Automatic Generation of University Course Timetabling Using Genetic Algorithm

Meysam Ahangaran[a], Hossein Pourbozorg[b], Mahnaz Talebi[c], Kosar Soleymani[d]

[a] PhD Student of Computer Science, Iran University of Science and Technology

Tel: +98-11-32466121, Email: ahangaran@iust.ac.ir

[b, c, d] Students of Mazandaran University of Science and Technology

pourbozorg@ustmb.ac.ir, mtalebi@ustmb.ac.ir, ksoleymani@ustmb.ac.ir

Abstract

University Course Timetabling Problem (UCTP) is a well-known Constraint Satisfaction Problem (CSP) problem that has exponential number of solutions based on course conflicts, teacher's empty times and other parameters. This is a NP-Hard problem. Scheduling is a major debate on planning which can be used in trains scheduling, classroom scheduling, traffic even in schools and universities. The Scheduling leads to organizing tasks and removing tasks interference which is important. The goal of solving UCTP is setting times for courses and teachers in weekdays in order to reach minimum courses conflicts. It is also ideal for teachers to have joint days for teaching in the least weekdays. Of course, subject to the restrictions of classes and teachers program this scheduling is very difficult. Generally, Evolutionary Algorithms (EA) are efficient tools to solve this problem. The final timetabling must be optimum which means that there is no conflicts if possible and best scheduling generate for teachers. In this paper we solve this problem based on genetic algorithm and implement this algorithm with DEAP python based toolbox on random dataset. The implementation results show that genetic algorithm is efficient tools that can reach to the near of global optimum.

Keywords: Course Scheduling, Constraint Satisfaction Problem, Genetic Algorithm, Optimization, University Course Timetabling Problem**.**

1. Introduction

The issue of tabulation of courses, in principle, includes allocating lessons per week holding periods and room class. If conditions such as failure to allocate one class at a time to several lessons, or lack of time in courses of a teacher are added to this issue, it becomes clear that the problem of designing scheduling, can be a constraint satisfaction problem (CSP). Due to the increasing number of students, new fields, lack of classrooms, conference rooms, laboratories and increasing number of courses offered to the students, we face many limitations, there for designing a time tables for this problem base on teachers, students and courses parameters is very difficult .Different methods have been used to solve these types of issues, some of which include: Graph coloring algorithm, Use of heuristic functions or experimental functions, Population-oriented or developmental methods [1]. In these methods, the evolutionary ideas and population improvement are used and a variety of methods in common include application of this schedule include: Genetic Algorithm, Ant colony algorithm, Memetic algorithm [2]. The algorithm used in this paper is genetic algorithm, which is one of the strongest and most widely used algorithms in search and optimization problems [3, 4]. One reason for the popularity of the genetic algorithm is that it does not require a mathematical model of high-level and advanced one. This algorithm was first introduced by John Holland based on the Darwin's evolutionary theory [5, 6]. Genetic algorithms follow the law of development. To start, these algorithms need a random primary generation which is a set of problem answers. Then the answers are better than before until reaching a global optimum level. In this process chromosomes those which have higher fitness transfer to the next generation with higher probability [5, 6]. In this study each Chromosome is a university course timetabling and solve this problem based on Genetic Algorithm in terms of features and constraints of the problem .the rest of paper has organized as follows. In section 2 we introduce general issues of genetic algorithm. In section 3 we define the problem of university course timetable. Section 4 is devoted to description of the proposed algorithm. Section 5 concerns numerical experiments and their results were studied. Finally, section 6 summarizes the paper.

2. Introduction to Genetic Algorithm

The experience of recent decades shows that genetic algorithm is one of the strongest methods inspired from the nature of genetics and natural selection phenomenon, which is one of the best forms of numerical optimization problems in science and engineering. This algorithm, with a heuristic random search, find the most appropriate answers from the coded information (the chromosomes). The Components of GA include: Chromosomes and genes, genetic population, fitness function and genetic operators [6]. Table 1 compares natural genetic systems and GA [5].

In the genetic algorithm, genetic population (society) there is the set of chromosomes, the genetic operators are crossover and mutation, and the quantitative parameters are population size, the mutation rate and so on. Other components are explained in Table 1.

Table 1 - Comparing genetic algorithm with natural Genetic systems.

Genetic algorithm	Natural Genetic systems
Individual: Possible responses of the issue that have been encrypted like a string of numbers.	**Chromosome:** Gene packages that transfer genetic information from one generation to another.
Fitness function: Is to evaluate the quality of a chromosome that has a mathematical formula.	**Environment:** the environmental conditions of population, which dictates the manner of evolution.
Breeding: for each individual, fitness function is calculated. According to the fitness function value, parents will be selected to produce new population.	**The principle of natural selection:** Survive and proliferation criteria of the organism is adaptation with the environment.
Crossover: is a process of taking more than one parent solutions and producing child solutions from them.	**Proliferation:** As a result of the chromosomes exchange, genetic information transfer to the Childs.
Mutation: one bite of chromosome is selected randomly and changes.	**Genetic Mutation:** Replacement of a gene with another one, in DNA chain.
Reproduction: Repeating the algorithm to achieve optimal solution or reach the termination condition.	**Reproduction**: the creation of new generations and evolution.

2.1 Description of the Genetic Algorithm Flowchart

Genetic algorithm has flowing stages:

1- Creating random population (The initial population) and evaluation.
2- Parental choice and their combinations to make children.
3- Selecting members of the population to apply a mutation and mutation population creation.
4- The main population integrate the children of the original population and creating new mutants.
5- If the termination condition is not achieved, we repeat stage 2.

The general flowchart of genetic algorithm and pseudo code of typical evolutionary algorithm is shown in figure 1 and figure 2 respectively [9].

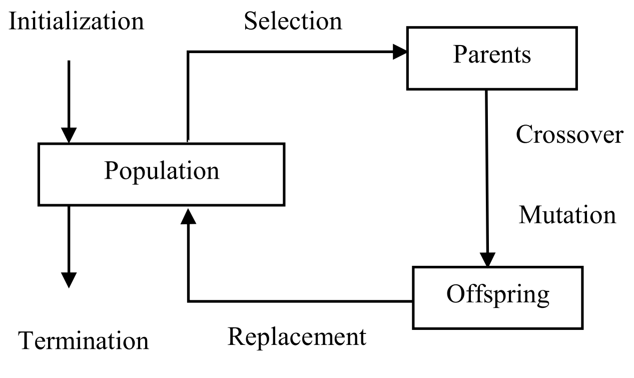

Figure 1- The flowchart of Genetic Algorithm

```
Algorithm: Evolutionary algorithm

BEGIN
INITIALISE population
EVALUATION each individual
  REPEAT UNTIL (termination condition)
   1. SELECT parents
   2. RECOMBINE pairs of parents
   3. MUTATE the resulting offspring
   4. EVALUATE new individuals
   5. SELECT individuals for the next generation
   END REPEAT
END
```

Figure 2 - Pseudo code of evolutionary algorithms

3. Problem Definition

In this problem we have a set of empty times for each teacher and list of courses and a series of conflicts between courses which has display by a graph (conflict graph). In the conflict graph nodes are as courses and each conflict between two courses is represented with and edge between related nodes. In the worst case, if all courses have conflict each other, the conflict graph is a complete graph with *N* nodes (N: number of courses). In this case we know that the number of edge for a complete graph is: $\frac{n*(n-1)}{2}$

The university course timetabling problem we want to find and optimum course scheduling in which all conflict are satisfied, therefore this is and optimization problem. Because the genetic algorithm is and optimization process, we can use this algorithm for solving this problem to reach the global optimum albeit genetic algorithm almost doesn't reach to the global optimum, but it can reach to the sub optimal points. In this problem the sub optimal point is a solution (course timetabling) that some conflicts are not satisfied. The input-output diagram of UCTP is shown in figure 3.

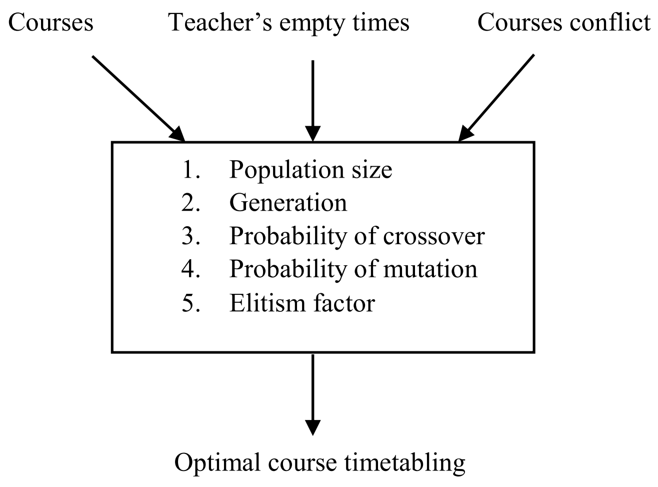

Figure 3 - Input – Output diagram of the Course scheduling model time.

The parameters of this problem are as following:

Inputs: Course Names, Teachers timetable, Courses conflict.

Output: Optimal Course timetabling.

Parameters:

1. Population size
2. Generations
3. Probability of mutation
4. Probability of crossover
5. Number of teachers
6. Number of courses

4. Our proposed algorithm

In the previous section we introduce UCTP parameters. In the following we identify the components of our algorithm. First, a set of teacher's empty times, list of courses and courses conflict is given to the algorithm. Then initial population is randomly produced including a set of courses timetable. After using parent selection method, a set of chromosomes are selected and combine them to produce new generation. Finally, chromosome with maximum fitness is introduced as the best answer. In the next section we review the components of our proposed algorithm.

4.1 Components of algorithm

Encoding: For creating individuals, we generate a number in range [0, 84]. Assume that university is open all days in week and 12 hours per day (from 8 to 20). Class time will be in the 84 hours (84=7*12). In this paper the real value is assigned to each problem parameter at random.

Length of the chromosome: The length of each chromosome is: $\sum_{n=0}^{p} c_n$, where p is the number of teachers and c_n is the number of courses that *n*'th teacher can offer.

Fitness: In this step, the following four parameters have been evaluate by weight -1. This weight shows that our aim is to maximize the fitness function, therefore we search state space to find global maximum.

1. Number of times that teacher can be present in the university.
2. Number of times that teachers have more than one class at the same time.
3. Number of times that two courses have conflict each other on the basis of conflict graph.
4. Number of times that two courses are at the same time.

Breeding: The fitness function for each individual is calculated and best individuals are selected as parents for creating next generation.

Selection: For parent selection we used tournament selection; this method works as follows: Choose some number of individuals (tournament size) randomly from the population and copy the best individual from this tournament group into the population and repeat n times [10]. In this paper, we figure out "tournament size=4" is suitable value for this problem.

Crossover: We used uniform crossover in our algorithm; in this method, each gene in the offspring is created by copying the corresponding gene from one or the other parent chosen according to a random generated binary crossover mask of the same length as the chromosomes. When there is a 1 in the crossover mask, the gene is copied from the first parent, and when there is a 0 in the mask the gene is copied from the second parent. A new crossover mask is randomly generated for each pair of parents. Therefore Offspring contain a mixture of genes from each parent. The number of effective crossing point L is not fixed [11]. In this paper, we set this parameter with "L=0.5" and crossover probability with 1.

Mutation: For mutation, we used shuffle indexes mutation; in this method, Shuffle the attributes of the input individual and return the mutant. The P parameter is the probability of each attribute to be moved. Usually this mutation is applied on vector of indices. In our implementation, we set "P=0.1" and mutation probability is 0.1.

Replacement: In the replacement step, we used the generation update method. In this method, each parent will be replaced by its child.

Elitism: We used elitism size=1. It means that we keep best individual in each iteration of generations.

Termination condition: In our algorithm we used the maximum production method. In this method, after N generation, algorithm is ended.

5. Simulation Results

For implementation of our propose algorithm we use Python programming language. We generate random datasets and apply genetic algorithm functions with Python based DEAP toolbox. For creating output diagrams we use another Python based toolbox named Matplotlib.

Distributed Evolutionary Algorithms in Python (DEAP) is an evolutionary computation framework for rapid prototyping and testing of ideas [7]. It incorporates the data structures and tools required to implement most common evolutionary computation techniques such as genetic algorithm, genetic programming, evolution strategies, particle swarm optimization, differential evolution and estimation of distribution algorithm. It is developed at University Laval since 2009.

Matplotlib is a plotting library for the Python programming language and its numerical mathematics extension NumPy library. It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits like wxPython, Qt, or GTK+ libraries. There is also a procedural "pylab" interface based on a state machine (like OpenGL), designed to closely resemble that of MATLAB, though its use is discouraged [8]. SciPy library makes use of matplotlib.

We simulate our proposed algorithm on random generated datasets with DEAP toolbox. Diagram of fitness changes in various generations is shown in figure 4. According to this figure standard deviation of population in each generation is constant and near to zero. This figure shows that average fitness increase, and near to the best solution (max diagram). Therefore the population converge to the global optimum and best solution is near to the global optimum (fitness=0).

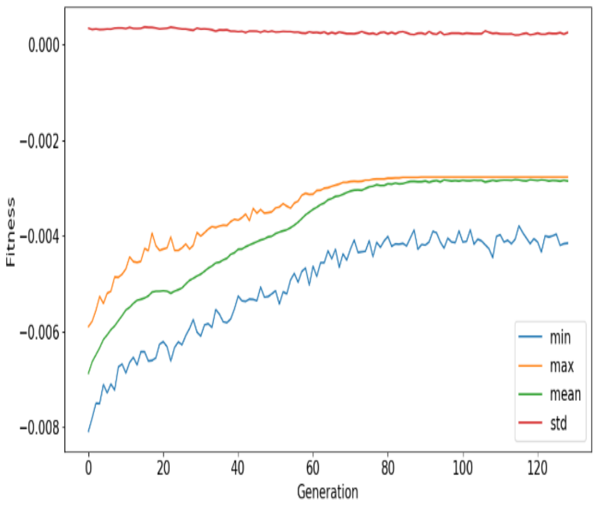

Figure 4 – Diagram of max, min, mean and standard deviation of fitness function in different generations.

The parameters value in our simulation are:
Number of generations = 128
Population size = 512
Number of courses = 32
Number of teachers = 32
Number of conflicts set = 16
Probability of courses for each teacher = 0.1
Probability of teacher empty time = 0.5
Probability of courses conflict = 0.1

In the next experiment we change probability of conflicts and fix other parameters. Figure 5 shows diagram of best fitness value based on conflict probability values. According to this diagram, with increasing conflict probability the value of best fitness decrease because when probability of conflict is large, algorithm cannot find optimal timetabling.

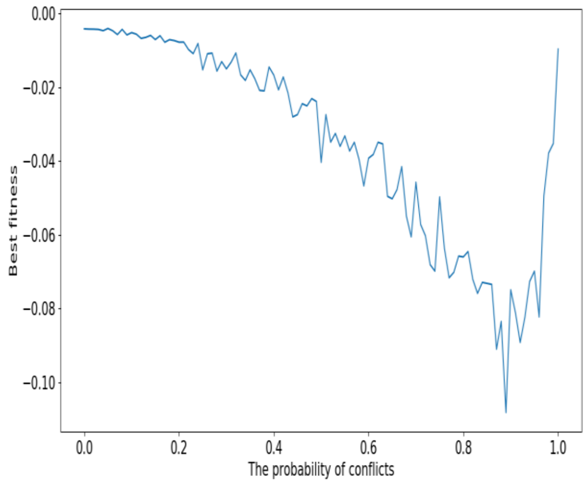

Figure 5 – Diagram of best fitness based on conflict probability variations.

The parameters value in this experiment are:
Number of generations = 8
Population size = 8
Number of courses = 32
Number of teachers = 32
Number of conflicts set = 16
Probability of courses for each teacher = 0.1
Probability of teacher empty time = 1

At next step we change probability of courses for teachers and fix other parameters. Diagram of this experiment is shown in figure 6. This diagram shows that when teachers have more empty times then finding course timetabling will be easier. Therefore we can see that this diagram is ascending function.

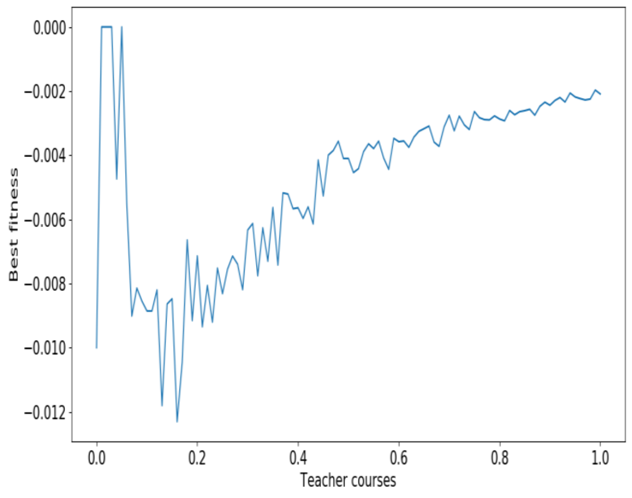

Figure 6 – Diagram of best fitness based on probability of courses for each teacher.

The parameters value in this experiment are:
Number of generations = 16
Population size =16
Number of courses = 16
Number of teachers = 16
Number of conflicts set = 16
Probability of teacher empty time = 1
Probability of courses conflict = 0.1

In the last experiment we studied behavior of algorithm when teachers empty times increase. Figure 7 shows diagram of best fitness based on probability of teacher's empty times. This diagram is also an ascending function because with increasing teacher empty times probability, algorithm can find better solution.

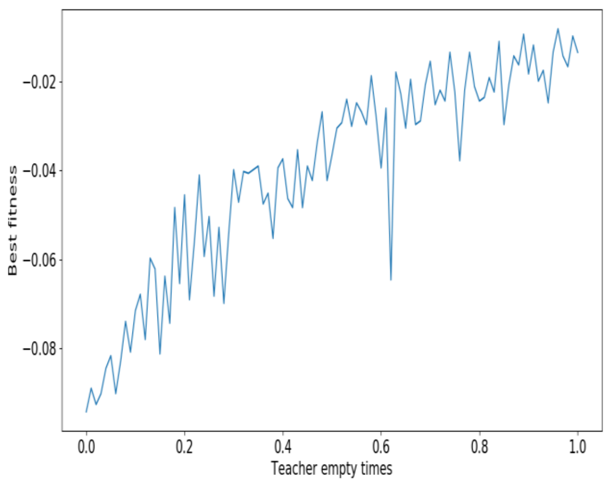

Figure 7 – Diagram of best fitness based on teacher empty times.

The parameters value in this experiment are:
Number of generations = 16
Population size =16
Number of courses = 16
Number of teachers = 16
Number of conflicts set = 16
Probability of courses for each teacher = 0.1
Probability of courses conflict = 0.1

6. Conclusion

In this paper we study University Course Timetabling Problem based on genetic algorithm. This problem has been widely studied in literatures but there is no optimal solution for this problem yet that reach to global optimum. Results show that solving this type of problems with evolutionary algorithms is very efficient. Also average evaluation function on different random datasets in various generations shows that this algorithm can improve its performance during iterations of each generation until reach to the sub-optimal solution. For future works we can improve this algorithm with another parameters of genetic algorithm such as different crossover algorithm or use several crossover methods.

References

[1] Carter M., "*A Comprehensive Course Timetabling and Student Scheduling System at the University of Waterloo*", Lecture Notes in Computer Science, Vol. 2079, 2001, pp. 64-82.

[2] Burke E., Elli man D., Wearer R., "*A Genetic Algorithm based University Timetabling System*", Proceedings of the 2nd East-West International Conference on Computer Technologies in Education, 1994, pp. 35-40.

[3] Russell S., Norvig P., *"Artificial Intelligence: A Modern Approach"*, 3rd Edition, 2009, Prentice Hall.

[4] Mitchell M., *"An Introduction to Genetic Algorithms"*, Mit press, 1998.

[5] Goldberg D., *"Genetic Algorithms in Search, Optimization and Machine Learning"*, Addison-Welsh, 1998.

[6] Whitely D., *"A Genetic Algorithm Tutorial"*, Journal of Statistics and Computing Vol. 4, 1994, pp. 65-85.

[7] Fortin, Félix-Antoine, et al. *"DEAP: Evolutionary algorithms made easy."* Journal of Machine Learning Research 13, 2012, pp.2171-2175.

[8] Hunter, John D. *"Matplotlib: A 2D graphics environment."* Computing in science and engineering 9, no. 3, 2007, pp. 90-95.

[9] Eiben, Agoston E., James E. Smith. *"Introduction to evolutionary computing."* Vol. 53. Heidelberg: springer, 2003.

[10] Matoušek, Radomil. *"Genetic Algorithm and Advanced Tournament Selection Concept"*, Nature Inspired Cooperative Strategies for Optimization (NICSO 2008). Ed. Natalio Krasnogor et al. Springer Berlin Heidelberg, 2009. pp. 189–196. Print. Studies in Computational Intelligence 236.

[11] Sivanandam, S. N., S. N. Deepa, "*Introduction to Genetic Algorithms*", Springer Berlin Heidelberg, 2008.