Time Table Optimisation

Jason Alvin Chesney | Lajiya Aleena Saji | Shivani M
Sanjay K M | Alvina Joanna

As part of our mandatory practical paper [BD1P4] Computing for Data Science Lab in the Master's course of Big Data Analytics at St, Joseph's University; our group was asked to implement an algorithm to better understand the working and applications of the same.
We chose to implement the genetic algorithm on a time table in order to get optimal time tables that had no classes. In this report we detail our process and outcome.

Problem statement

To build a model using Python that can optimize a time table for collision-free schedules for teachers and students.

Data collection

We selected a schedule with pre-included collisions in order to showcase the working of the algorithm. The time tables we considered had the following:

- Three teachers
- Three slots per day
- Three teaching days in a week

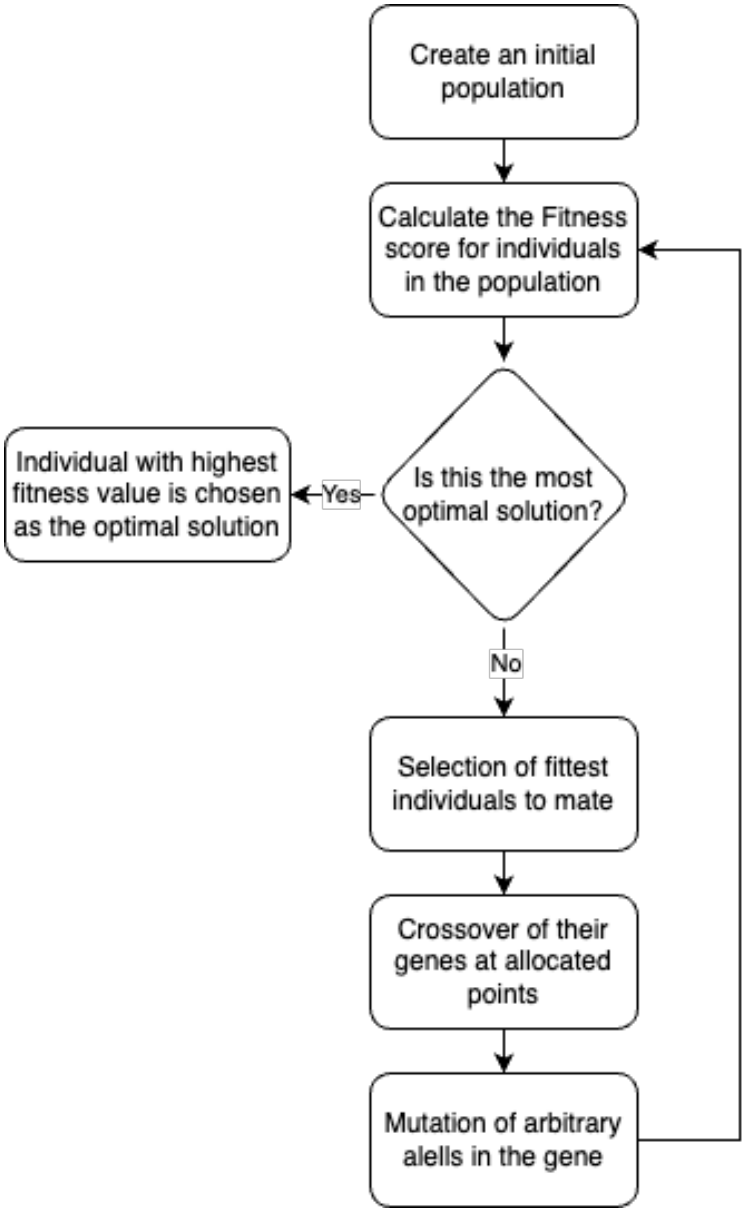
Data contraints

In the timetables that were generated and compared, we assumed the following as constraints.

- Hard constraints:
  - Two teachers cannot teach one class at the same time (in the same slot)
  - Two classes cannot be taught by one teacher in the same slot
- Soft constraints:
  - Each teacher is allowed at most three slots per week

Implementing the genetic algorithm

The algorithm is a heuristic search-based technique aimed at finding the most optimal solution based on the principle of evolution and biology.



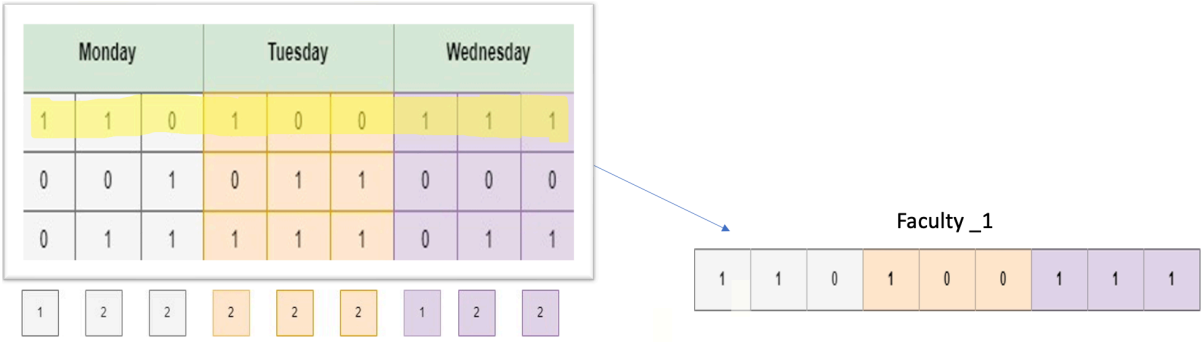
1. The initial population was composed of the timetable we had given as input with the parameters decided before hand. The timetable is then broken down to code each solution to a unique binary string with each bit in the string representing a parameter that we have.

2. We calculate the fitness function (a function used to measure the capability of an individual to compete with the competition in that population). The formula we used was:

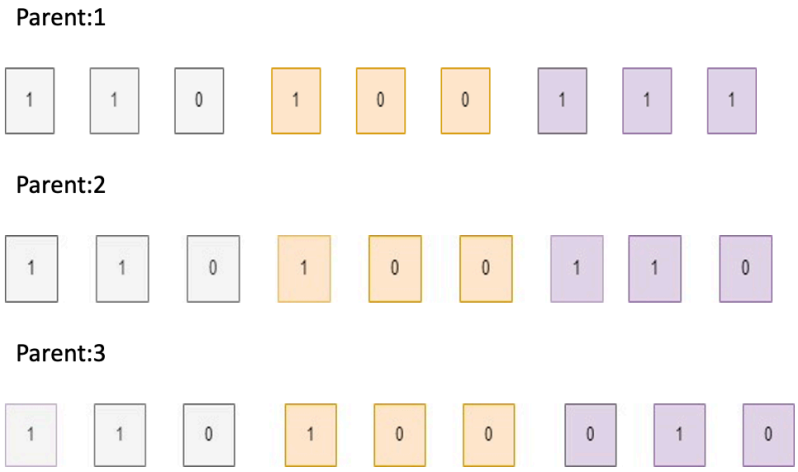
$$Fitness = \frac{1}{(1 + Colission\ count)}$$

Where, no collisions would give a perfect score of 1 and one or more collisions would give a score between 0 and 1 with the value indicating the probability that the individual will get selected in the next iteration/ generation.

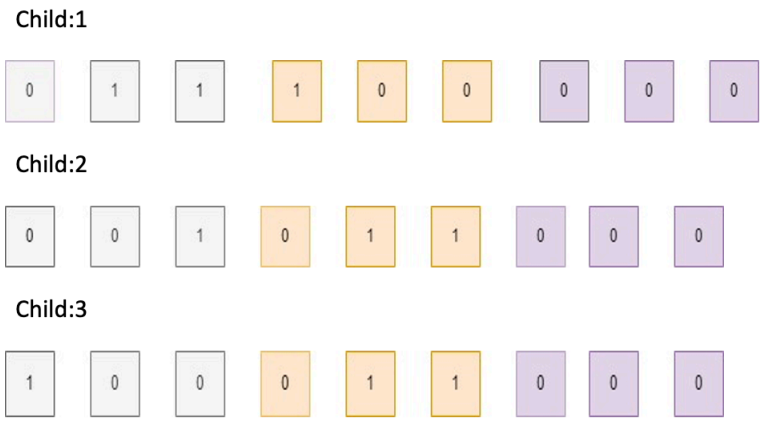
3. Selection was then done line by line for each individual



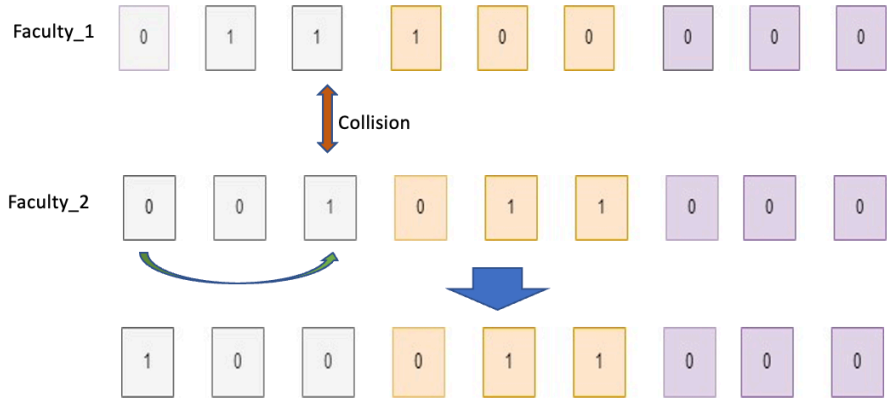
4. Next, the crossover is done by interchanging parts of the binary string at random points



This then results in offspring as below



5. Mutation is then introduced using the technique of one-bit flip where one allele individual's chromosome is flipped on the condition of the existence of a collision. This is done to ensure the we are getting our most optimal and collision-free solution.



These steps are iteratively executed to finally produce the optimal result (here, the optimal timetables)

Future work

This project was a very interesting project to work on in order to truly understand the algorithm, in the future such an application could be used in the following manner:

- In academic institutions to automate scheduling without having to manually consult with those involved for every collision
- Scaling of this project to include multiple parameters including number of classrooms required, number of students in a class, etc. could help with making the model more appropriate for a real-world situation
- Investigating other methods to implement the steps of the algorithm from the encoding of the individuals of the population to the type of mutation to be introduced in order to have a more efficient process

Acknowledgements

We're grateful to our faculty Mr. Amandeep Singh Khanna for the opportunity to work on this project and to those who contributed towards making this possible. Special mention of gratitude to Mr. Praveen Kumar, Ms. Jayati Kaushik and our loving classmates.

Reference literature

- A Review of Optimisation Algorithms for University Timetable Scheduling by students of the Umm Al Qura University
- Learning Genetic Algorithms with Python by Ivan Gridin