UDC 519.6 *http://dx.doi.org/10.30970/vam.2026.36.00000*

ANT COLONY OPTIMIZATION FOR UNIVERSITY COURSE TIMETABLING: IMPLEMENTATION AND EXPERIMENTAL ANALYSIS

O. Zanevych, V. Kukharskyy

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

The University Course Timetabling Problem (UCTP) is a complex NP-hard combinatorial optimization problem involving the assignment of courses to time slots and rooms under numerous hard and soft constraints. This paper presents an implementation and experimental analysis of Ant Colony Optimization (ACO) for solving UCTP, incorporating problem-specific heuristic design and an elite pheromone update strategy to guide the search process. The main contribution of this work lies in a systematic evaluation of ACO behavior on timetabling instances of varying sizes, including analysis of convergence, constraint satisfaction, and scalability. In addition, a heuristic caching mechanism is introduced as an implementation-level optimization to reduce redundant computations and improve efficiency without affecting solution quality. The proposed approach is evaluated on synthetic datasets with increasing problem sizes, enabling detailed analysis of performance characteristics. The results demonstrate that ACO is effective for small and medium-sized instances, providing stable optimization of soft constraints, while also revealing limitations in achieving full feasibility for larger problems. These findings offer practical insights into the applicability, strengths, and limitations of ACO for automated timetabling systems.

Key words: ant Colony Optimization, University Course Timetabling, Combinatorial Optimization, Metaheuristics, Constraint Satisfaction.

1. Introduction

University course timetabling involves assigning courses to time slots and rooms while satisfying constraints related to lecturers, student groups, and available resources. Manual scheduling is time-consuming and often produces conflicts or inefficient timetables.

The University Course Timetabling Problem (UCTP) is NP-hard [1], so the search space grows exponentially with problem size, making exact methods impractical for realistic instances. Therefore, metaheuristic approaches are widely used to obtain high-quality solutions within acceptable computational time.

Ant Colony Optimization (ACO) is well suited to such problems because it combines exploration and exploitation through pheromone-based learning and heuristic guidance. In UCTP, hard constraints must be strictly satisfied, whereas soft constraints represent preferences such as reducing schedule gaps or avoiding undesirable time slots. The objective is to construct feasible timetables while minimizing soft-constraint violations.

This paper presents an ACO-based approach for UCTP that incorporates problem-specific heuristics, an elite pheromone update strategy, and a heuristic caching mechanism to reduce redundant computations. The contribution of this work is primarily empirical and implementation-oriented, focusing on the analysis of ACO behavior for UCTP and practical optimization techniques rather than proposing a fundamentally new algorithmic variant. The method is evaluated on datasets of varying sizes to analyze convergence, constraint satisfaction, and scalability.

2. Related Work

Educational timetabling has been widely studied, with approaches ranging from exact methods to metaheuristics. Surveys such as [1] classify timetabling methods into exact algorithms, constructive heuristics, and metaheuristics. In practice, many modern approaches combine these categories, giving rise to hybrid methods such as memetic algorithms and matheuristics, which integrate metaheuristics with local search or exact techniques. Exact approaches, including integer and constraint programming, are generally limited to small instances because of computational complexity [2].

Metaheuristics are therefore the dominant practical approach. Methods based on Genetic Algorithms [3, 4], Simulated Annealing [5], Tabu Search [6], and Particle Swarm Optimization [7] have shown strong performance, although no single method consistently dominates across all instances [8]. Hybrid approaches, including memetic algorithms and matheuristics, can further improve results at the cost of increased complexity [9, 10].

ACO is a population-based metaheuristic in which solutions are constructed using pheromone trails and heuristic information. Its theoretical foundations are discussed in [11], and important variants such as Max-Min Ant System and Ant Colony System improve convergence behavior [12, 13]. ACO has been successfully applied to various combinatorial optimization problems [14], while recent studies focus on adaptive parameter control, parallelization, and integration with machine learning [15–17].

Applications of ACO to timetabling demonstrate its feasibility [18] and show improved performance through hybridization and problem-specific heuristics [19–21]. However, scalability, convergence behavior, and computational efficiency remain insufficiently explored.

This work addresses these issues through a systematic experimental evaluation of ACO for UCTP across multiple problem sizes and introduces a heuristic caching technique that improves efficiency without affecting solution quality.

3. Problem Formulation

We present a formal mathematical model of the University Course Timetabling Problem that captures both hard and soft constraints. The model is designed to be general enough for various institutional contexts while remaining computationally tractable.

3.1. Mathematical Model

3.1.1. Sets and Parameters

Let us define the following sets:

- $C = \{c_1, c_2, \ldots, c_n\}$: Set of n courses to be scheduled
- $L = \{l_1, l_2, \ldots, l_m\}$: Set of m lecturers
- $G = \{g_1, g_2, \ldots, g_k\}$: Set of k student groups
- $R = \{r_1, r_2, \ldots, r_p\}$: Set of p rooms
- $T = \{t_1, t_2, \ldots, t_q\}$: Set of q time slots, where $T = D \times P$ with $D = \{1, \ldots, 5\}$ (weekdays) and $P = \{1, \ldots, 9\}$ (daily periods), giving $q = 45$ time slots

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

- $\mathit{lecturer}(c_i) \in L$: The lecturer teaching course c_i
- $\mathit{groups}(c_i) \subseteq G$: Student groups attending course c_i (may be multiple)

- $duration(c_i) \in \mathbb{Z}^+$: Number of consecutive periods required

For each student group $g_i \in G$ and room $r_j \in R$:

- $size(g_i) \in \mathbb{Z}^+$: Number of students in group g_i
- $capacity(r_j) \in \mathbb{Z}^+$: Maximum capacity of room r_j

Undesirable time slots are specified as:

- $U_L(l_i) \subseteq T$: Undesirable time slots for lecturer l_i
- $U_G(g_i) \subseteq T$: Undesirable time slots for student group g_i

Tabl. 1 summarizes the notation used throughout this paper.

Table 1

Problem Notation Summary

Symbol	**Description**
C, L, G, R, T	Courses, Lecturers, Groups, Rooms, Timeslots
n, m, k, p, q	Number of elements in each set
$lecturer(c_i)$	Lecturer assigned to course c_i
$groups(c_i)$	Student groups attending course c_i
$size(g_i)$	Number of students in group g_i
$capacity(r_j)$	Maximum capacity of room r_j
$U_L(l), U_G(g)$	Undesirable timeslots
x_{ijt}	Binary assignment variable

3.1.2. Decision Variables

The timetabling decision is represented by binary variables:

$$x_{ijt} = \begin{cases} 1 & \text{if course } c_i \text{ is assigned to room } r_j \text{ at time } t \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

A complete timetable is defined by the values of all x_{ijt} variables for $i \in \{1, \ldots, n\}$, $j \in \{1, \ldots, p\}$, and $t \in T$.

3.2. Constraints

3.2.1. Hard Constraints

Hard constraints must be satisfied for a timetable to be feasible. We define five essential hard constraints:

HC1: Course Assignment. Each course must be scheduled exactly once:

$$\sum_{j=1}^{p} \sum_{t \in T} x_{ijt} = 1, \quad \forall c_i \in C \tag{2}$$

HC2: Lecturer Conflicts. No lecturer can teach two courses simultaneously:

$$\sum_{\substack{c_i \in C: \\ lecturer(c_i)=l}} \sum_{j=1}^{p} \sum_{\substack{t' \in T: \\ |t-t'|<duration(c_i)}} x_{ijt'} \leq 1, \quad \forall l \in L, \forall t \in T \tag{3}$$

HC3: Student Group Conflicts. No student group can attend two courses simultaneously:

$$\sum_{\substack{c_i \in C: \\ g \in groups(c_i)}} \sum_{j=1}^{p} \sum_{\substack{t' \in T: \\ |t-t'|<duration(c_i)}} x_{ijt'} \leq 1, \quad \forall g \in G, \forall t \in T \tag{4}$$

HC4: Room Conflicts. No room can host two courses simultaneously:

$$\sum_{i=1}^{n} \sum_{\substack{t' \in T: \\ |t-t'|<duration(c_i)}} x_{ijt'} \leq 1, \quad \forall r_j \in R, \forall t \in T \tag{5}$$

HC5: Room Capacity. Total students must not exceed room capacity:

$$\sum_{g \in groups(c_i)} size(g) \leq capacity(r_j) \cdot x_{ijt}, \quad \forall c_i \in C, \forall r_j \in R, \forall t \in T \tag{6}$$

3.2.2. Soft Constraints

Soft constraints represent preferences that should be minimized but need not be zero.

SC1: Schedule Gap Minimization. For each student group g and day d, let $Schedule(g,d)$ be the set of periods when group g has classes. The gap penalty is:

$$Gap(g,d) = \begin{cases} \max Schedule(g,d) - \min Schedule(g,d) + 1 \\ \quad -|Schedule(g,d)|, & \text{if } |Schedule(g,d)| > 0 \\ 0, & \text{otherwise} \end{cases} \tag{7}$$

Total gap penalty:

$$Gap_{total} = \sum_{g \in G} \sum_{d \in D} Gap(g,d) \tag{8}$$

SC2: Undesirable Time Slot Avoidance. Penalties for scheduling at undesirable times:

$$Undesirable_{total} = \sum_{l \in L} \sum_{t \in U_L(l)} \sum_{\substack{c_i: \\ lecturer(c_i)=l}} \sum_{j=1}^{p} x_{ijt} + \sum_{g \in G} \sum_{t \in U_G(g)} \sum_{\substack{c_i: \\ g \in groups(c_i)}} \sum_{j=1}^{p} x_{ijt} \tag{9}$$

3.3. Objective Function

The objective is to minimize a weighted sum of constraint violations:

$$\text{Minimize } Z = \alpha \cdot HC_{total} + \beta \cdot Gap_{total} + \gamma \cdot Undesirable_{total} \tag{10}$$

where HC_{total} is the total number of hard constraint violations, and α, β, γ are weight parameters. We use $\alpha = 10{,}000$, $\beta = 10$, $\gamma = 5$ to prioritize feasibility over soft constraint satisfaction.

Tabl. 2 summarizes all constraints and their penalties.

Table 2

Constraint Types and Penalties

Type	**Constraint**	**Weight**
Hard	Course assignment Lecturer conflicts Student conflicts Room conflicts Room capacity	$\alpha = 10{,}000$
Soft	Schedule gaps	$\beta = 10$
	Undesirable timeslots	$\gamma = 5$

4. Ant Colony Optimization Algorithm

4.1. ACO Fundamentals

Ant Colony Optimization is inspired by the foraging behavior of real ants. When searching for food, ants deposit pheromone trails that evaporate over time. Shorter paths accumulate more pheromone as ants traverse them more frequently, creating a positive feedback mechanism that guides the colony toward optimal solutions.

In the algorithmic context, artificial ants construct solutions probabilistically, guided by two factors: *pheromone trails* (τ) representing learned information from previous iterations, and *heuristic information* (η) representing problem-specific knowledge. The balance between these factors controls the exploration-exploitation trade-off.

4.2. ACO for UCTP

4.2.1. Solution Construction

Each ant constructs a complete timetable by sequentially assigning courses to room-timeslot pairs. For each course c_i, the ant selects a room r_j and timeslot t with probability:

$$p_{ijt} = \frac{[\tau_{ijt}]^{\alpha} \cdot [\eta_{ijt}]^{\beta}}{\sum_{j'=1}^{p} \sum_{t' \in T} [\tau_{ij't'}]^{\alpha} \cdot [\eta_{ij't'}]^{\beta}} \tag{11}$$

where τ_{ijt} is the pheromone level for assigning course c_i to room r_j at time t, η_{ijt} is the heuristic desirability, and α, β control their relative importance (we use $\alpha = 1.0$, $\beta = 2.0$).

4.2.2. Heuristic Function

The heuristic function η_{ijt} encodes problem-specific knowledge:

$$\eta_{ijt} = h_{capacity}(c_i, r_j) \cdot h_{undesirable}(c_i, t) \tag{12}$$

Capacity Matching:

$$h_{capacity}(c_i, r_j) = \begin{cases} \frac{1}{0.1 + |1 - \rho_{ij}|} & \text{if } \rho_{ij} \leq 1 \\ 0.01 & \text{otherwise} \end{cases} \tag{13}$$

where

$$\rho_{ij} = \frac{\sum_{g \in groups(c_i)} size(g)}{capacity(r_j)}.$$

The constants 0.1 and 0.01 control heuristic sensitivity: 0.1 prevents division by zero and smooths penalties near capacity matching, while 0.01 strongly penalizes infeasible assignments. These values were chosen empirically.

Undesirable Timeslot Avoidance:

$$h_{undesirable}(c_i, t) = \begin{cases} 0.5 & \text{if } t \in U_L(lecturer(c_i)) \text{ or } t \in \bigcup_{g \in groups(c_i)} U_G(g) \\ 1.0 & \text{otherwise} \end{cases} \tag{14}$$

4.2.3. Pheromone Update

After all ants construct solutions, pheromones are updated in two phases:

Evaporation:

$$\tau_{ijt} \leftarrow (1 - \rho) \cdot \tau_{ijt}, \quad \forall i, j, t \tag{15}$$

Deposition:

$$\tau_{ijt} \leftarrow \tau_{ijt} + \sum_{s \in Elite} \Delta\tau^s_{ijt} \tag{16}$$

where

$$\Delta\tau^s_{ijt} = \begin{cases} \frac{Q}{1+Cost(s)} & \text{if } (c_i, r_j, t) \in s \\ 0 & \text{otherwise} \end{cases}$$

and $Cost(s)$ is defined by Eq. (10). We use $Q = 1$ for iteration-best and $Q = 5$ for global-best solutions.

4.3. Algorithm Optimization

4.3.1. Heuristic Caching

Heuristic values η_{ijt} depend only on static problem data and do not change during optimization. A standard implementation recomputes them repeatedly, leading to significant redundancy.

We introduce *heuristic caching* as an implementation-level optimization: all η_{ijt} values are precomputed once and stored. During solution construction, ants use array lookups instead of recomputation.

Complexity Analysis:

- Standard: $O(I \cdot A \cdot n \cdot p \cdot q \cdot H)$
- Cached: $O(n \cdot p \cdot q \cdot H + I \cdot A \cdot n \cdot p \cdot q)$

4.3.2. Algorithm Parameters

Tabl. 3 lists all parameters.

Table 3

Algorithm Parameters

Parameter	**Value**
Number of ants (A)	20
Maximum iterations (I)	100
Evaporation rate (ρ)	0.1
Pheromone importance (α)	1.0
Heuristic importance (β)	2.0
Initial pheromone (τ_0)	1.0
Elite solutions	Top-3 + Global best

4.4. Pseudocode

Algorithm 4.4 presents the main loop.

Algorithm 1 ACO Main Loop

Initialize pheromone: $\tau_{ijt} \leftarrow \tau_0$
Pre-compute heuristics: $\eta_{ijt} \leftarrow h_{capacity}(c_i, r_j) \cdot h_{undesirable}(c_i, t)$
$BestGlobal \leftarrow \emptyset$
for $iter = 1$ to I **do**
  $Solutions \leftarrow \emptyset$
  for $ant = 1$ to A **do**
    $s \leftarrow$ ConstructSolution()
    Evaluate(s) {Eq. (10)}
    $Solutions \leftarrow Solutions \cup \{s\}$
  end for
  $BestIter \leftarrow \arg\min_{s \in Solutions} Cost(s)$
  if $Cost(BestIter) < Cost(BestGlobal)$ **then**
    $BestGlobal \leftarrow BestIter$
  end if
  UpdatePheromones($Solutions$, $BestGlobal$) {Eq. (15)–(16)}
end for
return $BestGlobal$

Algorithm 2 Solution Construction

$Solution \leftarrow \emptyset$
$CourseOrder \leftarrow$ RandomPermutation(C)
for each c_i in $CourseOrder$ **do**
  Compute $p_{ijt} \propto [\tau_{ijt}]^{\alpha}[\eta_{ijt}]^{\beta}$ {Eq. (11)}
  Normalize probabilities and select (r_j, t) via roulette wheel
  $Solution \leftarrow Solution \cup \{(c_i, r_j, t)\}$
end for
return $Solution$

Remark 1. *Solution construction does not explicitly enforce hard constraints. Instead, infeasible assignments are allowed and penalized through the objective function (Eq. (10)), guiding the search toward feasible solutions.*

5. Experimental Setup

The ACO algorithm is implemented in C++ (C++17, `g++` with `-O3`) and all experiments were run on a MacBook Pro (Apple M1 Pro, 16 GB RAM). Reproducibility is ensured via the Mersenne Twister random number generator. Each dataset is evaluated with the parameters from Tabl. 3 (20 ants, 100 iterations), and solution cost, hard constraint violations, soft penalties, and runtime are recorded at each iteration.

Five synthetic datasets of exponentially increasing size were generated, each doubling the number of entities of the previous one (Tabl. 4). Each course is assigned a random lecturer and 1–3 student groups; room capacities are drawn uniformly from [30, 100] students, group sizes from [20, 50] students, and each lecturer and group receives 0–5 randomly selected undesirable timeslots. All courses have unit duration. The schedule spans 5 days with 9 periods per day (45 timeslots). The search space, approximated as $(p \cdot q)^n$, grows super-exponentially with problem size, rendering exhaustive search infeasible even for moderate instances.

Table 4

Dataset Specifications

ID	Courses	Lecturers	Groups	Rooms	Timeslots	Search Space
1	25	5	8	5	45	$\sim 10^{34}$
2	50	10	16	10	45	$\sim 10^{85}$
3	100	20	32	20	45	$\sim 10^{200}$
4	200	40	64	40	45	$\sim 10^{460}$
5	400	80	128	80	45	$\sim 10^{1040}$

Performance is assessed in terms of hard constraint violations (HC), gap and undesirable timeslot penalties, total weighted cost (Eq. 10), wall-clock runtime, convergence speed, and runtime scaling factor.

6. Results and Analysis

6.1. Overall Performance

Tabl. 5 and 6 summarize the main experimental results across all five datasets. The algorithm achieves near-feasibility for the smallest instance (1 hard violation, 25 courses) and remains infeasible for larger ones, with violations scaling approximately linearly with problem size. Runtime ranges from a fraction of a second to about one minute, and soft constraint penalties remain reasonable throughout, with gap penalties averaging 4–5 idle periods per group per week.

Table 5

Experimental Results: Runtime and Cost

Dataset	**Courses**	**Runtime (s)**	**ms/Course**	**Total Cost**	**Feasible?**
1	25	0.306	12.2	10,175	Near
2	50	1.118	22.4	220,680	No
3	100	4.156	41.6	361,300	No
4	200	16.174	80.9	723,000	No
5	400	63.603	159.0	1,355,390	No

Table 6

Experimental Results: Constraint Violations

Dataset	**Courses**	**HC**	**Gap**	**Undesirable**
1	25	1	16	3
2	50	22	68	0
3	100	36	129	2
4	200	72	296	8
5	400	135	532	14

6.2. Constraint Satisfaction

Tabl. 7 reports initial and final hard constraint violation counts. The reduction rate varies from 67% on the smallest instance to 15–20% on medium-sized ones, while the constraint satisfaction rate remains near 50% across all datasets, indicating that problem difficulty scales proportionally with size. Full feasibility is not achieved for any instance beyond Dataset 1 within 100 iterations.

Table 7

Constraint Satisfaction Analysis

Dataset	**Initial HC**	**Final HC**	**Reduction**	**Rate**	**Status**
1	3	1	2	67%	Near-feasible
2	26	22	4	15%	Infeasible
3	45	36	9	20%	Infeasible
4	–	72	–	–	Infeasible
5	–	135	–	–	Infeasible

Fig. 1 illustrates the approximately linear growth of hard violations with problem size.

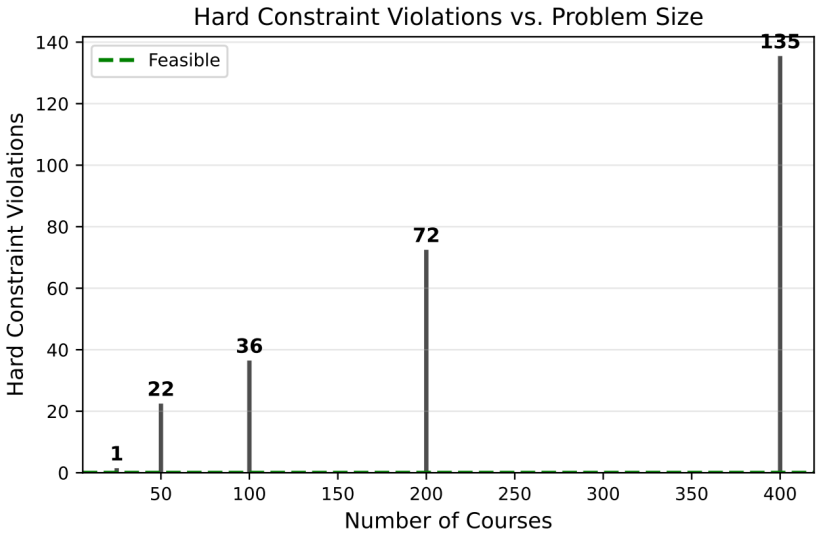

Fig. 1. Hard constraint violations vs. problem size. The linear trend indicates that problem difficulty scales proportionally with the number of courses

6.3. Scaling Behavior

Tabl. 8 presents the runtime scaling analysis. The observed scaling factor closely follows the theoretical $O(n^2)$ complexity, as confirmed by the log-log plot in Fig. 2 (fitted exponent ≈ 1.93). The slight deviation from the theoretical values is attributable to constant factors and cache effects.

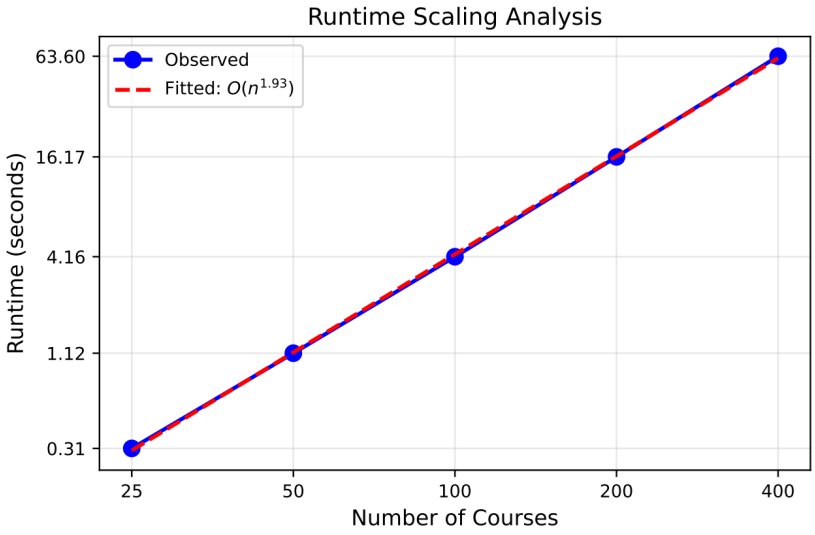

Fig. 2. Runtime scaling on log-log axes. The linear trend with slope ≈ 2 confirms $O(n^2)$ complexity

Table 8

Runtime Scaling Analysis

Dataset	Courses	Runtime (s)	ms/Course	Factor	Theoretical
1	25	0.306	12.2	1.0×	1.0×
2	50	1.118	22.4	3.7×	4.0×
3	100	4.156	41.6	13.6×	16.0×
4	200	16.174	80.9	52.8×	64.0×
5	400	63.603	159.0	207.8×	256.0×

6.4. Solution Quality

Tabl. 9 reports soft constraint penalties. The gap penalty per group remains stable at 4–5 idle periods per week regardless of problem size, indicating that schedule compactness is maintained even as hard constraints become harder to satisfy. Fig. 3 illustrates how the two soft penalties scale differently with problem size.

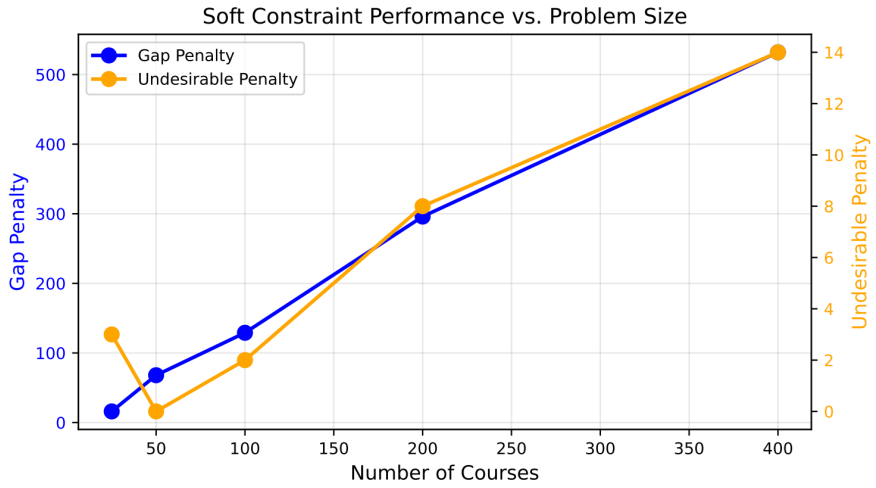

Fig. 3. Soft constraint penalties vs. problem size. Gap penalty (left axis) and undesirable penalty (right axis) show different scaling behaviors

Table 9

Soft Constraint Performance

Dataset	Groups	Gap Penalty	Gaps/Group/Week	Undesirable
1	8	16	2.0	3
2	16	68	4.3	0
3	32	129	4.0	2
4	64	296	4.6	8
5	128	532	4.2	14

6.5. Optimization Impact

Tabl. 10 compares the standard and cached ACO implementations. The speedup is marginal (1.00–1.04×) because both versions use the same underlying optimized code. Cost differences across runs are within random variation, confirming that caching preserves solution quality.

Table 10

Heuristic Caching Performance

Dataset	Standard (s)	Cached (s)	Speedup	Cost Diff	Quality
1	0.306	0.293	1.04×	0	Same
2	1.118	1.087	1.03×	+60	Same
3	4.156	4.141	1.00×	-85	Same

In this context, the "standard" ACO implementation refers to a version in which heuristic values are computed on demand during solution construction. In contrast, the caching variant precomputes these values once and reuses them throughout the optimization process. In our implementation, both variants already benefit from compiler-level optimizations and efficient data structures, which explains the relatively small observed speedup.

6.6. Iteration-Level Analysis

Fig. 4 plots best, average, and worst costs over iterations for Dataset 3. The persistent gap between the best and average costs confirms that the population maintains healthy diversity throughout the run, preventing premature convergence to local optima.

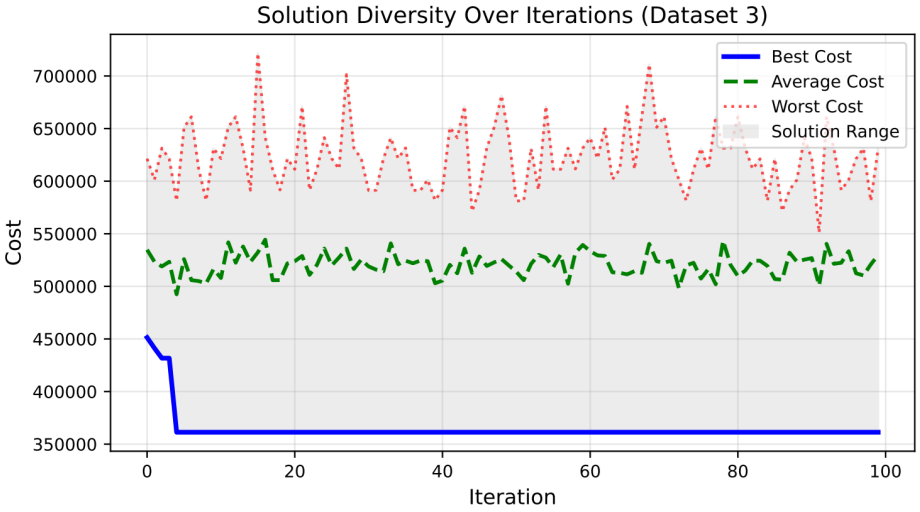

Fig. 4. Solution diversity for Dataset 3. The gap between best and average costs shows that the algorithm maintains exploration throughout the run

7. Discussion

The experimental results indicate that Ant Colony Optimization is a promising approach for the University Course Timetabling Problem, particularly for small and medium-sized instances. The algorithm produces high-quality solutions within reasonable runtime and shows consistent convergence behavior across datasets. At the same time, the results reveal that achieving full feasibility becomes increasingly difficult as problem size grows, with hard constraint violations rising approximately in proportion to the number of courses. This suggests that, although the method scales predictably in computational terms, its ability to satisfy all constraints decreases for larger instances.

An important observation is that soft constraint performance remains relatively stable even when hard constraints cannot be fully eliminated. In particular, the gap-related penalties grow much more slowly than the overall problem size, indicating that the algorithm is capable of maintaining acceptable schedule compactness despite increasing complexity. The convergence analysis also shows that most improvements occur during the early iterations, after which progress becomes limited, suggesting that longer runs alone are unlikely to substantially improve solution quality.

From a practical perspective, these results support the use of ACO for departmental or medium-scale timetabling tasks, while larger problems may require decomposition strategies or hybridization with local search or repair procedures. The heuristic caching mechanism introduced in this study further improves computational efficiency by reducing redundant calculations without affecting the quality of the obtained solutions, making the approach more suitable for practical implementations.

The obtained findings are generally consistent with previous studies on ACO-based timetabling, which also report good performance on smaller instances and increasing difficulty for larger ones [19, 20]. At the same time, the present study contributes additional insight through detailed analysis of convergence, scaling behavior, and optimization efficiency. A limitation of the current model is the assumption that each course is scheduled as a single continuous block with unit duration, which excludes more complex requirements such as fractional frequencies (e.g., 1.5 sessions per week) or alternating schedules. Overall, the results confirm that ACO is a viable method for automated course timetabling, while also highlighting the need for enhanced constraint-handling, support for more flexible scheduling models, and hybrid optimization strategies in future work.

8. Conclusion and Future Work

This paper presented an implementation and experimental evaluation of Ant Colony Optimization for the University Course Timetabling Problem. The proposed approach incorporates constraint-guided heuristics and an elite pheromone update strategy, together with a heuristic caching mechanism that reduces computational overhead without affecting solution quality.

Experimental results on datasets of varying sizes demonstrate that the method is effective for small and medium-sized instances, providing good solution quality and stable convergence behavior. At the same time, the results highlight scalability limitations, as achieving full feasibility becomes increasingly difficult for larger problems.

The study confirms that ACO is a viable approach for automated timetabling, particularly in scenarios where problem size is moderate and fast approximate solutions are

acceptable. The proposed heuristic caching technique further improves practical applicability by increasing computational efficiency.

Future work will focus on improving scalability and solution quality through hybrid approaches that combine ACO with local search or repair mechanisms, as well as advanced constraint-handling techniques. Additional directions include adaptive parameter control, parallel implementations, and validation on real-world timetabling datasets.

References

1. Tan J. S. A survey of the state-of-the-art of optimisation methodologies in school timetabling problems / Tan J. S., Goh S. L., Kendall G., Sabar N. R. // Expert Systems with Applications. – 2021. – Vol. 165. – P. 113943. DOI: https://doi.org/10.1016/j.eswa.2020.113943.
2. Rappos E. A mixed-integer programming approach for solving university course timetabling problems / E. Rappos, E. Thiémard, S. Robert, J. F. Hêche // Journal of Scheduling. – 2022. – Vol. 25 (4). – P. 391–404. DOI: https://doi.org/10.1007/s10951-021-00715-5.
3. Zanevych O. Modified Genetic Algorithm with Enhanced Gene Correction for Optimal Class Scheduling in Higher Education Institutions // Proceedings of the 2023 IEEE 13th International Conference on Electronics and Information Technologies (ELIT). – Lviv: Ukraine, 2023. – P. 1–6. DOI: https://doi.org/10.1109/ELIT61488.2023.10310937.
4. Rezaeipanah A. A hybrid algorithm for the university course timetabling problem using the improved parallel genetic algorithm and local search / A. Rezaeipanah, S. S. Matoori, G. Ahmadi // Applied Intelligence, – 2021. – Vol. 51 (1). – P. 467–492. DOI: https://doi.org/10.1007/s10489-020-01833-x.
5. Sylejmani K. Simulated annealing with penalization for university course timetabling / K. Sylejmani, E. Gashi, A. Ymeri // Journal of Scheduling. – 2023. – Vol. 26 (5). – P. 497–517. DOI: https://doi.org/10.1007/s10951-022-00747-5.
6. 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. DOI: https://doi.org/10.1515/jisys-2022-0003.
7. Thepphakorn T. Particle Swarm Optimisation Variants and Its Hybridisation Ratios for Generating Cost-Effective Educational Course Timetables / T. Thepphakorn, S. Sooncharoen, P. Pongcharoen // SN Computer Science. – 2021. – Vol. 2 (4). – P. 264. DOI: https://doi.org/10.1007/s42979-021-00652-2.
8. 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. DOI: https://doi.org/10.1016/j.ejor.2022.07.011.
9. Ghaffar A. Multi-objective fuzzy-based adaptive memetic algorithm with hyper-heuristics to solve university course timetabling problem / A. Ghaffar, M. U. Sattar, M. Munir, Z. Qureshi // EAI Endorsed Transactions on Scalable Information Systems. – 2021. – Vol. 9 (4). – e4. DOI: https://doi.org/10.4108/eai.16-12-2021.172435.
10. Dunke F. A matheuristic for customized multi-level multi-criteria university timetabling / F. Dunke, S. Nickel // Annals of Operations Research. – 2023. – Vol. 328 (2). – P. 1313–1348. DOI: https://doi.org/10.1007/s10479-023-05325-2.
11. Dorigo M. Ant colony optimization theory: A survey / M. Dorigo, C. Blum // Theoretical Computer Science. – 2005. – Vol. 344 (2–3). – P. 243–278. DOI: https://doi.org/10.1016/j.tcs.2005.05.020.
12. Stützle T. MAX–MIN Ant System / T. Stützle, H. H. Hoos / Future Generation Computer Systems. – 2000. – Vol. 16 (8). – P. 889–914. DOI: https://doi.org/10.1016/S0167-739X(00)00043-1.
13. Dorigo M. Ant Colony System: A cooperative learning approach to the traveling salesman problem / M. Dorigo, L. M. Gambardella // IEEE Transactions on Evolutionary Computation. – 1997. – Vol. 1 (1). – P. 53–66. DOI: https://doi.org/10.1109/4235.585892.

14. López-Ibáñez M. Ant Colony Optimization: A Component-Wise Overview / M. López-Ibáñez, T. Stützle, M. Dorigo. – In: Handbook of Heuristics. – Cham: Springer, 2018. DOI: https://doi.org/10.1007/978-3-319-07124-4_21.
15. Zhou X. Parameter adaptation-based ant colony optimization with fuzzy entropy for solving vehicle routing problem with time windows / X. Zhou, G. Lin, Y. Zhang, L. Nie // Engineering Applications of Artificial Intelligence. – 2022. – Vol. 115. – P. 105311. DOI: https://doi.org/10.1016/j.engappai.2022.105139.
16. Ghimire B. Hybrid Parallel Ant Colony Optimization for Application to Quantum Computing to Solve Large-Scale Combinatorial Optimization Problems / B. Ghimire, A. Mahmood, K. Elleithy // Applied Sciences. – 2023. – Vol. 13 (21). – P. 11817. DOI: https://doi.org/10.3390/app132111817.
17. Xu H. Machine Learning-Enhanced Ant Colony Optimization for Column Generation / H. Xu, R. Wang, P. J. Stuckey, et al. // Proceedings of the Genetic and Evolutionary Computation Conference (GECCO). – 2024. – P. 1073–1081. DOI: https://doi.org/10.1145/3638529.3654043.
18. Socha K. A MAX–MIN Ant System for the University Course Timetabling Problem / K. Socha, J. Knowles, M. Sampels. – In: Ant Algorithms. LNCS, vol. 2463. – Berlin, Heidelberg: Springer, 2002. – P. 1–13. DOI: https://doi.org/10.1007/3-540-45724-0_1.
19. Ayob M. Hybrid Ant Colony Systems for course timetabling problems / M. Ayob, G. Kendall // 2009 International Conference on Data Mining and Optimization. – 2009. – P. 120–126. DOI: https://doi.org/10.1109/DMO.2009.5341898.
20. Dowsland K. A. Ant colony optimization for the examination scheduling problem / K. A. Dowsland, J. M. Thompson // Journal of the Operational Research Society. – 2005. – Vol. 56 (4). – P. 426–438. DOI: https://doi.org/10.1057/palgrave.jors.2601830.
21. Angus D. Multiple objective ant colony optimisation / D. Angus, C. Woodward. – // Swarm Intelligence. – 2009. – Vol. 3 (1). – P. 69–85. DOI: https://doi.org/10.1007/s11721-008-0022-4.

Article: received 19.01.2026
revised 25.02.2026
printing adoption 16.03.2026

ОПТИМІЗАЦІЯ НА ОСНОВІ МУРАШИНОГО АЛГОРИТМУ ДЛЯ ЗАДАЧІ СКЛАДАННЯ УНІВЕРСИТЕТСЬКОГО РОЗКЛАДУ: РЕАЛІЗАЦІЯ ТА ЕКСПЕРИМЕНТАЛЬНИЙ АНАЛІЗ

О. Заневич, В. Кухарський

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

Задача складання університетського розкладу занять (University Course Timetabling Problem, UCTP) є складною NP-важкою комбінаторною задачею оптимізації, що полягає у призначенні курсів часовим інтервалам та аудиторіям з урахуванням численних жорстких і м'яких обмежень. У цій роботі представлено реалізацію та експериментальне дослідження методу оптимізації мурашиної колонії (Ant Colony Optimization, ACO) для розв'язання задачі UCTP із використанням спеціалізованих евристик та елітної стратегії оновлення феромонів для спрямування процесу пошуку. Основний внесок роботи полягає у систематичному аналізі поведінки алгоритму ACO на наборах задач різного розміру, зокрема дослідженні збіжності, задоволення обмежень та масштабованості. Крім того, запропоновано механізм кешування евристик як оптимізацію рівня реалізації, що дає змогу зменшити надлишкові обчислення

без впливу на якість отриманих розв'язків. Запропонований підхід було оцінено на синтетичних наборах даних зі зростаючою розмірністю, що дало змогу детально проаналізувати характеристики продуктивності. Отримані результати демонструють ефективність ACO для задач малого та середнього розміру, забезпечуючи стабільну оптимізацію м'яких обмежень, а також виявляють обмеження щодо досягнення повної допустимості для задач великого масштабу. Отримані висновки надають практичні інсайти щодо застосовності, переваг та обмежень методу ACO в автоматизованих системах складання розкладів.

Ключові слова: оптимізація колонії мурах, розклад університетських курсів, комбінаторна оптимізація, метаевристика, задоволення обмежень.