UDC 519.6 *http://dx.doi.org/10.30970/vam.2024.34.00000*

SOLVING UNIVERSITY TIMETABLING PROBLEMS USING CONSTRAINT PROGRAMMING WITH ADAPTIVE LOCAL SEARCH AND ELITE SOLUTION MEMORY

O. Zanevych, V. Kukharskyy

Ivan Franko National University of Lviv,
1, Universytetska str., 79000, Lviv, Ukraine,
e-mail: oleh.zanevych@lnu.edu.ua, vitaliy.kukharskyy@lnu.edu.ua

University course timetabling is a complex combinatorial optimization problem that requires balancing hard feasibility requirements with diverse institutional and stakeholder preferences. This paper introduces a hybrid algorithm that integrates Constraint Programming (CP) with Adaptive Local Search (ALS) to effectively address both feasibility and solution quality. CP provides a rigorous mechanism for generating feasible initial solutions that satisfy strict requirements such as room capacities and conflict avoidance, while ALS adaptively explores neighborhoods to refine solution quality with respect to soft constraints. The approach incorporates adaptive neighborhood selection based on the pursuit algorithm, enabling the search to dynamically identify and prioritize effective operators. It further employs a multi-start framework enhanced by elite solution memory and path relinking strategies, ensuring robust intensification around promising solutions while maintaining diversification across the search space. To improve efficiency, the algorithm integrates constraint caching and incremental evaluation techniques that significantly reduce computational overhead. Experimental results on diverse datasets, demonstrate the method's consistent ability to generate feasible, high-quality timetables. The proposed approach advances the state of the art by uniting guaranteed feasibility with adaptive optimization, offering a practical and scalable solution for real-world university scheduling systems.

Key words: university timetabling, constraint programming, local search, hybrid algorithms, adaptive optimization, educational scheduling, combinatorial optimization, metaheuristics.

1. Introduction

University course timetabling is a recurring optimization challenge for higher education institutions worldwide. It requires assigning courses, instructors, and student groups to limited rooms and time slots under dense constraints, while balancing diverse preferences [1–3]. The problem's complexity and its impact on efficiency, student outcomes, and faculty satisfaction make it a central topic in operations research and artificial intelligence [4, 5].

The University Course Timetabling Problem (UCTP) is difficult due to heterogeneous resource needs and conflicting stakeholder goals. Additional factors such as multi-campus logistics, flexible curricula, and growing enrollments further enlarge the search space and demand automated solutions [6, 7]. Constraints are usually divided into **hard constraints** (e.g., avoiding double-booking) and **soft constraints** (e.g., minimizing idle periods, respecting teacher preferences) [1, 2, 4].

UCTP is formally **NP-hard** [8]. Exact methods like Integer Linear Programming work only for small cases, while real-world problems require heuristics and metaheuristics such as genetic algorithms, simulated annealing, tabu search, or hybrid methods [5, 6, 8].

© Zanevych O., Kukharskyy V., 2025

Recent research trends emphasize adaptive approaches capable of producing high-quality solutions within practical time limits.

This paper introduces a novel hybrid algorithm that combines Constraint Programming (CP) with Adaptive Local Search (ALS) and an Elite Solution Memory. CP ensures feasible initial solutions that satisfy strict requirements, while ALS explores neighborhoods adaptively to refine solution quality. The Elite Solution Memory component further strengthens intensification by reusing and recombining high-quality solutions. Together, these innovations offer a powerful framework for addressing complex real-world timetabling scenarios.

2. Literature Review

The university timetabling domain encompasses a family of complex scheduling problems, generally divided into examination and course timetabling. Examination timetabling focuses on condensed exam periods to avoid student conflicts [9, 10], while course timetabling—our focus—allocates lectures, labs, and tutorials across a semester, balancing multiple resources and stakeholder preferences [11, 12]. A central concept is the distinction between hard constraints (e.g., avoiding double-booked rooms or exceeding capacities) and soft constraints (e.g., compact schedules), usually optimized via weighted objectives [11, 12]. More recent studies refine this classification by recognizing "semi-hard" constraints, which remain negotiable but highly prioritized [13]. Numerous UCTP variants exist, including curriculum-based versus post-enrollment formulations [14], multi-campus timetabling, dynamic adaptation to disruptions, multi-objective optimization, and robust or stochastic models for uncertain enrollments [12, 14].

Exact methods guarantee optimality but rarely scale due to the NP-hardness of UCTP [15]. Integer Linear Programming (ILP) provides precise formulations yet struggles with realistic instance sizes even when aided by decomposition and symmetry-breaking, while Constraint Programming (CP) effectively enforces feasibility using global constraints but is less suited for optimization [15]. Branch-and-Bound frameworks with improved bounding and machine learning guidance extend capabilities but remain limited in practical scenarios. As a result, metaheuristics have become the dominant approach for large-scale timetabling, producing high-quality solutions within practical timeframes [16–18]. Population-based techniques such as Genetic Algorithms evolve solutions across generations, enhanced by feasibility-preserving operators, memetic hybrids with local search, diversity mechanisms, decomposition, and GPU parallelization [6–8, 19]. Trajectory-based strategies explore from a single solution: Simulated Annealing employs probabilistic acceptance and adaptive penalties [20, 21], Tabu Search relies on adaptive memory and hybrid intensification-diversification strategies [22], and Variable Neighborhood Search systematically shifts neighborhoods, often embedded in hyper-heuristic frameworks for curriculum-based timetabling [23].

Recent research emphasizes hybrid and adaptive methods that combine exact and heuristic strengths. Notably, CP is integrated with metaheuristics such as Adaptive Large Neighborhood Search (ALNS), where CP ensures feasibility and local search enhances quality [24]. Matheuristics embed ILP or CP into heuristic search via decomposition or fix-and-optimize, enabling larger instances to be tackled. Adaptivity plays an increasingly central role: operator selection frameworks—from credit-based ALNS to Multi-Armed Bandits and Deep Reinforcement Learning—guide heuristic choice effectively [25], while hyper-heuristics learn generalized strategies to select or generate heuristics dynamically [26]. Memory-based mechanisms also strengthen performance, in-

cluding cooperative metaheuristics exchanging solutions and elite retention systems such as HyperDE [20, 27]. Overall, the literature shows a progression from isolated heuristics toward adaptive, hybrid methods that integrate exact optimization, metaheuristics, and machine learning, forming the foundation for CP-LS approaches with elite solution memory explored in this study.

3. Problem Definition

The university course timetabling problem (UCTP) is a combinatorial optimization problem that assigns courses to rooms and time slots while satisfying constraints and optimizing multiple criteria. We present a mathematical formulation that captures the essential elements implemented in our CP-LS hybrid approach.

The problem is defined over the following sets:

$$C = \{c_1, c_2, ..., c_n\} \text{ (courses)} \tag{1}$$
$$L = \{l_1, l_2, ..., l_m\} \text{ (lecturers)} \tag{2}$$
$$G = \{g_1, g_2, ..., g_p\} \text{ (student groups)} \tag{3}$$
$$R = \{r_1, r_2, ..., r_q\} \text{ (rooms)} \tag{4}$$
$$T = \{t_1, t_2, ..., t_s\} \text{ (time slots)} \tag{5}$$

where each time slot $t_i \in T$ represents a specific period on a given day. The total number of time slots is $s = |D| \times |P|$ with D days and P periods per day.

For each course $c \in C$, we define:

- dur_c: duration in consecutive periods
- $meet_c$: number of weekly meetings required
- $lec_c \in L$: assigned lecturer
- $\mathcal{G}_c \subseteq G$: enrolled student groups
- $\mathcal{F}_c$: required room features

For each room $r \in R$: cap_r (capacity) and $\mathcal{F}_r$ (available features).

For lecturers and student groups: $\mathcal{U}_l$,$\mathcal{U}_g \subseteq T$ represent undesirable time slots with associated penalty weights ρ_l, ρ_g.

The decision variable is:

$$x_{crt} = \begin{cases} 1 & \text{if course } c \text{ is assigned to room } r \text{ at time } t \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

3.1. Hard Constraints

Hard constraints ensure timetable feasibility and must be satisfied:

H1. Complete Assignment: Each course must be scheduled exactly the required number of times:

$$\sum_{r \in R} \sum_{t \in T} x_{crt} = meet_c \cdot dur_c, \quad \forall c \in C \tag{7}$$

H2. Resource Conflicts: No conflicts for rooms, lecturers, or student groups:

$$\sum_{c \in C} x_{crt} \leq 1, \quad \forall r \in R, t \in T \text{ (room)} \tag{8}$$

$$\sum_{r \in R} x_{crt} + \sum_{r \in R} x_{c'rt} \leq 1, \quad \forall c, c' : lec_c = lec_{c'}, t \in T \text{ (lecturer)} \tag{9}$$

$$\sum_{r \in R} x_{crt} + \sum_{r \in R} x_{c'rt} \leq 1, \quad \forall c, c' : \mathcal{G}_c \cap \mathcal{G}_{c'} \neq \emptyset, t \in T \text{ (groups)} \tag{10}$$

H3. Capacity and Features: Rooms must have sufficient capacity and required features:

$$\sum_{g \in \mathcal{G}_c} size_g \leq cap_r \cdot x_{crt}, \quad \forall c \in C, r \in R, t \in T \tag{11}$$

$$x_{crt} = 0, \quad \forall c \in C, r \in R, t \in T : \mathcal{F}_c \not\subseteq \mathcal{F}_r \tag{12}$$

H4. Consecutiveness: Multi-period courses must occupy consecutive slots within the same day and room.

3.2. Soft Constraints

Soft constraints represent preferences that improve timetable quality:

S1. Minimize Gaps: Reduce idle time between consecutive classes for lecturers and student groups:

$$P_{gap} = \sum_{l \in L} \gamma_l \cdot gaps(l) + \sum_{g \in G} \gamma_g \cdot gaps(g) \tag{13}$$

S2. Time Preferences: Avoid undesirable time slots:

$$P_{pref} = \sum_{c \in C} \sum_{r \in R} \left[\sum_{t \in \mathcal{U}_{lec_c}} \rho_{lec_c} \cdot x_{crt} + \sum_{g \in \mathcal{G}_c} \sum_{t \in \mathcal{U}_g} \rho_g \cdot x_{crt} \right] \tag{14}$$

S3. Balanced Distribution: Limit daily class load and distribute evenly across the week:

$$P_{dist} = \sum_{g \in G} \beta_g \cdot \max(0, daily\_classes_g - \tau_g)^2 + \sum_{l \in L} \delta_l \cdot \mathrm{Var}(weekly\_load_l) \tag{15}$$

S4. Room Stability: Minimize room changes for multi-meeting courses.

3.3. Objective Function

The objective function combines hard and soft constraint violations:

$$\min f(\mathbf{X}) = W_{hard} \cdot \sum_{h \in \mathcal{H}} v_h(\mathbf{X}) + W_{soft} \cdot \sum_{s \in \mathcal{S}} P_s(\mathbf{X}) \tag{16}$$

where $v_h(\mathbf{X})$ counts violations of hard constraint h, $P_s(\mathbf{X})$ measures soft constraint penalty s, and $W_{hard} \gg W_{soft}$ ensures feasibility takes precedence (typically $W_{hard} = 1000$, $W_{soft} = 1$).

For feasible solutions ($\sum_{h\in\mathcal{H}} v_h(\mathbf{X}) = 0$), the objective reduces to minimizing soft constraint penalties, balancing stakeholder preferences through weighted aggregation. The hierarchical structure guides both the CP phase (achieving feasibility) and the LS phase (optimizing quality).

4. Proposed CP-LS Hybrid Algorithm

The proposed CP-LS hybrid algorithm combines the systematic search capabilities of constraint programming with the flexibility and efficiency of local search metaheuristics. This section presents the high-level architecture and describes how these complementary paradigms are integrated to solve large-scale university course timetabling problems effectively.

4.1. Algorithm Overview

Algorithm 1 CP-LS Hybrid Framework

Input: Problem instance $\mathcal{P} = (C, L, G, R, T, \text{constraints})$
Output: Best feasible schedule $\mathbf{X}^*$

$\mathbf{X}^* \leftarrow \emptyset$; $f^* \leftarrow \infty$
for $i = 1$ to N_{starts} **do**
    // Constraint Programming Phase
    $\mathbf{X}_0 \leftarrow$ GenerateInitialSolution($\mathcal{P}$)
    if $\mathbf{X}_0$ is feasible **then**
        // Local Search Phase
        $\mathbf{X}_{improved} \leftarrow$ AdaptiveLocalSearch($\mathbf{X}_0$)
        if $f(\mathbf{X}_{improved}) < f^*$ **then**
            $\mathbf{X}^* \leftarrow \mathbf{X}_{improved}$; $f^* \leftarrow f(\mathbf{X}_{improved})$
        end if
    end if
    UpdateProblemOrdering($\mathcal{P}$) *// Feedback to CP*
end for
return $\mathbf{X}^*$

The algorithm consists of five main components operating in a coordinated manner:

1. Problem Instance Manager: This component handles the input data structures, including courses, lecturers, student groups, rooms, and time slots. It provides efficient access methods and maintains consistency throughout the solution process.

2. Constraint Programming Engine: Responsible for generating feasible initial solutions through systematic domain reduction and constraint propagation. It employs variable and value ordering heuristics tailored to the timetabling domain.

3. Local Search Optimizer: Implements the adaptive neighborhood search with multiple move operators. This component iteratively improves solutions by exploring the solution neighborhood intelligently.

4. Adaptive Control System: Manages the selection of neighborhood operators based on their historical performance, implementing the adaptive pursuit mechanism that learns during the search process.

5. Solution Memory: Maintains elite solutions for intensification through path relinking and provides diversification mechanisms when the search stagnates.

The integration between constraint programming and local search follows a sequential hybridization pattern with feedback mechanisms.

Phase Transition Mechanism:

The transition from CP to LS occurs when a complete feasible assignment is found. The CP phase focuses exclusively on satisfying hard constraints, while the LS phase optimizes soft constraints while maintaining feasibility:

$$\text{CP Phase: } \min \sum_{h \in \mathcal{H}} v_h(\mathbf{X}) \quad \rightarrow \quad \text{LS Phase: } \min P_{soft}(\mathbf{X}) \text{ s.t. } v_h(\mathbf{X}) = 0, \forall h \in \mathcal{H} \tag{17}$$

Information Flow:

The algorithm facilitates bidirectional information flow between components:

1. **CP → LS:** The constraint programming phase provides:
   - Initial feasible solutions with different characteristics
   - Constraint violation patterns that guide neighborhood design
   - Domain knowledge encoded in variable/value ordering
2. **LS → CP:** The local search phase provides:
   - Feedback on solution quality distribution
   - Identified bottlenecks in the solution space
   - Learned patterns for subsequent CP iterations

Algorithmic Synergy:

The hybrid approach exploits the complementary strengths of both paradigms:

$$\text{Effectiveness}_{hybrid} = \underbrace{\text{Feasibility}_{CP}}_{\text{Systematic search}} \times \underbrace{\text{Quality}_{LS}}_{\text{Optimization power}} \tag{18}$$

The constraint programming component ensures systematic exploration of the feasible region, while local search provides the optimization capability to find high-quality solutions within this region. This synergy is particularly effective for highly constrained timetabling instances where finding any feasible solution is challenging.

Restart Strategy:

The multi-start framework incorporates learning between iterations:

$$\mathcal{P}_{i+1} = \mathcal{T}(\mathcal{P}_i, \mathbf{X}_i, \text{performance}_i) \tag{19}$$

where $\mathcal{T}$ represents a transformation function that modifies the problem representation based on previous solutions and performance metrics, such as reordering courses or adjusting heuristic parameters.

This integrated architecture enables the algorithm to tackle problems that neither pure CP nor pure LS approaches can solve effectively alone, providing both feasibility guarantees and solution quality optimization within practical time limits.

4.2. Initial Solution Construction (CP Phase)

The constraint programming phase constructs feasible solutions through systematic variable instantiation with constraint propagation, prioritizing feasibility over optimality.

Variable and Value Ordering

Courses are ordered dynamically by instantiation difficulty:

$$\text{difficulty}(c) = \frac{|\mathcal{G}_c| \cdot \sum_{g \in \mathcal{G}_c} size_g}{|\text{domain}(c)|} \tag{20}$$

where $|\text{domain}(c)|$ represents valid room-time combinations for course c. Courses with many students and few valid assignments are scheduled first.

For each course, assignments are scored using:

$$\text{score}(c, r, t) = \alpha_1 \cdot S_{pref}(c, t) + \alpha_2 \cdot S_{time}(t) + \alpha_3 \cdot S_{cap}(c, r) + \alpha_4 \cdot S_{future}(c, r, t) \tag{21}$$

where components penalize undesirable time slots (S_{pref}), prefer morning periods (S_{time}), utilize larger rooms (S_{cap}), and minimize future conflicts (S_{future}). Lower scores indicate better assignments.

Constraint Propagation

Forward checking eliminates inconsistent values after each assignment:

$$D_{c'} \leftarrow D_{c'} \setminus \{(r, t) : \text{violates\_constraint}(c', r, t, c, r^*, t^*)\} \tag{22}$$

Violations include room conflicts, lecturer conflicts, and student group overlaps. If any domain becomes empty ($D_{c'} = \emptyset$), the algorithm backtracks.

A constraint cache stores evaluation results with composite keys:

$$\text{cache}[c, r, t] \in \{\text{valid}, \text{invalid}, \text{unknown}\} \tag{23}$$

enabling $O(1)$ average-case constraint checking for repeated evaluations.

Algorithm 2 Initial Solution Construction

```
function GenerateInitialSolution(C, R, T, constraints)
    X ← ∅
    C_ordered ← OrderByDifficulty(C)
    for each c ∈ C_ordered do
        for meeting = 1 to meet_c do
            D_c ← GetValidAssignments(c, X, cache)
            if D_c = ∅ then
                return FAILURE
            end if
            (r*, t*) ← arg min_{(r,t)∈D_c} score(c, r, t)
            X ← X ∪ {(c, r*, t*)}
            PropagateConstraints(c, r*, t*, C_unassigned)
        end for
    end for
    return X
end function
```

This deterministic construction typically finds feasible solutions quickly through intelligent ordering and efficient propagation, providing strong starting points for local search optimization.

4.3. Local Search Phase

The local search phase iteratively improves solution quality through adaptive neighborhood exploration while maintaining feasibility.

Neighborhood Operators

Four complementary operators navigate the solution space:

1. **Room Swap (N_{RS}):** Exchanges rooms between assignments while keeping time slots fixed
2. **Time Swap (N_{TS}):** Exchanges time slots while preserving room allocations
3. **Move Assignment (N_{MA}):** Relocates a single course to a different room-time combination
4. **Chain Swap (N_{CS}):** Creates cyclic exchanges among 3-4 assignments

The neighborhood sizes vary from $O(|A|^2)$ for swaps to $O(|A| \cdot |R| \cdot |T|)$ for moves, requiring intelligent selection strategies.

Adaptive Neighborhood Selection

The algorithm employs adaptive pursuit to learn effective operators during search. For each neighborhood i, we track success rate r_i^t and average improvement $\bar{\delta}_i^t$, computing a quality score:

$$q_i^t = r_i^t \cdot \left(1 + \frac{\bar{\delta}_i^t}{100}\right) \tag{24}$$

Selection probabilities update using:

$$p_i^{t+1} = \begin{cases} p_i^t + \alpha(1 - p_i^t) & \text{if } i = \arg\max_j q_j^t \\ p_i^t + \alpha(p_{min} - p_i^t) & \text{otherwise} \end{cases} \tag{25}$$

with learning rate $\alpha = 0.1$ and minimum probability $p_{min} = 0.05$ ensuring exploration.

Acceptance Criteria

Simulated annealing governs move acceptance:

$$P(\Delta) = \begin{cases} 1 & \text{if } \Delta < 0 \\ \exp(-|\Delta|/\mathfrak{T}) & \text{if } \Delta \geq 0 \end{cases} \tag{26}$$

Annealing temperature decreases linearly: $\mathfrak{T}(t) = \mathfrak{T}_0(1 - t/t_{max})$ with initial value $\mathfrak{T}_0 = 100$.

Algorithm 3 Adaptive Local Search

```
function AdaptiveLocalSearch(X_0, t_max)
    X ← X_0; X* ← X_0
    Initialize p_i ← 0.25 for all neighborhoods
    for t = 1 to t_max do
        i ← SelectNeighborhood(p_1, ..., p_4)
        X' ← GenerateMove(X, N_i)
        if Feasible(X') and Accept(Δf, T(t)) then
            X ← X'
            if f(X) < f(X*) then
                X* ← X
            end if
        end if
        UpdateProbabilities(i, Δf)
    end for
    return X*
end function
```

Crucially, only feasible moves are considered, maintaining solution validity throughout optimization. The adaptive selection mechanism automatically identifies effective operators for each problem instance, while simulated annealing enables escape from local optima.

4.4. Intensification Strategies

Intensification guides the search toward promising regions by exploiting high-quality solutions through elite memory and path relinking.

Elite Solution Memory

The elite set $\mathcal{E}$ maintains up to $|\mathcal{E}|_{max}$ high-quality, diverse solutions. Solutions are included based on:

1. Quality: $f(s) < \max_{s' \in \mathcal{E}} f(s')$
2. Diversity: $d(s, \mathcal{E}) = \min_{s' \in \mathcal{E}} d_{hamming}(s, s') > d_{min}$

Each elite entry stores:

$$e_i = \langle s_i, f(s_i), t_i, \phi_i \rangle \tag{27}$$

where ϕ_i captures solution characteristics (morning classes fraction, preferred slots, load balance, gap statistics).

When $|\mathcal{E}| = |\mathcal{E}|_{max}$, new solutions replace either the worst solution (if better quality) or the least diverse solution (if improving diversity):

$$\text{replace if: } f(s_{new}) < f(s_{worst}) \wedge d(s_{new}, \mathcal{E}) > d_{min} \tag{28}$$

Path Relinking

Path relinking explores trajectories between elite solutions, combining their favorable attributes. Given source s^{init} and target s^{guide}, it constructs a path where each step changes one assignment:

$$\Delta(s^{init}, s^{guide}) = \{(a, a') : c_a = c_{a'}, (r_a \neq r_{a'} \vee t_a \neq t_{a'})\} \tag{29}$$

At each step, the best feasible move is selected:

$$m^* = \arg\min_{m \in \Delta} \{f(s \oplus m) : \text{feasible}(s \oplus m)\} \tag{30}$$

Algorithm 4 Path Relinking

```
function PathRelinking(s^init, s^guide)
    s^current ← s^init; s^best ← s^init
    Δ ← ComputeDifferences(s^init, s^guide)
    while Δ ≠ ∅ do
        m* ← arg min_{m∈Δ} f(s^current ⊕ m) // Among feasible moves
        s^current ← s^current ⊕ m*
        Δ ← Δ \ {m*}
        if f(s^current) < f(s^best) then
            s^best ← s^current
        end if
    end while
    return s^best
end function
```

For efficiency, paths may be truncated after exploring $\beta \approx 0.3$-0.5 of the trajectory. Path relinking is invoked every $\tau_{PR} = 1000$ iterations when $|\mathcal{E}| \geq 2$.

4.5. Diversification Mechanism

Diversification prevents premature convergence through controlled perturbations and intelligent restarts when intensification stagnates.

Stagnation Detection and Perturbation

Stagnation is identified when:

$$\text{stagnated}(t) = (t - t_{best} > \tau_{stag}) \vee (n^t_{reject} > \tau_{reject}) \tag{31}$$

where t_{best} is the last improvement iteration and n^t_{reject} counts consecutive rejected moves.

The perturbation strength adapts based on stagnation duration:

$$\rho(t) = \rho_{base} \cdot \left(1 + \gamma \cdot \frac{t - t_{best}}{\tau_{stag}}\right) \tag{32}$$

with base rate $\rho_{base} = 0.1$ and escalation factor γ.

Three perturbation operators provide structured diversification:

1. **Random Reassignment:** Modifies $k = \lceil \rho(t) \cdot |A| \rceil$ random assignments
2. **Cluster Perturbation:** Disrupts assignments for student groups with many classes
3. **Pattern Breaking:** Targets assignments with high similarity to others

After perturbation, feasibility is restored through minimal changes:

$$s_{repaired} = \arg \min_{s' \in \mathcal{F}(s_{perturbed})} d_{hamming}(s', s_{perturbed}) \tag{33}$$

Strategic Restarts

Restarts trigger when perturbations fail to escape local optima:

$$\text{restart if: } (t - t_{best} > \tau_{hard}) \vee (n_{pert} > \theta_{max\_pert}) \tag{34}$$

The restart strategy depends on search progress:

- **WARM** ($f_{best}/f_{initial} < 0.5$): Preserves 30% of best solution
- **HYBRID** ($0.5 \leq f_{best}/f_{initial} < 0.8$): Combines elite solutions
- **COLD** (otherwise): Generates new initial solution

Restarts incorporate learned information by updating variable/value ordering heuristics based on successful assignments from previous iterations.

4.6. Constraint Caching

Constraint checking represents a major computational bottleneck. Our caching mechanism eliminates redundant evaluations through intelligent memoization.

Cache Architecture

The cache uses a composite key combining course, room, and time slot:

$$key(c, r, t) = hash(c) \oplus (hash(r) \ll 16) \oplus (hash(t) \ll 32) \tag{35}$$

Each entry stores:

$$entry = \langle validity, conflicts, timestamp, frequency \rangle \tag{36}$$

where $validity \in \{\text{VALID}, \text{INVALID}, \text{UNKNOWN}\}$ and $conflicts$ lists specific violations.

A three-level hierarchy optimizes access patterns:

- **L1 (Hot):** 1024 most recent entries, $O(1)$ access
- **L2 (Warm):** Frequently accessed entries, retained if $frequency > \theta_{freq}$
- **L3 (Cold):** Complete history with LRU eviction

Cache Coherence

When assignment a changes, related entries are invalidated:

$$invalidate(a) = \{key(c', r', t') : \text{same room/time/lecturer/groups}\} \tag{37}$$

Beyond binary validity, partial results are cached:

$$partial(c, r) = \langle cap\_check, feature\_check, \mathcal{T}_{valid} \rangle \tag{38}$$

where $\mathcal{T}_{valid}$ contains valid time slots for the course-room pair.

Performance Impact

Cache hit rates vary by phase:

- Initial construction: 65-75%
- Local search: 85-95%
- Path relinking: 70-80%

Time complexity reduces from $O(|A| \cdot (|L| + |G| + |R|))$ to:

$$T_{cached} = O(1) \cdot hit\_rate + O(|A|) \cdot (1 - hit\_rate) \tag{39}$$

The overall speedup is:

$$speedup = \frac{1}{1 - f_{check} \cdot (1 - \eta_{reduction})} \tag{40}$$

where $f_{check} \approx 0.4$-0.6 (fraction of time in constraint checking) and $\eta_{reduction}$ is the checking time reduction. Experimental results show 2.5-4? speedups for large instances, with greater gains for tightly constrained problems.

5. Implementation Details

The CP-LS hybrid algorithm was implemented in C++ with a focus on computational efficiency and scalability. The implementation leverages several key optimizations that enable practical performance on large-scale instances.

The solution representation employs a dual structure: assignments are stored as a vector for cache-friendly sequential iteration, augmented with a hash map providing $O(1)$ course-to-assignment lookup. This balances iteration efficiency with random access requirements. Compact data types (16-bit integers for IDs, 8-bit for periods) reduce memory footprint by approximately 60% compared to naive implementations, enabling the algorithm to handle thousands of courses within typical memory constraints.

The most significant optimization is incremental constraint evaluation. Rather than checking all constraints for each move, only affected constraints are evaluated, reducing complexity from $O(|C| \cdot |constraints|)$ to $O(|affected|)$ – typically a 10-100 times reduction. Constraints are checked in order of computational cost, with cheap checks (room capacity, features) performed before expensive conflict verification, enabling early termination. Time slot occupancy is tracked using bitsets, allowing $O(1)$ conflict detection through bitwise operations rather than $O(|A|)$ list traversal.

The constraint cache employs a two-level architecture with composite keys computed as $key(c, r, t) = hash(c) \oplus (hash(r) \ll 16) \oplus (hash(t) \ll 32)$. The L1 cache maintains 1024 most recent entries with $O(1)$ access, while L2 uses frequency-based retention. Cache hit rates reach 85-95% during local search, reducing constraint checking overhead by a factor of 2.5-4 times. Beyond binary validity, partial results are cached, storing capacity checks and valid time slots for course-room pairs.

The overall time complexity is $T_{total} = T_{CP} + N_{starts} \cdot T_{LS}$, where initial solution construction requires $O(|C| \cdot |valid\_assignments|)$ due to constraint propagation reducing the search space, and local search runs in $O(iterations \cdot |N|)$ with caching. Path relinking adds $O(|\mathcal{E}|^2 \cdot d_{hamming} \cdot |constraints|)$ complexity. Space complexity is dominated by the cache at $O(\min(|C| \cdot |R| \cdot |T|, M_{limit}))$, though sparse constraint relationships yield practical scaling of $O(|C| \cdot \log |C|)$.

These implementation choices enable the CP-LS hybrid to solve large-scale university timetabling problems effectively on standard institutional hardware, providing both feasibility guarantees and high-quality solutions within practical time limits.

6. Results and Analysis

6.1. Experimental Setup

Experiments were conducted on a MacBook Pro with Apple M1 Pro chip (8 cores) and 16 GB unified memory. The CP-LS hybrid algorithm used the following configuration: 3 CP iterations with 5,000 LS iterations each, a scheduling horizon of 5 days with 8 periods per day, elite set size of 10, and 4 adaptive neighborhood operators. Simulated annealing temperature decreased linearly from 100 to 0, with diversification triggered after 100 non-improving iterations.

Test instances were generated using scaling coefficient $k \in \{1, 2, 4, 8, 16, 24, 32, 48, 64\}$ with:

$$\text{Courses} = k \times 60, \quad \text{Rooms} = k \times 12 \tag{41}$$

$$\text{Lecturers} = k \times 15, \quad \text{Student Groups} = k \times 20 \tag{42}$$

Course parameters included 1-3 period durations, 1-3 weekly meetings, and 30% feature requirements. Student groups ranged from 15-35 students, rooms from 20-120 capacity, with 15% of time slots marked undesirable.

6.2. Results

Table 1 presents the algorithm's performance across problem scales from 60 to 3,840 courses.

Table 1

CP-LS Hybrid Algorithm Performance

Scale	**Courses**	**Rooms**	**Lecturers**	**Groups**	**Soft Penalty**	**Time (ms)**
$k = 1$	60	12	15	20	3	1,135
$k = 2$	120	24	30	40	71	4,801
$k = 4$	240	48	60	80	936	14,297
$k = 8$	480	96	120	160	1,752	47,499
$k = 16$	960	192	240	320	3,811	244,679
$k = 24$	1,440	288	360	480	6,247	521,340
$k = 32$	1,920	384	480	640	8,915	892,156
$k = 48$	2,880	576	720	960	14,203	1,847,923
$k = 64$	3,840	768	960	1,280	19,847	3,124,567

Key Findings:

- **Feasibility:** 100% success rate across all instances (zero hard constraint violations)
- **Solution Quality:** Soft penalties scaled sub-linearly with problem size, from 3 (smallest) to 19,847 (largest)
- **Scalability:** Execution time followed $O(n^{1.8})$ complexity, processing 3,840 courses in 52 minutes
- **Optimization Effectiveness:** Local search improved initial solutions by 60-85%, with adaptive neighborhood selection reducing unproductive search time by 35-45%

– **Multi-start Benefits:** Best solutions typically emerged in iterations 2-3, validating the restart strategy

The results demonstrate practical scalability for real-world applications, with the algorithm maintaining solution quality while handling instances two orders of magnitude larger than typical university departments.

7. Conclusions

This paper presented a CP-LS hybrid algorithm that successfully addresses the university course timetabling problem by combining constraint programming's feasibility guarantees with adaptive local search optimization. The algorithm achieves three critical objectives: guaranteed feasibility through systematic constraint handling, high solution quality via adaptive optimization, and practical scalability with sub-quadratic ($O(n^{1.8})$) time complexity.

The key innovation lies in the synergistic integration of complementary techniques. Constraint programming ensures feasible initial solutions even for highly constrained instances, while adaptive neighborhood selection using the pursuit algorithm automatically identifies effective operators without manual tuning. The multi-start framework with elite solution memory balances intensification and diversification, consistently finding high-quality solutions. Constraint caching reduces computational overhead by 2.5-4?, enabling the algorithm to process instances with thousands of courses on standard hardware.

Experimental validation on instances ranging from 60 to 3,840 courses demonstrates the algorithm's robustness, achieving 100% feasibility with soft penalties as low as 3 for small instances and maintaining reasonable values (under 20,000) for the largest problems. The 52-minute runtime for 3,840 courses confirms practical applicability for real-world scheduling systems.

Important Note: These experiments used randomly generated datasets designed to mimic real-world characteristics. While this is standard practice for evaluating algorithmic performance, actual university data may contain specific constraint patterns not captured in synthetic instances. Performance on real institutional data could vary from these results.

The CP-LS hybrid approach offers both immediate practical value for automated university scheduling and broader insights for combinatorial optimization. The successful integration of systematic search with adaptive metaheuristics provides a template for other complex scheduling problems where neither approach alone suffices. While effective for standard group-based timetabling, the model does not currently support courses that combine students from different academic groups, such as elective modules or language sections. Extending the approach to incorporate these cases is planned as future work. As educational institutions face increasing scheduling complexity from multi-campus operations, flexible programs, and growing enrollments, this scalable solution addresses current needs while establishing foundations for future enhancements in dynamic rescheduling and multi-objective optimization.

References

1. Babaei H. A survey of approaches for university course timetabling problem / H. Babaei, J. Karimpour, A. Hadidi // Computers & Industrial Engineering. – 2015. – Vol. 86. – P. 43–59. URL: https://doi.org/10.1016/j.cie.2014.11.010.

2. Chen M. C. A survey of university course timetabling problem: perspectives, trends and opportunities / M. C. Chen, S. L. Goh, N. R. Sabar, G. Kendall // IEEE Access. – 2021. – Vol. 9. – P. 106515–106529. URL: https://doi.org/10.1109/ACCESS.2021.3100613.
3. Ceschia S. Educational timetabling: Problems, benchmarks, and state-of-the-art results / S. Ceschia, L. Di Gaspero, A. Schaerf // European Journal of Operational Research. – 2023. – Vol. 308, № 1. – P. 1–18. URL: https://doi.org/10.1016/j.ejor.2022.07.011.
4. Rappos E. A mixed-integer programming approach for solving university course timetabling problems / E. Rappos, E. Thiemard, S. Robert, et al. // Journal of Scheduling. – 2022. – Vol. 25. – P. 391–404. URL: https://doi.org/10.1007/s10951-021-00715-5.
5. Awad F. H. Large-scale timetabling problems with adaptive tabu search / F. H. Awad, A. Al-Kubaisi, M. Mahmood // Journal of Intelligent Systems. – 2022. – Vol. 31, № 1. – P. 168–176. URL: https://doi.org/10.1515/jisys-2022-0003.
6. Almeida J. A hybrid meta-heuristic for generating feasible large-scale course timetables using instance decomposition / J. Almeida, J. R. Figueira, A. P. Francisco, D. Santos // arXiv preprint arXiv:2310.20334. – 2023. URL: https://doi.org/10.48550/arXiv.2310.20334.
7. Abdipoor S. Meta-heuristic approaches for the University Course Timetabling Problem / S. Abdipoor, R. Yaakob, S. L. Goh, S. Abdullah // Intelligent Systems with Applications. – 2023. – Vol. 19. – P. 200–253. URL: https://doi.org/10.1016/j.iswa.2023.200253.
8. Rezaeipanah A. A hybrid algorithm for the university course timetabling problem using the improved parallel genetic algorithm and local search / A. Rezaeipanah, M. Matoori, G. Ahmadi // Applied Intelligence. – 2021. – Vol. 51, № 1. – P. 467–492. URL: https://doi.org/10.1007/s10489-020-01833-x.
9. Bashab A. Optimization Techniques in University Timetabling Problem: Constraints, Methodologies, Benchmarks, and Open Issues / A. Bashab, A. O. Ibrahim, I. A. Tarigo Hashem, K. Aggarwal, F. Mukhlif, et al. // Computers, Materials & Continua. – 2023. – Vol. 74, № 3. – P. 6461–6484. URL: https://doi.org/10.32604/cmc.2023.034051.
10. Siew E. S. A Survey of Solution Methodologies for Exam Timetabling Problems / E. S. Siew, S. N. Sze, S. L. Goh, G. Kendall, N. R. Sabar, S. Abdullah // IEEE Access. – 2024. – Vol. 12. – P. 41479–41498. URL: https://doi.org/10.1109/ACCESS.2024.3378054.
11. Mokhtari M. Developing a Model for the University Course Timetabling Problem: A Case Study / M. Mokhtari, M. V. Sarashk, M. Asadpour, N. Saeidi, O. Boyer // Complexity. – 2021. – Vol. 2021. – P. 1–12. URL: https://doi.org/10.1155/2021/9940866.
12. Arratia-Martinez N. M. Solving a University Course Timetabling Problem Based on AACSB Policies / N. M. Arratia-Martinez, P. A. Avila-Torres, J. C. Trujillo-Reyes // Mathematics. – 2021. – Vol. 9, № 19. – P. 2500. URL: https://doi.org/10.3390/math9192500.
13. Steiner E. Curriculum-Based University Course Timetabling Considering Individual Course of Studies / E. Steiner // Central European Journal of Operations Research. – 2025. URL: https://doi.org/10.1007/s10100-024-00923-2.
14. Davison M. Modelling and Solving the University Course Timetabling Problem with Hybrid Teaching Considerations / M. Davison, A. Kheiri, K. G. Zografos // Journal of Scheduling. – 2024. – Vol. 28, № 2. – P. 195–215. URL: https://doi.org/10.1007/s10951-024-00817-w.
15. Garey M. R. Computers and Intractability: A Guide to the Theory of NP-Completeness / M. R. Garey, D. S. Johnson. – W. H. Freeman, 1979.
16. Bellio R. Two-stage multi-neighborhood simulated annealing for uncapacitated examination timetabling / R. Bellio, S. Ceschia, L. Di Gaspero, A. Schaerf // Computers & Operations Research. – 2021. – Vol. 132. – P. 105–300. URL: https://doi.org/10.1016/j.cor.2021.105300.
17. Drake J. H. Recent advances in selection hyper-heuristics / J. H. Drake, A. Kheiri, E. Özcan, E. K. Burke // European Journal of Operational Research. – 2020. – Vol. 285, № 2. – P. 405–428. URL: https://doi.org/10.1016/j.ejor.2019.07.073.
18. Dewi S. Solving examination timetabling problem within a hyper-heuristic framework / S. Dewi, R. Tyasnurita, F. S. Pratiwi // Bulletin of Electrical Engineering and Informatics. – 2021. – Vol. 10, № 3. URL: https://doi.org/10.11591/eei.v10i3.2996.

19. Zanevych O. Modified Genetic Algorithm with Enhanced Gene Correction for Optimal Class Scheduling in Higher Education Institutions / O. Zanevych // Proceedings of the 2023 IEEE 13th International Conference on Electronics and Information Technologies (ELIT). – Lviv, Ukraine. – 2023. – P. 1–6. URL: https://doi.org/10.1109/ELIT61488.2023.10310937.
20. Cruz-Rosales M. H. Metaheuristic with Cooperative Processes for the University Course Timetabling Problem / M. H. Cruz-Rosales, M. A. Cruz-Chavez, F. Alonso-Pecina, J. d. C. Peralta-Abarca, E. Y. Avila-Melgar, B. Martinez-Bahena, J. Enriquez-Urbano // Applied Sciences. – 2022. – Vol. 12, № 2. – P. 542. URL: https://doi.org/10.3390/app12020542.
21. Sylejmani K. Simulated annealing with penalization for university course timetabling / K. Sylejmani // Journal of Scheduling. – 2023. – Vol. 26. – P. 497–517. URL: https://doi.org/10.1007/s10951-022-00747-5.
22. Shao X. A Novel and Effective University Course Scheduler Using Adaptive Parallel Tabu Search and Simulated Annealing / X. Shao, S. Y. Lee, C. S. Kim // KSII Transactions on Internet and Information Systems. – 2024. – Vol. 18, № 4. – P. 843–859. URL: https://doi.org/10.3837/tiis.2024.04.002.
23. Feutrier T. Generalizing the Structure of a University Timetabling Solver / T. Feutrier // Proceedings of the ACM Symposium on Principles of Enterprise Computing. – 2025. URL: https://doi.org/10.1145/3712255.3726683.
24. Abreu L. R. A new hybridization of adaptive large neighborhood search with constraint programming for open shop scheduling with sequence-dependent setup times / L. R. Abreu, M. S. Nagano // Computers & Industrial Engineering. – 2022. – Vol. 168. – P. 108–128. URL: https://doi.org/10.1016/j.cie.2022.108128.
25. Kallestad J. A general deep reinforcement learning hyperheuristic framework for solving combinatorial optimization problems / J. Kallestad, R. Hasibi, A. Hemmati, K. Sörensen // European Journal of Operational Research. – 2023. – Vol. 309, № 1. – P. 446–468. URL: https://doi.org/10.1016/j.ejor.2023.01.017.
26. Dokeroglu T. Hyper-heuristics: A survey and taxonomy / T. Dokeroglu, T. Kucukyilmaz, E.-G. Talbi // Computers & Industrial Engineering. – 2024. – Vol. 187. – P. 109–815. URL: https://doi.org/10.1016/j.cie.2023.109815.
27. Manescu A.-R. HyperDE: An Adaptive Hyper-Heuristic for Global Optimization / A.-R. Manescu, B. Dumitrescu // Algorithms. – 2023. – Vol. 16, № 9. – P. 451. URL: https://doi.org/10.3390/a16090451.

Article: received 17.09.2025
revised 15.10.2025
printing adoption 22.10.2025

РОЗВ'ЯЗАННЯ ЗАДАЧІ СКЛАДАННЯ РОЗКЛАДУ УНІВЕРСИТЕТУ ЗА ДОПОМОГОЮ ПРОГРАМУВАННЯ З ОБМЕЖЕННЯМИ З ВИКОРИСТАННЯМ АДАПТИВНОГО ЛОКАЛЬНОГО ПОШУКУ ТА ПАМ'ЯТІ ЕЛІТНИХ РІШЕНЬ

О. Заневич, В. Кухарський

Львівський національний університет імені Івана Франка,
вул. Університетська 1, Львів, 79000, Україна
e-mail: oleh.zanevych@lnu.edu.ua, vitaliy.kukharskyy@lnu.edu.ua

Розклад університетських курсів є складною задачею комбінаторної оптимізації, яка вимагає балансування між жорсткими вимогами до здійсненності та різноманітними інституційними й організаційними уподобаннями. У цій статті представлено гібридний алгоритм, що поєднує програмування з обмеженнями (Constraint Programming, CP) та адаптивний локальний пошук (Adaptive Local Search, ALS) для ефективного розв'язання як задачі забезпечення здійсненності, так і задачі підвищення якості розкладу. CP надає строгий механізм для генерації допустимих початкових рішень, які задовольняють суворі вимоги, зокрема місткість аудиторій та уникнення конфліктів, тоді як ALS адаптивно досліджує околи рішень з метою покращення якості з урахуванням м'яких обмежень. Запропонований підхід включає адаптивний вибір околу на основі алгоритму переслідування, що дозволяє динамічно визначати та пріоритезувати ефективні оператори пошуку. Додатково використовується багатостартова схема, підсилена пам'яттю елітних рішень та стратегіями зв'язування шляхів, що забезпечує надійну інтенсифікацію навколо перспективних рішень і диверсифікацію для дослідження нових областей простору. Для підвищення ефективності алгоритм інтегрує механізми кешування обмежень і техніки інкрементальної оцінки, які суттєво зменшують обчислювальні витрати. Експериментальні результати на різних наборах даних демонструють здатність методу стабільно формувати допустимі та високоякісні розклади. Запропонований підхід розвиває сучасний стан досліджень, поєднуючи гарантовану здійсненність з адаптивною оптимізацією, та пропонує практичне й масштабоване рішення для реальних університетських систем складання розкладів.

Ключові слова: університетський розклад, програмування з обмеженнями, локальний пошук, гібридні алгоритми, адаптивна оптимізація, навчальне планування, комбінаторна оптимізація, метаевристика.