Fast University Timetabling via Room-wise CP-SAT Decomposition and Incremental Repair

Éber Júnio Borges Moreira[1] **, Sérgio Antônio Andrade de Freitas**[1]

[1]Department of Computer Science (CIC)
University of Brasília (UnB)
Campus Universitário Darcy Ribeiro, Brasília – DF, CEP 70910-900, Brazil

{ebermoreira,sergiofreitas}@unb.br

Abstract. *We propose a room-wise CP-SAT decomposition with incremental repair to speed up the course timetabling problem. We solve per-room subproblems and enforce cross-room instructor non-overlap via incremental global blocking. When infeasibility arises, we apply selective rollback and a controlled fallback that preserves completeness but can introduce instructor conflicts; all runs are checked by a global validator reporting coverage, violations, and per-lecture preference satisfaction. On 11 real instances from a large Brazilian public university (DS1–DS11, up to 6,278 lectures), we achieved 100% coverage in all runs and solved the largest case in seconds (4.45 s on DS11), with high preference satisfaction when fallback was not triggered (89.6% on DS11).*

1. Introduction

Academic timetabling in public universities is a recurring problem with a direct impact on administrative efficiency, infrastructure utilization, and service quality for academic communities. In practice, assigning lectures to timeslots and rooms must satisfy hard constraints while also aiming to satisfy time preferences and institutional criteria. This setting is commonly referred to as Course Timetabling (CTT). For large instances, obtaining feasible, high-quality timetables can require substantial computational effort, which hinders the operational use of exact approaches under short decision windows.

In our prior work, we proposed a monolithic CP-SAT formulation for academic timetabling and evaluated it on eleven real instances (DS1–DS11) for a large Brazilian public university [Moreira and De Freitas 2024], reporting first-solution and optimal-solution runtimes that grow sharply with instance size. However, for larger instances, searching for optimal solutions incurs a high computational cost, even with extensive `presolve` and `solver` parameters tuning.

In this paper, we propose an alternative approach, *Room-wise Decomposition with Incremental Repair* (RDIR), whose goal is to reduce runtime while preserving global consistency whenever possible and providing transparent reporting of global feasibility when fallback relaxations are required. The method decomposes the instance into per-room subproblems solved with CP-SAT, while preserving global constraints, particularly instructor non-overlap, through an incremental blocking mechanism over already-assigned timeslots. To handle situations where greedy incremental decisions may render a later subproblem infeasible, we introduce a repair procedure based on selective *rollback* and, as a last resort, a controlled *fallback*, always followed by global validation. In addition,

we employ a post-solution auditing stage that automatically checks hard constraint violations and measures preference satisfaction at the global level, enabling standardized comparisons across instances and runs.

Although our primary focus is optimizing academic–administrative processes, the proposed approach also improves operational efficiency by reducing rework during timetable generation and enabling more rational use of computational and administrative resources. The underlying strategies are general and may transfer to other allocation and planning problems.

Our contributions are as follows. This paper introduces a practical RDIR pipeline for CP-SAT timetabling with runtimes suitable for operational use: (1) a capacity- and slot-budget-aware room grouping heuristic; (2) an incremental global blocking mechanism that enforces instructor non-overlap across independently solved room subproblems; (3) a bounded repair loop based on selective rollback and a controlled fallback that preserves coverage when blocking leads to infeasibility; and (4) a global post-solution validator that reports coverage, hard-constraint violations, and per-lecture preference satisfaction to enable consistent, run-level comparisons.

The remainder of this paper is organized as follows. Section 2 discusses related work; Section 3 formalizes the problem and constraints; Section 4 describes the proposed approach; Section 5 presents the experimental methodology; Section 6 reports and discusses the results; Section 7 discusses threats to validity; and Section 8 concludes the paper and outlines future work.

2. Related Work

Course timetabling has been extensively studied due to its practical relevance and inherent combinatorial complexity. Surveys and state-of-the-art overviews describe a wide range of formulations and solution strategies, including exact methods (integer programming and constraint programming), metaheuristics, and hybrid approaches that combine modeling and search heuristics [Chen et al. 2021, Ceschia et al. 2023]. Standard benchmark formulations and datasets have played a key role in enabling reproducible comparisons across methods [Ceschia et al. 2023].

Constraint Programming (CP) is a prominent paradigm for timetabling, as it naturally expresses hard constraints and supports rich objective functions. CP-SAT, as implemented in Google OR-Tools, combines CP modeling with SAT-style inference and integer reasoning, and is widely adopted as a practical solver for combinatorial optimization [Google OR-Tools 2026]. In educational timetabling, CP-based and hybrid methods have shown competitive results, especially when complemented by domain reduction, presolve, and tailored search strategies [Ceschia et al. 2023].

A central challenge in scaling exact or CP-based approaches is the rapid growth of the search space with instance size. Decomposition mitigates this by splitting large instances into smaller subproblems solved independently or in a coordinated manner. In timetabling and related scheduling, decomposition is often paired with VLSN and repair mechanisms—fixing parts of a solution while re-optimizing a subset to restore feasibility or improve quality [Ahuja et al. 2002, Meyers and Orlin 2007]. Multi-neighborhood local search is another established line, exploring multiple neighborhood structures to improve solution quality on timetabling benchmarks [Di Gaspero and Schaerf 2003].

Regarding benchmark formulations, Curriculum-Based Course Timetabling (CB-CTT) has become a widely referenced model, supported by public instances, validators, and reference scores [Di Gaspero et al. 2007, Bonutti et al. 2012]. This benchmark ecosystem motivates approaches that balance feasibility, quality, and runtime scalability.

Our prior work proposed a monolithic CP-SAT formulation for academic timetabling and showed that both first-solution and optimal runtimes increase sharply with instance size on eleven real instances [Moreira and De Freitas 2024]. In this paper, we keep the same instances and constraints but replace monolithic optimization with an RDIR decomposition-and-repair pipeline aimed at reducing runtime.

3. Problem Definition

We address academic course timetabling, in which lectures must be assigned to discrete weekly time slots and rooms while satisfying feasibility constraints and improving preference satisfaction. Time is discretized into a finite set of slots. A solution is a set of assignments (e, k, r, t), where e denotes an event (course section), $k \in \{1, \ldots, L_e\}$ identifies the k-th lecture of event e, r is the assigned room, and t is the assigned time slot. Here, an event represents a course section that may require multiple weekly lecture occurrences, and k indexes these occurrences.

3.1. Sets and parameters

Let:

- $\mathcal{E}$ be the set of events (course sections) to schedule;
- $\mathcal{R}$ be the set of available rooms;
- $\mathcal{T}$ be the set of discrete time slots in the planning horizon.

For each event $e \in \mathcal{E}$:

- $L_e \in \mathbb{N}$ is the number of lectures to schedule for e;
- $\mathrm{prof}(e)$ is the set of instructors associated with e;
- $\mathrm{cap}(e)$ is the required capacity (number of enrolled students);
- $P_e \subseteq \mathcal{T}$ is the **set** of preferred time slots;
- $U_e \subseteq \mathcal{T}$ is the set of unavailable time slots.

For each room $r \in \mathcal{R}$:

- $\mathrm{cap}(r)$ is the room capacity.

3.2. Hard constraints

The following constraints define feasibility:

(H1) Room non-overlap. A room cannot host two lectures at the same time:

$$\forall r \in \mathcal{R}, \forall t \in \mathcal{T}: \quad |\{(e, k) : (e, k, r, t) \text{ is assigned}\}| \leq 1.$$

(H2) Room capacity. An event can only be assigned to rooms with sufficient capacity:

$$\forall (e, k, r, t) \text{ assigned}: \quad \mathrm{cap}(e) \leq \mathrm{cap}(r).$$

(H3) Instructor non-overlap. An instructor cannot teach more than one lecture at the same time:

$$\forall d,\ \forall t \in \mathcal{T}: \quad |\{(e,k,r) : d \in \text{prof}(e)\ \wedge\ (e,k,r,t) \text{ assigned}\}| \leq 1.$$

(H4) Unavailability. Lectures cannot be assigned to unavailable time slots:

$$\forall (e,k,r,t) \text{ assigned}: \quad t \notin U_e.$$

(H5) Completeness. Each event must have exactly L_e lectures assigned:

$$\forall e \in \mathcal{E}: \quad |\{(k,r,t) : (e,k,r,t) \text{ assigned}\}| = L_e.$$

3.3. Soft criteria and evaluation metrics

Beyond feasibility, we aim to satisfy time preferences. To ensure fair comparison across approaches, we report preference satisfaction *per lecture* as a common metric:

$$\text{Pref} = \frac{\#\{(e,k,r,t) \text{ assigned} : t \in P_e\}}{\#\{(e,k,r,t) \text{ assigned}\}}.$$

We also report the absolute number of lectures assigned outside preferred slots.

3.4. Note on global feasibility under controlled fallback

Our proposed method enforces instructor non-overlap (H3) through incremental global blocking across subproblems. However, when a subproblem remains infeasible after a bounded number of repair attempts, a *controlled fallback* may be invoked that relaxes global blocking for that subproblem to preserve completeness (H5). In such cases, H3 may be violated. To maintain transparency, all runs are followed by a global validator that reports any hard constraint violations, and the Results section explicitly quantifies the trade-off observed when fallback is triggered.

4. Proposed Approach

This section presents RDIR, which reduces runtime on large timetabling instances while preserving transparent global validation. The method (i) decomposes the instance into per-room CP-SAT subproblems, (ii) enforces cross-room instructor non-overlap via incremental global blocking, and (iii) handles local infeasibility through selective rollback and controlled fallback, always followed by global validation.

4.1. Overview

Given an instance $(\mathcal{E}, \mathcal{R}, \mathcal{T})$, the approach proceeds in four stages:

1. **Preprocessing:** parse events, instructors, room capacities, and preferred/unavailable time slots.
2. **Room-wise decomposition:** assign each event e to a room r, yielding subsets $\mathcal{E}_r \subseteq \mathcal{E}$.
3. **Incremental solving with global blocking:** solve each room subproblem with CP-SAT, updating a global occupancy map for instructors.
4. **Incremental repair and validation:** if a room subproblem becomes infeasible, trigger selective rollback and retry; if infeasibility persists, use controlled fallback and record the outcome; finally, run a global validator to measure feasibility and quality metrics.

4.2. RDIR heuristic

We decompose the global instance by assigning events to rooms before invoking CP-SAT. In our setting, each event e is assigned to a single room r for all of its lectures, reflecting the institutional practice adopted in the dataset and enabling a room-wise decomposition. The grouping heuristic aims to preserve feasibility and reduce conflicts by considering: (i) **capacity compatibility** (only rooms with $\mathrm{cap}(r) \geq \mathrm{cap}(e)$ are eligible) and (ii) **slot budget** (the sum of lectures assigned to a room cannot exceed $|\mathcal{T}|$). Optionally, to reduce local contention, the heuristic can try to avoid placing events with overlapping preferred time slots in the same room group. The output is a mapping $e \mapsto r$ and the induced subsets $\mathcal{E}_r$.

4.3. CP-SAT subproblem formulation

For each room r, we solve a CP-SAT subproblem that assigns the lectures of all events in $\mathcal{E}_r$ to time slots in $\mathcal{T}$.

Decision variables. For each event $e \in \mathcal{E}_r$ and each lecture index $k \in \{1, \ldots, L_e\}$, define an integer start variable:

$$s_{e,k} \in \mathcal{T}.$$

In CP-SAT, we model each lecture as a unit-length interval $(s_{e,k}, 1)$ and apply `NoOverlap` to avoid room collisions.

Local hard constraints. The subproblem enforces:

- **Room non-overlap (H1):** via `NoOverlap` on all unit intervals in room r.
- **Capacity (H2):** enforced by pre-filtering; events with $\mathrm{cap}(e) > \mathrm{cap}(r)$ are excluded (or the model is made infeasible).
- **Unavailability (H4):** for applicable events, prohibit $s_{e,k} \in U_e$.

Soft objective within subproblems. Subproblems include a preference-related objective component, encouraging assignments in preferred time slots. Because different internal objectives can lead to different trade-offs under decomposition, our evaluation emphasizes global, method-agnostic metrics.

4.4. Incremental global blocking for instructor consistency

To enforce instructor non-overlap across different rooms (H3) without solving a monolithic model, we maintain a global occupancy map:

$$B(d) \subseteq \mathcal{T},$$

where $B(d)$ stores all time slots already assigned to instructor d in previously solved rooms. When building the subproblem for room r, for each event $e \in \mathcal{E}_r$, lecture k, and instructor $d \in \mathrm{prof}(e)$, we add:

$$s_{e,k} \notin B(d).$$

After solving the subproblem for room r, we update $B(d)$ with the newly assigned time slots. This incremental mechanism enforces H3 across rooms under the assumption that all rooms are solved under global blocking.

4.5. Incremental repair: rollback and controlled fallback

RDIR with incremental blocking can fail when earlier decisions exhaust feasible time slots for instructors needed later, causing a room subproblem to become infeasible. We handle this with a bounded repair procedure:

Selective rollback. When room r is infeasible, we identify a small set of previously solved rooms that share instructors with $\mathcal{E}_r$, remove their assignments from the global state, rebuild the instructor occupancy map $B(d)$ from the remaining rooms, and retry solving r. This mitigates greedy commitment and often restores feasibility while keeping most assignments intact.

Controlled fallback. If infeasibility persists after a fixed number of attempts, we solve the room subproblem without applying global blocking constraints to preserve completeness (H5). As discussed in Section 3.4, this may violate instructor non-overlap (H3). Therefore, fallback is treated as a last resort and is explicitly counted and reported.

4.6. Global validation and reporting

After producing the final timetable, we run a global validator that checks: (i) room conflicts (H1), (ii) instructor conflicts (H3), (iii) capacity violations (H2), and (iv) unavailability violations (H4). We also compute coverage (H5) and per-lecture preference satisfaction:

$$\text{Pref} = \frac{\#\{(e,k,r,t) \text{ assigned} : t \in P_e\}}{\#\{(e,k,r,t) \text{ assigned}\}}.$$

This validation step provides transparent evidence of feasibility and quantifies trade-offs whenever repair or fallback is triggered.

4.7. Algorithm

In all experiments, rooms are processed in non-increasing order of room capacity (largest rooms first), considering only rooms whose assigned groups are non-empty. We set $A_{\max} = 4$ attempts per room before triggering fallback. During selective rollback, when a room subproblem is infeasible, we identify previously solved rooms that share at least one instructor with the current room group E_r and rollback up to $K = 3$ of them, prioritizing the most recently processed rooms. We choose $A_{\max} = 4$ and $K = 3$ as small constants to bound the worst-case repair effort per room and keep runtime predictable: increasing these limits can reduce fallback at the cost of more rollback rework and longer runtimes, whereas smaller limits may trigger fallback more often on highly coupled instances. After rollback, we rebuild the global instructor occupancy map $B(d)$ from the remaining room solutions and retry the current room; if infeasibility persists after $A_{\max}$ attempts, we solve the subproblem without global instructor blocking, as shown in Algorithm 1, which presents the pseudo-code of RDIR.

5. Experimental Methodology

The protocol was designed to enable direct comparison with our prior monolithic CP-SAT baseline on the same instance set. The instances vary in size as summarized in Table 1.

Algorithm 1 Room-wise Decomposition with Incremental Repair (RDIR)

Require: Events $\mathcal{E}$, rooms $\mathcal{R}$, time slots $\mathcal{T}$
Build room groups $\{\mathcal{E}_r\}_{r\in\mathcal{R}}$ using a capacity-aware heuristic
$B(d) \leftarrow \emptyset$ for all instructors d
$\mathcal{S} \leftarrow \emptyset$ ▷ set of room solutions
for each room r in a chosen order **do**
    attempt $\leftarrow 0$
    while attempt $< A_{\max}$ **do**
        Build CP-SAT subproblem for $(\mathcal{E}_r, \mathcal{T})$ with H1,H2,H4 and instructor blocking (H3) via $B(d)$
        Solve the subproblem
        if feasible **then**
            Update $\mathcal{S}$ and $B(d)$
            break
        end if
        Perform selective rollback on a small set of prior rooms sharing instructors with $\mathcal{E}_r$
        Rebuild $B(d)$ from $\mathcal{S}$
        attempt $\leftarrow$ attempt $+1$
    end while
    if subproblem still infeasible **then**
        Solve $(\mathcal{E}_r, \mathcal{T})$ **without** instructor blocking (controlled fallback)
        Add solution to $\mathcal{S}$ (flag fallback for reporting)
    end if
end for
Run global validation on the combined timetable from $\mathcal{S}$
Report runtime, coverage, violations, Pref, and repair/fallback counters

All instances use the same weekly discretization with $|\mathcal{T}| = 90$ time slots. We adopt $|\mathcal{T}| = 90$ because the institution's weekly timetable is discretized into 6 days (Monday–Saturday) with 15 schedulable periods per day. These 15 periods correspond to 5 morning, 6 afternoon, and 4 evening slots, yielding $6 \times 15 = 90$ time slots in total.

Table 1. Summary of dataset characteristics

DS	Events	Rooms	Required lectures
1	11	06	52
2	28	20	118
3	44	12	186
4	67	15	312
5	103	38	422
6	101	20	694
7	163	66	702
8	208	56	947
9	264	80	1396
10	605	145	2939
11	1473	188	6278

In addition to instance size, measured by the number of required lectures, we

report instructor-set statistics to characterize cross-room coupling. Let $|D|$ be the number of distinct instructors in an instance and let $N_L = \sum_{e \in \mathcal{E}} L_e$ denote the total number of required lectures, where $\mathcal{E}$ is the set of events and L_e is the number of lectures required by event e. (In Table 1, *Required lectures* corresponds to N_L.) We define a simple coupling proxy:

$$CR = \frac{N_L}{|D|}$$

Higher CR indicates stronger instructor coupling, which is consistent with the cases where rollback/fallback is more likely to be triggered.

5.1. Execution protocol

For each instance DSi ($i \in \{1, \ldots, 11\}$), we perform three repeated runs to account for wall-clock measurement noise. Each run executes the full pipeline: (i) preprocessing and room-wise grouping, (ii) incremental subproblem solving with global instructor blocking, (iii) repair procedures, and (iv) global validation and metric extraction.

5.2. Solver configuration

Each room subproblem is solved using CP-SAT under a per-subproblem time limit of 2 seconds and 8 search workers for parallel search. These settings bound worst-case local effort and keep the end-to-end pipeline operational across DS1–DS11; the repair mechanism is enabled with a bounded number of attempts per room before triggering fallback.

All experiments were executed on *AMD Ryzen 3 5400U with Radeon Graphics* with *8 GB RAM* running *Windows 10 Pro*. We used Python *3.12.9* and Google OR-Tools CP-SAT *9.9.3963*. Wall-clock time was measured as the end-to-end runtime of the full pipeline.

5.3. Metrics

We report metrics in three categories, computed from the global validator and run logs. A phase-level runtime breakdown (preprocessing/grouping, solving, repair, and validation) is not included in this version.

Runtime. We report (i) **total runtime** τ_{total}, (ii) **solver runtime** $\tau_{\text{solver}} = \sum_{j=1}^{N_{\text{sub}}} \tau_j$, and (iii) **per-subproblem runtime statistics** over $\{\tau_j\}$: min/median/mean/max.

Completeness and feasibility. Coverage (H5) is the fraction of scheduled lectures over the expected total:

$$\text{Coverage} = \frac{|\mathcal{A}|}{\sum_{e \in \mathcal{E}} L_e},$$

where $\mathcal{A}$ is the set of produced lecture assignments. For simplicity in the metric definitions, we write assignments as triples (e, r, t) and omit the lecture index k; equivalently, $\mathcal{A}$ denotes the set of produced lecture assignments (e, k, r, t) projected onto (e, r, t).

To define hard-constraint violation counters, let $c_{d,t} = |\{(e, r, t) \in \mathcal{A} : d \in \text{prof}(e)\}|$ be the number of lectures assigned to instructor d at

time slot t, and let $n_{r,t} = |\{(e, r, t) \in \mathcal{A}\}|$ be the number of lectures assigned to room r at time slot t. Hard-constraint violations are counted by the global validator:

$$\mathrm{InstrConf} = |\{(d, t) : c_{d,t} > 1\}|, \quad \mathrm{RoomConf} = |\{(r, t) : n_{r,t} > 1\}|.$$

$$\mathrm{CapViol} = |\{(e, r, t) \in \mathcal{A} : \mathrm{cap}(e) > \mathrm{cap}(r)\}|.$$

$$\mathrm{UnavailViol} = |\{(d, e, t) : (e, r, t) \in \mathcal{A},\ d \in \mathrm{prof}(e),\ t \in U_e\}|,$$

where U_e denotes the set of unavailable time slots associated with event e (e.g., induced by instructor unavailability and/or event-specific constraints used by the validator).

We also report robustness counters: $N_{\mathrm{infeasible}}$ (infeasible subproblems under blocking), N_{rollback} (rollback operations), and N_{fallback} (fallback activations).

Preference quality. We measure per-lecture preference satisfaction:

$$\mathrm{Pref} = \frac{\#\{(e, r, t) \in \mathcal{A} : t \in P_e\}}{\#\{(e, r, t) \in \mathcal{A}\}}, \qquad N_{\mathrm{PrefOutside}} = \#\{(e, r, t) \in \mathcal{A} : t \notin P_e\}.$$

5.4. Comparison baseline

We compare the proposed pipeline against our prior monolithic CP-SAT approach [Moreira and De Freitas 2024] using the same DS1–DS11 instance set. The primary comparison dimension is runtime scalability; we also report coverage, violations, and Pref to characterize feasibility and quality trade-offs under decomposition and repair.

6. Results and Discussion

This section reports the empirical performance of the proposed approach. Our primary goal is runtime reduction while maintaining completeness (coverage) and making any feasibility trade-offs explicit through global validation. Table 2 summarizes the main metrics per dataset: mean total runtime, coverage, global hard constraint violations (instructor and room conflicts), preference satisfaction per lecture, and robustness counters (rollback/fallback). Across all instances, the method achieved 100% coverage in all runs.

6.1. Runtime and scalability

Runtime increases with instance size, as expected. Nevertheless, the proposed approach remains operational even on the largest dataset (DS11), completing in approximately 4.45 seconds on average with low variance across runs. Several mid-sized datasets (e.g., DS7 and DS8) complete in well under one second, indicating that RDIR substantially reduces per-solve complexity compared to a monolithic model.

A consistent pattern across datasets is that most room subproblems solve quickly, while a small number of subproblems may reach the configured per-room time limit (2 seconds), creating a heavy-tailed distribution in the maximum subproblem time. This behavior is typical in decomposed combinatorial optimization, where local hardness can vary significantly by room group depending on instructor coupling and preference distributions [Gomes et al. 2000].

Table 2. Runtime values are reported as mean ± standard deviation over three runs. All other columns report validator counters and quality metrics from a single representative run.

Instance	Total time (s)	Coverage	Instr. conf.	Room conf.	Pref. (%)	Rollbacks	Fallbacks
DS1	0.034 ± 0.005	100%	0	0	100.00	0	0
DS2	0.119 ± 0.003	100%	0	0	76.27	0	0
DS3	0.154 ± 0.015	100%	0	0	91.94	0	0
DS4	0.223 ± 0.029	100%	0	0	90.71	0	0
DS5	0.316 ± 0.072	100%	0	0	88.86	0	0
DS6	1.821 ± 0.020	100%	322	0	55.62	5	2
DS7	0.473 ± 0.015	100%	0	0	85.66	0	0
DS8	0.591 ± 0.021	100%	0	0	86.80	0	0
DS9	2.216 ± 0.093	100%	366	0	70.87	5	3
DS10	7.090 ± 0.065	100%	218	0	79.09	3	1
DS11	4.447 ± 0.030	100%	0	0	89.58	0	0

6.2. Completeness, feasibility, and robustness

The method achieved full coverage (100%) on all instances and runs, demonstrating that decomposition combined with repair can preserve completeness at scale. For most datasets (DS1–DS5, DS7–DS8, and DS11), the incremental global blocking was sufficient to maintain instructor non-overlap, resulting in zero instructor conflicts and no need for rollback or fallback.

However, DS6, DS9, and DS10 triggered infeasibility in some room subproblems, requiring repair. In these datasets, selective rollback was frequently effective in restoring feasibility by undoing a small number of previously solved rooms that shared instructors with the infeasible subproblem. When infeasibility persisted after bounded attempts, the controlled fallback was activated.

Fallback solves the remaining room subproblem without global instructor blocking, preserving completeness at the cost of potentially violating instructor non-overlap (H3). As shown in Table 2, only DS6, DS9, and DS10 incur instructor conflicts (322, 366, and 218), while all other datasets keep H3 with zero conflicts. Thus, fallback is a last-resort mechanism for operational completeness and should be reported together with conflict metrics.

6.3. Preference satisfaction

Preference satisfaction is measured per lecture to ensure comparability across approaches. In datasets where fallback was not needed, preference satisfaction remained high, including DS11 with approximately 89.6% on average. In datasets where fallback was triggered (DS6 and DS9 in particular), preference satisfaction decreased substantially (e.g., 55.6% in DS6). This suggests that (i) stronger instructor coupling and (ii) repair/fallback decisions can constrain feasible time slots, reducing the ability to match preferences.

6.4. Comparison with the monolithic baseline

Our prior monolithic CP-SAT approach evaluated the same DS1–DS11 instances and reported first-solution and optimal-solution runtimes that grow sharply with instance size

[Moreira and De Freitas 2024]. The proposed decomposition-and-repair pipeline targets runtime scalability, achieving end-to-end runtimes in seconds even on DS11, while providing explicit global validation of feasibility and preference metrics. A full quantitative comparison against the baseline runtimes is presented in Table 3.

Table 3. Runtime comparison against the monolithic baseline (v1.0 from [Moreira and De Freitas 2024]; v2.0 from this work).

Instance	v1.0 Time 1st (s)	v1.0 Time OS (s)	v2.0 Total time (s)
DS1	0.001	4.37	0.034
DS2	0.005	24.04	0.119
DS3	0.011	62.35	0.154
DS4	0.026	290.96	0.223
DS5	0.118	922.58	0.316
DS6	0.393	2500.65	1.821
DS7	1.946	5686.50	0.473
DS8	7.432	16284.80	0.591
DS9	34.772	31438.80	2.216
DS10	216.290	50222.80	7.090
DS11	7825.07	50734.90	4.447

6.5. Discussion and implications

Overall, the results indicate that RDIR can deliver large runtime improvements with full coverage and transparent validation. For the majority of instances, global instructor consistency is preserved without repair. When fallback is triggered, the method preserves completeness under strict runtime bounds at the cost of potential instructor non-overlap violations, which are explicitly quantified by the global validator.

7. Threats to Validity

Our evaluation is limited to the datasets considered here and generalization to other institutions may vary. As discussed in ITC2007, timetabling requirements differ across institutions and formulations, and benchmark models often balance realism and generality by focusing on a core set of constraints while omitting institution-specific features [McCollum et al. 2010].

8. Conclusion and Future Work

We presented a room-wise CP-SAT decomposition with incremental repair for academic timetabling, targeting end-to-end runtimes suitable for operational use. Across eleven real instances, the pipeline achieved 100% coverage and, in most instances, preserved instructor and room feasibility under incremental global blocking. When bounded repair was insufficient, controlled fallback ensured coverage within runtime bounds but introduced instructor non-overlap violations, which were explicitly quantified by global validation. These results indicate that decomposition can yield substantial runtime gains, but maintaining global instructor feasibility under strict per-room limits remains a key challenge.

Future work will focus on reducing fallback-induced conflicts via stronger repair and conflict-aware grouping/ordering, and on broader empirical validation. This includes

sensitivity analyses of $A_{\max}$, K, the per-subproblem time limit, and the number of workers, as well as phase-level runtime breakdowns and comparisons on additional public benchmarks.

Artifact availability. Available upon reasonable request, subject to institutional approval and applicable policies.

References

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

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

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

Chen, M. C., Sze, S. N., Goh, S. L., Sabar, N. R., and Kendall, G. (2021). A survey of university course timetabling problem: Perspectives, trends and opportunities. *IEEE Access*, 9:106515–106529.

Di Gaspero, L., McCollum, B., and Schaerf, A. (2007). Curriculum based course timetabling (track 3). In *Proceedings of the ICAPS 2007 Workshop on Constraint Satisfaction Techniques for Planning and Scheduling (SSC/ITC track description)*. ITC-2007 Track 3 description; available online.

Di Gaspero, L. and Schaerf, A. (2003). Multi-neighbourhood local search with application to course timetabling. In *Practice and Theory of Automated Timetabling IV (PATAT 2002)*, volume 2740 of *Lecture Notes in Computer Science*, pages 262–275. Springer.

Gomes, C. P., Selman, B., Crato, N., and Kautz, H. (2000). Heavy-tailed phenomena in satisfiability and constraint satisfaction problems. *Journal of Automated Reasoning*, 24(1–2):67–100.

Google OR-Tools (2026). Cp-sat solver. `https://developers.google.com/optimization/cp/cp_solver`. Accessed on 2026-02-26.

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

Meyers, C. and Orlin, J. B. (2007). Very large-scale neighborhood search techniques in timetabling problems. In *Practice and Theory of Automated Timetabling VI (PATAT 2006)*, volume 3867 of *Lecture Notes in Computer Science*, pages 24–39. Springer.

Moreira, E. J. B. and De Freitas, S. A. A. (2024). A cp-sat approach for academic resource timetabling in higher education institutions: A case study at a major public university. In *2024 21st International Conference on Information Technology Based Higher Education and Training (ITHET)*, pages 1–8.