Programming a GA for CSP

Authors: Kenyon Leblanc, Wesley Bafile, David Wolfe

Scheduling with a Genetic Algorithm

We modeled our genetic algorithm on Model 5: Teacher Satisfaction Model. The constraints for this model include time conflicts for teachers and classrooms, the minimum/maximum classes that a teacher can teach, and time slots for credit hours. Preferences for this model include the board type that a teacher prefers (chalk or whiteboard), the time of day that a teacher prefers to teach, days of the week that a teacher prefers to teach, and specific class sections that teachers like or dislike teaching.

Our hypotheses were designed to be similar to the Teacher Satisfaction page. While using different dimensions, we based our hypotheses on the idea of filling up each of the class sections with teachers, classrooms, and times. If we treat classes as our variables, we can fill them with teacher, classroom, and time domains. We initially thought about doing the opposite, treating classes as domains, but it made the development process easier to interpret the problem as 3 smaller domains instead of one larger, more complex domain.

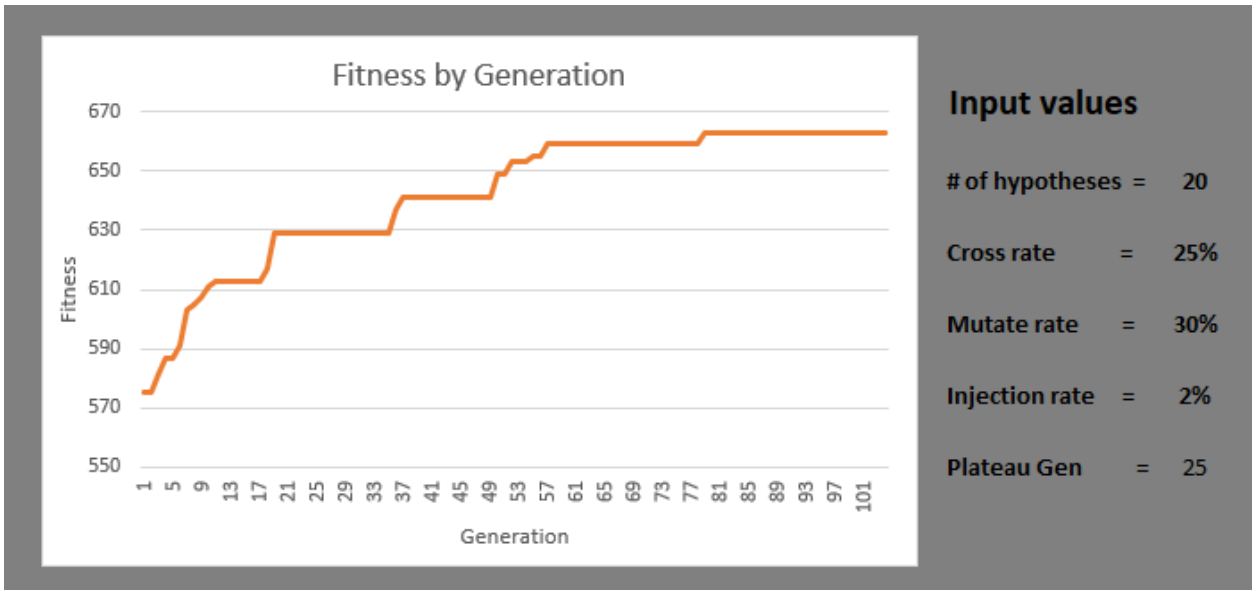
A hypothesis is a two-dimensional Python list that has the following dimensions: 29 rows, which represent the total number of courses; and 3 columns, which store the teacher, classroom, and time slot for each class section. By designing the hypothesis to build in row-major order, this made it so we could validate a hypothesis as it was built; if a row were to fail to meet a constraint, it would be tossed out and replaced until it met all constraints. This saves a lot of time and memory so entire hypotheses don’t have to

be built and scrapped until they are validated. We use the function check_row() to validate a hypothesis as it's being built, and validate_hypothesis() when an entire hypothesis needs to be checked. Validating the entire hypothesis is used when the population is crossed or mutations are introduced.

We evaluate the performance of each hypothesis in our fitness_check() function. The weights we used for teacher preferences were on a scale of 3/5/7, where 3 represents disliking, 5 is neutral, and 7 is liking. These weights were used for board, time of day, and days of week preferences. The original scores given to these categories in the sample data were 0/1/2, but we raised these scores in order to weigh them similarly to the course section preferences. The weights we used for course section preferences were 2/4/6/8, where 2 represents least preferred, and 8 represents most preferred. The sample data represents preferred courses in a way that a lower score means more preferred, but we reverse that in order to represent the fitness score as a growing number. We chose to go with an increasing fitness score to make it easier to understand. For a data set with these dimensions, domains, and variables, the theoretical maximum score is 841, which is the sum of all possible most preferred scores. This number is merely the theoretical best, as the actual maximum score obtainable is almost certainly different; this was shown in our experimental data. 841 represents the perfect score where every teacher loves their schedule, which is highly improbable since teachers have preferences.

We gauge hypotheses by their performance in fitness_check(), and sort them from greatest to least. We then take a predetermined ratio of the sorted list to move on to the next generation, and delete the rest. Generations are sorted before injections,

crossovers and mutations take place. This is done in elite_selection(), which sorts hypotheses from best to worst, and then gets rid of the non-elites. Crossover and mutation are designed to be fairly simple. Crossovers choose a random amount of points from each parent to compile into a new child. This is simulated by cloning a parent, choosing random points in the child hypothesis, and replacing them with the other parent's gene. Mutations are simulated by selecting a random number of genes in a hypothesis, and replacing them with randomized valid values. Injections are an occurrence where a new, randomly generated hypothesis replaces an existing hypothesis in the population. This serves a similar role to mutation in the sense that allows the introduction of variety into the gene pool, however the injection rather than changing a single point can introduce a whole new schedule. This has a much larger impact and can produce the variety needed to get out of a dead end path. We achieve injection by beginning with an injection rate, this is a chance that any given hypothesis could be replaced. If it is selected we replace it by using the generate_hypothesis() function.

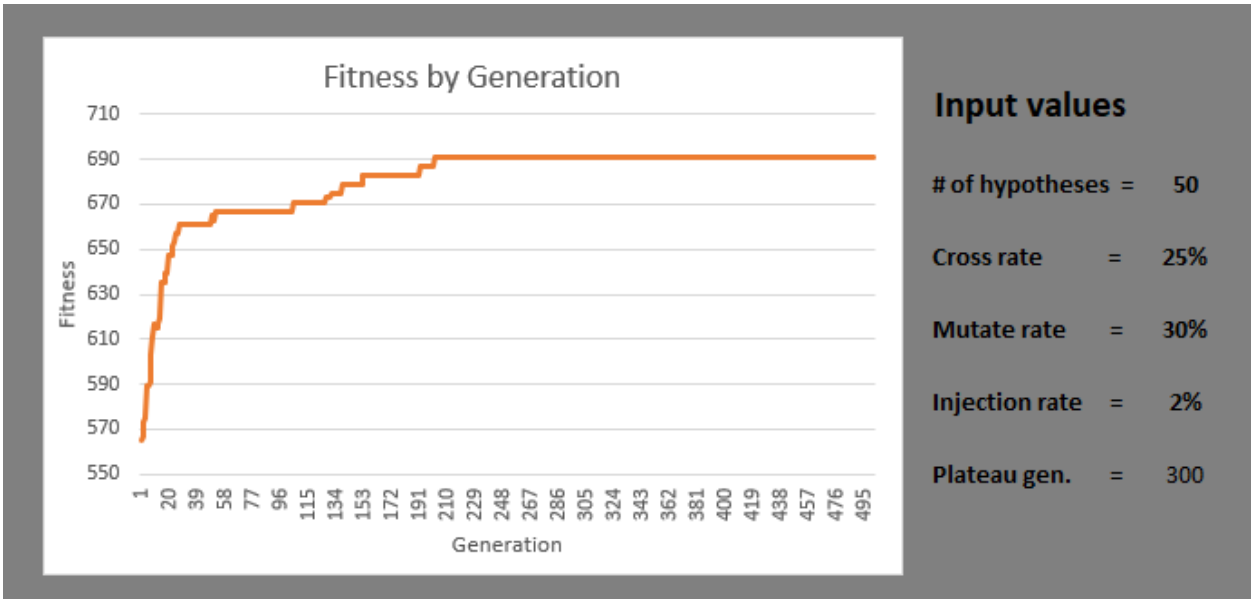


We decided to create a GA model where the population size can be changed to increase accuracy, at the cost of increasing time complexity. We found that a population of 50 hypotheses was a good population size given that it had a good balance between accuracy and time taken to produce a generation. We went with a crossover rate of 25% to have the population shift towards the more elite of each generation. We chose a mutation rate of 30% to reduce the chance of genetic drift, which is caused by too much bias towards the elites of a population. A mutation rate like this helps to decrease the chance of fitness scores plateauing. We also decided on an injection rate of 2% in order to not randomize the population too much, while also adding variability to prevent genetic drift.

The original paper took a different approach on scoring metrics. Thach chose to minimize the objective function, while we chose to maximize it, with a theoretical maximum score of 841. Thach's data also had a different rating scale, using 0-5. Our simulated data was scored from 2-5, and every course was marked, meaning we have to assume that every teacher was qualified to teach every course. Much like Thach's Model 5, we did not have to account for pure or applied classes, so we share a similar objective function. The main difference in our objective functions was the approach to producing such functions, as the original paper opted for a linear programming approach, while we used a genetic algorithmic approach.

After running our program the schedule is printed to the terminal as well as written to a CSV file, making the schedule easy to read. The program also writes the fitness and generation to a CSV, allowing us to look at the changing fitness score as generations progress. Examples of the outputs can be seen below.

Monday	Tuesday	Wednesday	Thursday	Friday
[Teacher: 3] [CRN: 1] [Room: 3] [Time: 15:00-16:50]	[Teacher: 2] [CRN: 1] [Room: 1] [Time: 14:30-15:45]	[Teacher: 3] [CRN: 1] [Room: 2] [Time: 17:30-18:45]	[Teacher: 2] [CRN: 1] [Room: 1] [Time: 14:30-15:45]	[Teacher: 3] [CRN: 1] [Room: 3] [Time: 15:00-16:50]
[Teacher: 3] [CRN: 1] [Room: 2] [Time: 17:30-18:45]	[Teacher: 1] [CRN: 2] [Room: 3] [Time: 07:00-08:15]	[Teacher: 9] [CRN: 3] [Room: 2] [Time: 07:00-08:05]	[Teacher: 1] [CRN: 2] [Room: 3] [Time: 07:00-08:15]	[Teacher: 2] [CRN: 2] [Room: 10] [Time: 11:30-12:45]
[Teacher: 2] [CRN: 2] [Room: 10] [Time: 11:30-12:45]	[Teacher: 1] [CRN: 2] [Room: 3] [Time: 10:00-11:50]	[Teacher: 5] [CRN: 3] [Room: 11] [Time: 19:00-20:50]	[Teacher: 1] [CRN: 2] [Room: 3] [Time: 10:00-11:50]	[Teacher: 9] [CRN: 3] [Room: 2] [Time: 07:00-08:05]
[Teacher: 9] [CRN: 3] [Room: 2] [Time: 07:00-08:05]	[Teacher: 5] [CRN: 3] [Room: 7] [Time: 17:00-18:50]	[Teacher: 9] [CRN: 4] [Room: 5] [Time: 10:45-11:50]	[Teacher: 5] [CRN: 3] [Room: 7] [Time: 17:00-18:50]	[Teacher: 5] [CRN: 3] [Room: 11] [Time: 19:00-20:50]
[Teacher: 9] [CRN: 4] [Room: 5] [Time: 10:45-11:50]	[Teacher: 3] [CRN: 6] [Room: 4] [Time: 13:00-14:50]	[Teacher: 10] [CRN: 4] [Room: 1] [Time: 15:00-16:50]	[Teacher: 3] [CRN: 6] [Room: 4] [Time: 13:00-14:50]	[Teacher: 9] [CRN: 4] [Room: 5] [Time: 10:45-11:50]



Taking a look at the graph, we see our fitness plateau as generations get high. This indicates that we are approaching the most optimized solution, however we may never meet it. Our fitness score appears to show logarithmic growth, eventually getting to a point where it is no longer increasing anymore.