# Modular Exam Scheduling using Genetic Algorithm

Dua' Saadeh
Department of Computer Science, University of Jordan, Amman, Jordan
Doaa.saadah@ju.edu.jo

Basel Abu-Jamous
Department of Computer Engineering, University of Jordan, Amman, Jordan
bwaaj2002@hotmail.com

Fahad Mustafa, Muath Al-Hijjawi
Department of Computer Science, University of Jordan, Amman, Jordan
fahadali1989@hotmail.com, muhejjawi27@hotmail.com

## ABSTRACT

Automated scheduling is considered to be one of the most challenging problems because it is very demanding in regards of performance and because of the existence of various methods to solve this problem. In this paper a solution to solve this problem by using a genetic algorithm re-enforced by a production system, although very performance demanding; this method allows for very good exam schedules to be generated.

Keywords: genetic algorithm, scheduling, distributed, production systems

## 1. Introduction

One of the problems in universities is building exams schedules. This problem is commonly named *Modular Exam Scheduling Problem* (MESP) [6]. Scheduling exam's time and place usually takes care of available rooms, supervisors and students' conflicts with other exams.
This problem gets bigger and bigger as the number of courses and students increases [1,8,9]. Some problems that arise when building an exams schedule are:

- Invalid cases:
    1. One supervisor supervises on two rooms at the same time.
    2. One course exam sessions held in different time slots.
    3. One course exam has no time to be held.
- Valid cases with serious problems:
    1. One student has more than one exam at the same time.
    2. One course exam which belongs to a faculty is held in a different faculty room.
    3. One supervisor which belongs to a faculty supervises on a room of different faculty.
- Valid cases with acceptable problems:
    1. One student has two exams at the same day.
    2. Unbalanced distribution of supervisors.
    3. Long term timetable.

After taking a look on Previously designed systems[2,3,4], they all try to solve this problem using straightforward genetic algorithms, although genetic algorithms are considered to be a good method to solve these kinds of problems, they are slow and very performance demanding.

The method proposed for solving this problem is also performance demanding but the difference is that we use production systems side by side with genetic algorithms. The use of production systems every few generations gives a significant boost to overall fitness of the schedule which reduces the amount of generations needed. We also differ from the previous approaches by:

- Taking more details into consideration regarding the scheduling process.
- Using production systems to fine tune the produced schedules in all its production phases.
- Using an accurate fitness function that reflects the actual status of the schedule as much as possible.

**1.1 The genetic algorithm**

It is an algorithm that imitates the biological reproduction operation. Every organism has a set of rules, describing how that organism is built up from the tiny building blocks of life. These rules are encoded in the *genes* of an organism, which in turn are connected together into long strings called chromosomes [5, 6, 7].

The philosophy behind this algorithm is:

Initially, we have random group of suggested instances of objects. The quality of these initial objects is too low because of the nature of the random process that generates them.

Each individual of that objects is called *a chromosome*. The group of the *chromosomes* on which the GA is applied is called *the population*.

Each *chromosome* (object) has many characteristics and properties that distinguish it. Each one of them is called *a gene*. In other words: We have a population of chromosomes, each chromosome consists of a group of genes that make it distinctive.

According to the nature of the problem, *a fitness* value is given to each chromosome. It represents the quality of that chromosome. Calculating the fitness is very important and sensitive and strict process. A better fitness function a better solution.

The steps that we may follow to apply genetic algorithms:

1. Specify your problem, defining your chromosome and gene. And decide what values a gene would take.

2. Create your initial population. This population is a set of chromosomes that are randomly created. The number of these initial chromosomes depends on the nature of the problem and on the abilities of the performing computer.

3. The fitness function: Put the fitness function which gives a chromosome a value depending on its quality (how much this chromosome satisfies conditions).

4. *Crossover*: It is an operation that is done between two chromosomes.

    a. Select two random chromosomes from the population.

    b. Select a random point within the chromosome.

    c. Divide both chromosomes from that point.

    d. The new chromosome is created by concatenating the first part of the first chromosome with the second part of the second chromosome.

5. *Mutation*: It is an operation that is applied on the chromosomes generated by the crossover operation. Mutation is done by changing the value of one random gene.

6. Recalculate the fitness for these new chromosomes and repeat points 4 & 5 (crossover and mutation).

The process of calculating fitness, creating new chromosomes using cross over and do mutation on these new chromosomes is called *reproduction* process.

## 2. Genetic Algorithm Re-enforced with Productions Systems (GARPS)

Since genetic algorithms are the main building block of the GARPS technique, it will be discussed first followed by the production systems.

The population is consisted of a group of exam tables where each table represents a chromosome. Each table, which is a 2D-array, considers the distribution of courses' exams and supervisors on rooms in different timeslots. The main architecture of the chromosome is shown in figure 1.

| | **Timeslot1** | **Timeslot 2** | …. |
|---|---|---|---|
| **Room1** | Course1<br>Superv A<br>Superv B | Course3<br>Superv E<br>Superv F | |
| **Room2** | Course2<br>Superv C<br>Superv D | Course4<br>Superv H<br>Superv B | |
| ⋮ | | | |

**Figure 1: Chromosome Architecture**

Crossover and mutation are applied on courses and supervisors separately. When applying them on courses we take into consideration that each course exam will not be held more than once, while supervisors can supervise more than one exam in different timeslots.

### 2.1 The fitness function

Each table will be given a fitness value between 0% and 100%, higher is better. Things taken in consideration in calculating the fitness:

- Course – room faculty (f1)

Each course should be held in a room of its faculty.

$$f1 = \frac{\text{courses in place}}{\text{all courses}} \times 100\% \qquad (1)$$

Weight = 0.12

- Supervisor-room faculty (f2)

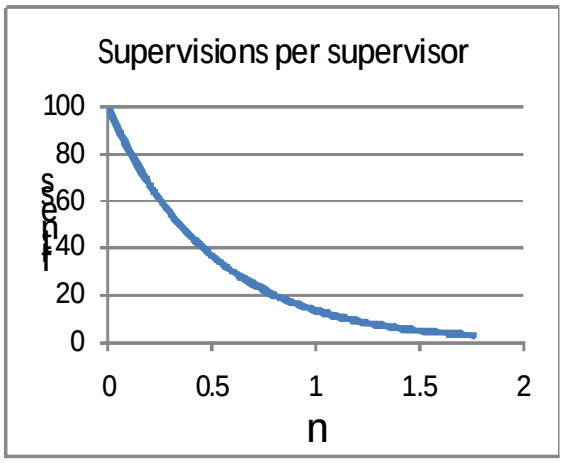


**Figure 2: Supervisions per supervisor**

Each supervisor should supervise on a room of his faculty.

$$f2 = \frac{\text{supervisors in place}}{\text{all courses}} \times 100\% \ldots\ldots (2)$$

Weight = 0.12

- Supervisions per supervisor (f3)

Supervisions should be distributed on supervisors in a fair way.

$$f3 = 100e^{-2n} \qquad \ldots\ldots (3)$$

$$n = \frac{\sqrt{\sigma}}{\bar{x}} \qquad \ldots\ldots (4)$$

Where σ is the variance and $\sqrt{\sigma}$ is the standard deviation

$$\sigma = \sum_i (x - \bar{x})^2 \quad \text{..... (5)}$$

x: number of supervisions by supervisor (i).

$\bar{x}$: Average number of supervisions per supervisor.

Weight = 0.05

- Free supervisors (f4)

Supervisors should have no lectures in the time they are required to supervise in.

$$f4 = \frac{\text{free supervisors}}{\text{all supervisions}} \text{ x } 100\% \quad \text{..... (6)}$$

Weight = 0.1

- Free rooms (f5)

Rooms should be free in the time they are needed for an exam.

$$f5 = \frac{\text{free rooms}}{\text{all sessions}} \text{ x } 100\% \quad \text{..... (7)}$$

Weight = 0.1

- Supervisors on courses (f6)

Each course exam should have two supervisors on it. No supervisors should be put on empty rooms.

$$f6 = \frac{\text{correct sessions}}{(\text{all rooms}) x (\text{all timeslots})} \text{ x } 100\% \quad \text{..... (8)}$$

Weight = 0.05

- Short tables (f7)

Shorter tables are preferred. A maximum timeline is assigned by the user for the exams. In the first half of that maximum timeline, a used room (holding an exam) is better than an empty one. The case is reversed in the second half. This is done by giving each room in a specific timeslot a "fitness value" then taking the average of all fitness values calculated.

Let f(i , j) be the fitness of the room i in the timeslot j.

$$f(i,j) = \begin{cases} a\sqrt{j}, & \text{room(i) is empty in timesl} \\ a\sqrt{N-j}, & \text{room(i) is used in time} \end{cases}$$

..... (9)

Where:

- N: Number of timeslots – 1
- i, j starts from zero
- $a = \frac{100}{\sqrt{N}}$

$$f7 = \frac{\sum_i \sum_j f(i,j)}{(\text{rooms}) x (\text{timeslots})} \quad \text{..... (10)}$$

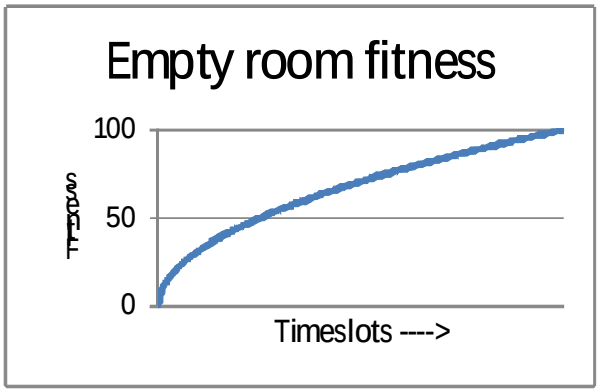


**Figure 3: Empty room fitness**

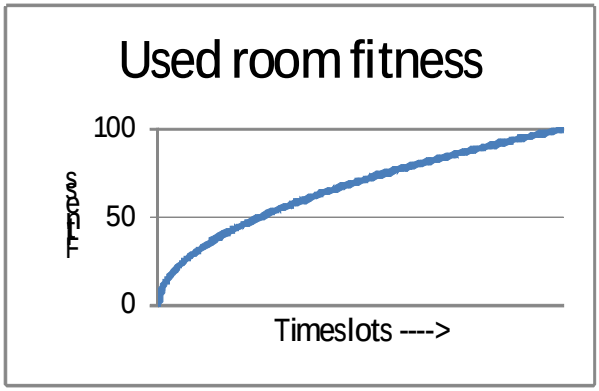


**Figure 4: Used room fitness**

Weight = 0.06

- Conflicts (f8)

Conflicts of students' exams are one of the most important reasons that caused the MESP.

This issue has many things to consider:

- Worst case is that a student has more than one exam in the same time.
- Another bad case but is acceptable that a student has more than one exam in the same day.
- Less bad case is that a student has a lot of time between two consecutive exams (more than 5 days for example).

The best period of time left between two consecutive exams is about 2.5 days.

A rough graph was drawn between fitness values and spaces between consecutive exams. Then an equation was set. That equation is:

$$f(x) = 30 + 37.052\, x^2 e^{-0.168x^2} - 30e^{-0.4x} \quad \ldots\ldots\ldots (11)$$

Where: (x) is the number of days between each two consecutive exams.

The whole conflicts fitness is calculated by averaging the fitness values of all students' exams.

Note that the maximum fitness (100%) is at (2.5 days) as discussed above. If the time between two consecutive exams was (0 days) then it's a real conflict, which results in zero fitness. If two exams where in the same day the time between them will be less that (1 day) which means a fitness value less than 42%.

Increasing free time between consecutive exams to values more than 2.5 days will result in decreasing the fitness value.

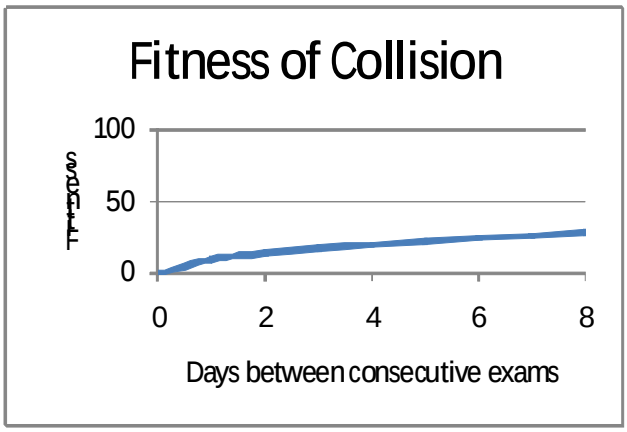


**Figure 5: Fitness of Collision**

Weight = 0.4

## 3. Results and analysis

The system has proved its capability of building highly reliable tables in significantly short time. The sample which was used in testing was as following:

- About 2000 students registered in 45 different courses.
- About 30 supervisors.
- About 20 exams rooms.

The initial population is 30 tables. The best table in the current generation (elite) survives and is transferred to the next generation. After experimenting with various crossover probabilities, the best choice for crossover probability is 0.75. Our experiments with various mutation probabilities concluded that a high mutation probability was needed to ensure faster fitness improvement, this probability is 0.15.

To improve the performance the system was implemented as a distributed system. The test results were:

The number of the students registered in the course is taken into consideration as well as the capacity of the rooms.

- One generation without production systems takes about $\frac{12 \times population}{number\ of\ computers}$ seconds….. (12)

- One generation with production systems takes about $\frac{40 \times population}{number\ of\ computers}$ seconds……(13)

**Table 1: A sample Schedule**

A sample of a result table after 10 generations of part of a single timeslot was:

Table 1 shows how different supervisors supervise on the exams at same time. It also shows that each course exam is to be held in a room (or more) of the same faculty.

| Course | Date | Supervisors | Location |
|---|---|---|---|
| **AI** | Thursday 1/1/2009 08:00 | John, Ahmad | IT105 |
| | | Ali, Mohammad | IT102 |
| **DB** | Thursday 1/1/2009 08:00 | Mustafa, Taha | IT301 |
| | | Fatima, Jack | IT101 |
| **Digital Logic** | Thursday 1/1/2009 08:00 | David, Michael | ENG102 |
| | | Tom, Mary | ENG104 |
| **Calculus** | Thursday 1/1/2009 08:00 | Sean, Mariam | SCI100 |
| | | Jane, Nour | SCI101 |
| **Anatomy** | Thursday 1/1/2009 08:00 | Sara, Anas | MED100 |
| | | Robin, Kyle | MED101 |

## 4. Conclusion

The solution used to solve the modular exam scheduling problem by using a genetic algorithm re-enforced by production systems proved itself by producing very high quality schedules in acceptable time. The use of pure genetic algorithms is performance demanding while the use of the production systems along with genetic algorithms in the GARPS technique helped with both the overall fitness which reduced the amount of generations needed to get the job done.
It is also worth to mention the role of distributed genetic algorithms in the improvement of the overall performance of the entire technique used.

## *5. Reference*


[1] A. J. Page and T. J. Naughton, "Framework for task scheduling in heterogeneous distributed computing using genetic algorithms," 15th Artificial Intelligence and Cognitive Science Conference, Castlebar, Ireland, September 2004, pp. 137–146

[2] Andrew J. Page, Thomas J. Naughton, "Dynamic Task Scheduling using Genetic Algorithms for Heterogeneous Distributed Computing," 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05) - Workshop 6, 2005, pp.189a.

[3] A. Schaerf and L.D. Gaspero, "Local Search Techniques for Educational Timetabling Problems", Proc. 6th International Symposium on Operations Research in Slovenia, Preddvor, Slovenia, 2001, pp. 13-23.

[4] B. Sigl, M. Golub, and V. Mornar, "Solving Timetable Scheduling Problem by Using Genetic Algorithms", Proc. 25th IEEE Int. Conf. on Information Technology Interfaces, Cavtat/Dubrovnik, Croatia, 2003, pp. 519-524.

[5] D. Abramson, H. Dang, and M. Krisnamoorthy, “Simulated Annealing Cooling Schedules for the School Timetabling Problem”, Asia Pacific Journal of Operational Research, World Scientific, 1999, vol. 16, pp. 1-22.

[6] D. Corne, H.-L. Fang , C. Mellish , “Solving the modular exam scheduling problem with genetic algorithms,” Proceedings of the 6th international conference on Industrial and engineering applications of artificial intelligence and expert systems, 1993, pp. 370 – 373

[7] E.K. Burke, D.G. Elliman, and R.F. Weare, “A Hybrid Genetic Algorithm for Highly Constrained Timetabling Problems”, Proc. 6th Int. Conf. on Genetic Algorithms, Pittsburg, Morgan Kaufmann, , 1995,pp. 605-610.

[8] Nguyen Duc Thanh ,"Solving Timetabling Problem Using Genetic and Heuristic Algorithms," Eighth ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing, 2007, pp. 472-477.

[9] Taisir Eldos , "A New Migration Model For Distributed Genetic Algorithms ," The 2005 International Conference On Scientific Computing (CSC'05) June 20-23