# A Non-Standard Genetic Algorithm Approach to Solve Constrained School Timetabling Problems

Cristina Fernández Bedoya[1], Matilde Santos[2]

[1]Lab. Electrónica y Automática. CIEMAT
Avda. Complutense, 22. 28040. Madrid – Spain
Cristina.fernández@ciemat.es

[2]Dto. Arquitectura de Computadores y Automática
Facultad de CC. Físicas – UCM. 28040. Madrid – Spain.

**Abstract.** In this paper a non-standard constructive approach is described to solve problems of timetabling, so necessary in every school and university. Genetic Algorithms have been used, but adapting them to the characteristics of the problem under consideration. Hard constrains, feasible timetables, have been considered in the problem formulation, forcing us to define a particular population generation and a new guided mutation operator. It has been proved that modified genetic algorithms can be very useful in this kind of problems for the easiness of including constrains and the effectiveness of its resolution, reaching optimum values in few iterations.

## 1. Introduction

Timetabling is one of the most common problems in every educational Institution. Every School, College or University throughout the world has to design every year feasible timetables that fulfill many required constrains. Most of the times this task is carried out manually or with the help of administration systems. In any case it is a tedious and time consuming task, while it is repetitive, routine and arduous. So it seems to be much more adequate for a computer.

On the other hand, form the Artificial Intelligence point of view, there has been a large concentration of efforts on university timetabling problems, in contrast with school courses and exams scheduling, where there is no currently no evolutionary school timetable software available [1].t This is despite the fact that there are several non-evolutionary school timetabling software available, as this problem has been tackled with operation research, linear programming, network flow, etc.

One of the main problem that appears when tackling of this task is how to introduce many constrains in an algorithm and how to assign to them the appropriate weights. Both hard and soft restrictions must be fulfilled but in a different level. In addition, computation time is usually very high. Evolutionary computing lends itself to multi-constrained problems, due to the facility of incorporating constraint violations into a single fitness function. These intelligent strategies can provide optimal solutions exploring a narrower search space, saving computational time.

Course timetabling is a multi-dimensional NP-Complete problem [2, 3] as it cannot be solved in polynomial time by the exhaustive evaluation of every timetable. Course timetabling is basically a combinatorial problem like the Traveler Salesman [5], with a set of items that have to be sorted to obtain the best configuration. Nevertheless the fitness function in the timetabling problem is more complicated, needing to take into account many restrictions with different priority.

Genetic Algorithms, as well as other heuristic methods, have been found to be very powerful as optimization tools for NP-complete problems. GA are based in the principles of evolution [4], and they explore the solution space moving towards the most promising areas of the search space. At the same time, it keeps different solutions, avoiding local minimums or cyclings.

The variables involved in a configuration are restricted to a discrete and finite set, and the kindness of a specific solution is measured by the order of these variables, which are strongly interdependent. This interrelationship is the main restriction that the solution has to meet, otherwise it is absolutely a non feasible timetable. This important characteristic forces to implement the resolution method so it is adjusted to this constraint. Therefore, GA can be an useful strategy for solving these problems but they are more efficient if they are adapted to the application.

The main objective of this work is to help administrative staff of public schools in Spain, where this task is usually performed manually and requires several days.

The paper is organised as follows. Section 2 describes the specific timetabling problem we are dealing with. In Section 3, the modifications applied to the Genetic Algorithms are explained. The simulation results are presented and analysed in Section 4. Finally, conclusions are summarised in the last section.

## 2. The Weekly Timetabling Problem

The timetabling problem consists of fixing a sequence of meetings between teachers and students in a prefixed period of time, typically a week. For the purpose of this paper, a simple problem of finding the optimal timetable for one academic year is considered. It works with a certain number $N_g$ of classes, each one representing a group of students taking an identical set of subjects and, typically, staying together all the week long. There will be certain number of subjects, $N_a$, that will be the same for every class, and each subject will have a different number of hours per week. The size of the search space is given by $(N_a!)^{Ng}$.

The timetable is split in time slots, for example, an hour or fifty minutes. The teaching of each subject has to be fitted to that slot, and the arrangement of these items will define each configuration. The number of items per day can be specified for each particular situation, as this will affect the fitness value. The number of available teachers per subject is also needed to be defined. If this number, $N_p$, is more than one, each of the available teachers will be assigned to each class.

There is a basic constraint that directs the searching toward reaching a solution, the feasibility of the timetable. There can not be more neither less hours of each subject than required, the same for teachers, etc. In addition, two other requirements are established:

1. Avoiding teachers meeting two classes in the same time, and vice versa. This constraint is more restrictive as the number of groups increases, leading to a non-feasible solution of the problem when the number of teachers is too low.

2. It is not allowed to have a timetable with the same course twice in the same day. The item size is considered as the maximum time that one course can be imparted per day.

Other soft constraints can be added, for example, allowing to have a particular subject at a specific hour or weekday; if two courses need the same resource, they should no be at the same time, etc.

## 3. The Genetic Algorithms Approach

In this section, a modified version of the standard GA is described, to be applied to the timetabling problem.

### 3.1. Codification of the problem

Each individual of the population contains all the information relative to a feasible timetable. The chromosome structure is a matrix in which the rows identify the class and the columns represent each time item (*gen*), arranged in increasingly order from first hour Monday morning (**Fig. 1**).

Each subject of the set $[1..N_a]$ is codified with a number. If there is more than one teacher qualified for that subject, a different code will be used to show that is the same subject although it can not be overlapped.

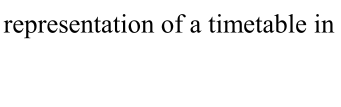


**Fig. 1.** Direct representation of a timetable in a chromosome

## 3.2. Population generation

A particular generation function has been defined to guarantee the feasibility of the generated chromosomes. The method is similar to that of an innocent hand extracting balls from a bag. In that “bag” the assigned codes of every subject are stored, but they can be repeated as many times as hours this course is going to be taught in a week. Choosing randomly each of these genes, it is possible to find different timetables for each class and each individual.

## 3.3. Fitness function

The aim is to maximize the fitness function described before, with each restriction $R_i$ weighted by a value. This weight will influence the evolution of the population with respect to each constraint.

$$F_{obj} = \Sigma \, w_i \, R_i \, . \qquad (1)$$

Each unfulfilled constraint is represented with a negative amount, so the optimal solution has a fitness value of zero.

## 3.4. Parents choice

This task is performed by the roulette method. Accordingly to the value of the fitness function, each individual is assigned to a proportional sector size, which represents the probability of being selected as a progenitor of the next generation. In consequence, the best individual has the higher probability of being selected but other individuals can also be selected. This will allow to achieve better solutions from individuals that have not been considered at that moment as the best configuration.

Once the parents have been selected, the program creates the next generation, but somehow they will have to be repaired to constitute feasible timetables. With the common crossover operator, as the chromosomes would be split and exchanged, there is an almost absolutely certainty that the resulting chromosome will not be feasible. In some papers, a crossover mechanism that respects this hard condition is explained [6] or a strategy that repairs the obtained chromosomes is described[5, 7], although this is quite time consuming. In this implementation the algorithm is always searching the space of the feasible solutions.

## 3.5. Mutation

The drawback of the crossover operator: it can violate some of the hard constraints, the necessity of repairing the springs, time consuming, …) does not appear in the mutation operation. Intrinsically, a timetable has all the necessary information to be an optimal solution. It needs just to be properly sorted. Hence, mixing information from different individuals is not going to improve any of those solutions, as each of them fulfills the constraints according to the disposition of the whole group of their genes.

Therefore, a parthenogenesis-like operator has been used. Each individual can produce a spring with a modified chromosome, generated by the reorganization of the parental genes. This reorganization can be performed in a random way or with certain guidance. Given a particular timetable, those genes that are penalizing the fitness function can be stored and then they can be randomly selected to establish the swapping point.

There are two swapping points that define the chromosome slice that is going to be exchanged. The depth of this slice (how many rows will be exchanged) is also a random parameter. One of these points is selected with a certain probability ($P_r$) from the set of conflictive genes:

```
If (code_repeated_same_day = true) then

          Fitness_f = Fitness_f - W;

          Conf_gen = [Conf_gen repeated_gen];

end

......

If (random(0..1) < Pr) then

          Swap_point = random(Conf_gen);

else

          Swap_point = random(1..number_genes);

end
```

This method helps to search better solutions, resulting in a guided influence on the generation of new solutions. The obtained chromosome will be quite similar to its parent, and most of the times, better, although some times it is possible to notice some oscillations (**Fig. 2**).

The algorithm has been implemented with Matlab, and a graphical user interface has been defined (**Fig. 3**). That allows to configure the genetic algorithms parameters, such as number of substitutions, population size, Pr, etc, and also school features as number of classes, number of hours per day, number of subjects, teachers assigned to each course, etc.

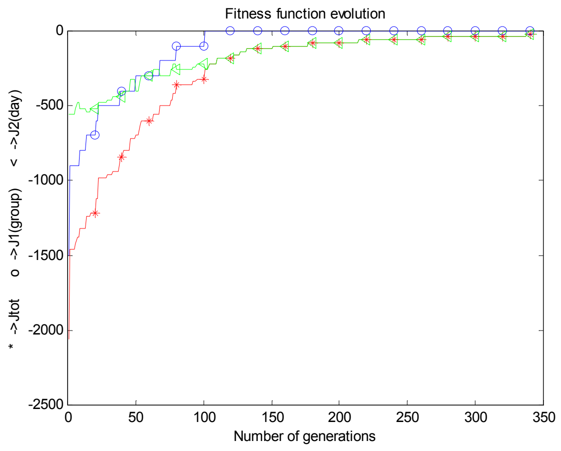
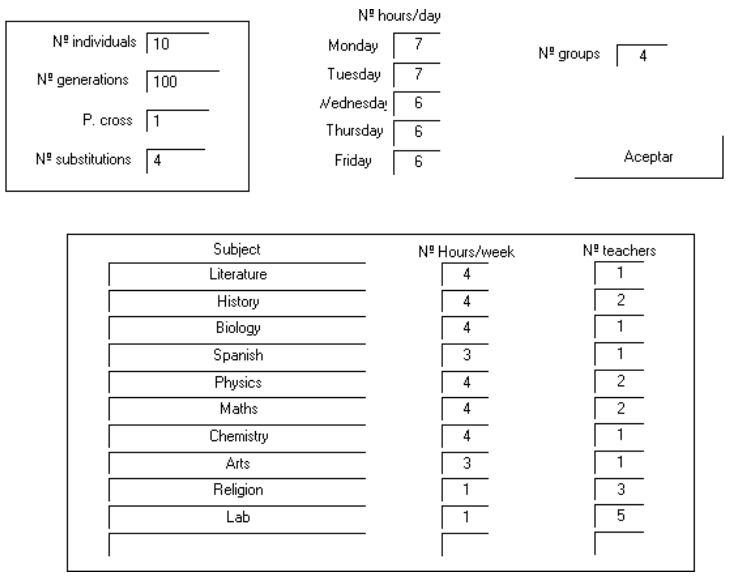


**Fig. 2.** Evolution of the fitness functions for the best individual in each generation. Both constraints: overlapping of subject in different classes (<), and same subject in the same day (o) are represented, as well as the total fitness value (*). The fitness subfunction oscillation is remarked.



| | |
|---|---|
| Nº individuals | 10 |
| Nº generations | 100 |
| P. cross | 1 |
| Nº substitutions | 4 |

Nº hours/day

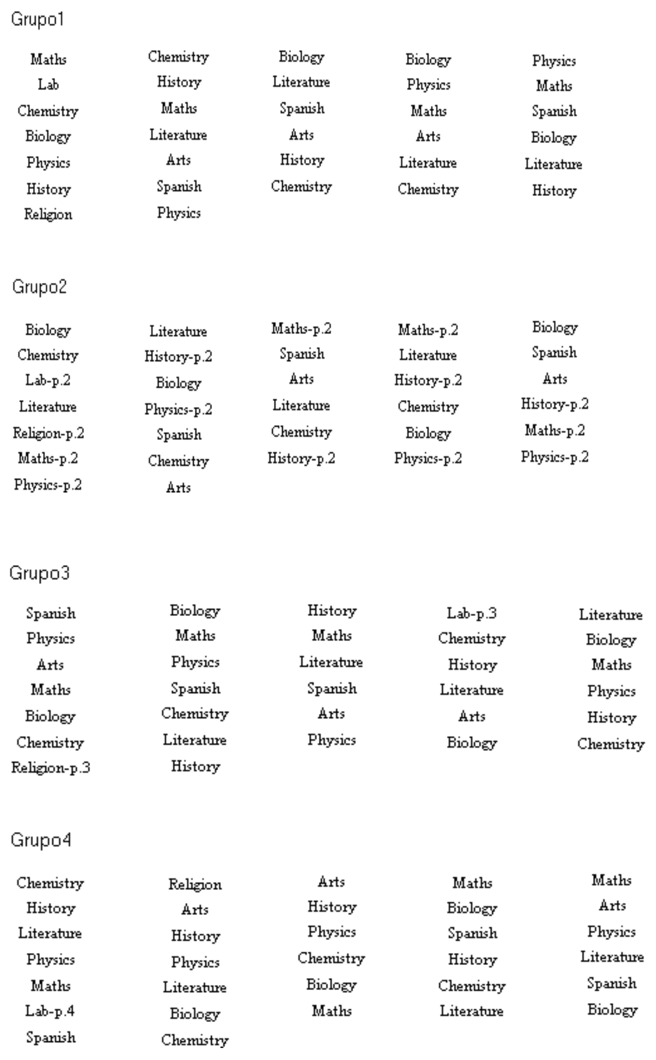
| | |
|---|---|
| Monday | 7 |
| Tuesday | 7 |
| Wednesday | 6 |
| Thursday | 6 |
| Friday | 6 |

| | |
|---|---|
| Nº groups | 4 |

| Subject | Nº Hours/week | Nº teachers |
|---|---|---|
| Literature | 4 | 1 |
| History | 4 | 2 |
| Biology | 4 | 1 |
| Spanish | 3 | 1 |
| Physics | 4 | 2 |
| Maths | 4 | 2 |
| Chemistry | 4 | 1 |
| Arts | 3 | 1 |
| Religion | 1 | 3 |
| Lab | 1 | 5 |
| | | |

**Fig. 3.** Graphical User Interface developed for the application

The maximum number of generations can be bounded, and when the program finds an optimal solution it shows the results as in **Fig 4**.



Grupo1

| | | | | |
|---|---|---|---|---|
| Maths | Chemistry | Biology | Biology | Physics |
| Lab | History | Literature | Physics | Maths |
| Chemistry | Maths | Spanish | Maths | Spanish |
| Biology | Literature | Arts | Arts | Biology |
| Physics | Arts | History | Literature | Literature |
| History | Spanish | Chemistry | Chemistry | History |
| Religion | Physics | | | |

Grupo2

| | | | | |
|---|---|---|---|---|
| Biology | Literature | Maths-p.2 | Maths-p.2 | Biology |
| Chemistry | History-p.2 | Spanish | Literature | Spanish |
| Lab-p.2 | Biology | Arts | History-p.2 | Arts |
| Literature | Physics-p.2 | Literature | Chemistry | History-p.2 |
| Religion-p.2 | Spanish | Chemistry | Biology | Maths-p.2 |
| Maths-p.2 | Chemistry | History-p.2 | Physics-p.2 | Physics-p.2 |
| Physics-p.2 | Arts | | | |

Grupo3

| | | | | |
|---|---|---|---|---|
| Spanish | Biology | History | Lab-p.3 | Literature |
| Physics | Maths | Maths | Chemistry | Biology |
| Arts | Physics | Literature | History | Maths |
| Maths | Spanish | Spanish | Literature | Physics |
| Biology | Chemistry | Arts | Arts | History |
| Chemistry | Literature | Physics | Biology | Chemistry |
| Religion-p.3 | History | | | |

Grupo4

| | | | | |
|---|---|---|---|---|
| Chemistry | Religion | Arts | Maths | Maths |
| History | Arts | History | Biology | Arts |
| Literature | History | Physics | Spanish | Physics |
| Physics | Physics | Chemistry | History | Literature |
| Maths | Literature | Biology | Chemistry | Spanish |
| Lab-p.4 | Biology | Maths | Literature | Biology |
| Spanish | Chemistry | | | |

**Fig. 4.** Optimum timetables obtained using the modified GA.

## 4. Analysis of simulation results

Dealing with the problem of four classes, with 32 hours per week each, 10 different subjects and 19 teachers, the algorithm has used 10 individuals, 30% substitutions each generation, and Pr = 1, requiring less than 500 iterations to obtain the optimum.

We have studied the increase in time when the size of the problem is on the rise. When increasing the number of classes, the number of iterations does not increase so quickly as the problem dimension. In **Fig. 5** the logarithm of the number of iterations and the logarithm of the feasible search space dimension are represented.

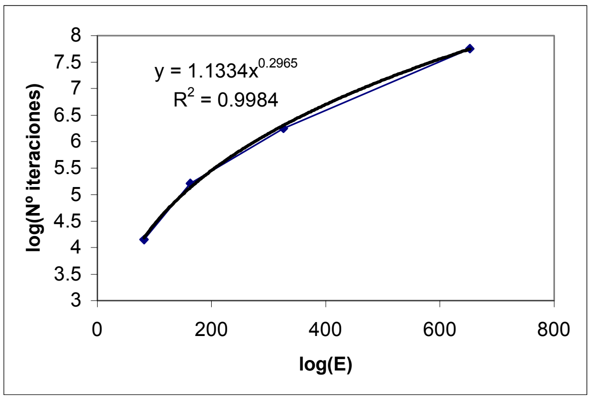


**Fig. 5.** Number of iterations related to the problem dimension.

The fitted curve gives the following relationship, with $D = (N_a!)^{Ng}$:

$$\text{N}^{\text{o}} \text{ iterations} = 1.1334\ D^{0.2965}\ . \qquad \textbf{(2)}$$

**Fig. 6** shows how the time increment varies linearly with the number of iterations, as it could be expected. So, that means that the penalization due to the increment of the dimension is very low working with this algorithm.

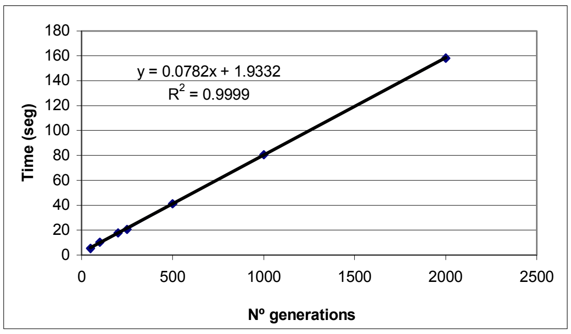


**Fig. 6.** Algorithm computation time vs. number of generations

The time penalization when increasing the population size is very high, about 1 second per additional individual. **Table 1** shows how the population increase does not lead to a better solution in less generations, when the population is more than 10 or 20 individuals. In no case it is going to be faster, but it may be a softer penalization if a parallel processor computer can be used to run the algorithm. Anyway, the increment of resources is not justified by the time saving.

**Table 1.** Iterations needed to reach the optimum according to population size.

| Population size | Nº iterations to optimum |
|---|---|
| 2 | May not find solution |
| 4 | May not find solution (>1000 iterations) |
| 10 | ~500 |
| 15 | ~400 |
| 20 | ~250 |
| 30 | ~250 |
| 50 | ~250 |

One of the main keys to design the algorithm for a particular problem is the definition of the weights associated to each constraint in the fitness function. It is important to consider not only the priority of a constraint but also to remark that if a timetable violates many of the same type of soft constraints, it will have worst fitness value that another that violates a very critical one. Adjusting these weights is a critical task for each particular problem. Comparing **Fig. 7** and **Fig. 2**, it is worth noting how the constraint with the highest weight is the one that improves faster, while the lowest one can move to worst values, so it will take longer to be fulfilled. The weights used here were -100 and –20, and when the lowest value is used for the constrain of repeated subject in the same day, the algorithm finds sooner the optimum.

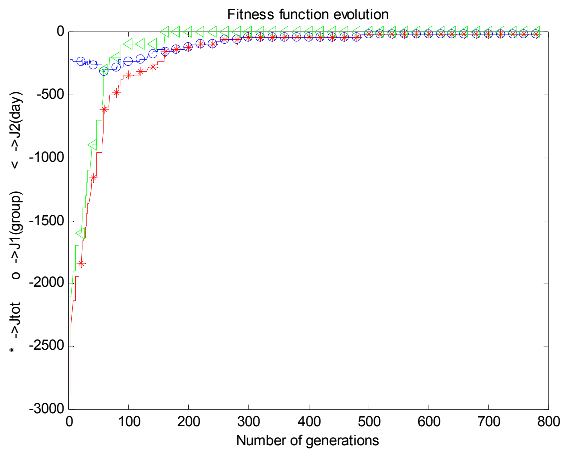


**Fig. 7.** Fitness function evolution with exchanged weights in fitness subfunctions.

## 5. Conclusions

A genetic algorithm has been adapted to a particular combinatorial problem with hard constrains. Some of the usual concepts of genetic algorithm have been modified to increase its effectiveness in this particular application. For example, in this case the crossover operator has not been used, and the mutation operation is less random than usual. That is, providing some intelligence, the solution can improve.

This strategy of using a guided mutation operator can reach optimal solutions in fewer iterations. These iterations are lower time consuming and less demanding, as no repairs need to be made in the new generated timetables.

In addition, it has been shown how this method reaches optimal solutions in a pretty fast way, requiring polynomial time when the dimension of the problem increases.

The main problem is the commissioning of the weights in the fitness function, because although including as many constrains as desired is very simple, giving them the appropriate influence in the global evaluation is a delicate task that affects the problem resolution.

It is important to notice that when a hard constrain has been included, despite it has a big influence in the algorithm, it helps to reduce the search space. So, although the design requires more efforts, the computational time is reduced and the effectiveness of the algorithm is improved.

## References

1.A. Meisels and N. Lusternik: Experimentas on networks of employee tiemtabling problems, in Proceedings of the 2nd Int. Conf. on the Practice and Theory of Automated Timetabling, 1997

2. M. W. Carter, G. Laporte: Recent Developments in Practical Course Timetabling. In Proceedings of the 2nd Int. Conf. on the Practice and Theory of Automated Timetabling, 1997

3. T. B. Cooper, J. H. Kingston: The Complexity of Timetable Construction Problems.

4. E. K. Burke, D. G. Elliman, and R. F. Weare: A genetic algorithm based university timetabling system. In Proceedings of the 2nd East-West International Conference on Computer Technologies in Education, volume 1, pages 35--40, 1994.

5. A. M. dos Santos, E. Marques, and L. S. Ochi: Desgin and implementation of a timetable system using a genetic algorithm. In Proceedings of the 2nd Int. Conf. on the Practice and Theory of Automated Timetabling, 1997

7. J P Caldeira and Agostinho C Rosa. School timetabling using genetic search. In In Proceedings of the 2nd Int. Conf. on the Practice and Theory of Automated Timetabling, 1997

6. E.K. Burke and M. Carter, editors. Proceedings of the 2nd International Conference on the Practice and Theory of Automated Timetabling, 1997.