Journal of Technology Research (JTR)
Volume 4, Issue 2, (2026), pp 093-110, ISSN 3005-639X

iCHAC: A Deterministic Tensor-Based Constraint-Aware Framework for Combinatorial Resource Allocation

Ahmed Jaha [*1], Anwar Alhenshiri[2]

[1]Department of Electronic Engineering - Misrata Industrial Technical College, Misrata, Libya.

[2]Department of Computer Science, Faculty of Information Technology, Misratah University, Misratah, Libya.

*Corresponding author email: a.jaha@it.misuratau.edu.ly; goha_99@yahoo.com

Received: 08-04-2026 | Accepted: 16-08-2026 | Available online: 20-08-2026 | DOI:10.26629/jtr.2026.09

ABSTRACT

Combinatorial resource allocation problems are NP-hard, appearing in a wide range of application domains, including academic timetabling and cloud resource management. Exact optimization methods guarantee optimal solutions but are often impractical due to limited scalability. Metaheuristic techniques are more flexible but frequently exhibit instability, introduce significant overhead, and produce varying outcomes. This paper introduces an improved Constraint-Aware Hierarchical Agglomerative Clustering (iCHAC), a deterministic tensor-based framework for solving multi-constraint resource allocation problems. The framework represents resource conflicts as vectors within a multi-dimensional conflict tensor, allowing simultaneous evaluation of multiple constraints through vectorized tensor operations. A tensor-based vectorized agglomeration process constructs feasible allocations directly, eliminating iterative constraint verification. This design reduces computational cost while preserving feasibility by construction. To lessen the influence of greedy ordering without introducing randomness, the framework also includes an expanded deterministic strategy bank that provides multiple deterministic allocation strategies. The proposed approach is evaluated using a timetabling dataset from the College of Industrial Technology (CIT), Misrata. The results show that iCHAC produces competitive scheduling quality, improves computational efficiency, and delivers fully reproducible results compared to traditional optimization methods and metaheuristics.

Keywords: Combinatorial Resource Allocation, Academic Timetabling, Hierarchical Agglomerative Clustering (HAC), Deterministic Algorithms, Vectorized Agglomeration.

iCHAC: إطار عمل حتمي قائم على الموتر مراعٍ للقيود لتخصيص الموارد التوافقية

أحمد جحا[1]، أنور الهنشيري[2]

[1]قسم علوم الحاسوب، كلية تقنية المعلومات، جامعة مصراتة، مصراتة، ليبيا

[2]قسم هندسة الكترونية، كلية التقنية الصناعية مصراتة، مصراتة، ليبيا

ملخـــــص البحـــــث

تُعدّ مسائل تخصيص الموارد التوافقية من المسائل الصعبة حسابيًا **(NP–hard)**، وتظهر في عدة مجالات مثل إعداد الجداول الدراسية الأكاديمية وإدارة موارد الحوسبة السحابية. توفر طرق التحسين التامة التقليدية حلولا مثلى، لكنها تعاني من ضعف قابلية التوسع، بينما تواجه أساليب الاستدلال الميتاهوريستية مجموعة من التحديات تتعلق بعدم الاستقرار العشوائي، وارتفاع العبء الحسابي، وصعوبة إعادة

الانتاج. تقترح هذه الورقة البحثية إطار عمل **iCHAC**، وهو إطار عمل حتمي قائم على الموتر لحل مسائل تخصيص الموارد متعددة القيود. تقوم الطريقة المقترحة بتمثيل تعارضات الموارد في صورة متجهات ضمن موتر تشابه متعدد الأبعاد، مما يتيح التقييم المتزامن للقيود المتعددة باستخدام عمليات الجبر الخطي المتجهي. تعمل خوارزمية **iCHAC** على استبدال آلية التحقق التكراري من القيود بآلية التجميع المتجهي، مما يقلل من الكلفة الحسابية مع ضمان الجدوى من خلال تصميمها البنائي. إضافة إلى ذلك، تم استخدام بنك استراتيجيات حتمية موسع للتخفيف من تأثير ترتيب العناصر على السلوك الجشع، دون الحاجة إلى الاعتماد على العشوائية. أظهر التقييم التجريبي لمجموعة بيانات الجدولة الزمنية من كلية التقنية الصناعية – مصراتة (**CIT**) أن خوارزمية **iCHAC** تحقق جدوى عالية وكفاءة حسابية محسنة ونتائج قابلة للتكرار بشكل كامل مقارنة بأساليب التحسين التقليدية والاستدلالية.

الكلمات الدالة: تخصيص لبموارد التوافقية، الجدولة الأكاديمية، التجميع الهرمي التراكمي (**HAC**)، الخوارزميات الحتمية، التجميع المتجهي.

1. INTRODUCTION

Efficient resource allocation under complex constraints is an essential problem in computer science, distributed systems, and operations research. Many practical applications, such as university course scheduling, cloud computing resource scheduling, logistics planning, and manufacturing systems, can be considered as combinatorial optimization problems, where limited resources are allocated to a set of entities according to multiple constraints [1].

For example, the university course timetabling problem (UCTP) requires assigning courses, instructors, rooms, and time slots without conflicting with institutional constraints. As the number of entities increases, the number of possible allocations grows exponentially, making such problems computationally impractical and general recognized as NP-hard [2].

1.1 Background and Motivation

Exact methods, including Integer Linear Programming (ILP), Mixed-Integer Linear Programming (MILP), and Constraint Programming (CP), have been used to tackle compatibility resource allocation problems. They yield optimal solutions, but fail to scale when applied to large cases, where the search space explodes exponentially [1]. To overcome these limitations, many researchers have turned to metaheuristics, which seek near-optimal solutions through stochastic exploration of the search space. Nature-inspired algorithms such as Genetic Algorithm (GA), Particle Swarm Optimization (PSO), and Ant Colony Optimization (ACO) have been widely applied to combinatorial optimization tasks in multiple fields [3], [4]. Although these methods commonly achieve high-quality solutions, they generally face challenges related to scalability, careful parameter tuning, stochastic variability, and reproducibility [5].

In recent years, learning-based optimization methods, including Reinforcement Learning (RL), Deep Reinforcement Learning (DRL), Graph Neural Networks (GNNs), and learning-guided hyper-heuristics, have been considered as viable alternatives for large-scale scheduling policies [6]. These approaches typically require a policy training phase before deployment and can achieve competitive scheduling [6]. Their effectiveness often depends on the quality of the learning process, the interaction or training environment, and mechanisms for incorporating or enforcing problem constraints [7].

Motivated by complementary strengths and limitations, this paper proposes iCHAC, a deterministic optimization framework that models multidimensional constraints with a multi-dimensional conflict tensor. Conflicts are encoded as vectors within the tensor, allowing feasibility verification through vectorized tensor operations [8]. This representation groups multiple constraint layers into a tensor, allowing them to be evaluated at the same time through vectorized agglomeration, speeding up the computation [8].

To mitigate ordering sensitivity and greedy behavior, the proposed framework incorporates a bank of deterministic strategies. iCHAC evaluates multiple predefined ordering strategies and selects the best-performing allocation according to the specified evaluation criteria, while preserving reproducibility. Experimental evaluation using the CIT academic timetabling dataset demonstrates that the proposed framework produces feasible solutions with stable and reproducible behavior while achieving high computational efficiency for combinatorial resource allocation problems.

2. RELATED WORK

Combinatorial optimization techniques have been evolved from exact methods through constructive and metaheuristic approaches to learning-based approaches over the years. This section reviews key studies, highlighting gaps that can be filled by the proposed framework.

2.1 Exact Optimization Techniques

In the past, exact optimization methods such as ILP, MILP, and CP have been applied to solve combinatorial allocation problems. These approaches yield mathematically optimal solutions, commonly applied in operations research for task scheduling and resource allocation. While these methods stay useful for small and medium-sized datasets and for theoretical evaluation, they are limited by scalability issues with large real-world datasets, making them improper for dynamic or real-time environments [9].

2.2 Metaheuristic Optimization Approaches

To overcome the scalability limitations of exact optimization methods, many researchers have increasingly adopted metaheuristic algorithms to solve combinatorial optimization problems. Metaheuristics are designed to explore large solution spaces efficiently and obtain near-optimal solutions in acceptable computational time. GAs have been inspired by the process of natural selection. Genetic algorithms are widely used, but finding suitable crossover operators for constrained problems is extremely difficult, repairing steps often consume more time than the search process itself [10]. PSO was inspired by swarming behavior. In general, PSO performs faster than the GA algorithm, but it has difficulty with discrete combinatorial tasks because velocity is a continuous concept that is difficult to relate to discrete courses without loss of information [11]. To simulate foraging, ACO has been developed as an effective routing tool [12]. ACO requires significant computing resources due to the need of simulating pheromone evaporation for a large number of ants, which often results in slow convergence rates in dynamic environments [13].

Despite their success, metaheuristic approaches present some challenges. First, these algorithms rely on stochastic search mechanisms, meaning that different runs of the same algorithm may produce different solutions. This stochastic instability has led to increasing concerns about reproducibility in evolutionary computation experiments [14]. Second, many metaheuristic algorithms require careful parameter tuning, including population size, mutation rates, and convergence thresholds. Improper parameter selection can significantly degrade algorithm performance [15]. In a, metaheuristics often produce invalid candidate solutions throughout the search process. Resolving these errors through repair functions or penalty costs adds significant computational overhead [15].

2.3 Constructive Heuristics and Clustering

Degree of Saturation method, or DSATUR, builds graph coloring solutions step by step. It builds solutions step by step. Despite the speed of this method, it is greedy and myopic, often falling into local optima trap [16]. Recent research indicated that the final solution depends on the initial plan [17]. A seminal study tried to improve it by destroying and rebuilding parts of the solution [18].

The relying of Hierarchical Agglomerative Clustering (HAC) on distance metrics like Euclidean distance makes it unsuitable for

binary constraints, limiting its use in scheduling problems [19]. In a related study, the researchers introduced constraint-based clustering by analyzing how 'must-link' and 'cannot-link' constraints affect the feasibility problem [20]. However, final exam timetabling application, implemented in research [21], was limited to 2D matrices and failed to handle complex multi-resource problems, struggling to represent complex multi-dimensional constraints that arise in real-world scheduling problems.

2.4 Learning-Based Scheduling Approaches

Recently, there has been a growing interest in learning-based optimization techniques for task scheduling and resource allocation. These approaches learn scheduling policies directly from data or from interaction with the environment. These techniques involve Reinforcement Learning (RL), Deep Reinforcement Learning (DRL), Graph Neural Networks (GNNs), Neural Coherence Optimization (NCO), learning-guided hyper-heuristics, and Learning-assisted large Neighborhood Search (LNS) [6][7]. Learning-based edge cloud computing scheduling frameworks have demonstrated competitive practical performance in the combined optimization of response time, power consumption, and operating costs under dynamic workloads [22].

Learning-based schedulers excel in large-scale scheduling problems, generating decisions rapidly after a policy training phase. However, their effectiveness depends on the quality of the learning process, careful hyperparameter tuning, and sufficient computational resources. Hard constraints are often handled indirectly through reward design, constrained action spaces, or masking strategies rather than explicit enforcement. Performance may degrade when deployment conditions differ from those experienced during training. [6] [7].

Instead of replacing deterministic constructivist approaches, learning-based optimization should be viewed as a complementary, not competing, research paradigm. Learning-based methods rely on statistical models, whereas deterministic constructivist approaches tackle each problem head-on, without prior knowledge. The iCHAC framework is a deterministic constructivist approach, built on explicit representation of multi-dimensional constraints, deterministic implementation, feasibility by construction, and reproducibility.

2.5 Research Gap

The literature reviewed above shows important growth in combinatorial resource allocation through exact optimization, metaheuristics, constructive heuristics, and learning-based scheduling approaches. Each research direction offers valuable capabilities but also has complementary limitations. Although exact methods offer optimal solutions with small- to medium- scale datasets, they face time challenges when dealing with large-scale datasets, limiting their scalability; metaheuristics improve scalability at the expense of stochastic behavior and reproducibility; constructive heuristics provide fast deterministic scheduling but generally struggle with complex, multi-dimensional constraints; while learning-based approaches require extensive policy training and generally enforce hard constraints indirectly through reward design or repair mechanisms.

As mentioned above, there is no unified deterministic constructive framework that rely on integrating tensor-based multiple constraint representation, feasibility-by-construction, and deterministic reproducibility within a single resource allocation methodology. To address this gap, the proposed iCHAC has been motivated. where tensor-based constraint modeling, vectorized agglomerative clustering, and ordering strategy bank are integrated to provide a scalable, deterministic, and reproducible solution for combinatorial resource allocation problems.

3. METHODOLOGY: THE UNIFIED iCHAC FRAMWORK

In the proposed methodology, traditional Hierarchical Agglomerative Clustering (HAC) paradigm has been improved by incorporating tensor-based constraints representation into a deterministic constructive framework. This methodology is domain-agnostic, composed of a fixed tensor-based core and adaptable modules, such as the strategy bank and hybrid resource allocation. Therefore, it can be applied to a variety of resource allocation problems, such as course scheduling. As shown in Figure 1, the iCHAC framework consists of three main components: the tensor-based modeling layer, the vectorized agglomeration process, and the deterministic strategy bank. These components interact to construct feasible clusters efficiently while ensuring deterministic reproducibility.

The iCHAC framework uses the tensor as a structure to hold various constraint layers together into a unified model. The proposed framework does not introduce new tensor algebra or continuous tensor operations; instead, it relies on simple vector-based operations over this discrete tensor representation.

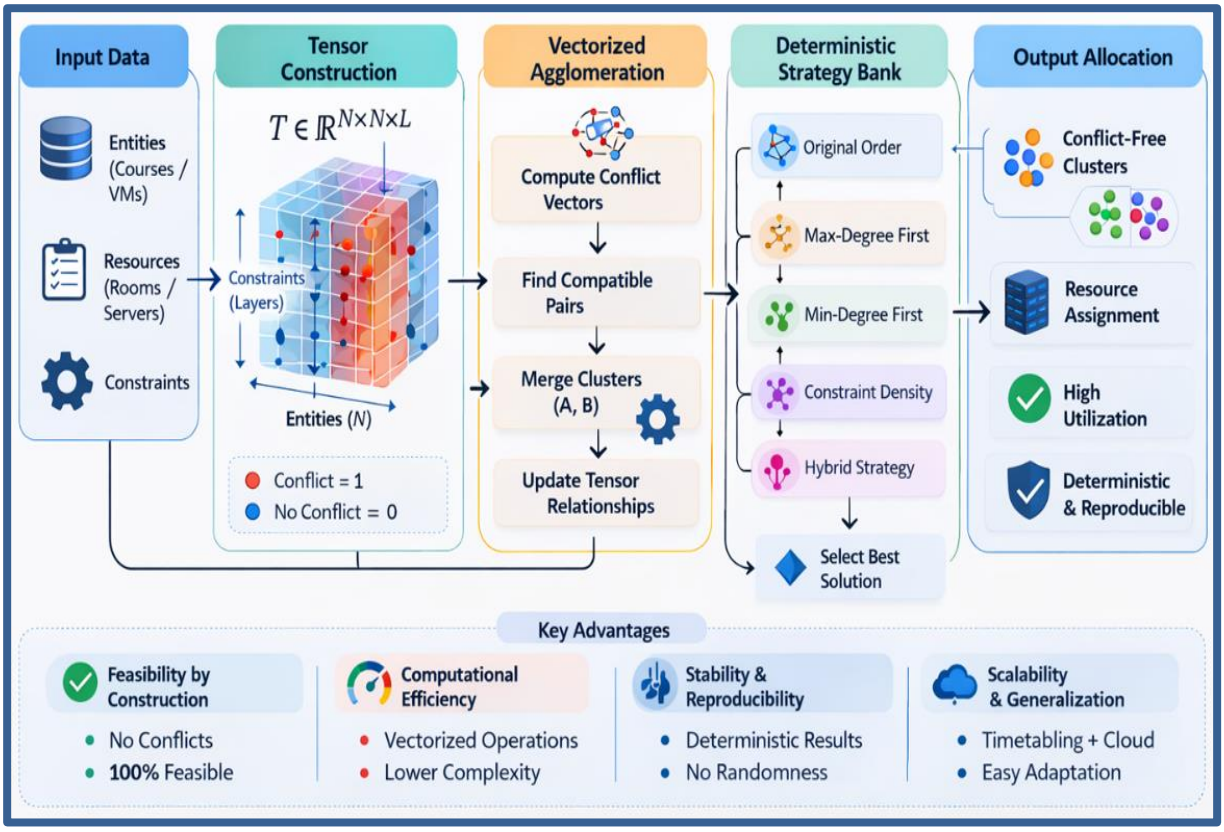

Fig 1. Overview of the iCHAC Framework.

To define an optimization problem formally, the following are assumed:

$E = \{e_1, e_2, \dots, e_N\}$ as a set of N distinct elements, like courses.

$R = \{r_1, r_2, \dots, r_M\}$ as a set of available resources, such as time slots.

Conflict Tensor T has been defined with dimensions $N \times N \times L$, where L represents the number of constraint layers, such as student conflicts.

The problem can be defined as partitioning E into a set of clusters $C = \{C_1, C_2, \dots, C_P\}$, where for each cluster C_x:

$$\sum_{\{e_i,e_j\}\subseteq C_x, i\neq j} \sum_{k=1}^{L} T_{i,j,k} = 0 \qquad (1)$$

Where all elements assigned to the same cluster, such as same time-slot, are conflict-free across all defined constraint layers.

3.1. Core: Multi-Dimensional Conflict Tensor

The conflict tensor is used exclusively as a modeling structure that embodies multiple constraint layers in a unified representation. It is not intended as a contribution to tensor mathematics, but rather as an efficient data organization mechanism for deterministic constructive allocation.

For any partitioning problem with N entities, a tensor of dimensions $N \times N \times L$ will be constructed, where L is the number of constraint layers. For example:

- Layer 1 formulates first conflict, such as student overlap.
- Layer 2 represents second conflict, like instructor overlap.
- Layer 3 deals with hard exclusion, such as lab requirement.

3.2. Vectorized Agglomeration

Standard clustering algorithms iterate through all data points to calculate the Euclidean distance [23].

Instead, iCHAC performs vectorized operations over the conflict tensor to evaluate multiple constraint layers simultaneously and construct feasible clusters through a tensor-based vectorized agglomeration process. Two entities (or sets), A and B, are merged according to the following vector-based conditions:

Merge Condition: Entities A and B will be merged if

$$\sum_{k=1}^{L} T_{ABk} = 0 \qquad (2)$$

Update Mechanism: When merged, the new cluster $C's$ relationship with any entity X will be updated via vector addition:

$$V_{CX} = V_{AX} + V_{BX} \qquad (3)$$

This mechanism takes advantage of the array optimization capabilities of modern processors (SIMD) to significantly reduce practical execution time without changing the asymptotic computational complexity.

3.3. Expanded Deterministic Strategy Bank

One of the major drawbacks of constructive approaches is their sensitivity to initial conditions, such as elements processing order [24]. To fix this, iCHAC sorts the data using at least 10 different strategies before clustering:

1. Conflict Strategies (1-2):

   MostConflicts/LeastConflicts strategy relies on degree centrality, where the most/least conflicting nodes, such as courses, will be processed first.

2. Capacity Strategies (3-4):

   MostCapacity/LeastCapacity strategy copes with capacity arrangements, where the most/least capacity nodes, like number of students per course, will be handled first.

3. Administrative Strategies (5-6):

   OriginalOrder/ReverseOrder strategy deals with the logic of data source, where the original/reverse order in which the data came from the source, such as registration department, will be treated first.

4. Workload Strategies (7-8):

   MostWorkload/LeastWorkload strategy is based on workload hours, where most/least workload, such as instructor's teaching hours, will be dealt with first.

5. Deterministic Stochasticity (9-10):

   FixedSeed1/FixedSeed2 strategy performs random sorts with fixed seeds to be reproducible.

iCHAC is executed multiple times, once for each strategy, and the best result is selected. This approach keeps the process deterministic and reproducible while making greedy construction less sensitive to processing order.

3.4. Hybrid Resource Allocation Layer

This layer differentiates between hard and dynamic resources, considering varying constraints when mapping conflict-free clusters.

- Static Mapping (Hard Constraints): Strictly specific resources are assigned to requester entities, such as allocating a physics lab to a physics course.
- Dynamic Mapping (dynamic constraints): Replaceable resources, such as general classrooms, are allocated using a more suitable mechanism to maximize utilization density.

3.5. Theoretical Justification:

This section offers the theoretical justification for the proposed framework.

3.5.1 Feasibility by Construction

The correctness of the proposed framework relies on the fact that clusters are constructed only when all constraint layers remain conflict-free. This construction mechanism provides a structural feasibility guarantee without requiring repair or backtracking. The following property formalizes this invariant.

Property 1 *(Structural Invariant)*:

The iCHAC algorithm always produces conflict-free clusters with respect to all defined constraint layers.

Justification:

The iCHAC algorithm constructs clusters incrementally through a deterministic agglomerative process. At each step, two entities (or clusters) A and B are merged only if their corresponding constraint vector satisfies the condition:

$$T(A,B,l) = 0, \quad \forall l \in L \qquad (4)$$

where T represents the multi-dimensional conflict tensor and L is the set of constraint layers (e.g., student conflicts, instructor conflicts, room conflicts).

This condition ensures that no constraint violation exists between A and B across any dimension. Since all merges strictly enforce this zero-conflict condition, no infeasible cluster can be formed during the construction process. Furthermore, once a merge is performed, the resulting cluster inherits its constraint relationships through vector addition, which preserves consistency across all constraint layers. Therefore, all constructed clusters remain conflict-free throughout the entire execution. Hence, the algorithm guarantees feasibility by construction, without requiring backtracking or repair mechanisms.

3.5.2 Complexity Derivation $\left(O(N^2)\right)$

1. Tensor Construction: Since it is only configured once initially, the time complexity will be quadratic $O(N^2)$.
2. Clustering Loop: For N elements, the loop is cycled N times, decreasing the number of unscheduled elements by 1 in each step. Then, the time complexity will be $O(N)$.
3. Scan & Update: To find zero in a row of N elements, The time complexity of scanning will be $O(N)$. Once a feasible candidate is identified, the vectors will be updated only one time with a time complexity of $O(N)$.
4. Total: $T(N) = O(N^2) + O(N) \times (O(N) + O(N)) = O(N^2)$.

In practice, execution time benefits from vectorized implementation through constant-factor acceleration without changing the asymptotic complexity. Consequently, iCHAC runs much faster than population-based metaheuristics. Metaheuristics slow down on

large problems since their execution time scales with population size and generations.

3.6 iCHAC Procedure

To provide a clear and structured representation of the proposed framework, the complete procedure of the iCHAC algorithm is summarized in Figure 2. The algorithm uses both of multi-dimensional conflict tensor and set ordering strategies as input. For each strategy, the entities are reordered and iteratively merged based on the zero-conflict condition across all constraint layers. The process continues until no more valid merges can be done. Then, the best solution is selected according to the evaluation criteria, ensuring both feasibility and efficiency.

.

```
Initialize clusters C = {e1,e2,...,eN}
for each strategy s ∈ S do
    Reorder entities
    while unclustered entities remain do
        Initialize a new cluster Ci with the first unclustered entity ei
        for each remaining entity ej do
            if T(Ci,ej,l) = 0, ∀ l ∈ L then
                Merge ej into cluster Ci
                Update tensor relationships by vector addition:
                T(Ci, X) ← T(Ci, X) + T(ej, X), ∀X ∈ C, X ≠ ej
                Remove ej from C
            end if
        end for
    end while
    Evaluate solution
end for
Return Best Solution
```

Fig 2. Pseudo Code of the iCHAC Algorithm.

The algorithm ensures that all generated clusters satisfy the defined constraints, as merging is performed only when no conflicts exist across all tensor dimensions. Multiple deterministic strategies are used to improve the robustness of the solution with reproducibility preserving.

4. AlGORITHMIC WALKTHROUGH (TENSOR EVOLUTION)

To illustrate the structural representation and evolution of the multi-dimensional conflict tensor, a hypothetical example has been traced: Assume that there are 20 students ($s_1 - s_{20}$), 10 Courses ($c_1 - c_{10}$), 4 Instructors ($i_1 - i_4$), 3 classrooms ($r_1 - r_3$), and 6 timeslots ($t_1 - t_4$). In addition, it is known that the capacity of classrooms r_1 and r_2 is 4 students, the capacity of classroom r_3 is 6 students.

Table 1 shows the students and the courses in which they were enrolled. As shown in Figure 3, the students conflict matrix captures pairwise conflict based on shared enrollments, where higher values indicate stronger conflicts.

Table 1. Students with their courses.

Student	Number of courses	First course	Second course	Third course
s1	2	c1	c3	-
s2	2	c2	c4	-
s3	2	c5	c7	-
s4	2	c6	c8	-
s5	2	c9	c10	-
s6	2	c1	c3	-
s7	2	c2	c4	-

s8	2	c5	c7	-
s9	2	c6	c8	-
s10	2	c9	c10	-
s11	2	c2	c4	-
s12	2	c6	c8	-
s13	2	c9	c10	-
s14	2	c2	c8	-
s15	2	c4	c6	-
s16	2	c2	c4	-
s17	2	c6	c8	-
s18	2	c9	c10	-
s19	2	c2	c8	-
s20	2	c4	c6	-

	c1	c2	c3	c4	c5	c6	c7	c8	c9	c10
c1	2	0	2	0	0	0	0	0	0	0
c2	0	6	0	4	0	0	0	2	0	0
c3	2	0	2	0	0	0	0	0	0	0
c4	0	4	0	6	0	2	0	0	0	0
c5	0	0	0	0	2	0	2	0	0	0
c6	0	0	0	2	0	6	0	4	0	0
c7	0	0	0	0	2	0	2	0	0	0
c8	0	2	0	0	0	4	0	6	0	0
c9	0	0	0	0	0	0	0	0	4	4
c10	0	0	0	0	0	0	0	0	4	4

Fig3. Students Conflict matrix.

Table 2 shows the instructors and the courses in which they were assigned. As illustrated in Figure 4, instructor conflicts are encoded similarly, ensuring that courses taught by the same instructor are not scheduled simultaneously.

Table 2. Instructors with their courses.

Instructor	Number of courses	First cours	Second course	Third course
i1	3	c1	c2	c3
i2	3	c4	c5	c6
i3	2	c7	c8	-
i4	2	c9	c10	-

	c1	c2	c3	c4	c5	c6	c7	c8	c9	c10
c1	1	1	1	0	0	0	0	0	0	0
c2	1	1	1	0	0	0	0	0	0	0
c3	1	1	1	0	0	0	0	0	0	0
c4	0	0	0	1	1	1	0	0	0	0
c5	0	0	0	1	1	1	0	0	0	0
c6	0	0	0	1	1	1	0	0	0	0
c7	0	0	0	0	0	0	1	1	0	0
c8	0	0	0	0	0	0	1	1	0	0
c9	0	0	0	0	0	0	0	0	1	1
c10	0	0	0	0	0	0	0	0	1	1

Fig 4. Instructors Conflict matrix.

Table 3 shows the rooms and the courses in which they were assigned. Fig 5 shows room-based conflicts, reflecting shared room constraints and capacity limitations.

Table 3. Rooms with their courses.

Room	Number of courses	First course	Second course	Third course	Fourth course
r1	3	c1	c3	c5	-
r2	3	c7	c9	c10	-
r3	4	c2	c4	c6	c8

	c1	c2	c3	c4	c5	c6	c7	c8	c9	c1
c1	1	0	1	0	1	0	0	0	0	0
c2	0	1	0	1	0	1	0	1	0	0
c3	1	0	1	0	1	0	0	0	0	0
c4	0	1	0	1	0	1	0	1	0	0
c5	1	0	1	0	1	0	0	0	0	0
c6	0	1	0	1	0	1	0	1	0	0
c7	0	0	0	0	0	0	1	0	1	1
c8	0	1	0	1	0	1	0	1	0	0
c9	0	0	0	0	0	0	1	0	1	1
c1	0	0	0	0	0	0	1	0	1	1

Fig 5. Rooms Conflict matrix.

Figure 6 demonstrates the combined 3D tensor, where each entry stores the constraint vectors associated with students, instructors, and rooms, enabling simultaneous evaluation of multiple constraint layers through vectorized operations.

	c1	c2	c3	c4	c5	c6	c7	c8	c9	c10
c1	(2,1,1)	(0,1,0)	(2,1,1)	(0,0,0)	(0,0,1)	(0,0,0)	(0,0,0)	(0,0,0)	(0,0,0)	(0,0,0)
c2	(0,1,0)	(6,1,1)	(0,1,0)	(4,0,1)	(0,0,0)	(0,0,1)	(0,0,0)	(2,0,1)	(0,0,0)	(0,0,0)
c3	(2,1,1)	(0,1,0)	(2,1,1)	(0,0,0)	(0,0,1)	(0,0,0)	(0,0,0)	(0,0,0)	(0,0,0)	(0,0,0)
c4	(0,0,0)	(4,0,1)	(0,0,0)	(6,1,1)	(0,1,0)	(2,1,1)	(0,0,0)	(0,0,1)	(0,0,0)	(0,0,0)
c5	(0,0,1)	(0,0,0)	(0,0,1)	(0,1,0)	(2,1,1)	(0,1,0)	(2,0,0)	(0,0,0)	(0,0,0)	(0,0,0)
c6	(0,0,0)	(0,0,1)	(0,0,0)	(2,1,1)	(0,1,0)	(6,1,1)	(0,0,0)	(4,0,1)	(0,0,0)	(0,0,0)
c7	(0,0,0)	(0,0,0)	(0,0,0)	(0,0,0)	(2,0,0)	(0,0,0)	(2,1,1)	(0,1,0)	(0,0,1)	(0,0,1)
c8	(0,0,0)	(2,0,1)	(0,0,0)	(0,0,1)	(0,0.0)	(4,0,1)	(0,1,0)	(6,1,1)	(0,0,0)	(0,0,0)
c9	(0,0,0)	(0,0,0)	(0,0,0)	(0,0,0)	(0,0.0)	(0,0,0)	(0,0,1)	(0,0,0)	(4,1,1)	(4,1,1)
c10	(0,0,0)	(0,0,0)	(0,0,0)	(0,0,0)	(0,0,0)	(0,0,0)	(0,0,1)	(0,0,0)	(4,1,1)	(4,1,1)

Fig 6. 3D-Conflict Tensor.

	c1,c4	c2	c3	c5	c6	c7	c8	c9	c10
c1,c4	(8,2,2)	(4,1,1)	(2,1,1)	(0,1,1)	(2,1,1)	(0,0,0)	(0,0,1)	(0,0,0)	(0,0,0)
c2	(4,1,1)	(6,1,1)	(0,1,0)	(0,0,0)	(0,0,1)	(0,0,0)	(2,0,1)	(0,0,0)	(0,0,0)
c3	(2,1,1)	(0,1,0)	(2,1,1)	(0,0,1)	(0,0,0)	(0,0,0)	(0,0,0)	(0,0,0)	(0,0,0)
c5	(0,1,1)	(0,0,0)	(0,0,1)	(2,1,1)	(0,1,0)	(2,0,0)	(0,0,0)	(0,0,0)	(0,0,0)
c6	(2,1,1)	(0,0,1)	(0,0,0)	(0,1,0)	(6,1,1)	(0,0,0)	(4,0,1)	(0,0,0)	(0,0,0)
c7	(0,0,0)	(0,0,0)	(0,0,0)	(2,0,0)	(0,0,0)	(2,1,1)	(0,1,0)	(0,0,1)	(0,0,1)
c8	(0,0,1)	(2,0,1)	(0,0,0)	(0,0.0)	(4,0,1)	(0,1,0)	(6,1,1)	(0,0,0)	(0,0,0)
c9	(0,0,0)	(0,0,0)	(0,0,0)	(0,0.0)	(0,0,0)	(0,0,1)	(0,0,0)	(4,1,1)	(4,1,1)
c10	(0,0,0)	(0,0,0)	(0,0,0)	(0,0,0)	(0,0,0)	(0,0,1)	(0,0,0)	(4,1,1)	(4,1,1)

Fig 7. 1st updated 3D-Conflict Tensor.

The steps of merging elements of the 3D Tensor can be summarized as follows:

Step 1 (Initialization): Scanning the constructed 3D tensor reveals the intersection of c_1 and c_4 is $(0,0,0)$.

Step 2: First Agglomeration. Courses c_1 and c_4 are merged. The new constraint vectors are the sum of the parents' vectors:

$$V_{c1,c4} = V_{c1} + V_{c4} \qquad (5)$$

Then, the first updated 3D-conflict tensor is obtained, as shown in Figure 7.

Step 3: Convergence. The process repeats until no further zero-conflict merges are possible. The final result groups the 10 courses into 4 conflict-free groups. As shown in Figure 8, after the first agglomeration step, the tensor is updated through vector addition, reflecting the merged constraints of the combined cluster.

	c1,c4,c7	c2,c5,c9	c3,c6,c10	c8
c1,c4,c7	(10,3,3)	(6,2,3)	(4,2,3)	(0,1,1)
c2,c5,c9	(6,2,3)	(12,3,3)	(4,3,3)	(2,0,1)
c3,c6,c10	(4,2,3)	(4,3,3)	(12,3,3)	(4,0,1)
c8	(0,1,1)	(2,0,1)	(4,0,1)	(6,1,1)

Fig 8. Final updated 3D-Conflict Tensor.

Finally, the ten courses are grouped into four sub-clusters as depicted in Table 4. The courses of each group will be placed at different rooms on the same time slot, and the scheduled courses can be accommodated in three rooms and four time slots without any conflicts between students, instructors, or rooms.

Table 4. Final Conflict-Free Course Schedule.

	Time slot 1	Time slot 2	Time slot 3	Time slot 4
Classroom 1	c1	5c	c3	
Classroom 2	c7	9c	c10	
Classroom 3	c4	2c	c6	c8

5. EXPERMENTAL EVALUATION AND COMPARATIVE ANALYSIS

This section presents a controlled experimental evaluation of the proposed iCHAC algorithm against Exact-Baseline model, GA, PSO, ACO, and DSATUR under a unified computational framework. To ensure a fair and unbiased comparison, all input/output handling, decoding mechanisms, and evaluation metrics were standardized across implementations, with differences confined strictly to the core optimization logic of each algorithm. The results aim to provide a clear and consistent assessment of performance in large-scale timetabling scenarios.

5.1. Experimental Setup and Hardware Environment

Establishing a rigorous and standardized benchmark is critical for evaluating the true operational efficiency of the proposed iCHAC algorithm against established metaheuristics. Therefore, all computational experiments were performed within a unified hardware and software environment. Scheduling algorithms were programmed and run using MATLAB 2019a on a standard Windows 10 workstation equipped with an Intel Core i7-4500U 2.4GHz processor and 12 GB of RAM.

The empirical dataset utilized for these experiments represents the actual, unedited academic load of the CIT during fall 2025/2026. This problem space encompasses 237 distinct class sessions, ranging from theoretical classes to specialized laboratory practicals, that need to be mapped onto 22 physical rooms and managed by a faculty of 48 instructors. To demonstrate the inherent randomness of evolutionary approaches and to ensure statistically significant data, the algorithms GA, PSO, ACO were each implemented for 10 independent runs. The DSATUR and iCHAC algorithms were subjected to the exact same 10-run testing protocol to verify deterministic reproducibility under identical experimental conditions.

In addition, the proposed iCHAC framework employs a dense tensor representation together with SIMD-style vectorized linear algebra operations available in MATLAB. This implementation enables simultaneous evaluation of multiple constraint layers and is the primary source of the observed practical execution-time improvements. No sparse graph representations (e.g., adjacency bit-sets or dynamically maintained zero-conflict candidate sets) were used in the current implementation. Therefore, the reported performance gains result from vectorized computation rather than graph-compression techniques.

5.2. Performance Metrics and Mathematical Formulations

The algorithms were evaluated across six specific dimensions. Each metric captures a vital aspect of the timetabling problem, moving beyond mere constraint satisfaction to assess the actual ergonomic quality of the generated schedules.

1. Computational Time (T): The raw wall-clock duration required for the algorithm to converge upon a final, feasible solution, measured in seconds.
2. Constraint Failures (Φ): This tracks the number of lectures the algorithm failed to schedule without breaching hard constraints, such as room capacity limits,

instructor double-booking, or the misallocation of theoretical courses to restricted labs.

$$\Phi = \sum_{i=1}^{L} x_i \qquad (6)$$

(Where $x_i = 1$ if lecture i cannot be feasibly scheduled, and 0 otherwise; L represents the total number of lectures).

3. Time Compaction / Slots (S): Evaluates the density of the schedule. A lower maximum slot count indicates a tightly packed schedule minimizing idle gaps and improving schedule compactness for both students and faculty.

$$S = \max(s_i) \ \forall i \in \{1,2,\ldots,L\} \qquad (7)$$

(Where s_i denotes the assigned time slot index for lecture i).

4. Spatial Standard Deviation (σ): Assesses the reproducibility and determinism of the algorithm across multiple independent runs. A value of 0 indicates exact determinism, meaning the algorithm yields the exact same spatial placements every time.

$$\sigma = \frac{1}{L}\sum_{i=1}^{L}\sqrt{\frac{1}{M-1}\sum_{j=1}^{M}(R_{i,j} - \bar{R}_i)^2} \qquad (8)$$

(Where M is the number of runs, $R_{i,j}$ is the room index assigned to lecture i during run j, and $\bar{R}_i$ is the mean room index).

5. Capacity Best-Fit (BF): Measures spatial efficiency by calculating the volume of wasted seating. Lower scores reflect a tighter, more optimized fit between enrolled student counts and physical room capacities.

$$BF = \frac{1}{L}\sum_{i=1}^{L}\left|C_{R_i} - N_i\right| \qquad (9)$$

(Where C_{R_i} is the physical capacity of the assigned room, and N_i is the student enrollment).

6. Load Balance (B): Quantifies fairness in the distribution of lectures across the rooms, based on the Gini Index, which measures the degree of inequality in the distribution of use between rooms. A minimal value here means the academic load is equitably distributed across the campus, preventing specific corridors from becoming heavily congested while others remain abandoned.

$$G = \frac{1}{2R^2\bar{U}}\sum_{i=1}^{R}\sum_{j=1}^{R}\left|U_i - U_j\right| \qquad (10)$$

(Where R is the total number of rooms, U_i is the total lectures hosted in room i, and $\bar{U}$ is the average of rooms usage).

5.3. The Combinatorial Explosion of Exact Methods

Before pivoting to heuristic solutions, an attempt was made to solve this specific dataset using Exact-Baseline model to ensure mathematically proven global optimum. The Exact-Baseline code was forcefully killed after running for three continuous hours, and it didn't even generate a single feasible schedule.

This failure perfectly illustrates that the combinatorial explosion inherent to NP-hard scheduling environments. In Exact-Baseline models, decision variables are strictly binary and defined as $X_{l,r,s} \in \{0,1\}$ where lecture l is assigned to room r at time slot s. Given the dataset dimensions (237 lectures × 22 rooms × a conservative bound of 40 time slots), the model generates approximately 208,560 binary decision variables. This results in an exponentially large combinatorial search space, further constrained by thousands of linear constraints required to prevent instructor and room overlaps. At this scale, Exact-Baseline model cannot traverse the search space within polynomial time. Therefore, it becomes computationally impractical, which led to the use of advanced metaheuristics and constructive frameworks.

5.4. Baseline Algorithms Configuration

To ensure a fair comparison, all baseline algorithms (GA, PSO, ACO, and DSATUR) were implemented within the same MATLAB environment, using identical input datasets, decoding procedures, feasibility checking routines, and evaluation metrics. The only component that differs among the algorithms is the optimization engine itself. For the evolutionary algorithms (GA, PSO, and ACO), the hyperparameter settings, as shown in Table 5, were selected according to widely adopted configurations reported in the literature and were validated through preliminary pilot experiments to ensure stable convergence and the consistent generation of feasible schedules. Since DSATUR is a deterministic constructive graph-coloring heuristic rather than an iterative evolutionary algorithm, it does not require population parameters or convergence tuning and was executed using its standard deterministic procedure.

The objective of these preliminary experiments was not to perform exhaustive hyperparameter optimization, but rather to identify parameter settings that reliably produced zero hard-constraint violations while avoiding unnecessarily excessive computational budgets. Accordingly, the stopping criteria for the evolutionary algorithms were defined as fixed computational budgets (population size and maximum generations/iterations), selected after confirming that further increases produced negligible improvements in scheduling quality while substantially increasing execution time. This protocol was applied consistently across all stochastic algorithms to ensure a fair experimental comparison under identical computational conditions.

According to commonly adopted values reported in the literature, the hyperparameters were selected and verified through preliminary pilot runs to ensure stable convergence while maintaining reasonable computational cost. The values of hyperparameters used for all reported experiments are summarized in Table 5.

Table 5. Hyperparameter Settings.

Algorithm	Parameter	Value
GA	Population size	100
	Number of generations	100
	Crossover operator	PMX
	Crossover rate	1.00
	Mutation operator	Swap
	Mutation rate	0.25
	Selection	Tournament
	Elitism	Yes
PSO	Swarm size	50
	Max iterations	100
	Inertia weight	0.70
	c1	1.50
	c2	1.50
	Velocity initialization	zero
ACO	Number of ants	50
	Max iterations	100
	α (pheromone importance)	1.0
	β (heuristic importance)	2.0
	ρ (Evaporation rate)	0.10

All algorithms were implemented within the same MATLAB environment using identical input data, decoding procedures, feasibility verification routines, and evaluation metrics. The only component that differs among the implementations is the optimization engine itself. While iCHAC naturally exploits vectorized tensor operations as an inherent part of its deterministic constructive design, the implementations of GA, PSO, and ACO follow their standard population-based iterative optimization paradigms, whose computational behavior is dominated by repeated candidate generation and evaluation. Consequently, the reported execution times primarily reflect the intrinsic computational characteristics of each optimization paradigm rather than differences in programming environment or software infrastructure.

5.5. Experimental Results

This section presents the experimental evaluation results of the algorithms under study: iCHAC, GA, PSO, ACO, and DSATUR. To obtain a robust and fair comparison, each algorithm was executed ten consecutive times under the same environmental conditions. Multiple runs have been conducted to evaluate the performance and the stability of each approach when dealing with the same constraints.

Tables 6–10 illustrate the detailed metrics for each individual run. These tables provide a more precise view of each algorithm's behavior across repeated runs, particularly regarding solution consistency, computational time, and strict adherence to hard constraints.

For a clearer comparison, the average performance across all executions is summarized in Table 11. This output helps the performance analysis of all algorithms in terms of execution time, number of time slots, spatial stability, capacity utilization, and fairness of load distribution.

Table 6. Results of iCHAC algorithm.

Run #	Time (s)	Fails	Slots	Spatial SD	Best Fit	Balance
1	0.2029	0.0	25	0.0	13.52	0.19
2	0.2000	0.0	25	0.0	13.52	0.19
3	0.2022	0.0	25	0.0	13.52	0.19
4	0.2000	0.0	25	0.0	13.52	0.19
5	0.2001	0.0	25	0.0	13.52	0.19
6	0.2054	0.0	25	0.0	13.52	0.19
7	0.1996	0.0	25	0.0	13.52	0.19
8	0.2051	0.0	25	0.0	13.52	0.19
9	0.2046	0.0	25	0.0	13.52	0.19
10	0.2003	0.0	25	0.0	13.52	0.19
Average	0.2020	0.0	25.0	0.0	13.52	0.19

Table 7. Results of GA algorithm.

Run #	Time (s)	Fails	Slots	Spatial SD	Best Fit	Balance
1	137.50	0.0	36	155.45	13.25	0.19
2	137.65	0.0	33	155.45	13.22	0.19
3	133.48	0.0	34	155.45	13.25	0.19
4	131.58	0.0	34	155.45	13.41	0.19
5	130.93	0.0	34	155.45	13.31	0.19
6	129.97	0.0	32	155.45	13.31	0.19
7	130.70	0.0	34	155.45	13.38	0.19
8	131.04	0.0	31	155.45	13.38	0.19
9	131.68	0.0	33	155.45	13.25	0.19
10	129.99	0.0	35	155.45	13.29	0.19
Average	132.45	0.0	33.6	155.45	13.30	0.19

Table 8. Results of PSO algorithm.

Run #	Time (s)	Fails	Slots	Spatial SD	Best Fit	Balance
1	36.76	0.0	36	148.68	13.35	0.19
2	37.19	0.0	35	148.68	13.19	0.19
3	36.99	0.0	33	148.68	13.16	0.19
4	37.35	0.0	33	148.68	13.22	0.19
5	38.30	0.0	31	148.68	13.41	0.19
6	37.24	0.0	33	148.68	13.29	0.19
7	37.46	0.0	33	148.68	13.35	0.19
8	37.30	0.0	31	148.68	13.29	0.19
9	37.54	0.0	34	148.68	13.19	0.19
10	43.02	0.0	32	148.68	13.31	0.19
Average	37.92	0.0	33.1	148.68	13.28	0.19

Table 9. Results of ACO algorithm.

Run #	Time (s)	Fails	Slots	Spatial SD	Best Fit	Balance
1	6.5107	0.0	31	116.77	12.95	0.20
2	6.5390	0.0	33	116.77	12.95	0.20
3	6.2216	0.0	33	116.77	13.00	0.20
4	6.0975	0.0	33	116.77	12.97	0.20
5	6.1305	0.0	32	116.77	12.95	0.20
6	6.0250	0.0	32	116.77	12.92	0.21
7	6.1878	0.0	34	116.77	12.97	0.20
8	6.0955	0.0	33	116.77	12.95	0.20
9	6.4695	0.0	32	116.77	12.97	0.20
10	6.1467	0.0	33	116.77	13.00	0.20
Average	6.2424	0.0	32.6	116.77	12.96	0.20

Table 10. Results of DSATUR algorithm.

Run #	Time (s)	Fails	Slots	Spatial SD	Best Fit	Balance
1	2.8566	0.0	25	0.0	13.39	0.19
2	3.3095	0.0	25	0.0	13.39	0.19
3	2.8260	0.0	25	0.0	13.39	0.19
4	2.6853	0.0	25	0.0	13.39	0.19
5	2.9407	0.0	25	0.0	13.39	0.19
6	2.7455	0.0	25	0.0	13.39	0.19
7	2.7548	0.0	25	0.0	13.39	0.19
8	2.6076	0.0	25	0.0	13.39	0.19
9	2.5794	0.0	25	0.0	13.39	0.19
10	2.9485	0.0	25	0.0	13.39	0.19
Average	2.8254	0.0	25.0	0.0	13.39	0.19

Table 11. Average Results for All Algorithms.

Algorithm	Time (s)	Fails	Slots	Spatial SD	Best Fit	Balance
iCHAC	0.2020	0.0	25.0	0.0	13.52	0.19
GA	132.45	0.0	33.6	155.45	13.30	0.19
PSO	37.92	0.0	33.1	148.68	13.28	0.19
ACO	6.2424	0.0	32.6	116.77	12.96	0.20
DSATUR	2.8254	0.0	25.0	0.0	13.39	0.19

5.6. Comparative Analysis and Discussion

The results presented in the previous section show that all the algorithms were able to produce feasible schedules without any failures. However, their behavior differed significantly when tested with the remaining performance metrics.

According to the computational time results, iCHAC achieved the lowest execution time, with an average execution time of 0.2020 seconds, which is approximately 656, 188, 31, and 14 times faster than the GA, PSO, ACO, and DSATUR algorithms, respectively. While the reduced runtime of iCHAC algorithm is a significant improvement, it's an ideal choice for real-time scheduling environments that demand rapid rescheduling in response to sudden changes such as room maintenance, instructor absences, or other unpredicted modifications.

Regarding time slots, the iCHAC and DSATUR algorithms was able to condense the schedule into exactly 25 time slots across all attempts, making it approximately 25% more efficient than other algorithms that require an average of 31 to 36 time intervals. This increased the efficiency of the iCHAC and DSATUR algorithms in utilizing resources and reducing waiting times for both lecturers and students.

With respect to spatial standard deviation, the iCHAC and DSATUR algorithms demonstrated full deterministic reproducibility with a perfect standard deviation of 0.00 across all trials, while the other algorithms exhibited significant spatial standard deviation of 155.45, 148.68, and 116.77 for GA, PSO, and ACO algorithms, respectively. iCHAC and DSATUR algorithms produce identical results across all trials, ensuring full reproducibility and providing consistency, verifiability, and predictability for formal academic or administrative scheduling.

In terms of capacity best-fit, DSATUR, GA, PSO, and ACO achieved values of 13.39, 13.30, 13.28, and 12.96, respectively, compared to 13.52 for iCHAC. Although ACO achieved the lowest Capacity Best-Fit value, this improvement was accompanied by the use of substantially more time slots (32.6 versus 25), while not improving room-load balance (Gini = 0.20 versus 0.19 for iCHAC). Since iCHAC explicitly prioritizes schedule compaction, deterministic feasibility, and balanced resource utilization, the slight increase in capacity mismatch represents an acceptable trade-off within the objectives of the proposed framework.

On the basis of load balancing, all algorithms performed similarly, with Gini indices ranging from 0.19 (achieved by iCHAC, GA, PSO, and DSATUR) to 0.20 (achieved by ACO). This indicates that the room allocation mechanism successfully distributes the workload across available resources in a fair and stable manner.

The experimental findings indicate that both iCHAC and DSATUR produce deterministic

constructive schedules. However, iCHAC achieves the same scheduling compactness and reproducibility with approximately fourteen times lower execution time.

In contrast to metaheuristic algorithms, which repeatedly explore candidate solutions through randomized search mechanisms, iCHAC operates within a deterministic constructivist framework that adheres to constraints, providing a compacted, failure-free schedule with fast execution times and full reproducibility.

The above mentioned properties make iCHAC particularly well suited for practical applications in dynamic environments that demand rapid response. Additionally, this framework can potentially be adapted to other fields of resource scheduling and allocation, such as healthcare operations, transportation systems, and distributed service architectures.

6. CONCLUSION AND FUTURE WORK

This paper introduced the iCHAC as an efficient and deterministic framework for solving combinatorial Resource Allocation Problems. By representing multiple constraint layers within a multi-dimensional conflict tensor, the framework enables simultaneous constraint evaluation through vectorized tensor operations.

In contrast to traditional metaheuristic algorithms that rely on stochastic search mechanisms, iCHAC implements a fully constructive and deterministic approach. It directly generates feasible solutions via constraint-aware clustering using vectorized operations. Furthermore, a deterministic strategy bank has been introduced to improve robustness while preserving full reproducibility across different trials.

The experimental results demonstrate that iCHAC framework achieves highly competitive performance across the evaluated metrics while maintaining deterministic reproducibility and computational efficiency. It consistently produced feasible timetables with no hard-constraint violations across all runs. With an average execution time of 0.2020 seconds, iCHAC outperformed the evaluated algorithms by factors ranging from 14× to 656×. Compared with the deterministic graph-coloring heuristic DSATUR, iCHAC achieved the same schedule compactness, identical deterministic reproducibility, and comparable resource utilization while reducing execution time by approximately fourteen times. In addition, it reduced the required number of time slots by up to 25% compared with the evaluated metaheuristic methods. Finally, a zero spatial standard deviation confirmed full deterministic reproducibility by producing identical schedules across repeated runs.

In conclusion, iCHAC represents a practical deterministic constructive alternative for combinatorial resource allocation, combining feasibility-by-construction, reproducibility, and high computational efficiency. The results of the experimental evaluation show that the proposed framework proposed framework achieves the same scheduling quality as DSATUR with a faster execution time.

Despite the above-mentioned advantages, several directions remain for future investigation. A comprehensive benchmark-based evaluation using publicly available datasets, including the International Timetabling Competition (ITC2007), the Udine Benchmark, and additional standard scheduling datasets, is currently underway and will be reported in a subsequent study dedicated to large-scale benchmarking and statistical validation. This ongoing evaluation is expected to provide broader evidence of the framework's generalizability across diverse scheduling scenarios and enable rigorous statistical comparison with state-of-the-art approaches.

Moreover, the proposed framework has been designed as domain-agnostic and can be applied to other combinatorial resource allocation problems such as virtual machine placement in

cloud computing and resource management in fifth-generation communication networks.

Finally, the current implementation adopts a dense tensor representation to maximize SIMD-style vectorized computation and achieve high computational efficiency. Although this design shows high capability for the problem sizes investigated in this study, dense tensor representations naturally require increasing memory as the number of scheduling entities grows. Therefore, future work will investigate sparse tensor and sparse adjacency representations to improve memory scalability while preserving the deterministic constructive scheduling framework. In addition, lightweight optimization mechanisms for handling soft constraints, such as instructor and student preferences, will also be explored.

REFERENCES

[1] Papadimitriou CH, Steiglitz K. Combinatorial optimization: algorithms and complexity. New York: Courier Corporation; 1998.

[2] Garey MR, Johnson DS. Computers and intractability: a guide to the theory of NP-completeness. San Francisco (CA): W.H. Freeman; 1979.

[3] Goldberg DE. Genetic algorithms in search, optimization, and machine learning. Reading (MA): Addison-Wesley; 1989.

[4] Blum C, Roli A. Metaheuristics in combinatorial optimization: overview and conceptual comparison. ACM Comput Surv. 2003;35(3):268-308.

[5] Sörensen K. Metaheuristics—the metaphor exposed. Int Trans Oper Res. 2015;22(1):3-18. doi:10.1111/itor.12001.

[6] Mazyavkina N, Sviridov S, Ivanov S, Burnaev E. Reinforcement learning for combinatorial optimization: a survey. Comput Oper Res. 2021;134:105400.

[7] Cappart Q, Chételat D, Khalil E, Lodi A, Morris C, Veličković P. Combinatorial optimization and reasoning with graph neural networks. J Mach Learn Res. 2023;24:1-61.

[8] Golub GH, Van Loan CF. Matrix computations. 4th ed. Baltimore (MD): Johns Hopkins University Press; 2013.

[9] Kristiansen S, Stidsen TJR. A comprehensive study of educational timetabling—a survey. Lyngby: DTU Management Engineering; 2013. Report No. 8.2013.

[10] Bonyadi MR, Michalewicz Z, Wagner M, Neumann F. Evolutionary computation for multicomponent problems: opportunities and future directions. In: Present practices and future scopes. Cham: Springer; 2019. p. 21-53. doi:10.1007/978-3-030-01641-8_2.

[11] Shami TM, El-Saleh AA, Alswaitti M, Al-Tashi Q, Summakieh MA, Mirjalili S. Particle swarm optimization: a comprehensive survey. IEEE Access. 2022;10:10031-10061. doi:10.1109/ACCESS.2022.3142859.

[12] Dorigo M, Stützle T. Ant colony optimization. Cambridge (MA): MIT Press; 2004.

[13] Rezvanian A, Vahidipour SM, Sadollah A. An overview of ant colony optimization algorithms for dynamic optimization problems. In: Optimization algorithms: classics and recent advances. London: IntechOpen; 2023. doi:10.5772/intechopen.111839.

[14] López-Ibáñez M, Branke J, Paquete L. Reproducibility in evolutionary computation. ACM Trans Evol Learn Optim. 2021;1(4):1-21. doi:10.1145/3466624.

[15] Osaba E, Del Ser J, Nebro AJ, Salcedo-Sanz S, Herrera F. A tutorial on the design, experimentation and application of metaheuristic algorithms to real-world optimization problems. Swarm Evol Comput. 2021;64:100888. doi:10.1016/j.swevo.2021.100888.

[16] Lewis R. A guide to graph colouring: algorithms and applications. Cham: Springer; 2016. doi:10.1007/978-3-319-25730-3.

[17] Kassoumeh A, Kartal Z, Arslan A. The effect of different initial solutions on the metaheuristic algorithms for the single allocation p-hub center and routing problem. PeerJ Comput Sci. 2025;11:e2840. doi:10.7717/peerj-cs.2840.

[18] Ruiz R, Stützle T. A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. Eur J Oper Res. 2007;177(3):2033-2049.

[19] Wani A. Comprehensive analysis of clustering algorithms: exploring limitations and innovative solutions. PeerJ Comput Sci. 2024. doi:10.7717/peerj-cs.2286.

[20] Davidson I, Ravi SS. The complexity of non-hierarchical clustering with instance and cluster level constraints. Data Min Knowl Discov. 2007;14:25-61. doi:10.1007/s10618-006-0053-7.

[21] Ziglam A, Emasallati A, Jaha A. Applying constrained hierarchical agglomerative clustering (CHAC) algorithm to implement final exam timetable. Int Sci Technol J. 2019;19:363-374.

[22] Zhang W, Ou H. Reinforcement learning based multi-objective task scheduling for energy

efficient and cost effective cloud edge computing. Sci Rep. 2025;15:41716. doi:10.1038/s41598-025-25666-1.

[23] Murtagh F, Contreras P. Algorithms for hierarchical clustering: an overview, II. Wiley Interdiscip Rev Data Min Knowl Discov. 2017;7(6):e1219. doi:10.1002/widm.1219.

[24] Xu Y. Joint reasoning for camera and 3D human pose estimation [dissertation]. Pittsburgh (PA): Carnegie Mellon University; 2022.