# A TIMETABLE PREDICTION FOR TECHNICAL EDUCATIONAL SYSTEM USING GENETIC ALGORITHM

[1]SANDEEP SINGH RAWAT, [2]LAKSHMI RAJAMANI

[1]Associate Prof., Department of Computer Science and Engineering, GNIT, Ibrahimpatnam, India

[2]Prof., Department of Computer Science and Engineering, Osmania University, Hyderabad, India

E-mail: sandeep4578@gmail.com, rlakshmi@yahoo.com

**ABSTRACT**

This paper deals with the implementation of a computer program, which employs Genetic Algorithms (GAs) in the quest for an optimal class timetable generator. The program is written in Java and incorporates a repair strategy for faster evolution. This paper also explains an example usage of Genetic Algorithms (GAs) for finding optimal solutions to the problem of Class Timetable. It is seen that the GA could be improved by the further incorporation of repair strategies, and is readily scalable to the complete timetabling problem. The system at present does not take care of other constraints like unavailability of lecturers, small size of rooms and time required by the lecturer to move from one class to other class, which is to be considered in the future up gradations. The automated class timetable is used at the dept. of Computer Science & Engineering, Guru Nanak Institute of Technology (GNIT), Hyderabad, India and in future it will be used by other faculty administrators and proposes solutions to be considered by the parties involved: administration, departments and students.

**Keywords:** *Genetic Algorithms, Timetable, Scheduling, and Automated*

## 1. INTRODUCTION

The class timetabling is a major administrative activity for a wide variety of institutions. A timetabling problem can be defined to be the problem of assigning a number of events into a limited number of time periods. A. Wren defines timetabling as follows: “*Timetabling* is the allocation, subject to constraints, of given to objects being placed in space time, in such a way as to satisfy as nearly as possible a set of desirable objectives [1]”.

In this paper, we concentrate on the class-timetabling problem. The problem is subject to many constraints that are usually divided into two categories: “hard” and “soft” [4]. This work was partially supported by the Research Committee of the Guru Nanak Institute of Technology (GNIT), under the research program, in order to produce high quality timetables with optimal constraint satisfaction and optimization of the timetable’s objectives at the same time.

### 1.1 Hard Constraints

Hard constraints are rigidly enforced. Examples of such constraints are:

No lecturer should have different classes at the same time slot.

There cannot be more than 2 classes for a subject on one day

For each time period there should be sufficient resources (e.g. rooms and lecturers) available for all the events that have been scheduled for that time period.

### 1.2 Soft Constraints

Soft constraints are those that are desirable but not absolutely essential. In real-world situations it is, of course, usually impossible to satisfy all soft constraints. Examples of soft constraints (in both exam and course timetabling) are:

Every staff should get at least one first hour

Lecturer having two theory subjects has no lab assignments

Lecturer having one theory may get two lab classes.

A particular class may need to be scheduled in a particular time period.

Lab Classes may not be in consecutive hours.

Lecturers may prefer to have all their lectures in a number of days and to have a number of lecture-free days.

### 1.3 The Class Timetable Problem

Time Table problems are mainly classified as constraint satisfaction problems, where the main goal is to satisfy all problem constraints, rather than optimizing a number of objectives. At present, science has no analytical solution method for all problem cases of this category, other than exhaustive search, which however cannot be applied but only to toy problems. Due to the immense search spaces, Automated timetable scheduling, on the other hand, is a task of great importance as it can save a lot of man-hours work, to institutions and companies, and provides optimal solutions with constraint satisfaction [3]. Scheduling is the arrangement of entities (people, tasks, vehicles, lectures, exams, meetings, etc.) into a pattern in space-time in such a way that constraints are satisfied and certain goals are achieved [1]. Constructing a schedule is the problem in which time, space and other (often limited) resources have to be considered in the arrangement.

Holland’s original schema was a method of classifying objects, then selectively “breeding” those objects with each other to produce new objects to be classified [5]. Created for the direct purpose of modeling Darwinian natural selection, the programs followed a simple pattern of the birth, mating and death of life forms. A top-level description of this process is given in figure 1[5], [6], [7].

```
Create a population of creatures.
Evaluate the fitness of each creature.
While the population is not fit enough:
{
Kill all relatively unfit creatures.
While population size< max;
{
Select two population members.
Combine their genetic material to create a new creature.
Cause a few random mutations on the new creature.
Evaluate the new creature and place it in the population.
}
}
```

*Figure 1: Top Level description of a GA.*

Genetic algorithms (GAs) are evolutionary algorithms that use the principle of natural selection to evolve a set of solutions toward an optimum solution. GAs are not only very powerful, but are also very easy to use as most of the work can be encapsulated into a single component, requiring users only to define a fitness function that is used to determine how “good” a particular solution is relative to other solutions.

The creatures upon which the genetic algorithm acts are composed of a series of units of information- referred to as genes. The genes, which make up each creature, are known as the chromosome. Each creature has its own chromosome.

A GA, as shown in figure 1 requires a process of initializing, breeding, mutating, choosing and killing. The order and method of performing each of these gives rise to many variations on Holland’s original schema.

### 1.4 Chromosome Encoding, Fitness, Crossover and Mutation

Holland encoded chromosomes as a string of binary digits. A number of properties of binary encoding work to provide simple, effective and elegant GAs. There are, however, many other ways to represent a creature’s genes, which can have their own implicit advantages. In order to get a problem into gene form, the substance of its solution must be represented as a collection of units of information [8]. This is true of many problems. For example, when designing a weekly budget, the amount spent on each item could be stored as a number in a column. This can be thought of as not just a list of values but a string of genes. The value in the first row might represent the amount of money to spend on rice, and the second row might be the amount of money to spend on caviar and so on. Each of these values might be converted from base 10 to base 2 to create a fixed width binary number. Hence the problem of minimizing your budget while maintaining your survival is translated into a genetic representation. A collection of possible budgets could be thus encoded, producing a population of Budget creatures. Random populations are almost always extremely unfit [8]. In order to determine which are fitter than others, each creature must be evaluated. In order to evaluate a creature, some knowledge must be known about the environment in which it survives. Depending on the way we structure the method of evaluating a chromosome we can either aim to generate the least costly population or the most fit; it is a question of minimizing cost or maximizing fitness. In the budgeting example, the heuristic concerning caviar can be represented with a cost. In optimization problems cost is not a measure of money, but a unit of efficiency [7], [8]. When discussing optimization techniques, the range of possible solutions is often referred to as the solution space and the cost/fitness of each point in the solution space is referred to as the altitude in the landscape of the problem. To looking for the global minimum of the cost is also to look for the lowest point in the lowest valley of the cost landscape. Similarly, to look for the global maximum fitness is to look for the highest point of the highest mountain in the fitness landscape.

In various GAs, the method of selecting creatures for breeding is handled in different ways. Holland's original model uses a method where the healthiest are most likely to breed. Other methods select any two creatures at random for breeding. Selective breeding can be used in conjunction with or in the absence of an Elitist Natural Selection Operator- in either case the GA can perform evolution [7]. Once parents have been chosen, breeding itself can then take place. A new creature is produced by selecting, for each gene in the chromosome, an allele from either the mother or the father. The process of combining the genes can be performed in a number of ways. The simplest method of combination is called single point cross-over [5], [7], [8]. This can be best demonstrated using genes encoded in binary, though the process is translatable to almost any gene representation. A child chromosome can be produced using single point crossover, as shown in figure 2. A crossover point is randomly chosen to occur somewhere in the string of genes. All genetic material from before the crossover point is taken from one parent, and all material after the crossover point is taken from the other [8].

```
Two parents have already been selected:
PARENT1:
101101010101001001001001110011100110101011101101
PARENT2:
010100111011010101110101001001101011001010010110
Choose a crossover point:
PARENT1:
 1011010101010010
01001001110011100110101011101101
PARENT2:
0101001110110101 01110101001001101011001010010110
Perform crossover to produce a child:
CHILD:
1011010101010010 01110101001001101011001010010110
Which then becomes, a whole new chromosome:
CHILD:
101101010101001001110101001001101011001010010110
```

*Figure 2: An Example of Crossover with Fully encoded Genes*

After crossover is performed and before the child is released into the wild, there is a chance that it will undergo mutation. The chance of this occurring is referred to as the mutation rate. This is usually kept quite small [8]. The purpose of mutation is to inject noise, and, in particular, new alleles, into the population. This is useful in escaping local minima as it helps explore new regions of the multi dimensional solution space [7]. Once a gene has been selected for mutation, the mutation itself can take on a number of forms. This, again, depends on the implementation of the GA. In the case of a binary string representation, simple mutation of a single gene causes that genes value to be complemented- a 1 becomes a 0 and vice versa.

In Holland's founding work on GAs he made mention of another operator, besides selection, breeding, crossover and mutation which takes place in biological reproduction. This is known as the inversion operator [8]. An inversion is where a portion of chromosomes detaches from the rest of the chromosome, then changes direction and recombines with the chromosome.

We present an approach to solve this large, highly constrained timetabling problem, based on evolutionary algorithms. In chapter two we describe the proposed system taking as an example the department of Computer Science and Engineering, GNIT. Third chapter gives the implementation details about the algorithm. While the fourth chapter shows the results of the program (written in java) and chapter five shows the conclusion and future works.

## 2. PROPOSED SYSTEM

The proposed system has 'L' no of lecturers, 'S' no of subjects and 'C' no of classes per subject per week. Each day has 'H' no of hours and we have five working days per week. The total no of time slots then become equal to 5*H. The problem then becomes assigning 'S*C' number of classes in the '5*H' time slots. For example in the department of Computer Science and Engineering, VITS, there are twelve lecturers and 24 subjects with 4 classes per subject per week giving a total number of 96 classes per week. Each day has six hours and five days per week giving 120 time slots. See Table 1 and Table 2.

*Table 1: Subjects allotted to Lecturers*

| LECTURER | SUBJECTS |
|---|---|
| RAM | C Language, PPL |
| Sandy | BDP, Disc. Struct |
| Patil | DS, OS |
| Paritosh | DC, CN |
| Tiwari | NM, SE |
| Pandey | ALC, DBMS |
| Deepthi | CC, SAD |
| Mohan | OOPS, WP |
| Harish | EG, ET Lab, C Lab |
| Shyam | RTS, DC Lab, SAD Lab |
| Radhe | AI, OOPS Lab, OS Lab |
| Ritu | CG, WP Lab, AI Lab |

*Table 2: Subjects per year*

| YEAR | SUBJECTS |
|---|---|
| First | C Lang., DS, Disc.Struct, NM, EG |
| Second | DC, BDP, SAD, ALC, PPL |
| Third | OOPS, OS, CG, DBMS, CN |
| Final | RTS, CC, WP, AI, SE |

## 3. GA IMPLEMENTATION

A program in java was developed which employed GA methods to perform Automated Timetabling. Consequently, the program was entitled "gaatt.java". The GA operates upon a population of timetables, which are maintained in memory. Each timetable is evaluated by testing the number of times it breaches each constraint. Thus timetables are evolved with a minimum number of constraint violations. A top-level description of "gaatt.java" is provided as figure 3. It can be seen that this structure is similar to the pseudo code given in figure 1 with the major difference being the incorporation of a repair strategy. Explanation of each of the components of the program is given in the remainder of this section.
Load all constraint data from a constraint file.

```
While the population size is less than the maximum:
{
Create a new timetable with no classes booked to it.
Repair the new timetable by using the constraint data.
Evaluate the cost of the new timetable by using the constraint data.
Enter the new timetable into the population.
}
While the cost of the best timetable is greater than zero:
{
Discard a portion of costly timetables.
Repeat until the population size is maximum:
{
Breed a new timetable.
Mutate the new timetable.
Repair the new timetable by using the constraint data.
Evaluate the cost of the new timetable by using the constraint data.
Enter the new timetable into the population.
}
}.
```

*Figure 3: Top Level Description of proposed system*

### 3.1 Constraint Data

In order to test for each of the types of hard constraint it is necessary to store sufficient detail about the department. This means that information concerning all lecturers; classrooms and classes must be maintained. The way in which each of these data types are implemented will now be given in detail. A class is a structured type with three fields. Each class has a certain size (predicted size), a lecturer number and a number indicating the code number of the group of related classes to which it belongs. The set of all lecturers is stored as an array. Similarly, there are arrays of classes and of classrooms. Each of these elements is identified uniquely by its position in the relevant array. The set of all information concerning lecturers, classes and classrooms is termed the constraint data.

The timetable for a single room is a two-dimensional array as shown in Table 3. Each Field describes (decode) some aspect of genetic information particularly the cost or number of breaches. Times at which there is no class booked hold a NULL booking, which has a value of zero.

*Table 3: Timetable for Room 301 at an intermediate stage depicting costs at various time slots.*

| **Room 301 Day/Time** | **8:45 9:35** | **9:35 10:25** | **10:25 11:15** | **12:05 12:55** | **12:55 1:45** | **1:45 2:35** |
|---|---|---|---|---|---|---|
| **Day1:** | 23 | 14 | 1 | 21 | 0 | 20 |
| **Day2:** | 3 | 13 | 19 | 22 | 2 | 0 |
| **Day3:** | 9 | 0 | 0 | 18 | 6 | 12 |
| **Day4:** | 15 | 5 | 8 | 16 | 0 | 11 |
| **Day5:** | 24 | 10 | 17 | 4 | 7 | 0 |

A class timetable stores information about what classes are booked in each room, at any hour of the day, on any day of the week. Each of these bookings (or NULL bookings) is one gene. A population is a collection of timetables. A population is itself a structured type with a number of fields. It contains a pointer to the least costly timetable in the population, (which has, in turn, a pointer to the next least costly). There is also a pointer to the most costly timetable in the population, as well as a field storing the average cost, and the average number of violations of each type of hard constraint.

### 3.2 Repair Strategy

A repair strategy is used which ensures that all classes appear exactly once. For robustness this is done in two stages. Firstly, any classes, which appear more than once, as shown in, figure 4. Secondly, any classes, which did not appear at all, are booked to a spare space (regardless of room size, etc) as shown in figure 5. If this repair strategy is applied to an empty timetable the result is a timetable with each class booked to a random time and place. As such, the repair strategy is also used for initializing a random population. The use of the repair strategy ensures that each class is booked exactly once. Hence, the number of hard constraints, which must be considered when timetables are being evaluated, is further reduced.

```
For each class:
            Set the Count to 0.
For each time:
    For each room:
        If the current class is booked at this location:
                Add 1 to the count.
                Add the location of this class to a
                linked list.
        If the class occurred more than once then:
            Keep doing the following until there is
            only one booking left:
            Randomly choose one of the bookings.
            Turn it into a NULL booking.
Free the linked list.
```

*Figure 4: Pseudo Code for the First Stage of the repair strategy*

```
For each class:
     For each time:
          Look in each room until either the class is
          seen or you get to the end.
          If you got to the end without finding the
          class then randomly find a
NULL booking and book that class to it.
```

*Figure 5: Pseudo Code for the Second Stage of the repair strategy*

## 3.3 Breeding Timetables

Timetables are randomly selected from the population and used for breeding. No favoritism is given to fitter timetables. A child timetable is bred by performing unity order based crossover on the parents. This means that each parent has an equal chance of providing each gene.

## 3.4 Mutating Timetables

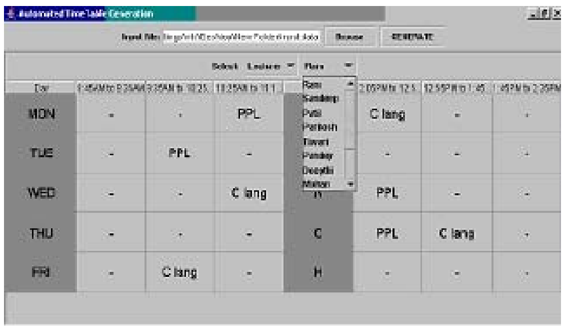
The method of mutation is given in figure 6. This means that the chance of any one gene undergoing mutation is approximately twice the mutation rate divided by one thousand. A mutation rate equal to ten, for example, implies that approximately twenty in every thousand genes will be mutated. This method should be scalable to the complete problem. The incorporation of further constraints, and the up scaling of the problem size should not require any changes to the overall architecture.

```
There is a fixed mutation rate.
For each gene
{ Randomly choose a number between 1 and
   1000.
  If the number is less than the mutation rate then
  { Randomly choose a gene from the current
  timetable and swap it with the current gene.
  }
}
```

*Figure 6: Pseudo Code for Method of Mutation*

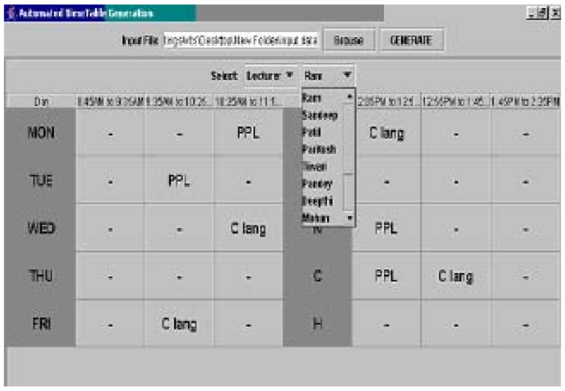
## 4. SIMULATION RESULTS

First we conducted a simulation experiment using the standard GA setup for the system described in the previous section. The experiment consisted of 10 independent runs. After the completion of the runs the program presents the timetables room wise and staff wise. Each output gives the different results. The output timetable of a sample run is shown below in figure 7 and figure 8.



| Day | 8:45AM to 9:35AM | 9:35AM to 10:25 | 10:25AM to 11:1 | | 12:05PM to 12:5 | 12:55PM to 1:45 | 1:45PM to 2:35PM |
|---|---|---|---|---|---|---|---|
| MON | - | - | PPL | | C lang | - | - |
| TUE | - | PPL | - | | - | - | - |
| WED | - | - | C lang | N | PPL | - | - |
| THU | - | - | - | C | PPL | C lang | - |
| FRI | - | C lang | - | H | - | - | - |

Figure 7: Output Timetable (room wise)



| Day | 8:45AM to 9:35AM | 9:35AM to 10:25... | 10:25AM to 11:1... | | 12:05PM to 12:5... | 12:55PM to 1:45... | 1:45PM to 2:35PM |
|---|---|---|---|---|---|---|---|
| MON | - | - | PPL | | C lang | - | - |
| TUE | - | PPL | - | | - | - | - |
| WED | - | - | C lang | N | PPL | - | - |
| THU | - | - | - | C | PPL | C lang | - |
| FRI | - | C lang | - | H | - | - | - |

Figure 8: Output Timetable (staff wise)

## 5. CONCLUSIONS AND FUTURE WORK

The work was done and implemented as per the requirements of the department. It has been seen that “gaatt.java” produced timetables void of hard constraint violations and the results were very promising. Ultimately, it is quite possible that optimal performance of the GA would require continual fine-tuning of the mutation rate. It is assumed that “gaatt.java” was executed on a serial machine. As such, larger populations take more time in between each selection for extinction so lower performance chromosomes are given more of a chance to breed. On a parallel machine larger populations could be expected to perform better. It cannot be ruled out that even on the current hardware a different tuning of the GA parameters could allow larger populations to perform better than minimally small ones.

The system at present does not take care of other constraints like unavailability of lecturers, small size of rooms and time required by the lecturer to move from one class to other class, which is to be considered in future work.

## REFERENCES

[1] A. Wren (1996), “Scheduling, Timetabling and Rostering – A Special Relationship?,”, in The Practice and Theory of Automated Timetabling: Springer Lecture Notes in Computer Science Series, Vol. 1153, pp. 46-75.

[2] Bagchi T.P., Multiobjective (1999) Scheduling By Genetic Algorithms, Kluwer Academic Publishers.

[3] Ehrgott M., Gandibleux X.(2000), A Survey and Annotated Bibliography of Multiobjective Combinatorial Optimization, OR Spectrum, Vol. 22, No. 4, Springer, pp. 425-460.

[4] Burke E.K., Newall J.P., Weare R.F(1998)., A Simple Heuristically Guided Search for the Timetable Problem, Proceedings of the International ICSC Symposium on Engineering of Intelligent Systems. ICSC Academic Press,Nottingham,.

[5] Buckles BP and Petry FE (1992): Genetic Al algorithms. Los Alamitos: The IEEE Computer Society Press.

[6] Melanie Mitchell, “An introduction to Genetic Algorithms”, Prentice Hall India.

[7] Gen M and Cheng R (1997): Genetic Algorithms and Engineering Design. John Wiley,NY.

[8] Davis L (Ed) (1991): Handbook of Genetic Algorithms. New York: Van Nostrand Reinhold.

[9] De Werra D (1995): Some Combinatorial Models for Course Scheduling. In Burke E and Ross P (Eds): First International Conference, dinburgh, U.K., August/September.

[10] Burke E and Ross P (Eds) (1996): Lecture Notes in Computer Science 1153 Practice Corne D and Ross P.

[11] Burke EK, Newall JP and Weare RF(1995) A Memetic Algorithm for University Exam Timetabling. In Burke E and Ross P (Eds).

## ACKNOWLEDGEMENT

This work is partially supported by the Research Committee of the Guru Nanak Institutions (GNI), Hyderabad, India. The authors would like to thankfully acknowledge the support and cooperation of the teaching and non-teaching staff of Guru Nanak Institute of Technology, Hyderabad and College of Engineering, Osmania University, Hyderabad during this work.

## BIOGRAPHY

**Mr. Sandeep Singh Rawat** received his Bachelor of Engineering in Computer Science from National Institute of Technology - Surat (formerly REC - Surat), India and his Masters in Information Technology from Indian Institute of Technology, Roorkee, India. He is pursuing his Ph.D. at Osmania University, Hyderabad., India.

He has presented three technical papers at international conferences and published paper in journals including IEEE Delhi Section and IEEE Computer Society Chapter, India.

Mr. Rawat is a member of ISTE and CSI. His research interest includes Data Mining, High Performance Computing and Machine Learning.

**Dr. Lakshmi Rajamani** received the Ph.D. degree in Computer Science & Engineering from the Jadavpur University. Currently, she is a professor and head of the department at University College of Engineering, Osmania University, Hyderabad, Andhra Pradesh, India.

She has presented many technical papers both at national and international conferences and published paper in journals.

Dr. Lakshmi is a member of ISTE and CSI. Her research interests include Artificial Intelligence, High Speed Computing and Database Technology.