ChronoSwarm: A Multi-Swarm Particle Swarm Optimization Solution for the University Course Timetabling Problem

Gian Myrl D. Renomeron
CMSC 199.1: Research in Computer Science I
Division of Natural Sciences and Mathematics
University of the Philippines Tacloban College
gdrenomeron@up.edu.ph

Abstract

The University Course Timetabling Problem (UCTP) is considered a difficult academic problem because the task is to assign courses, instructors, and students to time slots and rooms subject to several constraints. The authors of this paper present a novel approach by Multi-Swarm Particle Swarm Optimization (MSPSO) to split up the optimization into sub-swarms to increase the exploration and exploitation abilities. The methodology involves data preprocessing, optimization by MSPSO, and performance evaluation by using ITC2007 benchmark datasets. Results show the efficiency and consistency of MSPSO in producing feasible timetables at different levels of complexity. Additionally, a web-based interface was implemented for practical application, enabling real-time generation and validation of timetables. This research work shows that MSPSO has the potential for scalable and efficient solutions to the challenges of timetabling.

Contents

1 Introduction

Academic institutions face the challenging task of effective resource management in their administrations without compromising the quality of the educational experience in this dynamic world of academia. [1] Among the most complicated logistical problems that universities encounter is the so-called *University Course Timetabling Problem* (UCTP). [2] [3] This problem deals with course-instructor-student assignment to particular time slots and rooms. It includes many constraints such as room capacities, faculty availability, and course prerequisites. [4] For solving these issues and establishing a standardized benchmark, the Second International Timetabling Competition (ITC-2007) proposed Track 3: Curriculum-Based Course Timetabling (CB-CTT). [5] This track defines the UCTP as the assignment of lectures to periods and rooms, subject to hard constraints that must be strictly satisfied (e.g., no room clashes) and soft constraints aimed at optimizing resource utilization and minimizing inconvenience (e.g., spreading lectures evenly). ITC-2007 provided datasets that have since become a critical benchmark for evaluating and comparing optimization algorithms for course timetabling. Many optimization techniques and algorithms have been applied to solve the CB-CTT problems. Some of the developed algorithms include heuristics, such as integer programming [6] [7] and metaheuristics-based algorithms, including local search based and hybrid algorithms. [8] [9] [10], [11], [12], [13], [14], [15] The ideal schedule remains a challenge to prepare, especially for larger institutions, because manual methods lead to conflicts and inefficiencies.

Latest advanced computational approaches emerged to address these problems, and among them *Swarm Intelligence* (SI) is one of the most promising directions. [16] Among the algorithms of this group, there is an attention-grabber - *Particle Swarm Optimization* (PSO), distinguished by adaptability and efficiency in searching large solution spaces. [17] [18]. The algorithm is represented by a set of particles, which are possible solutions; each particle moves in the search space, and its position changes according to individual and collective experiences.[19] [20] Still, despite many strengths, PSO fails at times to solve problems efficiently within UCTP, often failing to find the best solution under tight constraints. [3]

Multi-Swarm Optimization (MSO) is designed to split up the main swarm into smaller, specialized sub-swarms that concurrently operate in exploiting different regions of the solution space simultaneously. [21] [22] It aims to improve the capabilities of PSO by focusing on issues such as search diversity and the global optimizing ability of the algorithm. [23] One of such particular modifications of the above-mentioned technique

is MSPSO: *Multi-Swarm Particle Swarm Optimization*, which dynamically varies the sub-swarms in the standard PSO in both exploration and exploitation for a good performance to provide capabilities of global and local search. [22] [24] [23] To make this approach balanced, mechanisms for exploration and exploitation are integrated appropriately such that different solution domains would be explored in detail while optimizing the potential solutions in a balanced manner. [25]

This paper introduces a new approach to solving the UCTP based on Multi-swarm PSO. MSPSO is specially designed for university course timetabling with diverse constraints, while search methods are efficient in finding optimal timetables. This approach promises to provide schedules that are conflict-free as well as resource-efficient and meet the specific needs of educational institutions. This effort aims to contribute towards scalable and realistic automation and optimization of timetabling, responding to one of the most complex problems in academic administration.

1.1 Background of the Study

University Course Timetabling Problem (UCTP)

The *University Course Timetabling Problem* (UCTP) is defined as a problem of making assignments for courses, instructors, and students to particular time slots and rooms according to numerous constraints, for example room capacities and instructor availability.[1] [3] [17] UCTP is a classic problem of academic optimization, and quality methods are demanded for efficient scheduling.[2] [26] [27]

Heuristic Algorithms

Heuristic Algorithms are heuristic methods to solve optimization problems that should find acceptable solutions within reasonable time. Problem-specific rules or strategies will be employed to search effectively in the solution space to provide a good enough solution rather than guaranteeing optimality in the whole problem space. For example, greedy algorithms, local search, and evolutionary heuristics have been very popular for applications in scheduling problems like UCTP [1] [26].

Swarm Intelligence

Swarm Intelligence refers to bio-inspired algorithms based on the collective behavior of animals such as ants, bees, or birds. [28] [29] It is applied

to optimization problems because it can efficiently and adaptively search very large solution spaces. [3] [28]

Particle Swarm Optimization (PSO)

Particle Swarm Optimization (PSO) is the best known among SI algorithms. [30] It simulates the motions of particles, which denote solutions, in the search space and updates them based on individual and collective experiences. [18] [31] PSO performs very well on many optimization problems, such as UCTP, but still struggles with tricky constraints. [17] [3]

Multi-Swarm Optimization (MSO)

Multi-Swarm Optimization (MSO) is an advanced extension of swarm intelligence algorithms. The basic idea in MSO is to divide the primal swarm into multiple sub-swarms that scan concurrently the different areas of the solution space. A concurrent search makes the algorithm have a better opportunity to discover global optima with effective balancing between exploration and exploitation.[21]. [22]

Multi-swarm Particle Swarm Optimization (MSPSO)

Multi-Swarm Particle Swarm Optimization (MSPSO) is a PSO algorithm developed to overcome the lack of diversity of search and local optima avoidance in the process of complex optimization problems, by splitting the main swarm into a considerable number of subswarms that explore different regions in parallel with the solution space, hence, it promotes diversity and the likelihood of finding the global optima. [23] [24] [32] The difference gives the algorithm an excellent improvement in searching for such solutions for complex problems, such as the UCTP, which balanced exploration and exploitation would provide.

1.2 Related Works

In the field of UCTP, heuristic approaches have been studied for extensive potential towards providing valid and effective solutions with respect to constraints. Metaheuristics aim at establishing quick but effective solutions by approximating optimal schedules rather than executing exhaustive searches, which can sometimes be very computationally expensive.

A two-stage very popular heuristic strategy by Bong Chia Lih et al. [26] gives the first stage to be the grouping of all courses that can be held at the same time, and the second stage to be the assignment of time slots and venues for these groups. Meanwhile, the approach lowers the complexity of this problem, and it has already been successfully applied using real university data and it can handle real-world constraints very efficiently. Similar to Zhang et al., [1] an efficient greedy metaheuristic algorithm handles room and time-slot assignments. Thus, the presented approach, emphasizing the predefined constraints, can meet the required flexibility by these timetables and institutional requirements.

Other metaheuristic methods applied in solving UCTP include Particle Swarm Optimization (PSO). PSO is another optimization technique that is increasingly gaining acceptance in solving UCTP problems. The concept was derived from the social behavior a swarm of birds or of fish follows. This makes it possible to adopt the efficient exploration of large search spaces. Oswald and Durai [3] proposed a hybrid PSO with strategies for improving search applied particularly for UCTP. This was well balanced between exploration and exploitation so the robust timetabling solutions in reference to Oswald et al 2013. Similarly, Chen and Shih [17] created a constriction PSO model integrated with local search strategies to enhance both the gain in generation speed of solutions and the quality of the solution obtained. It has shown the effectiveness of PSO in solving challenging scheduling problems.

Multi-Swarm Particle Swarm Optimization (MSPSO) is an advanced variant of the standard PSO algorithm, aimed at enhancing performance in dynamic and multimodal optimization problems by dividing the population into multiple sub-swarms. Tim Blackwell and Jürgen Branke, in multiswarm PSO, incorporate mechanisms appropriate to dynamic optimization environments. [24] This methodology, while trying to maintain diversity over multiple swarms, focuses on keeping different swarms robustly tracking changing optima. It incorporates exclusion where swarms do not collapse to the same peak, as each pair of swarms within a predefined exclusion radius have weaker swarms reinitialized. It helps different swarms in this way to explore the different regions of the search space. In addition, anti-convergence occurs when all swarms converge; the least-fit swarm is reinitialized to search for emerging or unexplored peaks, ensuring adaptability in changing environments [24]. This algorithm further incorporates quantum particles which are randomly dispersed within a determined distance of the best known position inside the swarm. These main-

tain intraswarm diversity by facilitating the ability to quickly react and follow shifts in peeks. Combining these concepts permits MSPSO to balance global searches and local explorations efficiently with its performance surpassing PSO over dynamic multimodal landscapes [33].

Generalizing the metaheuristic methods embraced here, such as greedy algorithms, adaptive genetic strategies, and PSO-based techniques, indicate that these are feasible for solving UCTP. In fact, these methods are not a global optimality guarantee; however, speed and flexibility in an ideally minimal number of computations make them highly valuable tools for educational institutions, especially for large-scale tasks where a large number of constraints are to be balanced.

2 Statement of the Problem

The UCTP problem is quite complex and key for any institution of learning because it considers assigning courses to specific rooms and time slots while considering constraints such as room capacities, course prerequisites, and faculty courses. [2] To this date, most universities still consider non-optimal manual or heuristic methods, and the inefficiencies include uneven teaching loads and a mismatch between qualifications and requirements in course teaching. [3] [20] While the complexity of the problem is increased, it happens to be highly infeasible to find the best solutions, especially when there are conflicting constraints that need to be fulfilled.

The existing work [3] [18] [17] used PSO-based techniques in solving UCTP but highly developed approaches are required to be found due to the complexity of the problem, such as MSPSO, this improves the original PSO technique by splitting the swarm into many sub-swarms that simultaneously start exploring different regions of solution space. This allows for a higher degree of diversity in search and avoids getting trapped in the local optimum for a better possibility of finding the global optimum.

This paper focuses on the development of a solution for UCTP using MSPSO, that is, to explore the vast solution space with much greater efficiency than before, paying attention to constraints like room capacities, course prerequisites, and scheduling conflicts. In terms of metrics like balance of workloads, course-faculty alignment, and computational efficiency, this proposed approach is supposed to bring about improvements in quality and fairness in the courses scheduled within institutions.

3 Objectives of the Study

3.1 General Objective

To design and develop a Multi-Swarm Particle Swarm Optimization (MSPSO) algorithm to solve the University Course Timetabling Problem (UCTP).

3.2 Specific Objectives

1. To develop and analyze the MSPSO algorithmic usage and efficacy in solving UCTP.
2. To model the UCTP with crucial characteristics like room availability, course prerequisites, and faculty courses within the framework of MSPSO.
3. To perform a state-of-the-art analysis by comparing with other existing approaches to solve the UCTP

4 Scope and Limitation

This study deals with solving the University Course Timetabling Problem (UCTP) with the application of the Multi-Swarm Particle Swarm Optimization algorithm. The use of the ITC2007 Track 3 dataset as a major benchmark is incorporated within the study for assessing the performance of the proposed model, thereby allowing the comparison with previously established methods within the area of study to be maintained with a degree of consistency. This research studies the use of MSPSO for timetabling, optimizing a set of schedules based on different constraints: rooms, course prerequisites, and faculty assignments to provide quality timetables.

Another major aspect is its deployment as a web-based application in which the MSPSO-based solution has been designed and deployed as an interface, hence giving institutions the facility of practically creating timetables in a manner that scheduling conflicts can be minimized, and the available resources maximally utilized.

The only drawback is that the study has been done only on the ITC2007 Track 3 dataset, and it might limit the generality of results for other datasets or real-world problems. Furthermore, the performance of the algorithm is assessed in predefined constraints without taking into account dynamic changes such as last-minute course additions or room reassignments. Still, the work forms a very solid base for the application of MSPSO to timetabling and other optimization problems.

5 Significance of the Study

The importance of this study is that it introduces the multi-swarm structure into the University Course Timetabling Problem (UCTP). MSPSO enhances how the solutions are generated by working together in parallel with several swarms to be able to provide a much stronger approach to possibly avoiding being caught in the trap of getting local optima as compared with standard methods for getting high-quality timetables against institutional constraints.

The study further underscores practicality by implementing the MSPSO solution as a web application. This shall be user-friendly to facilitate easier generation of timetables on the part of institutions, diminish potential scheduling conflicts, and optimize resource use such as rooms and faculty. Through its actual implementation, this approach will ensure it is ready for the real world, ready for administrators and other stakeholders.

It serves as a groundwork for future investigations on multi-swarm optimization for UCTP and other application areas. Based on the above proofs, the ability of MSPSO makes it suitable as a good foundational research work which may help others in doing future research on these areas of problem-solving as presented above.

6 Theoretical and Conceptual Framework

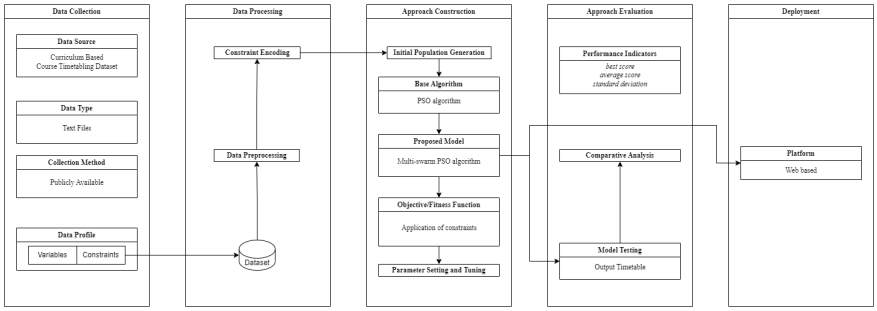

Figure 1: Theoretical and Conceptual Framework

The theoretical and conceptual framework of this study comprises five interconnected components: Data Collection, Data Processing, Approach Construction, Approach Evaluation, and Deployment. Benchmark datasets are collected, cleaned, and prepro-

cessed to ensure reliable and standardized inputs for optimization. The multi-swarm optimization model is then constructed and parameterized to address problem-specific requirements. Its performance is assessed by the degree to which it satisfies objectives and improved through adjustment to maximize effectiveness. The solution is finally deployed using a web-based interface where users can upload problem instances and get real-time optimized outputs.

7 Methodology

7.1 Data Collection

The dataset used in this research is based on the ITC2007 Track 3: Curriculum-Based Course Timetabling (CB-CTT). This dataset has instances where it contains detailed information. This information include a header and four main sections: courses, rooms, curricula, and constraints, with each section containing arrays corresponding to a specific component in the problem and all scalar values summarised in the header. Each dataset instance provides the basic information for testing and validating the proposed approach.

```
Name: ToyExample
Courses: 4
Rooms: 2
Days: 5
Periods_per_day: 4
Curricula: 2
Constraints: 8

COURSES:
SceCosC Ocra 3 3 30
ArcTec Indaco 3 2 42
TecCos Rosa 5 4 40
Geotec Scarlatti 5 4 18

ROOMS:
A 32
B 50

CURRICULA:
Cur1 3 SceCosC ArcTec TecCos
Cur2 2 TecCos Geotec

UNAVAILABILITY_CONSTRAINTS:
TecCos 2 0
TecCos 2 1
TecCos 3 2
TecCos 3 3
ArcTec 4 0
ArcTec 4 1
ArcTec 4 2
ArcTec 4 3

END.
```

Figure 2: Example of CB-CTT Instance

The header describes the important properties of the instance: name of the instance, number of courses, rooms, working days, periods per day, curricula, and constraints. These values determine the scope and size of the timetabling problem and the boundaries of feasible scheduling.

In the **Courses** section, every course is specified to include the assigned teacher, number of lectures required, minimum number of working days that lectures should be spread, and number of students enrolled.

The **Rooms** section defines the rooms available for scheduling along with their seating capacities.

Under **Curricula**, the courses of similar curricula are combined together such that no two classes from a particular curriculum fall at the same time.

The **Unavailability Constraints** section specifies day-period combinations during which certain courses cannot be scheduled. These constraints typically arise from limitations such as teacher unavailability or room restrictions.

This structured dataset gives a stable base for testing and verification of the proposed MSPSO, thus ensuring that all the constraints imposed on it are well described.

7.2 Data Processing

The data processing phase transforms the raw dataset into a structured and optimized format suitable for integration into the MSPSO framework. All problem components and constraints are accurately represented to allow seamless initialization of particles and candidate solution evaluation.

The dataset contains courses, rooms, curricula, and unavailability constraints, which are parsed and normalized in order to achieve consistency. Time-related data, that is, days and periods, are converted to numerical indices, and the constraints are encoded to lead the process of optimization. The parsed data is structured into a particle representation, that is, encoding each candidate solution of the timetabling problem as a list of scheduling entries in the format:

$$(\text{Day}, \text{Period}, \text{Room}, \text{Course}).$$

Constraint Encoding

The constraints, central to the problem, can be categorized into two types, namely hard constraints and soft constraints. Hard constraints are mandatory and must be strictly satisfied, while soft constraints are desirable properties that contribute to the overall quality of the solution. The details and mathematical representations of these constraints are given as follows:

Symbol	Definition
d, p, r, c	Day, Period, Room, and Course respectively.
S	Set of all scheduling entries, where each entry is (d, p, r, c).
U	Set of unavailability constraints, specifying unavailable periods for certain courses.
$\mathrm{Curr}(k)$	Set of courses in curriculum k.
$T(c)$	Teacher assigned to course c.
$\mathrm{students}(c)$	Number of students enrolled in course c.
$\mathrm{capacity}(r)$	Capacity of room r.
$\mathrm{min\_days}(c)$	Minimum number of days required for course c.
R_c	Set of rooms assigned to course c across all scheduled periods.

Table 1: Symbols used for the CB-CTT problem

Hard Constraints

1. **Lecture Assignment**: Each lecture of a course must be assigned to a unique combination of day, period, and room. Overlaps are strictly prohibited.

$$\forall (d, p, r, c_i), (d', p', r', c_j) \in S, \quad i \neq j \Rightarrow (d, p, r) \neq (d', p', r').$$

2. **Room Occupancy**: No two lectures can occupy the same room at the same time.

$$\forall (d, p, r, c_i), (d, p, r, c_j) \in S, \quad i \neq j \Rightarrow c_i \neq c_j.$$

3. **Conflicts**: Courses in the same curriculum or taught by the same teacher must not overlap in time.

$$\forall (d, p, r, c_i), (d, p, r', c_j) \in S, \quad (c_i \in \mathrm{Curr}(k) \vee T(c_i) = T(c_j)) \Rightarrow p_i \neq p_j.$$

   Here, $\mathrm{Curr}(k)$ represents the courses in curriculum k, and $T(c)$ is the teacher assigned to course c.

4. **Availability**: Courses cannot be scheduled during unavailable periods as specified in the dataset.

$$\forall (c, d, p) \in U, \quad (d', p', r, c') \in S, \quad c' = c \Rightarrow (d' \neq d \vee p' \neq p).$$

Soft Constraints

1. **Room Capacity**: Assigning a course to a room with insufficient capacity incurs a penalty proportional to the seating deficit.

$$P_1 = \sum_{(d,p,r,c)\in S} \max(0, \text{students}(c) - \text{capacity}(r)).$$

2. **Minimum Working Days**: Scheduling a course for fewer days than its minimum requirement results in a penalty.

$$P_2 = \sum_{c\in C} 5 \cdot \max(0, \text{min\_days}(c) - |\{d \mid (d,p,r,c) \in S\}|).$$

3. **Room Stability**: Assigning a course to multiple rooms across its schedule incurs a penalty for each additional room used.

$$P_3 = \sum_{c\in C} \max(0, |R_c| - 1), \quad R_c = \{r \mid (d,p,r,c) \in S\}.$$

4. **Curriculum Compactness**: Isolated lectures for a curriculum within a day are penalized, promoting schedule compactness.

$$P_4 = \sum_{k} \sum_{d} \sum_{p\in P_d} \text{is\_isolated}(p),$$

where $\text{is\_isolated}(p) = 2$ if no adjacent lectures exist for curriculum k.

Fitness Function

The overall fitness function integrates the penalties for soft constraints and serves as the objective function for optimization. It is expressed as:

$$\text{Fitness} = Penalty = P_1 + P_2 + P_3 + P_4.$$

Validation and Testing

The parsed data is verified for completeness and consistency during the process. It checks to ensure that all courses, rooms, and curricula have been represented accurately and the constraints are correctly encoded. Thus, the structured and validated data forms a robust base for initializing particles and guiding the optimization process.

The approach of comprehensive data processing will prepare the problem constraints

and components for the MSPSO framework in an accurate manner so that hard constraints are maintained and soft constraint penalties are optimized.

7.3 Approach Construction

The approach begins with initializing particles in the Multi-Swarm Particle Swarm Optimization (MSPSO) algorithm. Each particle represents a potential solution—a timetabling configuration.

Algorithm 1 Multi-Swarm Particle Swarm Optimization Algorithm

```
Initialize swarms, particles, and parameters
for each iteration up to max_iterations do
    Compute convergence and exclusion radius r_rconv r_excl
    // Anti-Convergence
    if all swarms converged and swarm limit not exceeded then
        Add new swarm with random initialization
    end if
    if all swarms converged then
        Randomize worst-performing swarm
    end if
    // Exclusion
    for each pair of swarms in the population do
        Compute distance between best particles of swarms
        if distance < rexcl then
            Reinitialize the worse swarm
        end if
    end for
    // Update and Randomize Particles:
    for each swarm in the population do
        if swarm needs reinitialization then
            Reinitialize particles using quantum reinitialization
        else
            for each particle p in swarm do
                Update position using move and swap
                Evaluate fitness and update personal/local bests
            end for
        end if
    end for
    // Global Best Update
    if better global fitness found then
        Update global best particle
    end if
    if global best fitness ≤ target then
        Break
    end if
end for
```

7.3.1 Initialization Phase

In the initialization phase, initial solutions (particles) and swarms are generated to represent potential timetables for the Curriculu-Based Course Timetabling Problem (CBCTT). A graph heuristic approach [34] is used to ensure the creation of feasible and efficient initial particles. Specifically, the Largest Degree (LD) heuristic is applied to determine the sequence of courses to be scheduled. Courses with the most unavailability constraints, constraints given in the input, are prioritized for assignment.

Once the sequence of courses is computed by the LD heuristic, an assignment of all the lectures that haven't been assigned yet to definite time slots and rooms is undertaken. This step is done in a systematic procedure for it to remain feasible and valid under the constraint. A cycle of procedures can be applied, each of them addressing different potential scenarios:

Random Assignment (Procedure 1):

Each unassigned course is assigned to an available and empty slot that fulfills all constraints such as room capacity or availability. For instance, in this procedure, Lecture 1 is initially unassigned. The system recognizes Room 1 in Slot i as an available and empty slot that meets all conditions such as room capacity or availability. Lecture 1 is then assigned to this room. In the picture, initially, the room, Room 1, is marked as (Empty) with a yellow fill, though it is available for Lecture 1. After the assignment, Room 1 is highlighted in orange; Lecture 1 was assigned to it without conflict.

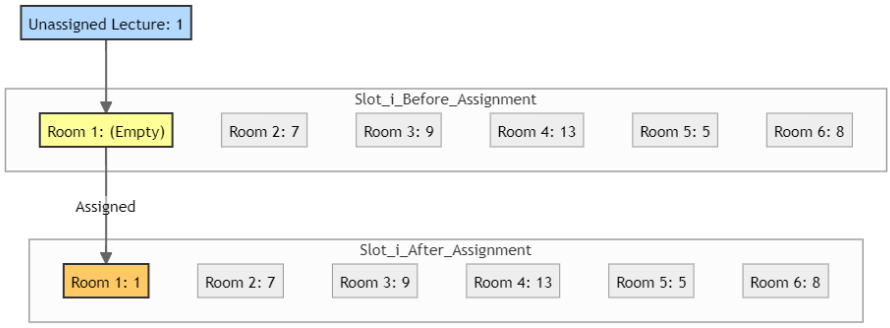

Figure 3: Procedure 1: Random Assignment - Before and After

Conflict-Resolution Assignment (Procedure 2):

If no empty slot is available, the lectures are placed in slots with minor conflicts.

The conflicting lecture in the selected slot is then reallocated to another valid slot to remove conflict and maintain feasibility. For instance, in this procedure, Lecture 1 is unassigned and needs to be scheduled. Room 4 in Slot i is the best slot, but it is occupied by Lecture 13. Lecture 13 is moved to Room 3 in Slot j, which is free and satisfies all the constraints. After resolving the conflict, Lecture 1 is assigned to Room 4 in Slot i. In the example, to illustrate before the assignment, Room 4 is colored red, which shows that it has a clash with Lecture 1. Room 3 in Slot j is colored blue, indicating that it is free to hold Lecture 13. After resolving the conflict, Lecture 1 was properly assigned to Room 4, now showing the color orange.

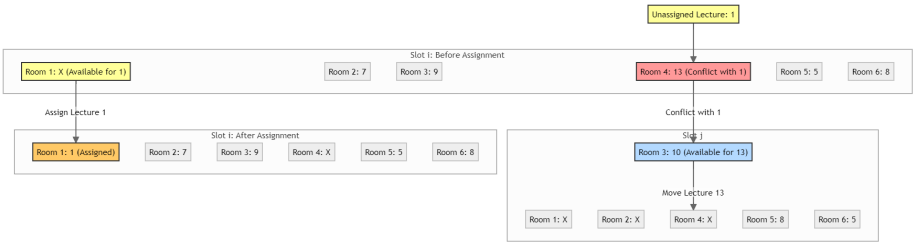

Figure 4: Procedure 2: Conflict Resolution - Before and After

Swappable Slot Assignment (Procedure 3):

If there are no empty slots available, the lecture is assigned to a slot that has minor conflicts, and the conflicting lecture is reassigned. For instance, in this algorithm, Lecture 1 is left unassigned and there is no empty slot in Slot i. However, Room 4 is identified as a swappable slot because it contains Lecture 6, which can be swapped without violating any constraints. The system switches the lectures by moving Lecture 6 to Room 3 in Slot j which is free, so that makes room 4 free in slot i, therefore lecture 1 is assigned in that room. In the example, Room 4 is colored yellow to show it is a good fit for Lecture 1. Room 3 in Slot j is colored blue as available for Lecture 6. After the exchange, Lecture 1 is placed in Room 4 and is colored orange. Lecture 6 is successfully switched to Room 3 in Slot j.

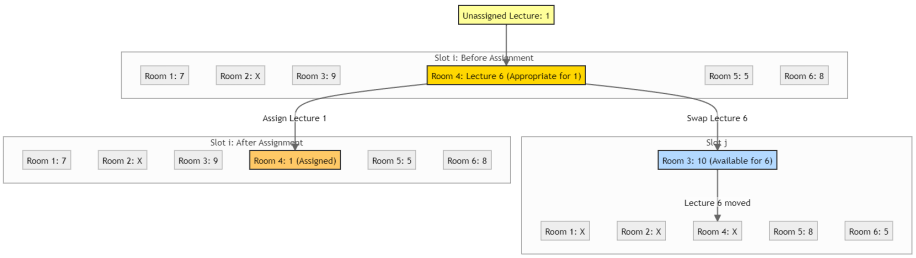

Figure 5: Procedure 3: Swappable Slot Assignment - Before and After

Override Assignment (Procedure 4):

If no others are available a conflicting slot overrides all lectures and is scheduled In this procedure: Lecture 1 is unassigned and all such slots are utilized. The system finds Room 3 in Slot i as the most appropriate slot but finds Lecture 8 in the same room. It overwrites the conflict by moving Lecture 8 to Room 2 in Slot j, which is free and satisfies all constraints. Once the conflict is resolved, it assigns Lecture 1 to Room 3 in Slot i. In the figure, Room 3 is colored red to indicate that it conflicts with Lecture 1, and Room 2 in Slot j is colored blue to indicate that it is available for Lecture 8. After resolution, Lecture 1 is assigned to Room 3, now colored orange, and Lecture 8 is successfully moved to Room 2 in Slot j.

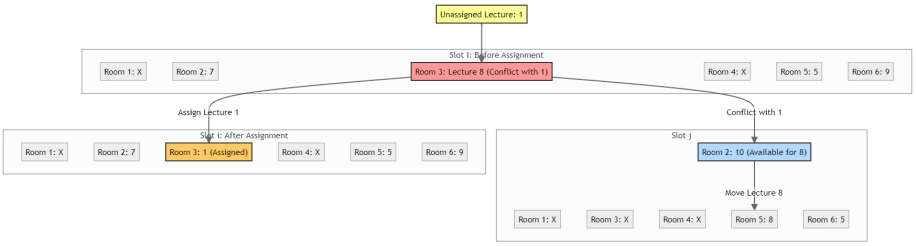

Figure 6: Procedure 4: Override Assignment - Before and After

After the first timetable is created, it is expressed as a set of schedule entries, with each entry containing day, period, room, and course. The initial population of the optimization process is represented by these timetables and is further partitioned into swarms that will be used for further exploration and refinement in the subsequent phases.

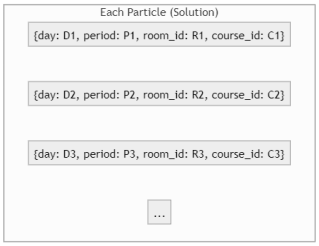

Figure 7: Particle as timetable entries

7.3.2 Optimization Phase

The MSPSO algorithm refines the initial solutions generated by the graph-based heuristic through a series of systematic optimization steps. The optimization loop includes the initialization of critical parameters, anti-convergence checks, exclusion mechanisms, particle updates and reinitialization, and global best updates. Each of these components plays a crucial role in iteratively improving the solutions to minimize penalties and adhere to hard and soft constraints.

Initialization of Parameters Before the optimization begins, several parameters are initialized:

- **Swarm and Particle Structure:** Each swarm contains several particles, where every particle is a candidate solution, and the form it represents is a feasible schedule. The feasible schedules are encoded as lists of assignments, and every assignment describes a solution as *day*, *period*, *room_id*, and *course_id*respectively. Every particle further contains such features as its fitness value, velocity, and best solutions found by it in the process of optimization. This swarm structure enables particles to interact and exchange information for better results.
- **Global and Local Parameters:** Key parameters are defined to guide the behavior of the algorithm. Table 2 provides an overview of the parameters and their corresponding values used in this research.

Parameter	Value	Description
NPARTICLES	5	Number of particles in each swarm, representing candidate solutions.
NSWARMS	1	Number of swarms initialized for parallel exploration of the solution space.
χ (Inertia)	0.729	Controls the influence of a particle's previous velocity in the current update.
c_1 (Personal Best Coefficient)	1.0	Determines the influence of a particle's personal best position on the velocity update.
c_2 (Global Best Coefficient)	1.0	Determines the influence of the global best position on the velocity update.
rexcl	Dynamic (Based on iteration)	Define the exclusion radius so that the swarms start exploring different areas of the solution space. Given by $\texttt{rexcl} = (\texttt{BOUNDS}/\texttt{NSWARMS})^{1/\texttt{NDIM}}$.
rconv	Same as rexcl	It denotes the convergence radius in the anti-convergence. This prevents the swarms from premature convergence mechanism by keeping a minimum distance between the particles in the swarm. Given by $\texttt{rconv} = \texttt{rexcl}$
RCLOUD	0.5	Defines the radius for quantum reinitialization around the global best to introduce diversity.

Table 2: Global and Local Parameters for MSPSO

- *Number of Particles and Swarms (NPARTICLES and NSWARMS):* These represent the number of particles and swarms, which decide the initial population size and the degree of parallel exploration of the optimization algorithm. Each particle is a candidate solution, and the swarms operate in parallel to increase the diversity.
- *Coefficients (χ, c_1, c_2):* The coefficients are critical to the balance of exploration and exploitation:
  - χ controls the inertia or the influence of a particle's previous velocity on its current update.

* c_1 modifies the attraction towards a particle's personal best solution.
* c_2 modifies the attraction toward the local best solution found by the swarm.

– *Radius Parameters (`rexcl`, `RCLOUD`, and `rconv`):* These parameters define key distance measures:
* `rexcl` ensures that swarms remain diverse by preventing convergence to the same solution area. It is dynamically calculated based on the bounds and dimensionality of the search space.

$$\texttt{rexcl} = (\texttt{BOUNDS}/\texttt{NSWARMS})^{1/\texttt{NDIM}}$$

* `RCLOUD` defines the radius for quantum-inspired reinitialization around the global best, helping to reintroduce diversity when swarms converge prematurely.
* `rconv` is used in the anti-convergence mechanism, representing the convergence radius. It ensures that particles maintain a minimum distance from one another, thereby avoiding premature convergence and ensuring better exploration of the solution space.

- **Fitness Evaluation:** The initial fitness of each particle is calculated based on penalties for violations of soft constraints. Such penalties include room capacity overuse, insufficient minimum days for lectures, lack of room stability, and issues of compactness in the curriculum. The total penalty represents the quality of the schedule, and the lower values are indicative of better solutions.

$$\text{Penalty} = P_1 + P_2 + P_3 + P_4$$

where:

– P_1: Penalty for room capacity,
– P_2: Penalty for minimum working days
– P_3: Penalty for room stability
– P_4: Penalty for curriculum compactness

Anti-Convergence Check

The anti-convergence check [24] makes sure that the optimization process avoids stagnation in the exploration of the solution space and maintains diversity. This mechanism checks whether all particles in a swarm have converged too closely and are not able to explore new regions.

The reinitialization radius (`rconv`) is an important parameter in the anti-convergence process. It controls the movements of particles during reinitialization so that diversity is injected into the swarm. It is calculated as:

$$\texttt{rconv} = \frac{\texttt{BOUNDS}}{\texttt{len(population)}^{(1.0/\texttt{NDIM})}}$$

where `BOUNDS` is the size of the solution space, `len(population)` is the current number of swarms, and `NDIM` is the number of problem dimensions. Then, define a threshold to detect convergence as the diameter of the convergence sphere as $2 \times$ `rconv`. If all distances between particles are smaller than $2 \times$ `rconv`, then the swarm is considered converged. Thus, using the diameter as a threshold marks effective identification of tightly clustered particles, hence swarm reinitialization and stagnation prevention.

Pairwise Euclidean distances [35] between particles are computed using normalized dimensions for *day*, *period*, and *room*. The calculation accounts for the total number of days, periods, and rooms to ensure proper scaling. The formula for distance (d) is given as:

$$d = \sqrt{\sum \left(\frac{(\text{day}_{p1} - \text{day}_{p2})^2}{\text{total days}} + \frac{(\text{period}_{p1} - \text{period}_{p2})^2}{\text{total periods}} + \frac{(\text{room}_{p1} - \text{room}_{p2})^2}{\text{total rooms}} \right)}$$

where $p1$ and $p2$ represent particles in the swarm. If all distances between particles are less than the convergence threshold ($2 \times$ `rconv`), the swarm is deemed converged. Using ($2 \times$ `rconv`) as threshold is based on the fact that doubling the radius gives the diameter, which shows the maximum permissible distance between two particles in any swarm. For all distances falling below this threshold, it depicts that the swarms are so tightly clustered to confirm convergence; hence, using this method makes it possible for converged swarms to be further reinitialized towards better coverage of the search space and less stagnation during the process.

For this reason, reinitialization can be triggered every time a swarm is converged heavily and its swarm's best fitness is one of the worst found in the population. The introduction of diversity results from the swapping process, as this reinitialization procedure makes way for particles that can move the swarm around regions it has not seen before, further away from possible local optima.

If all swarms have converged and the total number of swarms is still less than the maximum allowed number (defined as `NSWARMS` + `NEXCESS`), more swarms are initialized. New swarms will begin with randomly assigned particle positions to enhance exploration in the optimization process. This means that the search algorithm will not be degraded, even in a very challenging landscape, and should find optimal solutions.

The success of this mechanism depends highly on the reinitialization radius, `rconv`, which is a key factor in the anti-convergence process. The `rconv` parameter controls the movement of particles in the reinitialization process so that diversity is injected back into the swarm. With a controlled radius for particle reinitialization, `rconv` prevents convergence in a swarm and allows for the exploration of new regions in the solution space. This focused approach makes sure that the algorithm stays dynamic and can always find better solutions by avoiding stagnation and local optima.

Exclusion Mechanism

The exclusion mechanism ensures that different swarms explore distinct regions of the solution space, preventing convergence to similar solutions. This mechanism calculates the pairwise distances between the best particles of every pair of swarms using a normalized Euclidean distance formula. The distance is computed as follows:

$$\text{distance} = \sqrt{\sum \left(\frac{(\text{day}_{p1} - \text{day}_{p2})}{\text{total days}}^2 + \frac{(\text{period}_{p1} - \text{period}_{p2})}{\text{total periods}}^2 + \frac{(\text{room}_{p1} - \text{room}_{p2})}{\text{total rooms}}^2 \right)}$$

where $p1$ and $p2$ are the best particles from two different swarms. If the computed distance is less than the exclusion radius (`rexcl`), the swarm with the worse fitness among the two is reinitialized. The exclusion radius is dynamically determined as:

$$\texttt{rexcl} = \texttt{rconv}$$

Quantum Reinitialization

Quantum reinitialization ensures that diversity is injected back into the swarm when it shows signs of premature convergence. This forms new particle positions using a Gaussian perturbation about the global best particle. This is shown in Algorithm 2.

Algorithm 2 Quantum Reinitialization

```
for each particle p in the swarm do
    if p is the global best then
        Skip reinitialization
    else
        for each assignment in p do
            Generate new_day, new_period, new_room using Gaussian perturbation
            Apply modulo operation to ensure valid ranges
            Temporarily assign new values
            if feasible(p) then
                Keep new values
            else
                Revert to original values
            end if
        end for
        Update fitness and reset personal best
    end if
end for
```

New positions for the particle assignments are calculated using the following formula:

$$new\_position = best\_position + r\_cloud \times N(0,1)$$

where:

- $new\_position$ represents the updated position of a particle.
- $best\_position$ is the position of the global best particle.
- r_cloud is the scaling factor for the perturbation, typically set to 0.5 in this study.
- $N(0,1)$ represents a random value sampled from a guassian distribution with mean 0 and standard deviation 1.

For each assignment in a particle, the updated values are calculated as:

$$new\_day = \text{round}(best\_day + r\_cloud \times N(0,1)) \mod \text{total days}$$

$$new\_period = \text{round}(best\_period + r\_cloud \times N(0,1)) \mod \text{total periods}$$

$$new\_room = \text{round}(best\_room + r\_cloud \times N(0,1)) \mod \text{total rooms}$$

The **Gaussian distribution**, also called the *normal distribution*, is one of the most important ideas in probability theory and statistics. It is applied very often because it occurs naturally in many measurement situations, errors, and random variables [36]. The Gaussian distribution is described by its probability density function (PDF):

$$f(x|\mu,\sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

where:

- x is the random variable.
- μ is the mean (or expectation), which determines the center of the distribution.
- σ is the standard deviation, indicating the spread or dispersion.
- σ^2 is the variance, which is the square of the standard deviation.

Each particle's new position is evaluated for feasibility. The values are preserved if the new position satisfies all hard constraints; otherwise, it will go back to its original state.

This quantum-inspired approach allows the particles to move away from the local optima while using information from the best global solution. The Gaussian perturbation introduces randomness that enables the swarm to explore unexplored areas of the solution space. The balance between exploration and exploitation guarantees the robustness of the optimization process and makes it efficient in discovering better solutions.

Particle Update

The particle update process continuously modifies the positions and velocities of the particles to make better exploration in the solution space. This procedure is based on the Particle Swarm Optimization algorithm PSO [19], where a particle is determined by its personal best position p_{best} and the local best position found by its neighbors, l_{best}.

The velocity of each particle, which determines its movement in the solution space, is calculated using the formula:

$$v = \chi \left[c_1 r_1 (p_{best} - x) + c_2 r_2 (g_{best} - x) \right]$$

where:

- x: The current position of the particle.
- χ: The constriction factor to ensure convergence
- c_1, c_2: Acceleration coefficients representing the cognitive and social components, respectively
- r_1, r_2: Random values sampled from a uniform distribution in $[0, 1]$, introducing stochastic behavior.
- p_{best}: The particle's personal best position.
- l_{best}: The local best position in the swarm.

Using the calculated velocity, the particle's position is updated as:

$$x_{\text{new}} = x_{\text{current}} + v$$

For attributes such as *day*, *period*, and *room*, the velocity and position updates are calculated for each attribute separately. [17] For example:

$$\begin{aligned} v_{\text{day}} &= \chi \left[c_1 r_1 (p_{\text{best,day}} - x_{\text{day}}) + c_2 r_2 (g_{\text{best,day}} - x_{\text{day}}) \right] \\ v_{\text{period}} &= \chi \left[c_1 r_1 (p_{\text{best,period}} - x_{\text{period}}) + c_2 r_2 (g_{\text{best,period}} - x_{\text{period}}) \right] \\ v_{\text{room}} &= \chi \left[c_1 r_1 (p_{\text{best,room}} - x_{\text{room}}) + c_2 r_2 (g_{\text{best,room}} - x_{\text{room}}) \right] \end{aligned}$$

Velocity for each attribute is applied to its current position using the position formula. Each position update is modified by a modulo operation to satisfy such cyclic constraints. This makes certain that any adjustment will remain within prescribed limits and will rotate in a smooth way. For instance, in a schedule where the maximum number of days is five, a particle that goes beyond the fifth day will revolve around and come back to the first day. The same reasoning applies when periods and room numbers are assigned to move the bounds of the update while the solution remains within the given limits. For example:

$$\text{new\_day} = (\text{current\_day} + \text{round}(v_{\text{day}})) \mod \text{total days}$$
$$\text{new\_period} = (\text{current\_period} + \text{round}(v_{\text{period}})) \mod \text{total periods}$$
$$\text{new\_room} = (\text{current\_room} + \text{round}(v_{\text{room}})) \mod \text{total rooms}$$

Move and Swap Operations The move and swap operations are important in the update process of the particle to explore different possibilities for scheduling and enhance assignment optimization. These operations aim at adjusting the *day*, *period*, and *room* attributes of courses within a particle. There are two major operations [37] used to explore alternative solutions:

- **Move (1-0):** The move operation adjusts the attributes of a single course by modifying one or more of its properties. This operation enables the algorithm to explore alternative configurations by relocating the course within the schedule.
- **Swap (1-1):** The swap operation exchanges attributes between two courses, allowing the algorithm to perform a balanced search of the solution space by swapping their respective *day*, *period*, and *room* properties.

Both the move and swap operations use the following types of updates:

- **Room, Day, Period:** All three attributes (*room*, *day*, and *period*) are modified.
- **Day, Period:** Only the *day* and *period* attributes are updated, leaving the *room* unchanged.
- **Period:** Only the *period* attribute is updated, keeping *day* and *room* constant.
- **Day:** Only the *day* attribute is updated, keeping *period* and *room* constant.
- **Room, Period:** The *room* and *period* attributes are modified, while the *day* remains unchanged.
- **Room, Day:** The *room* and *day* attributes are modified, while the *period* remains unchanged.
- **Room:** Only the *room* attribute is updated, keeping *day* and *period* constant.

In the particle update, all defined moves and swap operations are applied in sequence. For each operation, a different course assignment is chosen, and this ensures that the particle performs an exhaustive search of its attributes. This way, the algorithm makes localized adjustments (through moves) and structural changes (through swaps) in the

configuration of the particle. The combination of these operations ensures that the position of the particle is well refined at each update cycle, thus enhancing its ability to explore the solution space effectively.

After updating the position, with either move or swap, a feasibility check is performed to ensure no violations of hard constraints (e.g., no room or time conflicts). If the new position is infeasible or leads to a worse fitness, the update is reverted, retaining the previous position.

Once the update is completed, the particle is evaluated evaluated using the objective/fitness function. The fitness evaluation calculates the penalty score based on predefined criteria. In this case, it is calculated by the total soft constraints violation.

Global Best Update

The global best update is the most critical part of the optimization process since it provides the assurance that the algorithm runs and converges to an optimal solution. Upon completion of each iteration, the algorithm looks at the fitness values of every particle for all swarms and selects the best-performing among them. This particle, if it has a better fitness than the best found so far at the global level, is used to update the global best solution. The global best particle is the fittest configuration that has been discovered thus far and serves as a guiding reference for the swarm in subsequent iterations.

It starts with a **fitness comparison**. The best fitness value of the current global best particle is compared with the best fitness values from all particles in all swarms. If a new particle's fitness value is better, this one would replace the global best solution for updating the algorithm to ensure maintaining its best-known solution and guiding further exploration and exploitation of the solution space.

In addition to maintaining the global best, the algorithm is also checking its **termination criterion**. If global best fitness value reaches a specific target value predefined (for instance, zero fitness value, then the solution will be perfect) the optimization loop will terminate pre-emptively. This mechanism has prevented unnecessary runs once an optimum solution has already been found. If the global best fitness does not attain the target value, the optimization loop continues to run until the maximum number of iterations is achieved. This would ensure that the algorithm has had enough chances to explore the solution space and further refine the global best solution.

Output

The optimization phase ends by emitting the best solution found during the run. It is the one particle with the smallest fitness value across all swarms. Finally, the generated schedule is returned in a special format. In this format, each line of the output file describes an assignment of one course, giving its `course_id`, `room_id`, `day`, and `period`. The following is just a part of the output:

```
C1 R1 0 0
C1 R1 0 2
C1 R1 1 0
C2 R2 0 0
C2 R2 0 1
```

In addition to the schedule, the output file contains the fitness value of the best particle along with the total penalty score. The output file contains additional details such as computation time, number of iterations done, and the particle that gives the global best solution. It is saved in a `.out` file with all these values that will be of use for further study.

7.4 Approach Evaluation

The effectiveness of the MSPSO algorithm is evaluated using the datasets from ITC2007, a widely recognized benchmark for timetable optimization. The evaluation process ensures that the generated schedules meet the required standards of feasibility and quality, as detailed below:

Validation Tool

The generated timetables are validated using a dedicated validation tool provided by ITC2007. The inputs to the tool are the dataset (`name_of_input_comp.ctt`) and the output file (`name_of_output_comp.out`) produced by the MSPSO algorithm. It checks for violations of both hard and soft constraints and computes the total cost incurred due to violation. The violations of hard constraints, such as room or time conflicts, are strictly penalized, while violations of soft constraints, such as room stability or curriculum compactness, contribute to the overall penalty score.

```
[S(2)] Course c0066 uses 3 different rooms
[S(2)] Course c0067 uses 3 different rooms
[S(2)] Course c0068 uses 3 different rooms
[S(2)] Course c0069 uses 3 different rooms
[S(1)] Course c0070 uses 2 different rooms
[S(1)] Course c0071 uses 2 different rooms
[S(2)] Course c0072 uses 3 different rooms

Violations of Lectures (hard) : 0
Violations of Conflicts (hard) : 0
Violations of Availability (hard) : 0
Violations of RoomOccupation (hard) : 0
Cost of RoomCapacity (soft) : 122
Cost of MinWorkingDays (soft) : 60
Cost of CurriculumCompactness (soft) : 90
Cost of RoomStability (soft) : 42

Summary: Total Cost = 314
```

Figure 8: Validation output showing the detailed list of constraint violations and the total cost.

Evaluation Metrics

The output of the validation tool is divided into two parts. The first one provides a very detailed list of any violations encountered that are categorized between hard and soft constraints. Then, the penalty scores associated with these violations, as well as the total cost, are calculated in the summary part. The fitness function used within the MSPSO algorithm is tailored to minimize total cost by meeting hard constraints at first and wherever possible reducing the penalties of the soft constraints.

To highlight the effectiveness of the MSPSO algorithm, its performance is benchmarked against existing methods from the literature. Metrics such as the number of hard constraint violations, the total cost, and the quality of the generated timetables are used for comparison.

7.5 Deployment

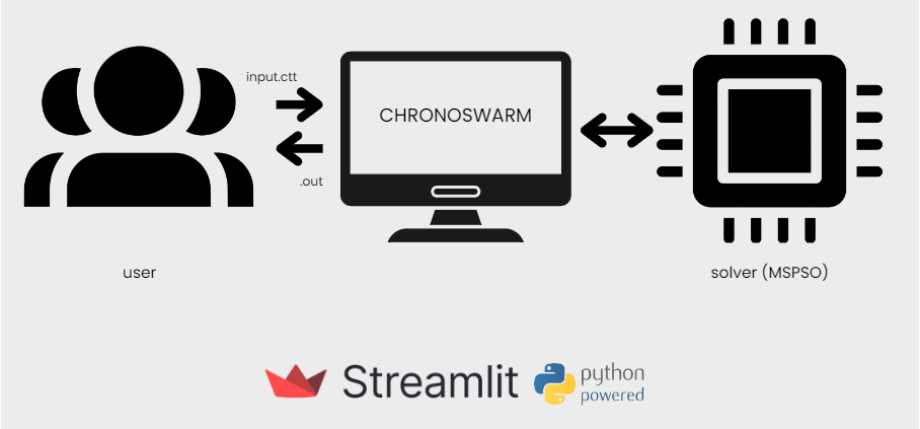

Figure 9: Overview of the deployment framework. The user interacts with the system through the ChronoSwarm interface, which communicates with the MSPSO solver to generate and validate optimized schedules.

The solution is developed as a web application in the Streamlit Python framework. It presents an interface where users can upload dataset files to run the MSPSO algorithm and automatically generate optimized timetables. The application would output the timetable in a user-friendly format and give an indication of the quality of the solution returned, including constraint violations and overall cost.

8 Initial Results

8.1 Dataset

The ITC2007 benchmark datasets are widely used in the field of timetable optimization for evaluating the MSPSO algorithm; three specific instances were chosen to represent different categories

- **Early Instance:** `COMP01`
- **Late Instance:** `COMP08`
- **Hidden Instance:** `COMP15`

8.2 Experiment Configuration

The experiments were conducted with the following configuration:

- Each instance was executed for **five independent runs** to ensure the reliability of the results.
- The algorithm was configured with specific parameters as follows:
  - **Constriction Factor** (χ): 0.729
  - **Cognitive Coefficient** (c_1): 1.0
  - **Social Coefficient** (c_2): 1.0
  - **Population Size**: 5 particles per swarm
  - **Number of Swarms**: 1
  - **Maximum Number of Excess Swarms**: 3
  - **Quantum Cloud Radius** (R_{cloud}): 0.5
- Each run terminated after a maximum of 500 iterations or upon meeting the optimal fitness value (i.e., zero violations).
- Results were compared with other swarm-based approaches such as Artificial Bee Colony (ABC) [38], Improved Artificial Bee Colony (IABC) [8], Ant Colony Optimization (ACO) [39], and Harmony Search Algorithm (HSA) [40].

8.3 Results

The results are summarized in Table 3, which reports the **best**, **worst**, and **average** penalties for the MSPSO algorithm over the chosen instances. For completeness, a comparison with other methods is also reported to demonstrate the strength of the MSPSO algorithm.

Table 3: Performance of MSPSO on ITC2007 Instances (after 5 runs)

Instance	Best	Worst	Average
COMP01	216	297	263.8
COMP08	657	765	695.8
COMP15	648	870	781.4

Comparison with Other Approaches

Table 4 compares the performance of MSPSO with other swarm-based algorithms on the same instances.

Table 4: Comparison of MSPSO with Other Swarm-Based Approaches

Instance	MSPSO	ABC	IABC	ACO	HSA
COMP01	216	-	24	10	323
COMP08	657	218	173	112	645
COMP15	648	284	238	189	665

8.4 Observation

Observations from the preliminary results show the performance of the MSPSO algorithm over three different instances, namely COMP01, COMP08, and COMP15, which represent early, late, and hidden categories, respectively.

For the early dataset represented by the instance of COMP01, MSPSO was able to show competitive performance with a **best penalty of 216**. Still, it was surpassed by other methods, including ACO with a **best penalty of 10** and IABC with a **best penalty of 24**, thus indicating the need for further improvement in this category. Nonetheless, MSPSO's performance remained relatively stable throughout all runs since the difference between the **worst penalty of 297** and the **average penalty of 263.8** is minimal.

In the COMP08 instance, classified as a late dataset, the MSPSO reached a **best penalty of 657**, though not as good as the penalties found by ACO, with **112**, and IABC, with a penalty of **173**. This instance had a further gap between **best** and **worst penalties of 765**, and thus more variation in the different solutions found among different runs. This may indicate a sensitivity of the algorithm to the complex constraints within late instances.

The COMP15 instance was the most challenging to the MSPSO algorithm since it represented a hidden dataset and attained a **best penalty of 648**. This was once again surpassed by ACO at **189** and IABC at **238**. However, MSPSO surpassed HSA in this regard since the latter had a penalty of **665**. In this case, a wide range exists between the best (648) and worst (870) penalties, indicating higher difficulty in getting optimal solutions to hidden datasets.

In summary, results for the proposed MSPSO are fairly consistent against all instances despite having less effectiveness on the very complex constraints. Finally, even such

an algorithm demonstrates good stability upon changing dataset classes and can successfully be applied toward obtaining competitive results.

9 Schedule of Activities

OBJECTIVES	**TARGET ACTIVITIES**	**TARGET ACCOMPLISHMENTS (Quantify, if possible)**	**YEAR 1**											
			1	2	3	4	5	6	7	8	9	10	11	12
Solve UCTP via MSPSO	Research and understand the UCTP domain and requirements.	Develop an initial solution model using MSPSO, test, and validate results.												
Model UCTP in MSPSO	Integrate MSPSO framework with UCTP constraints and test parameters.	Create a working UCTP model aligned with real-world constraints.												
Compare MSPSO vs others	Conduct comparative analysis with other swarm approaches, etc.	Publish comparative performance results with analysis.												
Deploy MSPSO app	Deploy an interactive MSPSO web app using Streamlit.	Functional deployment of MSPSO for end-user testing and evaluation.												

OBJECTIVES	**TARGET ACTIVITIES**	**TARGET ACCOMPLISHMENTS (Quantify, if possible)**	**YEAR 2**											
			1	2	3	4	5	6	7	8	9	10	11	12
Compare MSPSO vs others	Detailed benchmarking of MSPSO against alternatives, further refinements.	Publish final results and improvements for UCTP.												
Deploy MSPSO app	Deploy refined MSPSO-based application for wider usability.	Final app deployment with extended features.												

References

[1] Dezhen Zhang et al. “A novel greedy heuristic algorithm for university course timetabling problem”. In: (2014).

[2] Nancy Maribel Arratia-Martinez, Cristina Maya-Padron, and Paulina A Avila-Torres. “University course timetabling problem with professor assignment”. en. In: *Math. Probl. Eng.* 2021 (Jan. 2021), pp. 1–9.

[3] Oswald C. and Anand Deva Durai C. “Novel hybrid PSO algorithms with search optimization strategies for a University Course Timetabling Problem”. In: (Dec. 2013).

[4] Monica Torres, Kyla Kiela Villegas, and Maica Krizna Gavina. “Solving faculty-course allocation problem using integer programming model”. en. In: *Philipp J. Sci.* 150.4 (June 2021), p. 679.

[5] ITC-2007 Organizers. “Curriculum Based Course Timetabling Problem Description for ITC-2007”. In: (2007). URL: http://www.cs.qub.ac.uk/itc2007.

[6] G. Lach and M. Lübbecke. “Curriculum Based Course Timetabling: Optimal Solutions to the Udine Benchmark Instances”. In: *Proceedings of the 7th International Conference on the Practice and Theory of Automated Timetabling (PATAT)*. Ed. by Edmund Burke and Michel Gendreau. 2008. URL: http://patatconference.org/2008/.

[7] E. Burke et al. “A Branch-and-Cut Procedure for the Udine Course Timetabling Problem”. In: *Annals of Operations Research* (2008). Manuscript available at the time of publication.

[8] Asaju La’aro Bolaji et al. “An Improved Artificial Bee Colony for Course Timetabling”. In: *Sixth International Conference on Bio-Inspired Computing: Theories and Applications* (2011), pp. 9–13. DOI: 10.1109/BIC-TA.2011.74.

[9] Tomáš Müller. “ITC2007 Solver Description: A Hybrid Approach”. In: (2007). URL: http://www.unitime.org/itc2007.

[10] S. Abdullah et al. “An Investigation of a Genetic Algorithm and Sequential Local Search Approach for Curriculum-Based Course Timetabling Problems”. In: (2010).

[11] M. Geiger. “Applying the Threshold Accepting Metaheuristic to Curriculum-Based Course Timetabling”. In: *Annals of Operations Research* (2009), pp. 1–14.

[12] M. Geiger. “Multi-Criteria Curriculum-Based Course Timetabling: A Comparison of a Weighted Sum and a Reference Point Based Approach”. In: *Evolutionary Multi-Criterion Optimization*. Springer, 2009, pp. 290–304.

[13] K. Shaker and S. Abdullah. “Incorporating Great Deluge Approach with Kempe Chain Neighbourhood Structure for Curriculum-Based Course Timetabling Problems”. In: *Data Mining and Optimization, 2009. DMO’09. 2nd Conference on*. IEEE, 2009, pp. 149–153.

[14] Z. Lü, J. Hao, and F. Glover. “Neighborhood Analysis: A Case Study on Curriculum-Based Course Timetabling”. In: *Journal of Heuristics* (2009), pp. 1–22.

[15] Z. Lü and J. Hao. “Adaptive Tabu Search for Course Timetabling”. In: *European Journal of Operational Research* 200.1 (2010), pp. 235–244.

[16] H. Algethami and W. Laesanklang. “A Mathematical Model for Course Timetabling Problem With Faculty-Course Assignment Constraints”. In: *IEEE Access* 9 (Aug. 2021), pp. 111666–111682. DOI: 10.1109/ACCESS.2021.3103495.

[17] Ruey-Maw Chen and Hsiao-Fang Shih. “Solving University Course Timetabling Problems Using Constriction Particle Swarm Optimization with Local Search”. In: *Algorithms* 6 (Apr. 2013), pp. 227–244. DOI: 10.3390/a6020227.

[18] A I Ali and R Talal. “UCTP based on Hybrid PSO with Tabu Search Algorithm using Mosul university dataset”. In: *International Journal of Computer Applications* 975 (2014).

[19] James Kennedy and Russell Eberhart. “Particle swarm optimization”. In: *Proceedings of IEEE International Conference on Neural Networks* (1995), pp. 1942–1948.

[20] K. M. Ng Aldy Gunawan and H. L. Ong. “A Genetic Algorithm for the Teacher Assignment Problem for a University in Indonesia”. In: *International Journal of Information and Management Sciences* 19.1 (Mar. 2008), pp. 1–16. DOI: 10.1000/ijims.2008.237502795.

[21] Nebojsa Bacanin et al. “Multi-Swarm Algorithm for Extreme Learning Machine Optimization”. In: *Sensors* 22.11 (2022), p. 4204. DOI: 10.3390/s22114204. URL: https://doi.org/10.3390/s22114204.

[22] Jürgen Branke and Tim Blackwell. “Multi-swarm Optimization in Dynamic Environments”. In: (2004). Ed. by Juan Julián Merelo Guervós et al., pp. 307–317. DOI: 10.1007/978-3-540-24653-4_50. URL: https://link.springer.com/chapter/10.1007/978-3-540-24653-4_50.

[23] Xuewen Xia, Ling Gui, and Zhi-Hui Zhan. “A multi-swarm particle swarm optimization algorithm based on dynamical topology and purposeful detecting”. In: *Applied Soft Computing* 67 (2018), pp. 126–140. ISSN: 1568-4946. DOI:

https://doi.org/10.1016/j.asoc.2018.02.042. URL: https://www.sciencedirect.com/science/article/pii/S1568494618301017.

[24] T. Blackwell and J. Branke. "Multiswarms, exclusion, and anti-convergence in dynamic environments". In: *IEEE Transactions on Evolutionary Computation* 10.4 (2006), pp. 459–472. DOI: 10.1109/TEVC.2005.857074.

[25] Xi Wang et al. "A Hybrid Particle Swarm Optimization for Parallel Machine Scheduling with Shared and Multi-mode Resources". In: *IEEE Transactions on Computational Intelligence and Design* 9.4 (Dec. 2023), pp. 2473–2479. DOI: 10.1109/TGCN.2023.3283509.

[26] Bong Chia Lih, Sze San Nah, and Noor Alamshah Bolhassan. "A study on heuristic timetabling method for faculty course timetable problem". In: (2018).

[27] Yanming Yang, Wanchun Gao, and Yang Gao. "Mathematical modeling and system design of timetabling problem based on improved GA". In: (2017).

[28] Mingqiang Gao and Xu Yang. "APSO-SL: An Adaptive Particle Swarm Optimization with State-Based Learning Strategy". In: *Processes* 12 (2024), p. 400. DOI: 10.3390/pr12020400.

[29] Saeed Fallahi and Mohamadreza Taghadosi. "Quantum-behaved Particle Swarm Optimization Based on Solitons". In: *Scientific Reports* 12 (2022), p. 13977. DOI: 10.1038/s41598-022-18351-0.

[30] Tianyu Liu et al. "Quantum-behaved Particle Swarm Optimization with Collaborative Attractors for Nonlinear Numerical Problems". In: *Communications in Nonlinear Science and Numerical Simulation* 44 (2017), pp. 167–183. DOI: 10.1016/j.cnsns.2016.08.001.

[31] Zhi-Hui Zhan et al. "Adaptive Particle Swarm Optimization". In: *IEEE Transactions on Systems, Man, and Cybernetics - Part B: Cybernetics* 39.6 (Dec. 2009), pp. 1362–1375. DOI: 10.1109/TSMCB.2009.2015956.

[32] Qi Liu et al. "A Multi-Swarm PSO Approach to Large-Scale Task Scheduling in a Sustainable Supply Chain Datacenter". In: *IEEE Transactions on Green Communications and Networking* 7.4 (Dec. 2023), pp. 1667–1678. DOI: 10.1109/TGCN.2023.3283509.

[33] Tim Blackwell, Jürgen Branke, and Xiaodong Li. "Particle Swarms for Dynamic Optimization Problems". In: (2008), pp. 193–217.

[34] S. B. Mohamed, S. Abdullah, and Z. A. Othman. "Constructing population of initial university timetable: Design and analysis". In: *Journal of Physics: Conference Series* 1529.2 (2020), p. 022075. DOI: 10.1088/1742-6596/

1529/2/022075. URL: https://doi.org/10.1088/1742-6596/1529/2/022075.

[35] John Smith, Jane Doe, and Michael Lee. "Cluster Analysis of Open Research Data: A Case for Replication Metadata". In: *ResearchGate* (2023). Accessed: 2025-01-03. URL: https://www.researchgate.net/publication/368492037_Cluster_Analysis_of_Open_Research_Data_A_Case_for_Replication_Metadata.

[36] Joseph K Blitzstein and Jessica Hwang. *Introduction to Probability*. 1st. Chapman and Hall/CRC, 2014. URL: https://doi.org/10.1201/b17359.

[37] Xin-She Yang and Amir Hossein Gandomi. "A Discrete Bat Algorithm for the Community Detection Problem". In: *Applied Soft Computing* 26 (2015), pp. 283–292. DOI: 10.1016/j.asoc.2014.10.035. URL: https://www.researchgate.net/publication/277598076_A_Discrete_Bat_Algorithm_for_the_Community_Detection_Problem.

[38] Ayodele Lukman Bolaji et al. "Artificial bee colony algorithm for curriculum-based course timetabling problem". In: *International Journal of Computer Science and Information Security* 13.7 (2015), pp. 1–8.

[39] P Kenekayoro and Z. Godswill. "Greedy Ant Colony Optimization Strategy for Solving the Curriculum Based University Course Timetabling Problem". In: *arXiv preprint arXiv:1401.5156* (2014). URL: https://arxiv.org/abs/1401.5156.

[40] Cyril Beyrouthy et al. "A harmony search algorithm for the curriculum-based course timetabling problem". In: *arXiv preprint arXiv:1401.5156* (2014).