CS575: Final Project Report

Project Title: Timetable Generation using Genetic Algorithm (GA) and Bacterial Foraging OptimizationAlgorithm (BFOA)

Team Member: Rashmi A. Badadale

I. PROBLEM

The time tabling problem is categorized among highly constrained NP-hard problems. Timetable generation for schools and colleges has been very tedious task since a very long time. It's high time to automate this system in this world of instant technology development and communication. We often change the things that are not suitable and are those that require high manpower. This generation of timetable also requires a lot of effort and human power. Complications and overlapping make this over all a difficult task to implement. Therefore, I designed a program in python that generates this timetable using lesser resources than required by us. Timetable generation requires a system having some machine learning capabilities and thus requires some algorithms to be processed on the input given in order to deliver the required output. These algorithms form a platform for the program on how to work upon the given input. In this project, these algorithms are genetic algorithm (GA) and Bacterial Foraging Optimization Algorithm (BFOA). These are the algorithms, both intended to provide an optimized solution, i.e. the best of its kind considering all the parameters.

A lot of parameters are needed to be considered while designing the program. Satisfying all these constraints can only lead us to the optimized solution for timetable generation.

II. ALGORITHMS

In this project, I have used two major algorithms, Genetic Algorithm (GA) and Bacterial Foraging Optimization Algorithm (BFOA). The project was developed as an application for implementation of both the mentioned algorithms. In each subsection below, I have described briefly how each algorithm works. Both the algorithms use the concept of Dynamic programming and I preferred combining it with the heuristic approach, by letting the developed program taking real-time decisions on what solution would be the best for the next step of the process in order to reach an optimized solution. The multi-objective behavior of many problems has led to the innumerable conflicting solutions. These kinds of problems are termed as Multi-objective Problems (MOPs). However, simultaneous optimization of each of the objective is not possible. The main purpose of solving MOPs is to derive such a set of solutions which cannot be dominated by any other solutions. A solution is called as Pareto optimal if there is no existing solution whose constraint dominates the former's constraint. The set of all Pareto objective vectors is called as Pareto front PF [2]. A good solution set should be:

- As close as possible to the true Pareto front
- Uniformly distributed over the entire PF [2]

A. Genetic Algorithm

Genetic algorithm is a heuristic search method which duplicates the natural selection process. John Holland was the father of the first genetic algorithm, which he invented in 1970's [1]. It applies natural selection process components like mutation, crossover, substitution, etc. in order to provide the optimized search. Consider a problem having multiple objectives and needs an optimal solution. These kinds of problems are called as multi-objective optimization problems (MOO). These kinds of problems need a solution that satisfies all the objectives of the problem statement. Now, there will be a number of solutions satisfying certain objectives of the problem but a very few, or only one that satisfies the maximum no. of objectives of the problem. This one solution that satisfies all or maximum no. of objectives of the problem is known as optimized solution. In order to retrieve this optimal solution, we use genetic algorithm, which processes the input given by applying natural selection techniques and hence give us an optimal solution.

Following are the main features of the genetic algorithm:

- Generating population
- Fitness value determination
- Selection (Stochastic selection)
- Crossover
- Mutation/ Permutation

The following is the pseudo code for GA I am using:

1. Start
2. Generate population of *n, l-bit* chromosomes
3. for (each offspring that belongs to *n*)
   {
   f(n) = Calculate Fitness.
   Fitness_max = max(f(n))

```
        while (fitness = Fitness_max)
        {
                Choose 2 candidates
        }
        if(crossover=yes)
                perform crossover.
        else (form 2 exact copies of resp. parents)
    }

4. if (mutation = true)
   {
        Perform Mutation.
    }
5. Place resulting chromosomes in new population.
6.Replace current population with new population
7. End for loop
8.End
```

The algorithm is emphasizing mostly on one feature that helps converging the solution to optimality. In this case, timetable generation evaluates the fitness value by determining the number of conflicts it comes across in a single solution, timetables in this case.

All these constraints are checked and verified during the process of timetable generation.

B. Bacterial Foraging Optimization Algorithm

The bacterial foraging optimization algorithm provides an optimized approach which is built on the concept of searching and foraging motion of Escherichia Coli bacteria (E. coli). A virtual bacterium corresponds to a point in n-dimensional search space where each point may be potential solution to the timetabling problem. Chemotaxis stage is used to search for optimal solutions of the problem. In greed of the nutrient, the bacterium randomly walks to find better gradient of nutrition alternatively with swim (clockwise rotation) and tumble (counter clockwise rotation). There is a stage called swarming after chemotaxis where cell-to-cell interaction takes place. It is then followed by reproduction stage in which weaker bacteria are eliminated and the stronger one is supposed to split into two. Premature convergence is neglected due the following stages viz. elimination and dispersal.

The following is the pseudo code for the BFOA:

1. Start
2. Initialize Population (Chemotaxis)
3. For (each bacterium in population - Swarming)
    a. Mutate for Tumble process (Reproduction)
    b. Mutate for swim process
    c. Calculate Fitness values
    d. Selection based on fitness
4. Elimination (of unfit bacteria)
5. Dispersal (End)

III. Software Design and Implementation

In this project, I have combined both the algorithms, GA and BFOA to get an optimized solution. The need for combining both the algorithms originates from the disadvantages that emerge on implementation. The GA is a very fine algorithm for solving the multi-objective optimization problems, except for it causes the problem of over-fitting most of the times. Since the algorithm generate all the possible outcomes with respect to all parameters provided, it is bound to overfit. If we keep on training the algorithm rigorously, the usability for the same would be low. Also, even a small input would need a large time since, we are repopulating the potential solutions with new changes. From all the above-mentioned points, we can clearly see that GA starts off well but may hinder the performance as it proceeds failing to reach the global maxima in expected time period.

The disadvantage with BFOA is that if the bacteria's locomotive steps are longer than usual, and if the optimum value remains in the valley with steep edges, the search will jump out of the valley by swimming through it without stopping. Contrary to this, if the locomotive steps are too small, convergence can be slow, but if search finds any local minimum it may not deviate too far from it. Clearly, the BFOA does not gives an expected start off, but if a valid push is obtained, it can proceed very well thereby leading us to the optimal solution.

Considering both the situations, we can combine both the algorithms to bring up a hybrid approach in order to increase our performance and optimality. The genetic algorithm is used during the chemotaxis stage of the locomotion of the bacterium, which gives a certain direction to it, in order to move towards the optimized solution. Combining genetic algorithm and bacterial foraging optimization algorithm was a perfect match due to the selection of the fittest behavior of both algorithms. The hybridized algorithm considers a complete timetable solution in the form of a variable length chromosome. Each structure (chromosome) consists of all the parameters namely, room number, professor, department, courses, etc. The availability of slots in a day is encapsulated in the form of a vector, which is identified by a key parameter named as room and thus maintaining the liberty in accessing the time slots. In a single slot, no. of classes can be considered.

A. Software Design

During the generation process, we need to be careful of some conditions and constraints required to fulfil optimality of the problem. Active rules are applied for each constraint. As mentioned in earlier, constraints are classified into 2 fields [3]:

Hard constraint: The following 4 constraints should be necessarily satisfied the solution which is obtained using these algorithms:

1. A student should have only one class at a time.
2. A teacher should have only one class at a time.

3. A room should be booked only for one class at a time.
4. Some classes require classes to have equipment. For ex: audio visual equipment, projectors, etc.

Soft constraint: These are the constraints that are of no great concern but are still taken into contemplation. They do not need to be satisfied but the solutions are considered to be good if they are satisfied:
1. Courses must be eventually distributed.
2. Student should not have any free time between two classes on a day.
3. Scheduling of teachers should be well spread over the week.

The program uses genetic algorithm initially to generate the population and the BFOA performs the chemotaxis and swarming to generate the optimized solution from the selected individual chromosomes, provided by the GA.

B. Implementation and Tools Used

The implementation of the program uses classes and their respective functions as mentioned below. For demonstration purpose and to save time during the demo, I am providing the input for program by hardcoded values. But the future scope of this project can involve taking a one-time input from the user in order to get personalized results.

Input class:

{[Rooms], [Max_capacity], [No. of students], [Time_slots], [Professor], [Courses]}

Output class:

{[Generation 1, Gen 2, Gen3,…, Gen n], [Table representation of fittest timetable after each generation]}

Fitness Function:

Calculate the fitness value based on the number of conflicts occurring for a timetable generated. I have considered two conditions that lead to formation of any conflicts in the timetable. The two parameters are as below:

a) If number of students attending the course is greater than max_capacity of the room. Then, increment conflict.

b) When two rooms are occupied at the same time, check if they do not have the same classroom and the course. And if the same instructor is placed at different classes at the same time. If yes, increment conflict.

I studied GA and BFOA from the programming aspects and used some code fragment for reference purposes which I have mentioned in the section for References. I have implemented the combination of both the algorithms by myself from the scratch. The tools and software I have used are Python Anaconda, Jupyter Notebook for python execution, vim. I have imported just 2 packages in my program, Random and PrettyTable. The former one for generating random individuals for population initialization and the later for displaying the output in a user-friendly tabular format.

C. Performance Evaluation

Genetic Algorithm Analysis:

If *N* is the total no. of chromosomes in the population

G is the number of generations until the termination condition

L is the size of each individual

S is the storage space for a population

Then,

Time Complexity = O (N*G*L)

May vary based on what fitness function and selection parameters are being used.

Space Complexity = O (2S)

Twice the storage since both old & new population needs to be saved.

BFOA Analysis:

If N is the size of the population

Sorting used for selection – Quick sort (nlogn)

Nc - # of Chemotactic steps, Nre - # of reproductive steps, Ns - # of swimming steps, Ned - # of elimination and dispersal steps.

S is the storage space for a population

Then,

Time complexity =

O(Nc*NlogN) * (O(Nre) + O(Ns) + O(Ned))

Time Complexity = O(NlogN)

Since size of population and step sizes remain constant, we can ignore them. Also, complexity may vary when different parameters are considered or depending on what algorithm is used for selection/sort.

Space Complexity = O(S)

Since the population remains constant throughout the process.

ATTACHMENTS

- https://youtu.be/ZP6F36sMZA4
- https://github.com/rashmi1112/CS575_Project/blob/master/GABFOA_tt.py
- https://drive.google.com/file/d/1-t3N3fUGOh15cTkGYX_i2sJ_Fq3s2Qte/view?usp=sharing

REFERENCES

[1] Anisha Jain, Ganpathy S C Aiyer, Harshita Goel, Rishabh Bhandari," A literature review on time table generation algorithms based on genetic algorithm and heuristic approach", International Journal of Advanced Research in Computer and Communication Engineering, Vol. 4, Issue 4, April 2015

[2] Kaustuv Nag, Tandra Pal, Member, IEEE, and Nikhil R.Pal, Fellow, IEEE," ASMiGA: An Archive Based Steady- State Micro Genetic Algorithm", IEEE Transactions on cybernetics, Vol.45, No.1, January 2015

[3] Dipesh Mittal, Hiral Doshi, Mohammed Sunasra, Renuka Nagpure,"Automic time table generation using genetic algorithm", International Journal Advanced Research in Computer and Communication Engineering, Vol. 4, Issue 2, February 2015

[4]https://towardsdatascience.com/continuous-genetic-algorithm-from-scratch-with-python-ff29deedd099

[5]https://stackoverflow.com/questions/60849278/genetic-algorithm-implementation-errors-time-table-scheduling