Dominik Czarnota

Modular exam scheduling problem with genetic algorithms

Abstract

Solving exam scheduling problems for large set of data is a complex problem which is usually solved manually by specific person on universities. Such process often takes days or even weeks of iterations of finding better and better solution. This work describes implementation and results of work presented in [1] using DEAP (Distributed Evolutionary Algorithms) library in Python [2].

Modular exam scheduling problem

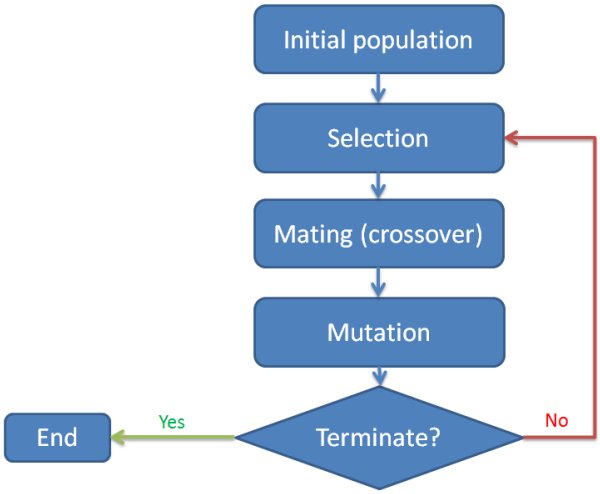
In MESP it is assumed that *e* exams have to be distributed into *t* timeslots such that the chosen terms will be satisfied. That means that no student will have two exams at the same time and that it is unlikely to happen that a student has two exams one after another. The number of constraints may vary depending on the university, number of courses, exams, rooms and so on.

The computational complexity of such problem is t^e. For example in the given testcase, there are 20^{18} possible solutions. Searching all the space for the best solution would take a lot of time and because of that I have decided to solve it with genetic algorithms.

Genetic algorithms

Genetic algorithm is a type of evolutionary algorithm which is a search heuristic that mimics the process of natural selection in biology. This method is often used to find satisfying solutions to complex problems which cannot be solved using standard methods because of too big computational complexity. The word “satisfying” was used because the solution found using GA may not be the global maximum but a good enough one (which might be a global maximum).

Schema of genetic algorithm:



The population is a set of individuals (chromosomes). Each individual represents a solution to the given problem. The initial population is a population of randomized individuals.

Next, a number of genetic operators is used till the termination condition is met. The usual genetic algorithms consists of selection, crossover and mutation operators.

The selection operator chooses *N* individuals which are then cloned to fill up new generation. The crossover operator takes two individuals and mixes them up (e.g. swaps half of individuals). The mutation operator takes one individual and randomly changes some of its parts (e.g. flips a bit or randomize value in particular range).

The terminate constraint can be anything – usually it is a condition based on execution time (e.g. this must return results in 3 minutes) or number of generations in which the algorithm didn't find better individual.

DEAP

"DEAP is a novel evolutionary computation framework for rapid prototyping and testing of ideas. It seeks to make algorithms explicit and data structures transparent. It works in perfect harmony with parallelisation mechanism such as multiprocessing and SCOOP." [2]
The library doesn't define complete approaches/algorithms that would require just defining the evaluation (fitness) function and the individual representation. Instead it gives some abstraction to register general functions in the toolbox like genetic operators (mutation, crossover, selection) and then it is up to the user to define the genetic algorithm loop and do whatever they want inside. This approach gives flexibility in testing different methods or parameters because everything is configured in one place. The toolbox also defines the most general genetic operators like one/two point crossover, flipping bit mutation or tournament selection.

Individual representation

In this project, the genetic algorithm individual is represented as a list of integer numbers of length *e* (number of exams to be scheduled), where each element is a number between 1 and *t* (the number of timeslots available). Each timeslot represents a different term on which an exam can occur (e.g. Monday 12:00). The position of number in the list corresponds to particular exam.

For example, having such data:

```
exams = ["Physics", "Math", "Programming"]
individual = [5, 2, 3]
```

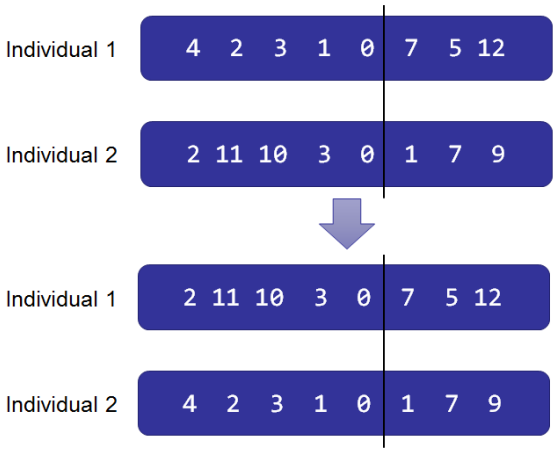
means that Physics occurs at timeslot 5, Math at 2 and Programming at 3.

The mapping between timeslot number and day and hour depends on the settings – one can change "*timeslots_per_day*" variable to determine how many terms are there for exams on each day. In this project this is set to 4. The number of days is defined in "*DAYS*" variable and is set to 5.

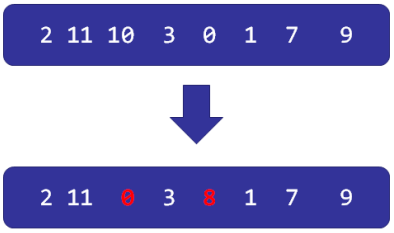
An alternative representation of individual could be that the position in the chromosome (individual) represents the timeslot and the element is a set of exams. This solution has been proposed at [3]. However this leads to situation where it is hard to define operations such as crossover that would not produce invalid individuals.

Chosen operators

For this algorithm I have chosen one point crossover, which occurs with a given probability. It swaps one part of two individuals until randomized crossover point (black line).



The mutation I have used changes randomly chooses an exam and randomizes its timeslot value (randomizes value between 0 and *t*).



I have tested four different selection methods – choosing N best individuals, choosing random N individuals, choosing individuals based on a tournament (chooses *k* individuals from the population at random and with a given probability chooses best individual from the pool/tournament) and also a roulette selection (the better the individual is the bigger is the chance it will be reproduced to the new generation).

Evaluation/fitness function

The evaluation function takes an individual and returns an inversion of *punishment* value. Particular individual is calculated by adding up all punishments and returning decimal value of *1.0 / (1.0 + punishments)*. The punishments are defined as:

```
STUDENT_TAKING_TWO_EXAMS_AT_ONCE = 500
STUDENT_TAKING_MORE_THAN_TWO_EXAMS_IN_ONE_DAY = 10
STUDENT_TAKING_EXAMS_ONE_AFTER_ANOTHER = 5
STUDENT_TAKING_TWO_EXAMS_IN_ONE_DAY = 3
```

Test case

The test case consists of 230 students:

- 100 students from first year
- 70 students from second year
- 50 students from third year

Each year has its own 4 separate exams. Each student also has one of three modules – which consists of two subjects.

To add some more reality to the data, 10 of second year students also have one exam from first year and 5 of third year students have one exam from second year.

Program

The program has been written in Python 2.7 using DEAP module as a dependency (it can be installed using Python Install Package by *pip install deap*).

There are two scripts that can be launched with Python (*python <script>*) in the src directory:

- Main.py – launches the algorithm once with a given parameters (defined in the file)
- Find_best_params.py – launches the algorithm with a range of different parameters (defined in the file) – prints out the results onto standard output

The exams are defined in exams.py file and the students data in testcase.py file. The genetic algorithm is defined in ga.py file.

Results

A set of different parameters ranges has been tested:

```
mutation_probabilities = (0.01, 0.03, 0.05, 0.1, 0.2, 0.3)
mutation_change_probabilities = (0.01, 0.03, 0.05, 0.1, 0.2, 0.3)
crossover_probabilities = (0.01, 0.05, 0.1, 0.2, 0.3, 0.5)
select_methods = (select_best, select_random, select_roulette, select_tournament)
```

Out of 6*6*6*4 = 864 algorithm runs (with 100 individuals in population and 50 generations/evolutions) only 38 runs got the best fitness function value of 0.00135869565217 which is 735 punishment value.

Example result that gave this solution – bestSelection, 0.1 mutation probability, 0.03 mutation change probability, 0.5 crossover probability:

```
Printing individual
Raw: [2, 13, 0, 19, 17, 8, 13, 3, 15, 1, 18, 12, 10, 6, 11, 4, 11, 4]
Fitness: (0.001358695652173913,)
(735, True, {'exam after another': 0, 'two exams at one day': 245, 'two exams at once': 0, 'more than two exams at one day': 0})
Exam Algebra liniowa i geometria analityczna is on day 0 - timeslot 2
Exam Analiza matematyczna is on day 3 - timeslot 1
Exam Matematyka dyskretna is on day 0 - timeslot 0
Exam Fizyka 1 is on day 4 - timeslot 3
Exam Programowanie obiektowe is on day 4 - timeslot 1
Exam Architektury komputerów is on day 2 - timeslot 0
Exam Systemy dynamiczne is on day 3 - timeslot 1
Exam Fizyka 2 is on day 0 - timeslot 3
Exam Przetwarzanie obrazów cyfrowych is on day 3 - timeslot 3
Exam Bazy danych is on day 0 - timeslot 1
Exam Lingwistyka formalna i automaty is on day 4 - timeslot 2
Exam Analiza i modelowanie oprogramowania is on day 3 - timeslot 0
Exam extra1 is on day 2 - timeslot 2
Exam extra2 is on day 1 - timeslot 2
Exam extra3 is on day 2 - timeslot 3
Exam extra4 is on day 1 - timeslot 0
```

Exam extra5 is on day 2 - timeslot 3
Exam extra6 is on day 1 - timeslot 0

The individual is not invalid (it would be, if it had one or more “two exams at once” value). There are 245 “two exams at one day” – this is correct, as since there are only 5 exam days and 230 of students have 6 exams and 15 of students have 7 exams, so such situation when there are two exams at one day must occur at least 245 times. The timeslot presented here is a “daily timeslot” – so 0 corresponds to 8:00-10:00, 1 to 10:00-12:00 and so on.

Also as students from different years have different modules (so some students have extra1 and extra2 exams, some have extra3 and extra4 and some have extra5 and extra6) those exams were placed on different days, so it is less likely that they would produce “more than two exams at one day” punishment.

Full log of the results can be seen in “full_results.txt” file (along with printed individual). The parameters for the best found results can be found in “best_results.txt” and “just_results.txt” contains all results’ parameters along with fitness function value.

References:

1. D. Corne, H.L. Fang, C. Mellish “Solving the Modular Exam Scheduling Problem with Genetic Algorithms”, DAI Research Paper No. 622
2. https://github.com/DEAP/deap
3. Abramson & Abela “A Parallel Genetic Algorithm for Solving the School Timetabling Problem”, Technical Report, Division of I.T., C.S.I.R.O, April 1991