Project Report

The program employs a genetic algorithm to optimize exam scheduling by maximizing the fulfillment of various constraints. For this I have utilized various techniques and logics through trial and error which have yielded satisfying results.

Initially, random time slots, classrooms, and invigilators are assigned to exams. Each exam is represented by an object storing its details like course code, invigilator, classroom, and start time. These exam objects are organized into schedules, forming a population of schedules. The fitness function checks for all hard and soft constraints and returns a value in negative. The smaller the negative value, the worse the fitness. All constraints are written from scratch and are easy to understand. Violation of a hard constraints incur a higher penalty (1 point) for violation compared to soft constraints (0.5 points). The smaller the negative fitness value, the better the schedule's compliance.

With this, I run the program for a given amount of generations, in each generation. I first pick 2 parents using roulette wheel function, population // 2 times. This is because 2 parents are required to breed which halves the population. Roullete_wheel_selection function uses random.choices to randomly select a parent based off it's probability which is found using it's fitness. Roullete_wheel_selection2 basically picks a number between 0 and total fitness and then adds fitness of each schedule until the summed fitness is greater than the randomly picked number. The latest schedule who's fitness crossed the threshold is considered as a parent.

I use a crossover function from one of the 4 crossover functions I made. From hit and trial method, it would seem that crossover2 and crossover4 work best. Crossover2 splits the parents in 3 segments and swaps the segment in the middle. Crossover4 applies a probability on each bit to see if the bits of both parents should be swapped or not. Then the children created after the crossover are run through a probability to see if they will mutate or not. There are 4 mutation functions. Through hit and trail, I found mutation2 to perform the best in most cases. Mutation2 simply removes a random gene (Exam object) from the chromosome (Schedule object) and places it in a random index in the chromosome.

After fitness evaluation of children, they are added to the population. The population is sorted by fitness, and only the fittest schedules are retained, maintaining the population size through slicing. This process iterates for a specified number of generations, with each generation displaying the best schedule. The schedule with the highest fitness across all generations is selected as the best schedule and presented in a tabular format.

By : Muhid Qaiser (22i-0472) and Ahmed Zubair (22i-0525)