PROGRAM STRUCTURES AND ALGORITHMS

Time Table Scheduling Using Genetic Algorithm



**INFORMATION SYSTEMS**

JAY NIRMAL

KARAN KATHAYAT

# INTRODUCTION

The genetic algorithm is a method for solving both constrained and unconstrained optimization problems that is based on natural selection, the process that drives biological evolution. The genetic algorithm repeatedly modifies a population of individual solutions and selects the best solution for that particular generation.

# PROBLEM

We have created a genetic algorithm to optimize the time table schedule for a class. The schedule is for the professor, the course, the room number, group / batch of class and time and day schedule. The schedule should be such that there are no “collisions” or clashes in schedule for all the participating components. We have to optimize the schedule to fit in all participating components.

# GENETIC ALGORITHM

The genetic algorithm performs two basic operations:

1. Randomly selects ‘m’ pairs of parents from the current population and produces ‘M’ new chromosomes by performing crossover on its parent pair.
2. Randomly selects ‘M’ chromosomes from the current population and replaces them with new population. We don’t select chromosome for replacement if it is among the best chromosomes in the population for that particular generation.

These two operations are run in a loop until the best chromosome reaches a fitness value equal to 1 (meaning that all classes in the schedule meet the requirement). Thus we ensure that the best chromosome is picked so that we have an optimal solution for the time table scheduling.

When we make a classroom schedule, we must take into consideration many requirements.

1. Hard requirements (if you break one of these, then the schedule is infeasible):

* A class can be placed only in an unoccupied classroom.
* No professor or student can have more then one class at a time.

1. Some soft requirements (can be broken, but the schedule is still feasible):

* Preferred time of class by professors.
* Preferred classroom by professors.
* Distribution (in time or space) of classes for student groups or professors.
* Rest time for professor if he has taught more than one class in a day

# IMPLEMENTATION

Fitness Function : The fitness function calculates the fitness of a gene. We have kept this function recursive and evaluated the fitness of all genes using the expression :

**Double fitness = 1 / (double) ( 1 + number of clashes).**

After looping through all the genes, the “FITTEST” gene is returned.

Genetic Expression :

Mutation : A mutation operation is very simple. It just takes a class randomly and moves it to another randomly chosen slot. The number of classes which are going to be moved in a single operation is defined by the mutation size in the chromosome's parameters

We have designed a mutation function having objects of Population and Timetable. We loop through the current population with fitness and for a gene having fitness say ‘x’, create new individual and swap the genes. Once we have mutated, we check the fitness of the newly formed child gene and check if it is the fittest gene or not. If not, we add this gene to our population. Example :

PARENT | 2 | 4 | 5 | 3 | 8 | 5 | 9 | 5| 0 |

Swap index : 3 and 5

CHILD | 2 | 4 | 5 | 5 | 8 | 3 | 9 | 5 | 0 |

Crossover : We apply crossover to the newly formed population and find the fittest gene for the concerned population. We have to check now whether to apply crossover to this gene or not based on the elitism and crossover rate. If the conditions are met, we create an offspring and find the second parent for mutation by looping through the population again. We have used one point crossover in our algorithm. Example :

CROSSOVER POINT

↓

PARENT CHROMOSOME 1 | 0 | 1 | 0 | 0 | 0 | 1 |

PARENT CHROMOSOME 2 | 0 | 1 | 1 | 0 | 1 | 1 |

CHILD CHROMOSOME 1 | 0 | 1 | 1 | 0 | 1 | 1 |

CHILD CHROMOSOME 2 | 0 | 1 | 0 | 0 | 0 | 1 |

# 

# 

# 

# 

# 

# 

# 

# 

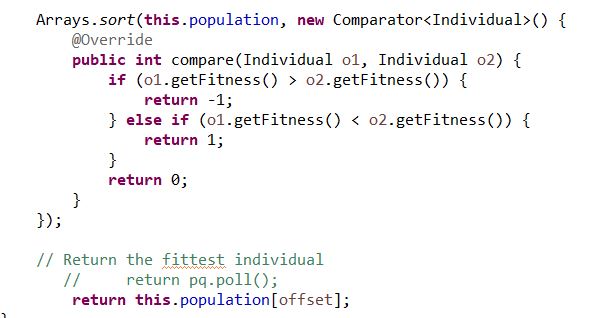
# CODE SNIPPETS

FITNESS FUNCTION :

# 

:

SORTING :



# 

# 

# 

# OUTPUT

# 

# 

# 

# 

# 

# 

# 

# TEST CASE

3 TEST CASES PASSED

↓↓↓↓↓

# 

# DATA

|  |  |  |  |
| --- | --- | --- | --- |
| TRIAL | POPULATION | GENERATION | FITNESS |
| 1 | 20 | 76 | 1 |
| 2 | 50 | 37 | 1 |
| 3 | 100 | 16 | 1 |
| 4 | 200 | 17 | 1 |
| 5 | 500 | 18 | 1 |
| 6 | 1000 | 13 | 1 |

# RESULTS

1. We noticed that the number of generations it took for the algorithm to find optimal solution fluctuate a lot depending on the population
2. After conducting test trials for the algorithm having population {20,1000}, the number of generations it took for our algorithm to achieve optimal solution greatly stabilized after crossing population count 100.
3. The execution time of the algorithm increases with increase in number of iterations.

# 

# CONCLUSION

Some difficulties are encountered when applying GAs to constrained combinatorial optimization problems. The most relevant of them is that crossover and mutation operators, since they may not always give a feasible solution.

# REFERENCES

1. Wikipedia
2. Tutorial Point
3. Coursera
4. <https://arxiv.org/ftp/arxiv/papers/1410/1410.4838.pdf>
5. <https://www.researchgate.net/publication/2253354_A_Genetic_Algorithm_To_Solve_The_Timetable_Problem>
6. GeeksForGeeks