Artificial Intelligence - Course Project Fall 2018

Solving and Optimizing Academic Timetabling as a CSP with Genetic Algorithm

Aminah Ankoush and Khulood Sabri

Birzeit University (BZU), Department of Electrical & Computer Engineering

1152161@student.birzeit.edu, 1150697@student.birzeit.edu

1. Introduction

Developing timetables for scheduling courses of educational institutions is considered to be one of the most exhausting and well-known search problems. Due to its huge state space, it cannot be solved using classical search methods. There are various approaches to solve it. Experiments have shown that local search methods give reasonable solutions to this problem. This paper will discuss an approach of representing timetabling as a constraint satisfaction problem that is then solved using the Genetic Algorithm (GA).

2. Problem specification

University timetabling problem could be defined as a set of constraints that are usually divided into two categories: "hard" and "soft". Hard constraints are rigidly enforced while Soft constraints are those that are desirable but not absolutely essential. These constraints need to be satisfied with limited, available resources. Basically, there are four elements that compromise university timetable: courses, lecturers, classrooms, and time slots.
In our problem, we consider real-world constraints that are encountered at our department of Electrical and Computer Engineering. The input provided to this problem is:

- A list of all courses and labs offered by the computer engineering at this semester
- A list of faculty members of the computer engineering program with a list of five favorite courses and five labs for each instructor.
- A list of time slots, where classes in the SMW are one hour, classes in the TR are 1.5 hours, and labs are three hours. The labs can be assigned on SMW at 2:00 or TR at 8, 11, or 2.
- A list of rooms and labs available in Al-Masri building and in other buildings.

The set of constraints required will be discussed in details later.

3. Problem Formalization

3.1. Chromosome Representation

To solve the timetabling problem using genetic algorithm, the schedule elements need to be represented as a chromosome. In our chromosome representation, every gene header represents a course's section. The value of gene is a sequence of three values: *Instructor*, *Room* and *Time Slot.* Figure 1 illustrates this structure.

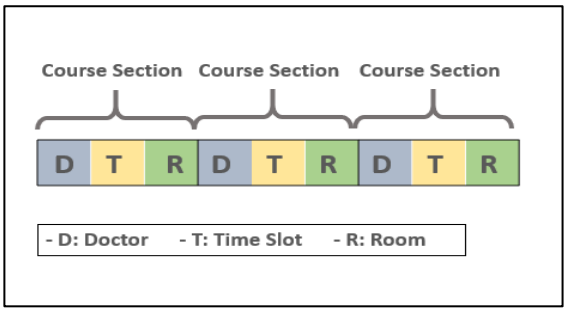

Figure 1: Chromosome Representation

According to this representation, the length of the chromosome is given by the following formula:

$$\textbf{\textit{Chromosome Length}} = \sum_{i=0}^{n} \boldsymbol{S_{ci}} * \mathbf{3}$$

Where n is the number of courses and S is the number of sections for course c_i.

3.2. Population

An initial population of size 1000 is generated. Each section is assigned randomly to one instructor who prefers to teach this course. This constraint helps in increasing the fitness of the chromosome and so decreases the number of iterations needed to converge to an acceptable solution. The other two values of gene (room, time slot) are assigned completely to random values. Through genetic algorithm, Population grows to a maximum size of 3000. When the population size exceeds 3000, a number of chromosomes are removed according to a criterion discussed later, in order to maintain the size limit to 3000 chromosomes.

3.3. Parents Selection

Two chromosomes are selected every iteration to generate two new children (off-springs). The tournament algorithm is used to determine the parents. In this algorithm, a random sub-group is selected from the population and the best two chromosomes of it are chosen. The size of the sub-group here is 60% of the population size. By experiment, smaller sizes give worse results since the randomness in selection increases.

3.3. Crossover Operators

Crossover is applied with a high rate equal to 0.6 in order to continuously generate new off-springs. Two types of crossover were used with equal probabilities (0.5 for both):

- Multi Point Crossover: where two points are chosen randomly, and the segment between these two points is swapped between the two parents.

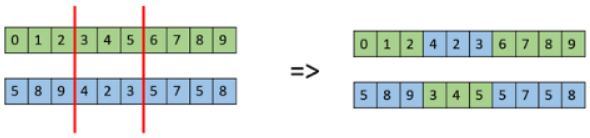

Figure 2: Multipoint Crossover

- Uniform Crossover: where randomly some genes from different locations are swapped between the two parents.

Figure 3: Uniform Crossover

These two types of crossover are used rather than the classical one-point cross over, in order to explore more solutions. Experimentally, they give better results in our problem.

3.4 Mutation Operators

Mutation is used with a lower rate equal to 0.3, in order to avoid convergence to local optima and get new children. As in crossover, two types of mutation are used with equal probabilities:

- Replace one randomly selected gene in the parent, by another randomly generated gene.

- Replace one randomly selected value of a gene in the parent by another randomly generated value. (i.e. change either instructor, time slot or room value of a gene rather than changing the whole gene).

3.5. Removing Criteria

As said earlier, there is a need to get rid of some chromosomes in order to avoid infinitely growing population. Here, the tournament algorithm is used again but this time to decide which chromosomes will be removed. Appling this algorithm leads to removing the worst chromosomes of a randomly selected sub-group. The purpose is to get some randomness but never remove chromosomes with really good fitness. The determined size of the selected subgroup is 20% of the population size. Larger sizes experimentally do not lead to any better results.

3.6. Fitness Function

Fitness function evaluates how close a given solution is to the optimum solution of the desired problem. Each chromosome is given a score out of maximum fitness 100%. 60% of this fitness is given to the hard constraints (equally weighted -15% for each) and 40% to the soft constraints (distributed subjectively between them). Soft fitness is not computed till all hard constraints are satisfied. Otherwise, their fitness value has no meaning.

3.6.1. Hard Constraints

1. There are no conflicts in assigning courses for the same instructor. The mathematical measurement of this constraint is defined by the equation:

$$fitness = \frac{number\ of\ conflicted\ sections}{total\ number\ of\ sections} * 15\%$$

2. The assigned courses and labs for each instructor are from his list of favorites.

$$fitness = \frac{\sum_{i=0}^{n} isFromFavorite(C_{Si}, Di)}{n} * 15\%$$

where *n: total number of sections for all courses.*
C: Course.
S: Section.
Di: Instructor assigned to section Si.

$$isFromFavorite(C, D) = \begin{cases} 1, & D\ teaches\ C \\ 0, & O.W \end{cases}$$

3. * Each instructor teaches at least 6 courses hours.

   let n to be number of instructors, and Xi be the number of hours assigned to instructor i.

$$F(X_i) = \begin{cases} 1 & , X_i \geq 6 \\ \frac{X_i}{6} & , X_i < 6 \end{cases}$$

$$fitness = \frac{\sum_{i=0}^{n} F(X_i)}{n} * 7.5\%$$

   * Each instructor can teach at least 12 hours/weak and at max 18 hours/weak. Each course has three hours while the lab has 2 hours.

$$G(X_i) = \begin{cases} \frac{X_i}{12} & , X_i < 12 \\ 1 & , 12 \leq X_i \leq 18 \\ \frac{18}{X_i} & , X_i > 18 \end{cases}$$

$$fitness = \frac{\sum_{i=0}^{n} G(X_i)}{n} * 7.5\%$$

4. No four or more consecutive lectures (without breaks) are allowed for the same instructor.

   let n to be number of instructors, and Xi be the maximum number of consecutive hours for instructor i.

$$F(X_i) = \begin{cases} 1 & , X_i \leq 3 \\ \frac{3}{X_i} & , X_i > 3 \end{cases}$$

$$fitness = \frac{\sum_{i=0}^{n} F(X_i)}{n} * 15\%$$

3.6.2. Soft Constraints

1. Minimizing the number of courses of the same level that have the same time slot.

   Let L be the number of levels, Xi the number of conflicted sections in level i, and Si number of sections in level i.

$$fitness = \left(1 - \frac{\sum_{i=0}^{L} \frac{X_i}{S_i}}{L}\right) * 12\%$$

2. Reducing the number of lectures at 8:00.

   Let n be the number of instructors and Xi the number of days start at 8:00 for instructor i.

$$fitness = \left(1 - \frac{\sum_{i=0}^{n} \frac{X_i}{5}}{n}\right) * 10\%$$

3. Minimize waiting time between lectures.

   Let n be the number of instructors, Bi be the total break hours for instructor i, and Li be the total lecture hours per week for instructor i.

$$fitness = \left(1 - \frac{\sum_{i=0}^{n} \frac{B_i}{B_i + L_i}}{n}\right) * 5\%$$

4. Making one day off for each instructor.
   Let N be the number of instructors, and Xi be the number of off-days per week for instructor i.

$$F(X_i) = \begin{cases} 1 & , X_i \geq 1 \\ 0 & , X_i = 0 \end{cases}$$

$$fitness = \frac{\sum_{i=0}^{n} F(X_i)}{n} * 5$$

5. Minimizing the number of rooms reserved from outside Al-Masri Building.

$$fitness = \frac{\#\ of\ sections\ taken\ at\ AlMasri}{Total\ number\ of\ sections}$$

4. Performance and Evaluation

Our program usually satisfies all the hard constraints in around 12000 iterations (fitness = 60). It usually satisfies 85% of the hard and soft constraints in less than 15000 iterations, and 90% of them in about 35000 iterations. After that, the progress becomes very slow, it needs another 35000 iterations to increase the fitness by only 2% and reach 92%. 90% however is really a good and very satisfying result in our case since a 100% satisfaction is not even a possible solution for many reasons. One of them is that some constraints limits the possibility of satisfying others. For example, in order not to give an instructor more than three consecutive busy hours, the program has to give him a break in between, which affects the efficiency of minimizing break hours. So, our program tries to find a tradeoff between these constraints. Another reason is that some constraints cannot be satisfied completely. For example, since the number of courses in the same level is large it has to be some conflicts between them. Actually, it is not even desired not to have any conflicts at all. One problem that faced our program, is that it sometimes converged to a local optimum while the fitness was still very small. However, this problem is solved easily by random restart. Our program counts the number of iterations which did not come with any progress. If the count is over some limit (30000 iterations), and the fitness is not good enough yet (less than 60), the current population is replaced by a new initial population.

.

5. Bonus Elements

Many features and enhancements were added to the program to make it more realistic and sensible. First, number of sections were added to each course. Second, a specific room is determined for each lab as in the real case. These additions make the constraints harder to be satisfied but they give reasonable results. Moreover, all soft constraints were implemented to optimize the solution to its best case.

For simplicity, the program is provided with a very good Graphical User Interface that allows user to determine the favorite courses for each instructor, or he can simply use already existing data written in a file. In addition to that, it allows the user to see the schedule of each instructor in a fancy design that actually shows the efficient satisfaction of hard constraints together with some soft constraints. Besides that, it provides a course browser similar to the one provided by RITAJ to clearly see the result of assigning courses to instructors and rooms with no conflicts.

6. Conclusion

This project is a very good practice on using local search methods to solve problems with large state space. We practically well understood the main aspects of solving problems using genetic algorithm. We actually saw the effect of choosing fitness function on the quality of solutions produced. Good ones get almost to the desired solution in a reasonable time while bad ones usually converge to unacceptable solutions. We also noticed how randomness in genetics plays an important role in preventing solution from converging to a local optimum.

Finally, our design and solution of the problem meets the specifications required and shows very good results. For future work, more features and constraints could be added to make it closer to the real case problem.

7. References

[1] http://www.secretgeek.net/content/bambrilg.pdf , accessed on Nov 23,2018.

[2] https://www.academia.edu/2876050/Solving_University_Timetabling_As_a_Constraint_Satisfaction_Problem_with_Genetic_Algorithm, accessed on 23 Nov, 2018.

[3] https://towardsdatascience.com/introduction-to-genetic-algorithms-including-example-code-e396e98d8bf3, Nov 25,2018.

[4] https://www.tutorialspoint.com/genetic_algorithms/ accessed on Nov 25, 2018.

[5] https://towardsdatascience.com/how-to-define-a-fitness-function-in-a-genetic-algorithm-be572b9ea3b4, Nov 28,2018.