[7] Particle Swarm Optimization (PSO) for University Timetable Scheduling

Team Members [Team 22]

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Members | Name | ID | Level | Department |
| Team Leader | عبدالله خالد عبدالله | 20210543 | 3 | AI |
| 2nd Member | عمر عبدالله دياب عبدالله | 20210609 | 3 | AI |
| 3rd Member | محمد ايمن عبدالفتاح احمد | 20210750 | 3 | AI |
| 4th Member | امنيه خليل محمد | 20210181 | 3 | AI |
| 5th Member | اميره اشرف عبدالحميد | 20210184 | 3 | AI |
| 6th Member | روان عزت محمد عزت | 20210341 | 3 | AI |

**1) Project Idea:**

The course timetabling problem investigated in this work involves resource allocation constraints; under limited resources the conditions with regard to teachers, courses, students, and teaching facilities have to be met. The constraints vary with the environment, system and other factors of each college. Usually, constraints are divided into hard and soft constraints.

Description : The Project aims to find the best university timetable scheduled with taking considerations of constraints (Constrained Optimization Model)

* It provides a panel for defining course name, no. of periods per day, teacher name, etc.
* Any number of subjects or subject groups can be added into the provided fields.
* By using pso (particle swarm optimization), we can find the best timetable for the university

**2)Similar Applications:**

An efficient train scheduling algorithm on a single-track railway system

🡪 Research Paper Link: <https://www.researchgate.net/publication/323242756_An_efficient_train_scheduling_algorithm_on_a_single-track_railway_system>

Timetable Management website

🡪 Link: <https://www.addmengroup.com/ecas/time-table-software.htm>

Addmen is a leading provider for smart educational software for schools and top universities that permits you to automate all aspects of administration and ECAS (Enhanced Campus Automation Software) is one such dynamic school automation software that helps you organize one of the many structured elements of an institute like time-table management.

Managing time-table manually involves calculated logic and distribution of periods among faculty, class and subject. It is a very tedious and complex process and one cannot afford to make mistakes as the whole institute can go haywire. So we need a campus automation software like ECAS to take care of all these tasks without any errors.

A screen shot of a computer

Description automatically generated

**3)Main Functionalities:**

A computer code with text

Description automatically generated with medium confidence

1. **self.timetable\_data = timetable\_data**: This line assigns the **timetable\_data** argument passed to the **Particle** class constructor to the **timetable\_data** attribute of the **Particle** object. This data contains information about courses, including their names, rooms, teachers, and potentially time slots.
2. **self.position = None**: This line initializes the **position** attribute of the **Particle** object to **None**. The **position** attribute represents the current position of the particle in the search space. It will be later updated by the **initialize\_position** method.
3. **self.initialize\_position(num\_courses, num\_slots)**: This line calls the **initialize\_position** method to randomly initialize the position of the particle based on the number of courses (**num\_courses**) and the number of available slots (**num\_slots**). This method ensures that each course is assigned to a time slot.
4. **self.velocity = np.random.rand(num\_courses)**: This line initializes the **velocity** attribute of the **Particle** object with a random velocity vector. The velocity vector represents the direction and magnitude of movement of the particle in the search space. It has the same length as the number of courses.
5. **self.best\_position = self.position.copy()**: This line initializes the **best\_position** attribute of the **Particle** object to a copy of the current position. The **best\_position** attribute represents the best position the particle has encountered so far during the optimization process. Initially, it is set to the same value as the current position.
6. **self.fitness = None**: This line initializes the **fitness** attribute of the **Particle** object to **None**. The **fitness** attribute stores the fitness value of the particle, which represents how well the particle's current position satisfies certain optimization criteria. It will be calculated during the optimization process.

Top of Form

A computer screen shot of a code

Description automatically generated

1. **def initialize\_position(self, num\_courses, num\_slots):**: This line defines the **initialize\_position** method within the **Particle** class. It takes three arguments:
   * **self**: A reference to the current instance of the class.
   * **num\_courses**: The total number of courses to be scheduled.
   * **num\_slots**: The number of available time slots per day.
2. **num\_days = 5**: This line assumes that there are 5 days in a week. This value is used to distribute the courses evenly over the days of the week.
3. **courses\_per\_day = num\_courses // num\_days**: This line calculates the number of courses that should be scheduled per day, assuming an equal distribution over the 5 days.
4. **remaining\_courses = num\_courses % num\_days**: This line calculates the remaining courses that need to be distributed after evenly distributing **courses\_per\_day** over the 5 days.
5. **position = np.zeros(num\_courses, dtype=int)**: This line initializes an array named **position** with zeros, where each element represents a course. The length of the array is **num\_courses**, indicating the total number of courses. Each element of this array will later be assigned a time slot representing when the corresponding course will be scheduled. The data type of the array elements is set to **int** to ensure compatibility with integer-based indexing and calculations.

Top of Form

A computer code with text

Description automatically generated with medium confidence

1. **for day in range(num\_days):**: This loop iterates over each day of the week.
2. **start\_index = day \* courses\_per\_day + min(day, remaining\_courses)**: This line calculates the starting index in the **position** array where courses for the current day should be placed. It ensures that courses are evenly distributed over the days of the week, taking into account any remaining courses that need to be distributed.
3. **end\_index = start\_index + courses\_per\_day + (1 if day < remaining\_courses else 0)**: This line calculates the ending index in the **position** array for the courses of the current day. It adds **courses\_per\_day** to the **start\_index**, and if the current day has more remaining courses than the others, it adds an additional course to ensure proper distribution.
4. **assigned\_slots = self.assign\_time\_slots(end\_index - start\_index, num\_slots)**: This line calls the **assign\_time\_slots** method of the **Particle** class to assign time slots to the courses scheduled for the current day. The number of courses for the day is determined by **end\_index - start\_index**, and the total number of available time slots is passed as **num\_slots**.
5. **position[start\_index:end\_index] = assigned\_slots**: This line assigns the time slots obtained from **assigned\_slots** to the corresponding courses in the **position** array for the current day.
6. **self.position = position**: Finally, this line sets the **position** attribute of the **Particle** object to the array **position**, which now contains the assigned time slots for all courses.

A close-up of a computer code

Description automatically generated

1. **def assign\_time\_slots(self, num\_courses, num\_slots):**: This line defines the **assign\_time\_slots** method within the **Particle** class. It takes three arguments:
   * **self**: A reference to the current instance of the class.
   * **num\_courses**: The number of courses for which time slots need to be assigned.
   * **num\_slots**: The total number of available time slots.
2. **return np.random.choice(num\_slots, num\_courses, replace=False)**: This line generates an array of random time slots for the given number of courses. Here's how it works:
   * **np.random.choice**: This NumPy function randomly selects elements from a given array (in this case, **num\_slots**) based on certain parameters.
   * **num\_slots**: The array from which to randomly select time slots.
   * **num\_courses**: The number of time slots to select. Since each course needs a time slot, **num\_courses** determines the length of the output array.
   * **replace=False**: This parameter ensures that each time slot is selected only once, meaning no two courses are assigned the same time slot. If **replace** were set to **True**, it would allow for duplicate selections, which is not desired in this context.

A computer code with black text

Description automatically generated

1. **def convert\_to\_time\_slots(timetable\_data, numerical\_timetable):**: This line defines the **convert\_to\_time\_slots** function. It takes two arguments:
   * **timetable\_data**: This is a list containing information about each course, including details such as the course name, room, teacher, and time slot.
   * **numerical\_timetable**: This is a list of indices representing the time slots assigned to each course. Each index corresponds to a particular time slot.
2. **time\_slots = []**: This line initializes an empty list named **time\_slots** which will store the time slots corresponding to the courses.
3. **for slot\_index in numerical\_timetable:**: This line initiates a loop over each element (slot index) in the **numerical\_timetable** list.
4. **time\_slots.append(timetable\_data[slot\_index]['time\_slot'])**: Within the loop, this line accesses the **timetable\_data** list using the current **slot\_index** to retrieve the corresponding course's information. From this information, specifically the **'time\_slot'** value, it extracts the time slot assigned to the course and appends it to the **time\_slots** list.
5. **return time\_slots**: Finally, this line returns the list **time\_slots** containing the time slots corresponding to each course. The order of the time slots in this list aligns with the order of courses in **numerical\_timetable**, making it easy to interpret the timetable.

A computer screen shot of a computer code

Description automatically generated

A computer screen shot of text

Description automatically generated

1. **def fitness\_function(position, timetable\_data):**: This line defines the **fitness\_function**. It takes two arguments:
   * **position**: A list representing the current schedule of courses, where each element is the index of the time slot assigned to a course.
   * **timetable\_data**: A list containing information about each course, including details such as the course name, room, teacher, and time slot.
2. **fitness\_score = 0**: This line initializes the **fitness\_score** variable to keep track of conflicts in the timetable.
3. The function iterates over each course in the **position** list, checking for conflicts:
   * It first checks if the **course\_index** is valid within the range of **timetable\_data**. If it's not valid, it skips to the next iteration.
   * It retrieves the room and time slot assigned to the current course from **timetable\_data**.
   * It then checks for conflicts related to room and teacher assignments with other courses scheduled at the same time.
4. The function returns the **fitness\_score**, which represents the total number of conflicts found in the timetable.

A computer code with text

Description automatically generated with medium confidence

1. **inertia\_term**: This term represents the inertia of the particle. It is calculated by multiplying the particle's current velocity by the inertia weight (**inertia\_weight**). This term allows the particle to retain some of its current velocity, helping to maintain directionality during optimization.
2. **cognitive\_term**: This term represents the cognitive component of the particle's movement. It encourages the particle to move towards its own best-known position (**particle.best\_position**). The amount of influence is determined by the cognitive parameter (**cognitive\_param**). The random value multiplied by **(particle.best\_position - particle.position)** introduces randomness to the movement, promoting exploration.
3. **social\_term**: This term represents the social component of the particle's movement. It encourages the particle to move towards the global best-known position (**global\_best\_position**). If **global\_best\_position** is not **None**, it calculates the difference between the global best position and the particle's current position, scaled by a random value and the social parameter (**social\_param**). This term facilitates exploitation of the best solutions found by other particles. If **global\_best\_position** is **None**, it sets the **social\_term** to a zero vector to avoid any influence from global best position.

A black and white text

Description automatically generated

1. **Particle's Position**: Each particle in the swarm represents a potential solution to the optimization problem. The position of a particle is a point in the search space that corresponds to a particular solution.
2. **Particle's Velocity**: The velocity of a particle determines its movement direction and magnitude. It guides the particle's exploration of the search space towards potentially better solutions.

The **update\_position** function is responsible for updating the position of a particle based on its velocity. Here's how it works:

A computer screen shot of a program

Description automatically generated

1. **Particle Initialization**:
   * **particles = [Particle(num\_courses, num\_slots, timetable\_data) for \_ in range(num\_particles)]**: This line creates a list of particles, where each particle is initialized with random positions and velocities.
2. **Initialization of Variables**:
   * **global\_best\_position = None**: Initializes the global best position as **None**.
   * **global\_best\_fitness = float('inf')**: Initializes the global best fitness value as positive infinity.
   * **inertia\_weight = 0.5**, **cognitive\_param = 1.5**, **social\_param = 2.0**: Initialize parameters used in updating particle velocities.
3. **PSO Iterations**:
   * **for \_ in range(max\_iterations):**: Loops through the specified maximum number of iterations.
     + Nested loop:
       - **for particle in particles:**: Iterates over each particle in the swarm.
         * Updates the fitness of each particle based on its current position using the **fitness\_function**.
         * If the current fitness of the particle is better than its previous best fitness, update the particle's best position and check if it's better than the global best fitness. If so, update the global best fitness and position accordingly.
       - After updating particle fitness and best positions, update the velocities and positions of all particles in the swarm using the **update\_velocity** and **update\_position** functions, respectively.
4. **Rounding Positions**:
   * Rounds the global best position to the nearest integers for better interpretation.
   * Returns the rounded global best position if it's not **None**, otherwise returns **None**.

This function essentially performs the optimization process using the Particle Swarm Optimization (PSO) algorithm, iterating over particles, updating their positions and velocities, and tracking the global best solution found throughout the iterations.

**(4) A literature review of Academic publications (papers) relevant to the idea:**

🡪 Research Paper Link: <https://en.wikipedia.org/wiki/Particle_swarm_optimization>

particle swarm optimization (PSO)is a computational method that optimizes a problem by iteratively trying to improve a candidate solution with regard to a given measure of quality. It solves a problem by having a population of candidate solutions, here dubbed particles, and moving these particles around in the search-space according to simple mathematical formula over the particle's position and velocity. Each particle's movement is influenced by its local best known position, but is also guided toward the best known positions in the search-space, which are updated as better positions are found by other particles. This is expected to move the swarm toward the best solutions.

PSO does not use the gradient of the problem being optimized, which means PSO does not require that the optimization problem be differentiable as is required by classic optimization methods such as gradient descent and quasi-newton methods. However, metaheuristics such as PSO do not guarantee an optimal solution is ever found.

A diagram of a graph

Description automatically generated with medium confidence

A particle swarm searching for the global minimum of a function

🡪 Research Paper Link: <https://www.academia.edu/4791311/AUTOMATED_TIMETABLE_GENERATOR_USING_PARTICLE_SWARM_OPTIMIZATION>

Kennedy and Eberhart developed Particle swarm optimization algorithm in 1995. It is a technique based on particles, each particle has velocity in search space in order to find feasible solution and adjust its position according its own previous experience and neighbors’ experiences. The initial position and velocity of each particle is anomaly determined. When each particle to move a new position then it will remember its personal best (Pbest) and own information. Each particle will also exchange its information to other particles and remember its global best (Gbest). Each particle has fitness value. Fitness value of each particle’s current position is compared with Pbest and Gbest. If it is better than Pbest and Gbest then update the Pbest and Gbest.

A diagram of a particle

Description automatically generated

🡪 Research Paper Link: <https://link.springer.com/chapter/10.1007/978-3-642-37213-1_27>

The main procedure and the procedure for one update of position in the flying process of i-th particle xi are as follows:

A white text on a white background

Description automatically generated

A white text with black text

Description automatically generated

🡪 Research Paper Link: <https://staff.fmi.uvt.ro/~daniela.zaharie/ma2017/projects/applications/timetabling/PSO%2Btimetabling1.pdf>

the process of velocity update :

V i+1 k = V i k + C1 · r1.(Pi k − Xi k) + C2 · r2 · (Gi − Xi k).

movement of the particles is processed by the following equation:

Xi+1 k = Xi k + V i+1 k, i = 0, 1, ··· , M − 1

where M is the particle size,

−Vmax ≤ V i+1 k ≤ Vmax (Vmax is the maximum velocity),

and r1 and r2 are random variables such that 0 ≤ r1, r2 ≤ 1.

If a solution is better than Gi , Gi will be replaced by this solution to represent Gi+1 , Otherwise, there will be no change for the global best solution, i.e. Gi = Gi+1. These recursive steps will go on unless we reach the termination condition. In this paper, we propose a method based on self-mutation concept for PSO to solve the discrete problem of timetable scheduling

🡪 Research Paper Link: <https://link.springer.com/article/10.1007/s42979-021-00652-2>

Scheduling or timetabling is an important problem and active area of research with applications in many fields , e.g., transportation , sport , healthcare (e.g., nurse rostering , surgery ), industrial production , employee , educational institutes (e.g., secondary school , university).

Most of the UCTP constraints can be mainly classified into two groups including hard constraints and soft constraints . The most important constraints are hard constraints, each of which must be satisfied to guarantee the candidate timetables to be the feasible timetables . Soft constraints can be more relaxed, some violations of soft constraints can be accepted but it should be minimized

🡪 Research Paper Link: <https://www.researchgate.net/publication/284607866_Solving_University_Course_Timetabling_Problems_Using_Constriction_Particle_Swarm_Optimization_with_Local_Search>

PSO is a promoting scheme for solving complex problems such as course timetabling. A new PSO named standard PSO (SPSO) has been applied and investigated. Thus, this work discusses the application of different types of PSO (PSO and SPSO) to find solutions to solve optimization problems in university course timetabling. Concurrently, to reduce the computational complexity, particle encoding is designated on the basis of timeslot rather than study hour. Moreover, an interchange heuristic is included to explore the neighboring solution space and enhance solution quality. The solutions found in accordance with the characteristics of the problem have been able to improve the satisfaction of the teachers and classes toward the schedule. Any conflicts between the teacher schedules, the class schedules, or the classroom schedules were also handled in this work.

**(5)Details of Algorithm :**

PSO algorithm used to find optimal time table by minimizing the conflicts of the fitness function ,the solution space is he solution space is constrained by various factors such as the availability of rooms, teachers, and time slots, as well as the need to avoid conflicts between courses.

First : initialize particles that used in the algorithm by taking time table data , number of slots and number of courses

Second : initialize position of the particle by using array of zeroes and then it updated with courses and time slots dimensions (multidimensions array )

Third : Each particle in the PSO algorithm represents a potential timetable schedule, and its position corresponds to a point in the solution space. As the algorithm iterates, particles move through the solution space in search of better solutions.

Forth : update velocity for each particle to allow movement in solution space to find best schedule of timetable

Fifth : apply fitness function to check the conflicts of rooms and teachers as its not allowed that both courses have the same room in the same time slot and cannot have same teacher teach different courses at the same time slot so we tend to minimizes these conflicts

Sixth : Apply PSO algorithm to find optimal or near-optimal timetable by applying comparisons between particles by using position and velocities to find the global best particle which has best timetable scheduled

Finally : print the optimal timetable for user

**(6) Development Platform:**

- Specify the programming languages and libraries used for implementation (e.g., Python, NumPy for numerical computations, tkinter for gui).

- Mention any additional tools or frameworks utilized for experimentation and analysis (e.g., Jupyter Notebooks)