Use of genetic algorithms in university schedule creation

Adrian Barbe
Stefan cel Mare University of Suceava

Introduction

A schedule is very important to manage an activity, especially when the activity was carried out in a large organization and held for long-term time or routine. Within the scope of the university, the course schedule is also critical.
University schedule is an entity that depends on multiple factors such a room availability, professor availability, or students' availability. Usually, there's a lot of manual work when creating the schedule and eventual conflict resolution in the schedule could affect a lot of people. That's why using programming algorithms is the best choice for solving this sort of challenge.
This is a task that could not be solved by the means of deterministic algorithms, or it will take too much time. Here's the place for use for such a heuristic algorithm as genetic.

Genetic algorithm

A genetic Algorithm is a heuristic method to search for an optimal solution for a problem that doesn't require the same result on every calculation or the best result. The method will find a good solution by the initial seed of the data (randomly) and eventual crossovers with another solution to create new solutions which could lead to a possible improvement. After that, a special method will apply the so-called mutation to the new solutions so that they have parts of a solution from the parents but not really the same as the parent. The process begins with the creation of a random population of valid solutions which are called chromosomes, then the genetic algorithm will calculate the fitness costs of each chromosome in the population. After that two chromosomes will be chosen to crossover and produce offspring. Then the offspring will be mutated. This process will be repeated till the stop condition is reached. This condition could vary from case to case. Some of them are reaching some generation number, after some time spent, fitness score stagnation or fitness score threshold. To not lose already found good solutions usually a selection mechanism is applied. There are a few of them: elitism, roulette wheel, stochastic universal sampling, tournament, and truncation. Sometimes a reinsertion logic is applied. When there are fewer offspring than parents, select the best parents to be reinserted together with the offspring.

The genetic algorithm can be represented as following major steps:
1. Represent the problem in a variable that forms a chromosome of a fixed length, choose the size of the population, the crossover probability, and the mutation probability.
2. Define a fitness function to measure the performance, or fitness, of an individual chromosome in the problem domain. The fitness function establishes the basis for selecting chromosomes that will be mated during reproduction.
3. Randomly generate an initial population of chromosomes of a predefined size.
4. Calculate the fitness of each individual chromosome.

5. Select a pair of chromosomes for mating from the current population. Parent chromosomes are selected with a probability related to their fitness. Highly fit

chromosomes have a higher probability of being selected for mating than less fit chromosomes.
6. Create a pair of offspring chromosomes by applying the genetic operators – crossover and mutation.
7. Place the created offspring chromosomes in the new population.
8. Repeat Step 5 until the size of the new chromosome population becomes equal to the size of the initial population.
9. Replace the initial (parent) chromosome population with the new (offspring) population.
10. Go to Step 4 and repeat the process until the termination criterion is satisfied.

Genetic algorithm design for university schedule problem

To represent the scheduling problem in the algorithm we need next entities:

- Professor (Id, Name)
- Course (Id, Name, LabRequired – a flag that indicates if computer class is needed)
- Room (Id, Name, IsLab, NumberOfSeats) – the classroom used for the classes
- Groups – a list of Group (Id, Name, NumberOfStudents)
- NumberOfSeats – a calculated field which results from the added groups
- Duration – the duration of the course (default = 2)
- Hour – represents the hour in a day from 8 to 20
- Day – represents the day in the week from 1 to 5 (1 corresponds to Mon and 5 – to Fri)

Chromosome design

Our chromosome represents the so-called CourseClass. This is a collective entity that has reference to Course, Groups, NumberOfSeats, Duration. Since the course and the assigned groups to them are predefined (and thus the number of seats and duration) these fields are populated during the initial seeding and are not changed during the crossover of mutation. The variable chromosome fields are Professor, Room, Day, and Hour. During the seed, a random even hour number is assigned because we have an initial restriction of a hard-defined hourly grid.

Crossover design

We used a uniform crossover. The Uniform Crossover uses a fixed mixing ratio between two parents. Unlike one-point and two-point crossover, the Uniform Crossover enables the parent chromosomes to contribute to the gene level rather than the segment level. If the mixing probability is 0.5, the offspring has approximately half of the genes from the first parent and the other half from the second parent, although cross-over points can be randomly chosen. We considered using Alternating-position and Cut and Splice crossover. The crossover method is a matter of fine-tuning, and we will consider this in the following studies.

Mutation design

We started our algorithm design by using the Uniform mutation method. This mutation method replaces the value of the chosen gene with a uniform random value selected between the user-specified upper and lower bounds for that gene.

For the selection, we used the Elite selection method. It selects the chromosomes with the best fitness and a small portion of the best individuals in the last generation is carried over (without any changes) to the next one.

Fitness calculation design

For the fitness calculation, we used 5 criteria – whether the class is a lab (IsComputerEnough), whether the seat amount in the classroom is enough (IsSeatEnough), and if there is no overlapping between rooms (IsRoomNotOverlapped), professors (IsProfessorNotOverlapped), and student groups (IsStudentNotOverlapped). Each positive result for criteria increases the score by 1, and each negative result resets it to 0, except IsComputerEnough criteria which just divides by 2 the existing score. The evaluation function for the chromosome returns a value from 0 to 1.0 which is a result of the score divided by the number of the criteria.

Algorithm termination

An important but often overlooked moment is the criteria of termination. We've chosen 2 criteria that should be triggered together – Fitness Threshold Termination and Fitness Stagnation Termination. Fitness Threshold Termination means that the genetic algorithm will be terminated when the best chromosome reaches the expected fitness. Fitness Stagnation Termination – the genetic algorithm will be terminated when the best chromosome's fitness has no change in the last generations specified.
Since we cannot be satisfied with a schedule where there are overlaps or other problems, we've chosen Fitness Threshold Termination as 1.0. To make sure that there are no possible other solutions we set stagnation termination to 5 generations. In future studies, we plan to use other best chromosomes to return multiple best results.

Results

We used 5 test items for the initial estimation of the algorithm. We used a population of 150 to a maximum of 320 chromosomes. We were able to increase the effectiveness by increasing the population count to 100+ chromosomes. Usually, we were able to obtain the best chromosome with fitness 1.0 during a maximum of 50 generations in less than 0.8 seconds on a 2.6GHz Core i7 CPU. The average number of needed generations was 25 and the time – was 0.47 seconds.
We generated results output in a form of an HTML page which allowed us to analyze better the results.

Conclusions

The use of a genetic algorithm allowed us to create a simple scheduling system that could be improved without linear code complexity increase. This basic algorithm has a lot of room for improvement – e.g., optimize the time slots allocation, or optimize the course distribution according to their complexity. But, at the same time, the obtained solution is viable and usable and could be used as a part of a complex university enterprise resource planning system.

Project GitHub link – https://github.com/adrianbarbe/UniScheduleGA