized destroy-and-repair operators designed to exploit the structural properties of timetables and support effective exploration of the search space.

The proposed approach was evaluated on two real-world datasets derived from manually constructed timetables. The experimental results showed that the solver consistently produced near-feasible solutions with substantially fewer hard constraint violations than the manual timetables while achieving comparable soft constraint quality. Although the generated timetables still require manual refinement, the results indicate that the proposed solver can effectively support practical timetable construction and significantly reduce the complexity of manual timetabling. The main limitation of the experimental evaluation is the use of only two datasets, as additional real-world data were not available.

Future work should focus on extending the constraint model to improve the practical quality of generated timetables. One important extension is the inclusion of additional teacher-oriented requirements, particularly the minimization of teacher daily spans and the reduction of fragmented teaching timetables. Another important direction is

improving lesson distribution across the biweekly timetable, aiming for more stable, evenly spaced lessons rather than relying solely on current constraints. These changes should be implemented with a new parameter setting, also focused on destroy operator scopes.

Several algorithmic improvements could further increase the quality and robustness of the proposed solver. The simulated annealing framework could be improved by adopting more advanced reheating and diversification strategies to better escape local optima. Performance gains could be achieved by parallelizing selected parts of the search process, particularly candidate evaluation and independent neighborhood exploration. Another possible extension, though requiring substantial refactoring of the current architecture, would be the incorporation of hybrid optimization techniques such as constraint programming or evolutionary algorithms.

An important direction for future work is the practical deployment of the solver within the timetable creation process at SŠIPF. Due to the relatively short runtime of the optimization, the solver can generate a large number of alternative timetables, allowing school administrators to choose solutions that best match practical preferences not explicitly captured by the model. In practice, the generated timetables would still primarily serve as high-quality drafts, requiring final manual corrections and adjustments. Further work is also needed on integration with existing school information systems, particularly on reliable import and export handling for the Škola OnLine information system, whose timetable representation requires additional preprocessing and postprocessing support.

Overall, the proposed solver demonstrates that large neighborhood search combined with simulated annealing can effectively address complex real-world high school timetabling requirements. This is confirmed by the school administrators, who are satisfied with the current results. While they confirmed that additional requirements regarding event distribution and teacher spans are needed, they deemed the timetables usable and of at least comparable quality to the real manually created timetables. They were particularly impressed by the successful decrease in constraint violations. Although the generated timetables still require manual refinement, the approach provides a practical foundation for supporting timetable construction in real educational environments, saving the school valuable time and resources.

Bibliography

1. CESCHIA, Sara; DI GASPERO, Luca; SCHAERF, Andrea. Educational timetabling: Problems, benchmarks, and state-of-the-art results. *European Journal of Operational Research*. 2023, vol. 308, no. 1, pp. 1–18.
2. POST, Gerhard; AHMADI, Samad; DASKALAKI, Sophia; KINGSTON, Jeffrey H.; KYNGAS, Jari; NURMI, Kimmo; RANSON, David. An XML format for benchmarks in high school timetabling. *Annals of Operations Research*. 2012, vol. 194, no. 1, pp. 385–397.
3. APPLIED SOFTWARE CONSULTANTS. *aSc TimeTables*. n.d. Available also from: `https://www.asctimetables.com/`. Accessed: 2026-04-08.
4. BAKALÁŘI SOFTWARE S.R.O. *Rozvrh hodin*. n.d. Available also from: `https://www.bakalari.cz/Timetable`. Accessed: 2026-04-08.
5. SKOLARIS SOFTWARE S.R.O. *Skolaris: Online school timetable software*. n.d. Available also from: `https://skolaris.net/`. Accessed: 2026-04-08.
6. RUIZ-TORRUBIANO, Rubén; KNOPP, Sebastian; KRYSTALLIDIS, Andreas; WOLF, Lukas Matthias. A scheduling perspective on modular educational systems in Europe. *Heliyon*. 2024, vol. 10, no. 21, e39694.
7. POST, Gerhard; KINGSTON, Jeffrey; AHMADI, Samad; DASKALAKI, Sophia; GOGOS, Christos; KYNGAS, Jari; NURMI, Kimmo; MUSLIU, Nysret; PILLAY, Nelishia; SANTOS, Haroldo. XHSTT: An XML archive for high school timetabling problems in different countries. *Annals of Operations Research*. 2011, vol. 218, pp. 295–301.
8. DE WERRA, D. An introduction to timetabling. *European Journal of Operational Research*. 1985, vol. 19, no. 2, pp. 151–162.

9. COOPER, Tim B.; KINGSTON, Jeffrey H. The complexity of timetable construction problems. In: BURKE, Edmund; ROSS, Peter (eds.). *Practice and Theory of Automated Timetabling*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1996, pp. 281–295.
10. KRISTIANSEN, Simon; SØRENSEN, Matias; STIDSEN, Thomas R. Integer programming for the generalized high school timetabling problem. *Journal of Scheduling*. 2015, vol. 18, no. 4, pp. 377–392.
11. AVELLA, Pasquale; D'AURIA, Bernardo; SALERNO, Saverio; VASIL'EV, Igor. A computational study of local search algorithms for Italian high-school timetabling. *Journal of Heuristics*. 2007, vol. 13, no. 6, pp. 543–556.
12. FONSECA, George H. G.; SANTOS, Haroldo G.; CARRANO, Eduardo G. Integrating matheuristics and metaheuristics for timetabling. *Computers & Operations Research*. 2016, vol. 74, pp. 108–117.
13. TALBI, El-Ghazali. *Metaheuristics: From Design to Implementation*. Hoboken, NJ, USA: John Wiley & Sons, 2009.
14. FERNANDES, Carlos; CALDEIRA, João; MELICIO, Fernando; ROSA, Agostinho. Evolutionary algorithm for school timetabling. In: *Annual Conference on Genetic and Evolutionary Computation*. 1999, p. 1777.
15. TASSOPOULOS, Ioannis X.; BELIGIANNIS, Grigorios N. Using particle swarm optimization to solve effectively the school timetabling problem. *Soft Computing*. 2012, vol. 16, no. 7, pp. 1229–1252.
16. CHRISTIAENS, Jan; VANDEN BERGHE, Greet. Slack induction by string removals for vehicle routing problems. *Transportation Science*. 2020, vol. 54, no. 2, pp. 417–433.
17. KIRKPATRICK, S.; GELATT, C. D.; VECCHI, M. P. Optimization by simulated annealing. *Science*. 1983, vol. 220, no. 4598, pp. 671–680.

18. SCHRIMPF, Gerhard; SCHNEIDER, Johannes; STAMM-WILBRANDT, Hermann; DUECK, Gunter. Record breaking optimization results using the ruin and recreate principle. *Journal of Computational Physics*. 2000, vol. 159, no. 2, pp. 139–171.

19. BELLIO, Ruggero; CESCHIA, Sara; DI GASPERO, Luca; SCHAERF, Andrea. Two-stage multi-neighborhood simulated annealing for uncapacitated examination timetabling. *Computers & Operations Research*. 2021, vol. 132, p. 105300.

20. GOH, Say Leng; KENDALL, Graham; SABAR, Nasser R. Simulated annealing with improved reheating and learning for the post enrolment course timetabling problem. *Journal of the Operational Research Society*. 2019, vol. 70, no. 6, pp. 873–888.

21. YU, Vincent F.; WINARNO; MAULIDIN, Achmad; REDI, A. A. N. Perwira; LIN, Shih-Wei; YANG, Chao-Lung. Simulated annealing with restart strategy for the path cover problem with time windows. *Mathematics*. 2021, vol. 9, no. 14, p. 1625.

22. CESNET. *MetaCentrum* [`https://metavo.metacentrum.cz/en/index.html`]. 2025. Accessed: 2026-05-11.

23. HOOS, Holger H.; STÜTZLE, Thomas. *Stochastic Local Search: Foundations and Applications*. Morgan Kaufmann, 2004. The Morgan Kaufmann Series in Artificial Intelligence.

A Archive

Files submitted into the master thesis archive in the Information System of Masaryk University:

- *thesis.pdf*: Text of the thesis in PDF format.
- *source.zip*: Source code of the implemented solver.
- *data.zip*: Anonymized datasets of the timetables.
- *timetables.zip*: Example output timetables of the solver.

B Input format

The solver uses an XML input derived from the XML High School Timetabling format (XHSTT) [2, 7]. However, it does not accept arbitrary valid XHSTT instances. The implemented loader works only with a restricted profile of XHSTT that follows several additional conventions. In our case, the input must contain a usable `Instance` with the sections `Resources`, `ResourceGroups`, `Events`, and `Constraints`; only the first instance is read. From XHSTT, the loader actually uses mainly `Resource`, `ResourceGroup`, `Course`, `AssignResourceConstraint`, `AssignTimeConstraint`, and `AvoidUnavailableTimesConstraint`. By contrast, many components of XHSTT, such as `SolutionGroups`, `Solutions`, `Reports`, event-level `Time`, and event-level `ResourceGroups`, are not used, with many of them ignored because the format is used for data input only. A minimal skeleton of the accepted input has the following form:

```
<HighSchoolTimetableArchive>
  <Instances>
    <Instance Id="FullInstance">
      <ResourceGroups> ... </ResourceGroups>
      <Resources> ... </Resources>
      <Events> ... </Events>
      <Constraints> ... </Constraints>
    </Instance>
  </Instances>
</HighSchoolTimetableArchive>
```

Classes are represented by `ResourceGroup` elements of type `Group` whose identifier begins with `CLASS_`. Division groups are represented by `ResourceGroup` elements of type `Group` whose identifier begins with `DIV_`. Their names have the form `CLASS|divN`, for example `BD1|div0`, `BD1|div1`, and `BD1|div2`. A necessary requirement is that each class has exactly one full-class partition. Concrete student groups are represented by `Resource` elements of type `Group` whose identifiers begin with `G_`. The class and division references are stored inside the nested `ResourceGroups` element.

A typical encoding of one class and its groups can be seen below. In the example, we can see a single class `MOA4`, represented by the resource group `CLASS_48`. The class has two division groups `DIV_132`

and DIV_133. The first division contains the full-class trivial partition group G_247, while the second division contains the two groups G_248 and G_249.

```
<ResourceGroups>
  <ResourceGroup Id="CLASS_48">
    <Name>MOA4</Name>
    <ResourceType Reference="Group" />
  </ResourceGroup>

  <ResourceGroup Id="DIV_132">
    <Name>MOA4|div0</Name>
    <ResourceType Reference="Group" />
  </ResourceGroup>
  <ResourceGroup Id="DIV_133">
    <Name>MOA4|div1</Name>
    <ResourceType Reference="Group" />
  </ResourceGroup>
</ResourceGroups>

<Resource Id="G_247">
  <Name>MOA4:full</Name>
  <ResourceType Reference="Group" />
  <ResourceGroups>
    <ResourceGroup Reference="CLASS_48" />
    <ResourceGroup Reference="DIV_132" />
  </ResourceGroups>
</Resource>

<Resource Id="G_248">
  <Name>MOA4:sk1</Name>
  <ResourceType Reference="Group" />
  <ResourceGroups>
    <ResourceGroup Reference="CLASS_48" />
    <ResourceGroup Reference="DIV_133" />
  </ResourceGroups>
</Resource>

<Resource Id="G_249">
  <Name>MOA4:sk2</Name>
  <ResourceType Reference="Group" />
  <ResourceGroups>
    <ResourceGroup Reference="CLASS_48" />
    <ResourceGroup Reference="DIV_133" />
  </ResourceGroups>
</Resource>
```

Rooms are represented by `Resource` elements of type `Room`. Room types are recognized only through membership in room resource groups whose identifiers begin with `RT_`. The same convention is used for room admissibility of lessons: allowed room types are loaded only from `AssignResourceConstraint` constraints that reference room groups. A typical room definition can be seen below.

```
<ResourceGroup Id="RT_PC">
  <Name>PC classroom type</Name>
  <ResourceType Reference="Room"/>
</ResourceGroup>

<Resource Id="R_PC2">
  <Name>PC2</Name>
  <ResourceType Reference="Room"/>
  <ResourceGroups>
    <ResourceGroup Reference="RT_PC"/>
  </ResourceGroups>
</Resource>
```

Events begin with the letter `E` followed by at least one digit. The numeric prefix is interpreted as the base event identifier. Time references are expected to match the internal timeslot model of the solver, namely a fixed number of 10 days and 10 periods. Lunch events are not given on input, as the solver creates them automatically. All-day events must have the duration of 10.

Lessons are modeled using courses. Every event must reference a `Course`, and this course must exist in `EventGroups`. Each academic event must contain exactly one student-group reference with the identifier prefix `G_`. Teacher assignment is recognized only through event resource references with the prefix `T_`. A valid course, teacher, and event therefore look as follows.

```
<EventGroups>
  <Course Id="C_25">
    <Name>Programming</Name>
  </Course>
</EventGroups>

<Resource Id="T_Turing">
  <Name>Alan Turing</Name>
  <ResourceType Reference="Teacher" />
</Resource>
```

```
<Event Id="E12">
  <Name>Programming</Name>
  <Duration>2</Duration>
  <Course Reference="C_25"/>
  <Resources>
    <Resource Reference="T_Turing">
      <Role>Teacher</Role>
    </Resource>
    <Resource Reference="G_248">
      <Role>Group</Role>
    </Resource>
    <Resource>
      <Role>Room</Role>
        <ResourceType Reference="Room" />
      </Resource>
  </Resources>
</Event>
```

While the given example specifies that the referenced event `E12` requires a room, it does not specify in which room types it is allowed. Allowed room types for an event are encoded by `AssignResourceConstraint` elements applied to a specific event. The admissible room types are listed in the nested `ResourceGroups` element. Each listed `ResourceGroup` references a room-type group with the prefix `RT_` and has the role `Room`. The event is referenced inside `AppliesTo` using an `EventReference`. The bounds `Minimum` and `Maximum` are 1, as each event requiring resource `Room` requires exactly 1 room.

```
<Constraints>
    <AssignResourceConstraint Required="true">
    <AppliesTo>
      <EventReference Reference="E12" />
    </AppliesTo>
    <ResourceGroups>
      <ResourceGroup Reference="RT_PC">
        <Role>Room</Role>
      </ResourceGroup>
    </ResourceGroups>
    <Minimum>1</Minimum>
    <Maximum>1</Maximum>
  </AssignResourceConstraint>
</Constraints>
```

Teacher availability and teacher time preferences are encoded using `AvoidUnavailableTimesConstraint` applied to teacher resources. A constraint with `Required="true"` is interpreted as a hard teacher unavailability (which can not be broken during the search process), whereas a constraint with `Required="false"` is interpreted as a soft non-preferred timeslot ($\mathcal{S}_9$).

```
<Constraints>
  <AvoidUnavailableTimesConstraint Required="true">
    <AppliesTo>
      <Resource Reference="T_Turing"/>
    </AppliesTo>
    <Times>
      <Time Reference="D1_P0"/>
      <Time Reference="D1_P1"/>
      <Time Reference="D6_P0"/>
      <Time Reference="D6_P1"/>
    </Times>
  </AvoidUnavailableTimesConstraint>
</Constraints>
```

Fixed event timeslots are encoded by `AssignTimeConstraint` elements applied to individual events. The event is referenced inside `AppliesTo` using an `EventReference`. The assigned timeslot is listed in the `Times` element as a `Time` reference. An example in which the previously defined event is set to period 2 of day 8 is given below.

```
<Constraints>
  <AssignTimeConstraint Required="true">
    <AppliesTo>
      <EventReference Reference="E12" />
    </AppliesTo>
    <Times>
      <Time Reference="D8_P2" />
    </Times>
  </AssignTimeConstraint>
</Constraints>
```

C Parameters and constraint violation weights

Symbol	Value	Description
β_1	1.5	Weight of teacher unavailability impact
β_2	1.3	Weight of room type scarcity
β_3	0.2	Weight of week order diversification
β_4	100	Priority boost for lunch events
β_5	1,000	Priority boost for all-day events

Table C.1: Weighting parameters used in the event scoring function $score(e)$ of Chapter 5.

Constraint	Name	Weight	Alt. weight
Hard constraints			
$\mathcal{H}_1$	Unassigned event	20,000	same
$\mathcal{H}_2$	Closed holes	10,000	12,000
$\mathcal{H}_3$	Late start of class day	10,000	same
$\mathcal{H}_4$	Long day of class	10,000	same
$\mathcal{H}_5$	Multiple lunches for group day	10,000	12,000
$\mathcal{H}_6$	Long day of teacher	10,000	same
$\mathcal{H}_7$	Missing teacher lunch break	10,000	same
Soft constraints			
$\mathcal{S}_1$	Non-unique lessons	2,000	same
$\mathcal{S}_2$	Odd-even mismatch	5	same
$\mathcal{S}_3$	Events after k_7	120	140
$\mathcal{S}_4$	Events after k_6 on Friday	630	730
$\mathcal{S}_5$	Events at k_0	650	750
$\mathcal{S}_6$	Events at k_9	400	350
$\mathcal{S}_7$	Non-trivial partition at k_3	5	same
$\mathcal{S}_8$	Not all groups of partition at k_1	50	same
$\mathcal{S}_9$	Non-preferred teacher time	10	same

Table C.2: Constraint violation weights of the objective function $f(\Sigma)$.

Symbol	Description
T_0	Initial temperature of simulated annealing.
w_d	Weight of the random events destroy operator.
w_r	Weight of the random ordering repair operator.
R_d	Reheating duration for the temperature jump.

Table C.3: Hyperparameters considered during parameter setting in Chapter 7.

Parameter	First phase	Second phase	Final HC
Random events destroy			
FractionMin	0.003	0.002	0.002
FractionMax	0.02	0.008	0.008
Targeted class destroy			
ClassesMin	1	1	–
ClassesMax	6	6	–
TargetRate	0.4	0	–
TargetedLongDays	0.4	0	–
ChanceToTakeAnotherSlice	0.66	0.7	–
Global parameters			
LNS iterations	500,000	800,000	120,000
Cooling rate	0.9999935	0.99999717	–
Reheat temperature	–	2,000	–
PoorΔ*Limit*	10,000	10,000	0
Candidate score threshold for continued candidate generation (Algorithm 2)			
Late start (soft)	800	800	800
Bonus to candidates for decrease in first period of class day (until $\mathcal{H}_3$ not hit)			
StagnationReheat	–	10,000	–
Maximal iterations with no improvement before reheating			
StagnationReturnToRecord	23,000	23,000	–
Maximal iterations with no improvement before returning to previous record			

Table C.4: Preset parameters of the solver for both phases used in Chapter 7.

Destroy operator	Repair operator	Weight
Random events	Random ordering	0.001
Targeted class	Feasible starts count	0.7

Table C.5: Transitional operator weight multipliers.

D Alternative destroy operator scope

The alternative destroy-operator scope was evaluated using the same methodology as in Chapter 7. The experiments were conducted on the 2025/26 dataset, with 10 independent runs performed for each configuration. Two parameters were considered: *FractionMax*, which controls the scope of the random events destroy operator, and *ClassesMax*, which defines the scope of the targeted class destroy operator. A small grid search over both parameters was performed to assess their joint influence on solution quality. The results are presented in Table D.1 for the first phase and Table D.2 for the second phase.

The original configuration, determined from preliminary experiments under the original constraint violation weights, used $FractionMax = 0.02$ and $ClassesMax = 6$ in the first phase, and $FractionMax = 0.008$ and $ClassesMax = 6$ in the second phase. In both instances, the configurations closest to the original configurations achieved promising results but were outperformed by a number of different parameter settings. For the first phase, the original configuration achieved the second best results, with $FractionMax = 0.025$, $ClassesMax = 8$ having the mean final score decreased by an additional 4%. In the second phase, the original configuration was replaced by $FractionMax = 0.008$, $ClassesMax = 3$, which provided the highest decrease in the soft constraints and a better final score by $\approx 2\%$.

These results are partly attributable to the changed constraint weights, which alter the objective landscape and therefore the effect of each destroy configuration. At the same time, they suggest that the preliminary setting of the destroy-scope parameters was not sufficiently fine-grained. The relationship between destroy scope and solution quality appears to be sensitive to the constraint violation weights configuration and to interactions between search parameters. Further parameter setting of the destroy-operator parameters would therefore be necessary before drawing stronger conclusions about their optimal values under both the original and alternative constraint violation weights.

FractionMax	ClassesMax	Final score	$\sum \mathcal{H}_i$	$\mathcal{H}_2$	$\mathcal{H}_5$	weighted $\sum \mathcal{S}_i$
	4	517,462	17.7	15.1	2.1	305,463
0.015	6	507,536	16.9	14.0	2.3	304,737
	8	487,898	14.5	11.6	2.5	313,899
	4	464,706	13.5	9.9	3.0	302,707
0.02	6	453,091	11.9	8.8	2.8	310,292
	8	460,733	12.7	10.0	2.5	308,334
	4	468,418	12.8	10.3	2.1	314,818
0.025	6	469,123	13.2	10.1	2.2	310,733
	8	435,561	10.7	8.4	1.8	307,161

Table D.1: Impact of destroy operator scopes on the first phase.

FractionMax	ClassesMax	Final score	$\sum \mathcal{H}_i$	$\mathcal{S}_1$	$\sum_{i>1} w_i S_i$
	3	336,031	8.6	16.1	204,631
0.005	5	338,295	8.4	17.4	204,695
	7	334,770	8	18.2	203,770
	3	323,328	7.5	16.1	202,128
0.008	5	340,605	8.8	16.6	204,205
	7	340,616	8.2	17.7	207,016
	3	339,646	8.5	17.7	204,246
0.012	5	336,068	8.3	16.9	204,468
	7	330,080	7.8	17.7	202,680

Table D.2: Impact of destroy operator scopes on the second phase.

E Expanded tabular results

This appendix provides expanded results of the final experiments used for the final solution assessment and alternative parameter setting in Chapter 7.

This chapter contains four tables:

- Table E.1: final assessment of the 2025/26 instance.
- Table E.2: final assessment of the 2024/25 instance.
- Table E.3: final assessment of the 2025/26 instance (alternative parameter setting).
- Table E.4: final assessment of the 2024/25 instance (alternative parameter setting).

Id	Final score	$\mathcal{H}_2$	$\mathcal{H}_3$	$\mathcal{H}_5$	Rest of $\mathcal{H}_i$	$\mathcal{S}_1$	$\mathcal{S}_2$	$\mathcal{S}_3$	$\mathcal{S}_4$	$\mathcal{S}_5$	$\mathcal{S}_6$	$\mathcal{S}_7$	$\mathcal{S}_8$
1	302,855	7	0	3	0	13	1,663	596	52	35	96	272	35
2	307,155	7	0	3	0	17	1,656	596	45	40	90	271	33
3	325,445	9	0	3	0	12	1,668	610	52	37	99	279	42
4	315,655	7	0	3	0	15	1,666	595	59	43	94	261	38
5	319,330	8	0	3	0	16	1,657	605	57	32	92	267	32
6	278,130	4	0	3	0	12	1,658	598	55	46	90	266	44
7	304,295	7	0	3	0	15	1,641	592	55	34	88	270	35
8	276,055	4	0	3	0	14	1,640	591	53	39	92	279	40
9	311,050	7	0	3	0	13	1,655	609	59	38	96	265	42
10	296,560	6	0	3	0	14	1,663	595	52	41	90	267	42
11	262,150	5	0	1	0	9	1,655	612	43	50	99	259	39
12	281,170	7	0	1	0	13	1,641	588	38	52	89	273	34
13	284,175	6	0	1	0	15	1,650	613	44	48	100	269	42
14	282,955	6	0	1	0	15	1,642	615	42	48	100	267	39
15	282,210	6	0	1	0	16	1,647	589	49	51	85	265	39
16	284,780	6	0	2	0	14	1,671	602	42	47	90	265	37
17	281,670	5	0	2	0	14	1,634	604	44	54	93	260	34
18	297,115	7	0	1	0	16	1,644	613	46	49	98	261	40
19	290,875	7	0	1	0	14	1,650	619	34	54	101	265	42
20	280,990	6	0	1	0	14	1,651	609	40	56	92	261	39
21	305,980	7	0	2	0	17	1,640	603	45	51	92	284	34
22	302,935	7	0	2	0	18	1,637	603	42	43	97	286	35
23	296,435	7	0	2	0	13	1,615	609	47	39	102	284	42
24	310,125	7	0	2	0	17	1,653	623	48	43	105	282	30
25	297,755	5	1	2	0	18	1,645	607	42	47	101	286	37
26	296,830	7	0	2	0	18	1,630	588	39	45	88	290	33
27	295,225	6	0	2	0	19	1,643	597	43	48	90	286	33
28	309,920	8	0	2	0	17	1,631	592	43	48	89	287	28
29	307,730	8	0	2	0	16	1,638	597	41	50	87	284	27
30	285,085	6	0	2	0	16	1,625	591	45	46	82	298	30
31	303,990	4	0	5	0	17	1,653	611	49	43	91	277	36
32	294,635	3	0	5	0	17	1,650	605	49	43	95	273	32
33	289,965	3	0	5	0	15	1,664	598	50	46	89	277	30
34	289,400	3	0	5	0	13	1,655	595	52	48	91	283	39
35	274,300	3	0	3	0	13	1,663	609	59	37	106	287	37
36	297,150	3	0	5	0	16	1,657	617	47	45	102	273	36
37	311,450	4	0	5	0	17	1,663	608	52	51	94	273	26
38	303,865	4	0	5	0	15	1,664	608	50	46	96	277	28
39	296,535	3	0	5	0	18	1,650	582	53	49	85	281	36
40	302,395	3	0	5	0	16	1,650	612	53	49	100	273	42
41	325,010	7	0	1	0	27	1,663	603	58	52	93	269	29
42	309,255	6	0	1	0	27	1,656	581	56	58	78	275	34
43	294,700	6	0	1	0	24	1,637	597	44	52	81	271	32
44	316,960	7	0	1	0	25	1,659	597	48	59	88	277	37
45	319,650	8	0	1	0	21	1,645	586	48	59	94	273	31
46	311,295	6	0	1	0	25	1,642	603	49	62	91	271	36
47	317,145	6	0	1	0	26	1,649	607	56	55	100	266	34
48	299,620	5	0	1	0	23	1,639	605	59	53	95	271	37
49	323,955	7	0	1	0	27	1,636	597	45	62	96	267	35
50	318,500	7	0	1	0	25	1,657	611	46	57	94	273	38

Table E.1: Expanded results of solution quality in the final assessment of 2025/26 instance, as used in Table 7.10.

Id	Final score	$\mathcal{H}_2$	$\mathcal{H}_5$	Rest of $\mathcal{H}_i$	$\mathcal{S}_1$	$\mathcal{S}_2$	$\mathcal{S}_3$	$\mathcal{S}_4$	$\mathcal{S}_5$	$\mathcal{S}_6$	$\mathcal{S}_7$	$\mathcal{S}_8$
1	228,650	4	0	0	19	1,654	595	40	15	80	266	54
2	227,310	3	0	0	16	1,662	619	42	14	108	262	53
3	223,625	3	0	0	16	1,660	602	45	20	90	257	49
4	220,420	3	0	0	12	1,664	618	43	19	102	260	48
5	233,255	4	0	0	15	1,675	625	40	19	96	256	53
6	232,445	4	0	0	16	1,675	600	47	19	87	262	40
7	213,955	2	0	0	18	1,685	596	43	19	87	244	51
8	244,575	4	0	0	17	1,679	607	56	19	95	252	49
9	225,425	3	0	0	17	1,662	598	45	25	83	261	45
10	234,280	4	0	0	15	1,679	604	44	19	87	267	42
11	220,495	2	2	0	7	1,662	590	50	30	82	277	44
12	228,195	3	2	0	11	1,666	595	37	24	86	281	35
13	244,360	4	2	0	7	1,660	606	44	25	105	274	40
14	228,670	4	2	0	5	1,658	601	37	23	92	270	37
15	239,940	4	2	0	6	1,670	616	37	29	99	272	51
16	231,065	3	2	0	8	1,645	606	44	22	96	270	47
17	239,505	4	2	0	5	1,663	610	47	22	101	276	46
18	221,485	3	2	0	6	1,655	595	39	25	89	278	40
19	231,530	4	2	0	5	1,666	617	32	21	105	270	40
20	202,690	3	0	0	7	1,664	613	35	13	107	292	41
21	193,145	3	0	0	6	1,653	597	37	13	90	296	40
22	204,125	3	0	0	7	1,648	585	44	23	89	293	39
23	231,255	3	2	0	8	1,652	604	44	21	99	259	49
24	199,680	3	0	0	6	1,671	597	44	18	87	293	40
25	203,735	3	0	0	6	1,665	610	40	19	98	292	40
26	187,155	2	0	0	7	1,679	595	37	14	94	290	38
27	207,565	4	0	0	7	1,643	612	37	7	101	300	43
28	196,915	4	0	0	5	1,670	582	34	18	81	291	35
29	188,605	2	0	0	6	1,660	588	41	21	87	293	40
30	203,715	3	0	0	7	1,651	580	47	21	88	290	39
31	195,570	3	0	0	7	1,640	602	36	10	95	299	49
32	195,105	3	0	0	7	1,659	596	41	10	88	302	45
33	207,125	3	0	0	10	1,663	600	43	16	87	294	61
34	185,940	2	0	0	7	1,628	585	41	16	84	294	46
35	183,250	1	0	0	10	1,654	582	49	15	77	294	45
36	193,570	3	0	0	6	1,644	602	36	10	95	296	49
37	211,120	3	0	0	7	1,652	626	44	10	113	294	57
38	211,870	3	0	0	12	1,652	595	41	16	96	296	42
39	203,825	3	0	0	9	1,653	602	44	9	94	300	53
40	199,785	3	0	0	7	1,651	604	42	9	97	298	49
41	218,295	3	0	0	17	1,652	605	44	9	90	243	53
42	233,300	5	0	0	12	1,669	597	43	22	86	245	46
43	249,140	5	0	0	17	1,666	609	47	19	95	244	51
44	213,030	3	0	0	12	1,663	606	41	18	93	243	41
45	224,520	4	0	0	11	1,679	615	43	15	100	247	45
46	232,425	5	0	0	11	1,666	606	43	16	96	247	45
47	237,750	5	0	0	12	1,683	617	45	19	93	249	43
48	216,810	3	0	0	12	1,663	606	46	20	90	249	51
49	238,735	5	0	0	12	1,668	616	41	20	99	239	57
50	215,945	4	0	0	10	1,663	606	44	17	96	250	45

Table E.2: Expanded results of solution quality in the final assessment of 2024/25 instance, as used in Table 7.10.

Id	Final score	$\mathcal{H}_2$	$\mathcal{H}_3$	$\mathcal{H}_5$	Rest of $\mathcal{H}_i$	$\mathcal{S}_1$	$\mathcal{S}_2$	$\mathcal{S}_3$	$\mathcal{S}_4$	$\mathcal{S}_5$	$\mathcal{S}_6$	$\mathcal{S}_7$	$\mathcal{S}_8$
1	255,205	0	0	2	0	22	1,684	653	23	35	116	285	46
2	277,880	1	0	2	0	26	1,663	662	25	30	128	287	38
3	262,010	1	0	2	0	18	1,685	666	20	34	127	279	48
4	264,585	1	0	2	0	20	1,670	667	18	30	136	283	44
5	259,940	1	0	2	0	19	1,677	652	20	35	121	285	33
6	269,405	1	0	2	0	22	1,666	661	21	33	128	291	44
7	269,555	1	0	2	0	20	1,690	657	25	32	135	285	44
8	273,165	1	0	2	0	20	1,678	666	27	35	130	285	53
9	259,595	1	0	2	0	16	1,674	653	21	39	125	275	42
10	265,355	1	0	2	0	21	1,668	653	20	35	124	289	38
11	308,920	3	0	0	0	31	1,652	653	42	45	124	246	44
12	284,610	2	0	0	0	28	1,673	651	38	40	126	253	40
13	305,010	3	0	0	0	36	1,666	635	32	45	113	254	37
14	278,620	2	0	0	0	27	1,661	640	41	41	111	247	39
15	264,400	2	0	0	0	25	1,653	648	22	41	118	251	41
16	295,235	3	0	0	0	26	1,663	661	36	41	132	240	39
17	272,755	2	0	0	0	28	1,658	646	31	39	111	249	41
18	289,255	2	0	0	0	31	1,670	644	35	44	122	259	44
19	280,675	3	0	0	0	22	1,671	643	33	47	113	242	44
20	280,170	2	0	0	0	26	1,641	659	32	44	126	249	40
21	314,555	4	0	4	0	8	1,648	657	27	48	122	275	51
22	318,455	4	0	4	0	11	1,645	663	23	45	130	274	40
23	326,305	5	0	4	0	12	1,661	652	22	46	117	272	37
24	342,035	5	0	4	0	17	1,649	661	25	45	125	270	43
25	325,365	5	0	4	0	12	1,649	651	28	36	123	268	43
26	339,695	5	0	4	0	19	1,651	650	20	44	124	278	41
27	340,065	5	0	4	0	17	1,653	648	27	45	121	274	38
28	318,895	4	0	4	0	16	1,655	642	26	44	106	272	46
29	320,960	4	0	4	0	13	1,669	652	26	41	132	271	41
30	334,720	5	0	4	0	16	1,662	648	24	42	123	274	45
31	282,710	2	1	1	0	15	1,639	663	33	44	129	271	42
32	317,130	5	1	1	0	21	1,645	632	29	44	111	261	42
33	295,700	4	1	1	0	15	1,636	645	27	44	118	262	38
34	311,755	5	1	1	0	16	1,650	645	27	47	117	269	39
35	294,170	3	1	1	0	21	1,635	656	23	39	129	273	32
36	305,220	5	1	1	0	9	1,641	658	29	46	131	265	41
37	307,155	5	1	1	0	15	1,640	657	28	38	122	267	40
38	302,090	4	1	1	0	17	1,635	647	30	43	119	267	44
39	287,095	4	1	1	0	16	1,637	623	22	44	107	266	37
40	296,595	3	1	1	0	17	1,646	659	35	44	121	261	38
41	314,490	5	0	2	0	19	1,675	653	27	36	121	267	46
42	310,620	5	0	2	0	20	1,679	640	22	38	116	283	41
43	316,680	5	0	2	0	21	1,667	645	28	39	112	271	36
44	286,715	4	0	2	0	12	1,656	658	21	36	127	277	43
45	296,780	4	0	2	0	16	1,667	652	27	36	122	281	47
46	311,345	5	0	2	0	18	1,664	669	24	32	125	279	54
47	301,785	4	0	2	0	21	1,658	662	21	32	127	277	33
48	323,420	5	0	2	0	21	1,681	675	27	32	135	271	44
49	305,255	3	0	2	0	23	1,677	667	31	34	131	272	43
50	320,105	5	0	2	0	22	1,670	659	22	38	125	277	36

Table E.3: Expanded results of solution quality in the final assessment of 2025/26 instance with alternative parameter setting, as used in Table 7.13.

Id	Final score	$\mathcal{H}_2$	$\mathcal{H}_5$	Rest of $\mathcal{H}_i$	$\mathcal{S}_1$	$\mathcal{S}_2$	$\mathcal{S}_3$	$\mathcal{S}_4$	$\mathcal{S}_5$	$\mathcal{S}_6$	$\mathcal{S}_7$	$\mathcal{S}_8$
1	227,830	2	0	0	16	1,672	648	20	15	123	260	51
2	233,380	2	0	0	20	1,676	652	14	15	127	260	50
3	223,790	2	0	0	17	1,678	642	16	13	122	258	42
4	235,565	3	0	0	16	1,657	644	20	13	117	264	50
5	229,265	3	0	0	14	1,670	656	19	7	121	261	46
6	239,225	4	0	0	11	1,669	655	18	12	124	267	47
7	236,110	2	0	0	16	1,674	676	17	14	144	258	50
8	245,350	4	0	0	14	1,668	658	19	12	120	254	55
9	223,440	2	0	0	15	1,668	660	20	8	126	250	55
10	248,290	4	0	0	15	1,692	660	18	12	123	268	58
11	227,065	2	2	0	5	1,644	660	13	16	123	271	51
12	235,235	2	2	0	9	1,678	641	19	17	116	277	50
13	261,335	4	2	0	7	1,671	652	21	16	128	274	44
14	239,770	2	2	0	9	1,671	650	19	17	125	269	54
15	267,000	4	2	0	9	1,664	668	16	19	130	276	47
16	254,005	3	2	0	9	1,664	658	19	17	129	269	49
17	250,830	3	2	0	5	1,671	682	16	17	140	273	44
18	258,030	4	2	0	7	1,660	662	15	16	127	270	46
19	251,935	3	2	0	10	1,668	645	19	21	113	275	55
20	246,280	3	0	0	11	1,658	649	16	10	121	270	45
21	231,680	4	0	0	7	1,664	658	23	5	129	290	42
22	209,560	3	0	0	6	1,644	651	15	8	119	290	43
23	207,235	3	0	0	6	1,662	649	13	4	125	295	47
24	225,390	4	0	0	5	1,668	659	15	9	130	288	43
25	226,480	4	0	0	6	1,667	652	22	7	119	291	49
26	227,360	4	0	0	9	1,639	651	15	7	120	295	47
27	224,890	4	0	0	8	1,641	643	16	9	117	297	36
28	229,790	4	0	0	7	1,670	654	18	12	122	298	31
29	225,495	4	0	0	8	1,669	646	13	11	118	294	44
30	250,195	5	0	0	14	1,672	660	21	13	127	251	53
31	206,415	3	0	0	6	1,643	647	14	6	117	300	49
32	214,475	3	0	0	8	1,659	645	18	8	116	298	53
33	219,330	3	0	0	9	1,638	642	19	10	121	298	41
34	217,125	3	0	0	9	1,635	653	15	5	129	296	44
35	215,810	3	0	0	8	1,648	650	16	9	121	288	47
36	214,620	3	0	0	8	1,644	632	13	17	114	296	46
37	190,330	1	0	0	7	1,648	664	15	0	138	296	48
38	212,160	1	0	0	13	1,644	652	26	8	131	296	47
39	212,825	3	0	0	7	1,661	642	15	8	125	298	49
40	215,765	3	0	0	8	1,642	646	14	7	120	289	44
41	256,805	5	0	0	13	1,674	660	18	11	129	249	45
42	255,175	5	0	0	14	1,674	653	20	8	124	247	43
43	256,095	5	0	0	15	1,662	645	20	11	118	247	42
44	235,935	3	0	0	13	1,678	672	15	14	133	243	45
45	217,725	3	0	0	9	1,672	645	13	14	119	245	44
46	248,810	5	0	0	10	1,665	657	17	11	126	269	48
47	260,070	5	0	0	13	1,667	661	22	12	127	247	49
48	243,015	5	0	0	9	1,666	648	16	12	119	247	48
49	225,305	2	0	0	15	1,664	674	16	13	126	249	37
50	249,845	5	0	0	12	1,660	652	16	14	116	247	45

Table E.4: Expanded results of solution quality in the final assessment of 2024/25 instance with alternative parameter setting, as used in Table 7.13.

F Examples of timetable outputs

This appendix contains selected timetable visualizations generated by the proposed solver for a real-world 2025/26 instance obtained from the Czech High School of ICT, Postal Services, and Finance in Brno (SŠIPF). The timetables are shown in a compact graphical form to illustrate the structure of generated solutions, including class partitions, parallel group teaching, and lunch breaks. Each figure represents the timetable of a single class for one week of the biweekly timetable.

The timetable in Figure F.1 displays a typical timetable after soft constraint minimization. The daily structure starts at period 1 and often employs open holes at the end of days, with a shorter Friday afternoon. The only downside is the non-uniqueness of lesson ($\mathcal{S}_1$) for ENG at the periods 5, 6 on Monday, caused by a high number of ENG lessons for this group. This violation is not detrimental to the solution quality.

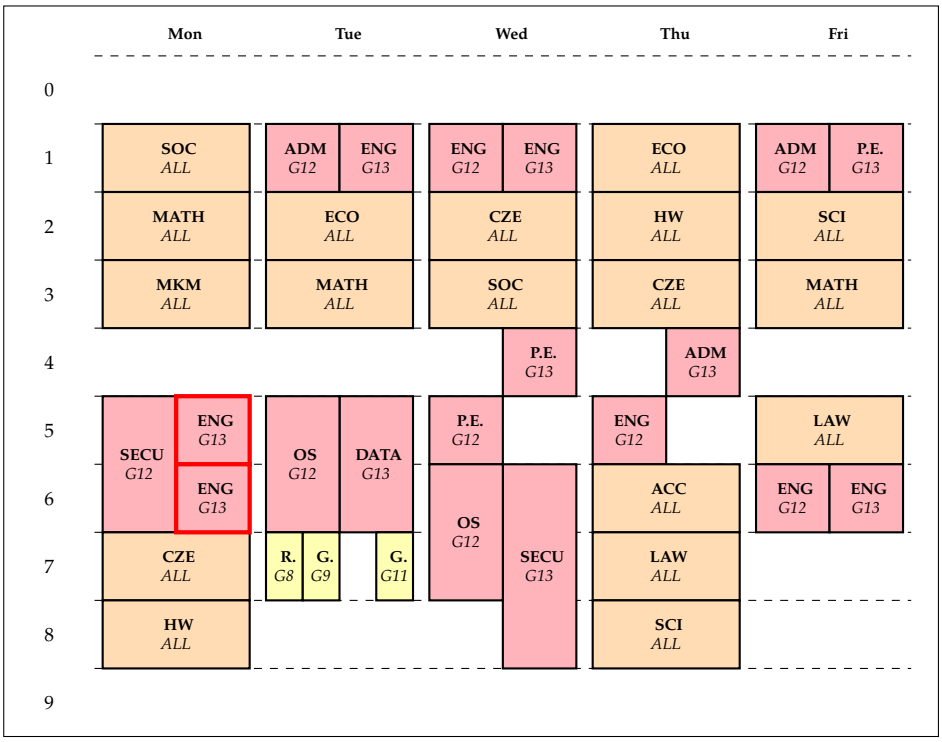

Figure F.1: Example one week timetable for a BD2 class.

While the vast majority of timetables provide reasonable results, the timetable in Figure F.2 showcases the limitations of the solver. The displayed timetable contains a closed hole violation ($\mathcal{H}_2$) at period 5 on Thursday, as well as three non-unique lesson violations ($\mathcal{S}_1$) in the same timeslot. These violations are caused by the small number of events sharing the relevant non-trivial partitions, which gives the solver little flexibility when assigning them. The ends of other days contain open holes, indicating that the solver has trouble assigning the events with non-trivial partitions of different days to other places within the timetable structure. Since this timetable contains 1 of the 2 hard constraints in the solution and 3 of the 17 non-unique lesson violations ($\mathcal{S}_1$), it would require manual correction before practical use.

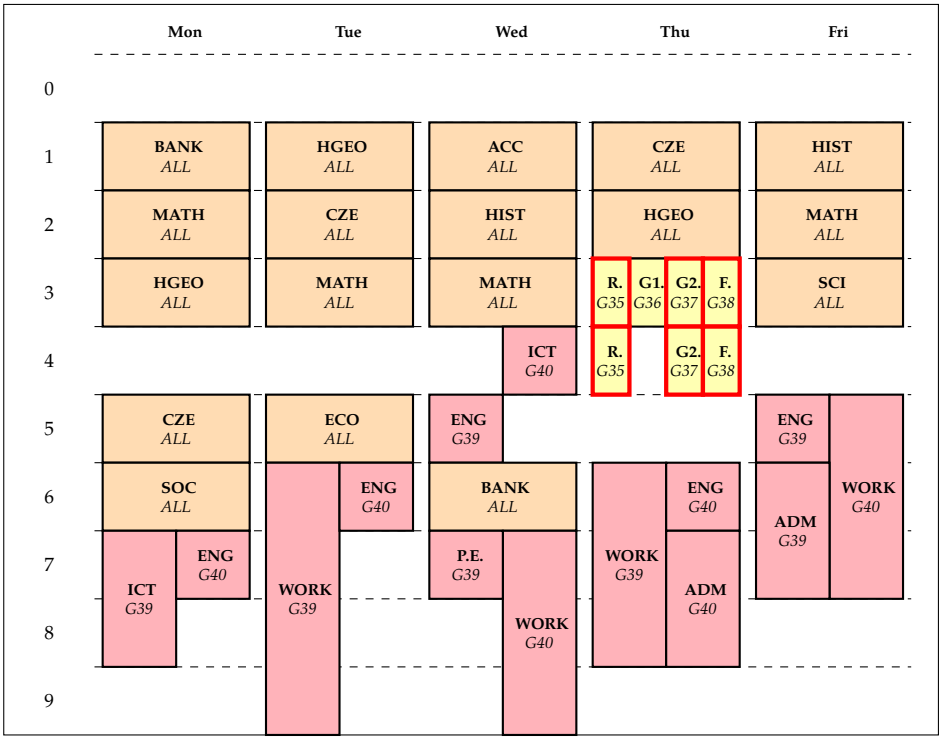

Figure F.2: Example one week timetable for a BP2A class.

The timetable in Figure F.3 shows a class timetable with multiple longer events with fixed placements. It also contains not all groups at k_0 constraint violation ($\mathcal{S}_8$) on Tuesday and a high number of event at k_9 violations ($\mathcal{S}_6$), mostly caused by the number of events for this class and their duration as a whole.

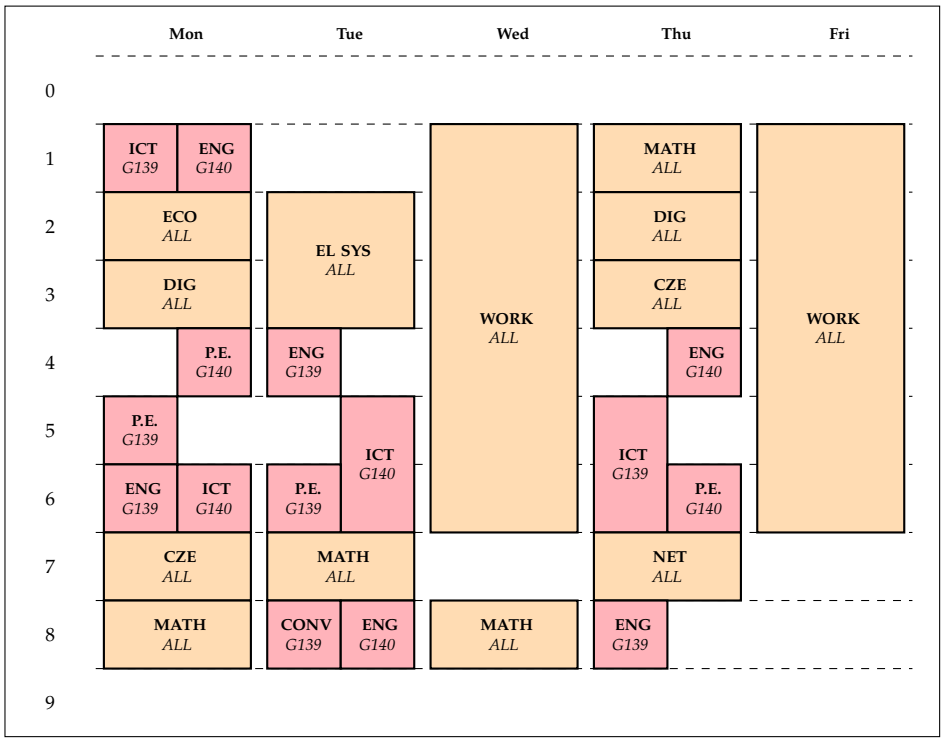

Figure F.3: Example one week timetable for a `IZT3` class.