# **Reference material for final report**

Based on the information from the paper, we'll adjust the focus of your Dynamic Timetable Generator to cater to the scenario where student sectioning is not a concern, and teachers move between fixed classes based on their slots. Here are the key points we can extract from the provided text to guide the development:

1. \*\*UCTP Constraints for Teachers\*\*:

- Focus on assigning teaching events (classes) to certain times and rooms while considering constraints related to teachers, such as their availability and preferences for specific time slots.

- Understand that teachers are floating, meaning they can move between classes based on their assigned slots rather than being fixed to specific classes.

2. \*\*Genetic Algorithms (GA)\*\*:

- Implement genetic algorithms as a practical solution for timetable generation, focusing on optimizing the assignment of teaching events while considering the constraints of teachers' availability and preferences.

3. \*\*Multi-Depth GA (MDGA)\*\*:

- Design a variation of GA called Multi-Depth GA (MDGA) that addresses the scalability and complexity challenges of UCTP with student sectioning.

- Use three levels of objective functions (shallow, medium, and deep) to handle different levels of problem complexity, specifically focusing on the floating nature of teachers and the optimization of their schedules.

4. \*\*Objective Function and Mutation\*\*:

- Develop objective functions and mutation operations tailored to the needs of the timetable generation problem for teachers with floating schedules.

- The objective function should minimize constraint violations related to teachers' schedules while maximizing optimization based on their preferences and availability.

5. \*\*Experimental Analysis\*\*:

- Conduct experiments to analyze the performance of MDGA in handling the timetable generation problem for floating teachers.

- Evaluate the scalability, efficiency, and effectiveness of MDGA compared to traditional GA approaches, especially in scenarios where the number of variables (teachers, classes, time slots) is large.

6. \*\*System Organization\*\*:

- Organize the system to handle the generation of timetables for floating teachers, focusing on optimizing their schedules based on availability and preferences.

- Develop a user-friendly interface for teachers to input their preferences and view their optimized schedules.

By aligning your project with these points, you can create a Dynamic Timetable Generator that effectively handles the scheduling of teaching events for floating teachers based on their availability and preferences within the constraints of the university timetable.

Based on the mathematical models and constraints described in the paper, we can tailor the development of your Dynamic Timetable Generator to focus on the scheduling of teaching events for floating teachers within the given constraints. Here are some key points derived from the provided equations:

1. \*\*UCTP Formulation\*\*:

- Define a set of teaching events (events) consisting of a lecturer (l), a class (c), and a set of students (S).

- Represent each event as e = (l, c, S), where e is an event, l is the lecturer, c is the class, and S is the set of students.

2. \*\*Timetable Mapping\*\*:

- Understand that a timetable is a mapping of all events into several or all packets (q = (t, r)), where q represents a time and room combination.

- Define a mapping of a packet and an event as p = (q, e), where p is a pair consisting of a packet q and an event e.

3. \*\*Constraints\*\*:

- Implement both hard constraints (HC) and soft constraints (SC) relevant to the scheduling of teaching events for floating teachers.

- Consider constraints such as:

- No conflict of lecturers (HC1).

- No conflict of classes (HC2).

- Scheduling lecturers within their time constraints (HC4).

- Scheduling some lecturers in their preferred time (HC5).

- Spread of lecturer assignments (SC1).

- Spread of class events (SC2).

- Scheduling some lecturers in their preferred time (SC4).

- Time constraints between events for a lecturer (SC5).

- Student assignment spread (SC6).

4. \*\*Objective Functions\*\*:

- Minimize violation counts for each constraint to optimize the timetable generation process.

- Develop objective functions corresponding to each constraint, such as f1(p, p0), f2(p, p0), ..., f12(z, p, p0), as described in the paper.

5. \*\*Optimization Algorithm\*\*:

- Consider using a genetic algorithm or a similar optimization algorithm to efficiently handle the timetable generation problem while considering the constraints and objectives outlined in the paper.

By incorporating these elements into your Dynamic Timetable Generator, you can create a system that effectively generates optimized timetables for floating teachers while satisfying the specified constraints and objectives.

Got it, so in addition to the constraints and formulation mentioned earlier, we'll include the aspect where certain sets of students and classes are fixed pairs, meaning specific classrooms are allocated to a predefined set of students, and this allocation remains unchanged. This adds another layer of constraints to the timetable generation process. Here's how we can integrate this into the development plan:

1. \*\*Fixed Student-Classroom Pairs\*\*:

- Identify and define the fixed pairs of students and classrooms in the system.

- Ensure that the allocation of these fixed pairs remains constant throughout the timetable generation process.

2. \*\*Constraints Update\*\*:

- Update the constraints to reflect the fixed pairs, ensuring that these pairs are scheduled together in the specified classrooms without any changes.

- Consider constraints such as:

- Fixed allocation of specific classrooms to certain classes.

- Fixed allocation of specific classrooms to predefined sets of students.

3. \*\*Objective Functions Adjustment\*\*:

- Modify the objective functions to optimize the timetable while adhering to the constraints related to fixed student-classroom pairs.

- Ensure that the optimization algorithm considers the fixed pairs and avoids making changes to their allocations.

4. \*\*Timetable Generation Algorithm\*\*:

- Implement an algorithm that takes into account the fixed pairs and optimally schedules the remaining events around these fixed allocations.

- Ensure that the algorithm maintains the integrity of fixed pairs while generating the timetable.

5. \*\*Testing and Validation\*\*:

- Test the system extensively to validate that the fixed pairs are correctly handled and that the timetable generation process produces optimal results while respecting all constraints, including the fixed allocations.

By incorporating these adjustments, your Dynamic Timetable Generator will be able to effectively handle the scheduling of teaching events for floating teachers while also managing the fixed pairs of students and classrooms without altering their allocations.

Yes, that makes perfect sense. Generating timetables based on departments and from the perspective of floating teachers aligns with the typical approach in academic institutions. Here's how you can approach it:

1. \*\*Department-Based Timetables\*\*:

- Organize the timetables by departments, ensuring that each department has its own dedicated timetable.

- This allows for clear separation and organization, making it easier for departments to manage their schedules.

2. \*\*Floating Teachers Perspective\*\*:

- Since teachers are floating and can teach multiple subjects or classes across different departments, consider their availability and preferences when scheduling.

- Design the timetable generation process to optimize the allocation of floating teachers to classes based on their availability, subject expertise, and preferences.

3. \*\*Integration with Fixed Student-Classroom Pairs\*\*:

- Ensure that the timetables generated for departments also consider the fixed pairs of students and classrooms, maintaining consistency in class allocations.

- Teachers should be assigned to classes within these fixed pairs while adhering to their floating status and departmental responsibilities.

4. \*\*User-Friendly Interface\*\*:

- Develop a user-friendly interface where teachers can input their availability, preferences, and constraints.

- The interface should allow teachers to view their assigned classes and make adjustments if necessary, considering their floating status.

5. \*\*Real-Time Updates and Notifications\*\*:

- Implement a system for real-time updates and notifications to inform teachers about any changes or updates in their timetables.

- This ensures effective communication and minimizes disruptions in the scheduling process.

By considering these aspects, your timetable generator can effectively generate timetables based on departments while catering to the floating nature of teachers and maintaining the integrity of fixed student-classroom pairs.

# MDGA

The encoding strategy used in MDGA for UCTP involves direct chromosome encoding, which mimics the real-world representation of university timetabling. Here's a breakdown of the encoding process based on the provided information:

1. \*\*Direct Chromosome Encoding\*\*:

- Direct chromosome encoding directly represents the timetable structure in the genetic algorithm.

- The timetable includes R rooms and timeslots, where each timeslot represents a specific day and shift (e.g., 7 AM to 4 PM for 6 days).

- This encoding strategy uses enumeration encoding for rooms and timeslots:

- Rooms are encoded as integers from 1 to R, where each integer represents a specific room (Room 1 to Room R).

- Timeslots are encoded as integers from 1 to 60, where each integer represents a specific timeslot (e.g., 1 for 7 AM Monday, 2 for 8 AM Monday, and so on).

2. \*\*Chromosome Structure\*\*:

- The chromosome structure consists of gene blocks, each representing a teaching event.

- A gene block comprises five parts: time, room, lecturer, class, and students.

- The length of each gene block is equal to the number of events in the timetable, indicating the number of teaching events to be scheduled.

3. \*\*Fixed Parts in Chromosome\*\*:

- In the chromosome, the parts related to lecturer, class, and students are fixed, as they are determined by the specific teaching events.

- Only the packet (time and room) is considered in the search space for genetic algorithm operations.

4. \*\*GA Operations on Packets\*\*:

- All genetic algorithm operations such as mutation and crossover are applied to packets, which consist of time and room information.

- This means that the genetic algorithm manipulates the packets to optimize the timetable while keeping other parts fixed.

Overall, this encoding strategy allows the genetic algorithm to work directly with the timetable structure, focusing on optimizing the assignment of timeslots and rooms for teaching events while considering fixed aspects related to lecturers, classes, and students.

Let's break down the problem definition and the objective functions introduced in the research for solving the student sectioning UCTP.

1. \*\*Problem Definition\*\*:

- The primary goal is to solve the student sectioning UCTP, which involves assigning students to classes while considering their preferences and additional constraints.

- Due to the large number of students, traditional methods become computationally intensive and challenging to guarantee the satisfaction of hard constraints (HCs).

2. \*\*Objective Function Division\*\*:

- \*\*Shallow Objective Function (HCs Focus)\*\*:

- Equation 28 represents the shallow objective function, which prioritizes maintaining hard constraints (HCs) with a large weighting.

- The goal of the shallow objective function is to ensure that critical constraints are always satisfied, even at the expense of computational complexity.

- Minimize \( V\_{HC} = \sum\_{i=1}^{5} V\_i(P) \) focuses on minimizing the violation of hard constraints.

- \*\*Medium Objective Function (Class-level SCs Focus)\*\*:

- Equation 29 represents the medium objective function, which focuses on solving class-level soft constraints (SCs) in addition to respecting HCs.

- Minimize \( V\_{class} = \sum\_{i=1}^{10} V\_i(P) \) aims to minimize the violation of soft constraints related to class-level preferences and constraints.

- \*\*Deep Objective Function (Student-level SCs Focus)\*\*:

- Equation 30 represents the deep objective function, which prioritizes solving student-level soft constraints (SCs) while also addressing HCs.

- Minimize \( V\_{student} = \sum\_{i=1}^{12} V\_i(P) \) focuses on minimizing the violation of soft constraints related to individual student preferences and constraints.

3. \*\*Objective Function Evaluation\*\*:

- Each objective function \( V\_{HC} \), \( V\_{class} \), and \( V\_{student} \) is evaluated based on the sum of violation counts ( \( V\_i(P) \) ) across different constraints.

- The objective is to minimize these violation counts, indicating a higher level of satisfaction with constraints and preferences.

By dividing the objective function into shallow, medium, and deep levels, the MDGA approach can effectively balance the satisfaction of hard constraints (HCs) while also addressing soft constraints (SCs) at different levels of granularity, ultimately leading to better solutions for the student sectioning UCTP.

Let's rephrase the Deep Objective Function (Student-level SCs Focus) into the perspective of teachers rather than students:

The Deep Objective Function in the context of teachers would prioritize addressing soft constraints (SCs) at the teacher level while also considering hard constraints (HCs). The goal is to minimize the violation of constraints related to individual teacher preferences and constraints.

Here's the rephrased description:

"Equation 30 represents the deep objective function, which prioritizes solving soft constraints (SCs) at the teacher level while also addressing hard constraints (HCs). The objective is to minimize the violation of constraints related to individual teacher preferences and constraints."

Here's a rephrased version of the MDGA Procedure section:

The MDGA procedure integrates the satisfaction of hard constraints (HCs) during the initial population generation using a greedy initialization approach. This results in an initial population with minimal HC violations. Subsequently, soft constraints (SCs) satisfaction is implemented in the core of the GA procedure.

For each generation (t), Algorithm 1 outlines the GA steps such as elitism, selection, crossover, and mutation. Initially, a population (P) is created using greedy initialization, with the population size (PopSize) determining the number of individuals. Elite individuals are identified and preserved throughout the generations to maintain the best solutions.

Elitism ensures that the best individual from P is retained in the new population P0, along with the best M individuals from P, where M represents the number of elite individuals. Roulette wheel selection is employed to choose parents for crossover fairly based on their evaluations. The selected parents undergo crossover with a probability (Pc) to produce offspring, which are then evaluated using class-level evaluation. The best individuals from the parents and offspring become the new parents.

Mutation is divided into two stages to gradually solve the problem. In stage 1 mutation, new parents undergo mutation with a probability (Pm) to produce offspring, evaluated using class-level evaluation. The best individuals from parents and offspring are selected as the new parents. Once the population size reaches PopSize, stage 2 mutation commences with the same mutation probability (Pm) but with student-level evaluation for parents and offspring. Again, the best individuals are chosen as new parents, and the process continues until P0 reaches PopSize.

Finally, P0 replaces P, and the procedure iterates from selection to mutation until the desired number of generations is achieved.

The crossover technique employed in MDGA is a multi-point crossover, where the number of affected genes (Nc) corresponds to all genes violating constraints. The crossover process unfolds as follows:

1. Two individuals are selected from the population as parents.

2. Nc events that violate constraints are selected from the first parent.

3. One event is chosen from the selected violating events.

4. A random event is chosen from the second parent, provided it has the same room capacity as the event selected from the first parent, regardless of constraint violations.

5. The selected event from the first parent is swapped with the corresponding event from the second parent.

6. Steps 3 to 5 are repeated until all selected events from the first parent have been swapped.

This multi-point crossover strategy ensures that constraint-violating events are exchanged between parents, facilitating the exploration of new solutions while maintaining feasibility.

Let me know if there's anything more you'd like to add or modify!

Here's the explanation of the Mutation steps as described:

### Mutation Steps:

\*\*M1 (Moving):\*\*

- Choose an event that violates constraints.

- Move this event to an unused packet (packet not occupied by any event), chosen from packets with appropriate room capacity.

- Illustrated in Fig. 3.

\*\*M2 (Swapping):\*\*

- Select a constraint-violating event as the first event.

- Identify other events with the same subject as the first event.

- Randomly select an event from these identified events.

- Swap the first event with the target event. Keep the new individual if violations decrease; otherwise, revert the swap.

- Illustrated in Fig. 4.

\*\*M3 (Comparing):\*\*

- Choose an event violating constraints.

- Randomly select two events with the same room capacity, regardless of violations.

- Swap the violating event with the event that results in lower violations.

- Keep the new individual if violations decrease; otherwise, cancel the swap.

- Illustrated in Fig. 5.

These mutation steps aim to iteratively improve offspring by moving, swapping, and comparing events, ultimately reducing violations and enhancing the quality of solutions.

Here is the Algorithm 1 GA Procedure based on your description:

```

Require: GA in generation t

if t = 1 then

Greedy Initialization population P[PopSize]

end if

P0 = ∅ //empty population

// elitism

Put the best individual of P into P0

Put the best M individuals of P into P0

count = 1 + M

while count < PopSize do

// selection

Select idv1 and idv2 from P with roulette wheel

// crossover

Crossover idv1 and idv2 with probability Pc

[idv1, idv2] = best 2 individuals of parents & offsprings

// stage 1 mutation

Mutate idv1 and idv2 with probability Pm

[idv1, idv2] = best 2 individuals of parents & offsprings

count = count + 2

end while

// stage 2 mutation

Select idv1 and idv2 from P with roulette wheel

Mutate idv1 and idv2 with probability Pm

[idv1, idv2] = best 2 individuals of parents & offsprings

Put idv1 and idv2 into P0

Replace P with P0

```

This algorithm outlines the steps for a Genetic Algorithm (GA) procedure, including initialization, elitism, selection, crossover, and mutation stages for generating new individuals and updating the population.

Does this implementation align with your expectations? Any adjustments needed?

Great! For data collection, we need to gather information about the departments, classes, teachers, rooms, and any constraints that need to be considered in the timetable generation. We'll also need to collect data about the availability preferences of teachers.

For encoding, we'll follow the direct chromosome approach as discussed earlier. The chromosome will represent the timetable, with each gene block representing an event consisting of time, room, teacher, class, and students. Time and room will be enumerated, while the other parts will be fixed based on the data collected.

Let's start by gathering the necessary data about the departments, classes, teachers, rooms, and constraints.

Sure, let's start by structuring the data collection process based on the requirements you've outlined.

1. \*\*Timeslots Data Collection:\*\*

- Number of days per week: Institution head enters the number of days the timetable needs to be scheduled for each week.

- Number of slots per day: Institution head specifies the number of time slots available each day.

2. \*\*Rooms Data Collection:\*\*

- Department name: Institution head enters the name of the department.

- Semester: Institution head selects or enters the semester for which the rooms are allocated.

- Number of batches: Institution head specifies the number of batches for the selected semester.

- Room details: For each room, the institution head enters the room name or code, the allotted batch or batch code, and the purpose of the room (lecture, lab, project, etc.).

3. \*\*Courses Data Collection:\*\*

- Department: Institution head selects the department to which the course belongs.

- Semester: Institution head selects the semester for the course.

- Course details: For each course, the institution head enters the course name, course code or subject code, and selects or enters the allotted teachers for that course.

4. \*\*Departments Data Collection:\*\*

- Department name: Institution head enters the name of each department offered by the institution.

- Number of batches: Institution head specifies the number of batches under each department.

5. \*\*Teacher Profile Data Collection:\*\*

- Preferred timeslot: Teachers enter their preferred timeslots, such as forenoon or afternoon.

- Position: Teachers enter their position or designation, such as assistant professor, associate professor, etc.

This data collection process will help in structuring the initial information required for timetable generation. Once this data is collected and stored appropriately, we can proceed with the encoding and further stages of timetable generation. How would you like to proceed with implementing this data collection process?