A Hybrid Genetic Algorithm for Curriculum Based Course Timetabling

Soolmaz Massoodian[1], Afsaneh Esteki[2]

February 2008

Abstract. In this paper we describe a genetic algorithm-based approach with two main stages for solving the course timetabling problem. A local search is applied to the algorithm at each stage. The first stage eliminates the violations of hard constraints, and the second one attempts to minimize the violations of soft constraints while keeping the number of hard constraint violations zero.

Keywords. Genetic Algorithm, Local Search, Timetable, Course Timetabling, Optimization

1. Introduction

The course timetabling problem involves assigning lectures to the time periods and lecture rooms. A number of soft and hard constraints are to be satisfied. A timetable is feasible if and only if all the hard constraints are satisfied. The goal of our algorithms is to find a timetable with no violations of hard constraints and minimum possible violations of soft ones in a given time.

The course timetabling problem can be different from one university to another. In this paper, our focus is on the curriculum based course timetabling problem described in the track 3 of the second international timetabling competition (ITC2007).

We present a system for automated construction of timetables. Our algorithm is based on Genetic Algorithm, with a local search which is applied to the algorithm in some generations and in different domains.

In the problem presented in ITC2007, it was assumed that for all the input datasets, it is possible to find at least one feasible timetable. [1] So our focus in the first stage is on the hard constraints only.

[1] soolmaz_massoodian@yahoo.com
BSc, Computer Engineering,software. The University of Isfahan. Isfahan, Iran
[2] esteki_afsaneh@yahoo.com
BSc, Computer Engineering,software. The University of Isfahan. Isfahan, Iran

2. Description of the Problem

Below is the description of the problem according to the website of the second international timetabling competition [1].
The problem consists of the following entities:
Days, Timeslots, and Periods. We are given a number of teaching days in the week (typically 5 or 6). Each day is split in a fixed number of timeslots, which is equal for all days. A period is a pair composed by a day and a timeslot. The total number of scheduling periods is the product of the days times the day timeslots.
Courses and Teachers. Each course consists of a fixed number of lectures to be scheduled in distinct periods; it is attended by given number of students, and is taught by a teacher. For each course there is a minimum number of days that the lectures of the course should be spread in, moreover there are some periods in which the course cannot be scheduled.
Rooms. Each room has a capacity, expressed in terms of number of available seats. All rooms are equally suitable for all courses (if large enough).
Curricula. A curriculum is a group of courses such that any pair of courses in the group have students in common. Based on curricula, we have the conflicts between courses and other soft constraints.

The timetables are feasible if and only if the following hard constraints are satisfied:
Lectures: A missing or extra lecture of a course.
Conflicts: Two conflicting lectures[3] in the same period. Three conflicting lectures count as 3 violations: one for each pair.
Room Occupancy: Two lectures in the same room at the same period. Any extra lecture in the same period and room counts as one more violation.
Availabilities: Each lecture in a period unavailable for that course.

The following soft constraints should be satisfied if possible:
Room Capacity: For each lecture, the number of students that attend the course must be less or equal than the number of seats of all the rooms that host its lectures. Each student above the capacity counts as 1 point of penalty.
Minimum Working Days: The lectures of each course must be spread into a minimum number of days. Each day below the minimum, counts as 5 points of penalty.
Curriculum Compactness: Lectures belonging to a curriculum should be adjacent to each other (i.e., in consecutive periods). For a given curriculum we account for a violation every time there is one lecture not adjacent to any other lecture within the same day. Each isolated lecture in a curriculum counts as 2 points of penalty.
Room Stability: All lectures of a course should be given in the same room. Each distinct room used for the lectures of a course, but the first, counts as 1 point of penalty.

[3] Lectures belonging to the courses of the same curriculum or taught by the same teacher

3. The algorithm

3.1 pseudo code

A pseudo code of the algorithm is presented here:

```
//Reads all the date from file
ReadData();

//Assigns integer values to the lectures
EncodeData();

//Sorts the courses to place lectures of more constrained courses in the timetable first
sortCourses(CourseList);

//Creates a semi-random population of timetables
Initialize();

for(All timetables)
        //Calculates the fitnessHard value
        fitnessHARD(timetable);

generations=1;
do {

//SELECTION POOL

//Selects the better half of the population for the first half of the new population
SelectBetterHalf();

//Selects the other half of the new population using tournament-3
SelectOtherHalf();

//CROSSOVER

//Crossover is not performed on the better half of population
for(i=population_size/2;i<population_size;i++)
{
        Choose a random number a;
        if(a<=crossover rate)
        //parent1 will be replaced by the offspring of parent1 and parent2
                Crossover(parent1,parent2,parent1);
        If (a feasible timetable has not found yet)
        // Calculate only the violations of hard constraints for timetables;
                fitnessHARD(i);
        else
```

```
        // Calculate the actual fitness of the timetables;
                fitness(i);
}

If(Maximum and average fitness of the population are very close)
        //To avoid convergence
        Increase mutation rate;

//sorts the chromosome pool to prevent performing mutation on a good
timetable created by crossover
sortpool(pool);

//MUTATION

//Mutation is not performed on good chromosomes
for(i=population_size/4;i<population_size;i++)
{
        Choose a random number a;
        if(a<=mutation_rate)
                mutation(i);

        If (a feasible timetable has not found yet)
                fitnessHARD(i);
        else
                fitness(i);

}

If(Max fitness of this generation>Max fitness of the previous generation)
        If(A feasible timetable has not found yet)
                Local search is applied to best chromosome;
        else
                Local search is applied to the columns of the best chromosome;

} while(!( (time is over) OR (a timetable for which all soft and hard constraints
are satisfied is found)))
```

3.2 How it works

The timetables represent the chromosomes. The terms chromosome and timetable are used in this paper interchangeably. The columns in the timetables represent the periods (day and timeslot) and the rows represent the rooms, therefore each location of a timetable defines a room and a period. Every integer number assigned to a course lecture will be placed in an empty timetable's location (See Fig.1). The circled numbers 23, 24, 25 and 26 in Fig.1 belong to the lectures of the same course. Using this method, we never have a missing or extra lecture or more than one lecture in the same location. So the only hard constraints to consider will be conflicts and availability.

The algorithm sees the timetables as 1D arrays of integers (See Fig.2). The input data is first encoded using the permutation encoding. Then a population of chromosomes in the initialization step is generated to form the first generation. In each generation the operations mutation and crossover are applied to a subset of population which does not contain the best chromosomes. The population size is fixed; meaning in each generation, for every good chromosome added to the population, a bad chromosome is discarded.

Since we want to find the best possible timetable in a limited time, we focus on the hard constraints first. Our algorithm has two main stages. In the first stage, we only consider the hard constraints and try to find a timetable in which all the hard constraints are satisfied. This means the fitness function we use at this stage, calculates only the violations of hard constraints, therefore the fitness value it assigns to each timetable is not the actual fitness of it, but it works for comparing the timetables. We refer to this fitness as fitnessHard.

To save the time, the local search is applied only to the best chromosome of the population. We apply the local search every time an improvement is made in the maximum fitnessHard of the population. In the end of the first stage we have at least one feasible timetable.

The algorithm goes to the second stage if and only if a feasible timetable is found. In this stage the main fitness function is used and the actual fitness for each timetable is calculated. Every time an improvement is made in the maximum fitness of the population, a local search is applied to the columns of best chromosome of the population. In this local search, for calculating the fitness of the timetable to which the local search is being applied, after every exchange of values within the same column, a different fitness function is used. This fitness function calculates only the violations of soft constraints. Since the only possible violations of hard constraints in our algorithm are conflicts and availability constraints, a feasible timetable can never become infeasible by exchanging values in any pairs of its locations which are within the same column. So we don't need to calculate the hard constraints violations.

	Mon 8-9	Mon 9-10	.....	Wed 13-14	.....	Thu 8-9	.....	Fri 17-18	Fri 18-19
Room 1	23	24	.....	56	.....	45	.....	8	22
Room 2	76	12	.....	53	.....	33	.....	39	13
Room 3	29	59	.....	28	.....	81	.....	44	75
.....	18	42	.....	15	.....	14	.....	34	80
.....	89	9	.....	62	.....	5	.....	72	49
Room n-1	41	30	.....	78	.....	25	.....	57	86
Room n	79	60	.....	71	.....	63	.....	10	26

Fig.1.A timetable

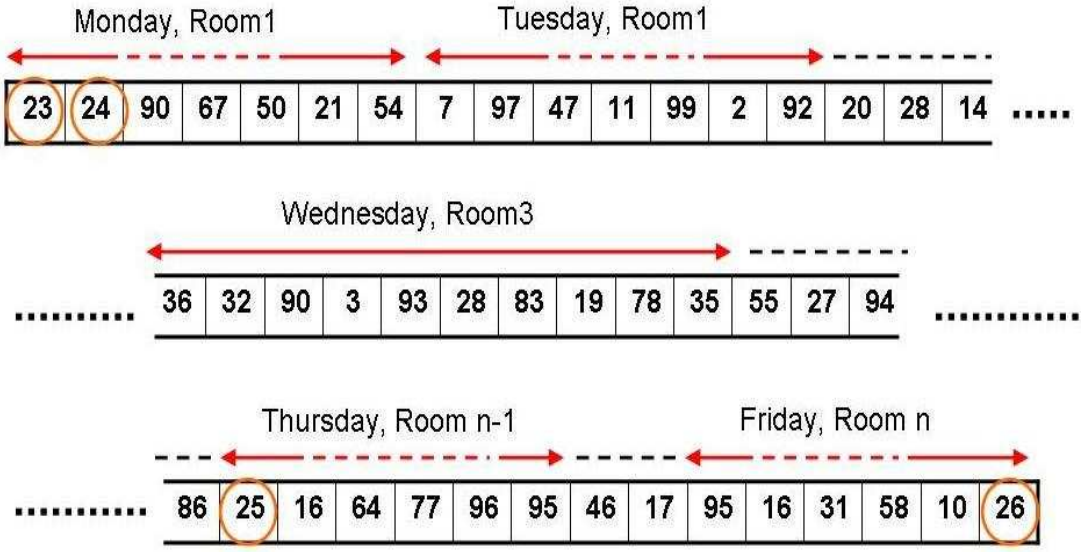

Fig.2.Chromosomes as 1D arrays of integer

3.3 Encoding

To encode the data, we use permutation encoding [2]. In this method the chromosomes are a sequence of integer values, from 1 to the number of genes, with no missing or repeated number. A location having a value greater than the total number of lectures is considered an empty location.

We assign to each course as many integer numbers as the number of its lectures, starting from 1. So the biggest integer number assigned is equal to the total number of lectures. After assigning values to the courses, the course list is sorted.

3.4 Initialization

The algorithm in the initialization step creates a semi-random population of chromosomes. The algorithm does not place a course lecture in a timeslot which is not available for that course, and we try to keep this constraint satisfied in the next generations as much as possible.

Since the course list is sorted, the integer numbers belonging to the courses which are more constrained will be placed in the timetable first. Integer numbers which start immediately after the last number assigned to a lecture will be assigned to empty locations of the timetable. So in each timetable we have all the numbers from 1 to the total number of timetable locations, with no missing or repeated number.

3.5 Evaluation

A fitness value is assigned to each timetable based on its number of violations of hard and soft constraints. In each generation the chromosomes are evaluated and then the chromosome pool is sorted. We use a fitness function to calculate the fitness of each timetable [3]. Our fitness function considers a much bigger penalty for a violation of a hard constraint than one of a soft constraint. We don't want to lose a feasible timetable for an unfeasible one with a smaller number of soft constraints under any circumstances.

$$Fitness = \frac{1000}{0.999 \times \left(\sum H_i \times hard_i \right) + 0.001 \times \left(\sum S_i \times soft_i \right) + 1}$$

Here H_i and $hard_i$ are the penalty value imposed to the violation of the hard constraint ith and the total number of its violations respectively. S_i and $soft_i$ are the penalty value imposed to the violation of the soft constraint ith and the total number of its violations respectively.

3.6 Selection

In each generation we want the better chromosomes to replace the very bad ones. So the algorithm doesn't perform the crossover and mutation operations on the best chromosomes. In the selection step, the better half of the population is directly sent to the next generation. The other half is selected using a Tournament-K method [2]. In this method, K chromosomes are chosen from the population, and the one with a greater fitness value will be copied into the next generation. This is repeated until the desired number of chromosomes is copied into the next generation.

3.7 Crossover

Partially Mapped Crossover (PMX) [4] is employed in this algorithm. In PMX, the two parents are broken in 2 points. The part between the two points from the first parent is copied directly into the child, and the rest is copied into the child from the second parent in a way it does not cause any missing or extra integer number. Using this method we never have a timetable with a missing or repeated lecture. Since the population size is fixed, the child will be replaced the first parent. Crossover will not be applied to the best chromosomes.

3.8 Mutation

The mutation function randomly exchanges the values in every pair in a random number of location pairs of a timetable. The mutation function checks availabilities before every exchange of values, not to cause any violation of availability constraint. The mutation rate is not fixed. In each generation the maximum and average fitness values are calculated. If the two numbers are close enough to each other, the mutation rate is increased to avoid any chance of an early convergence.

4. Results and conclusions

In this paper a course timetabling problem was presented and a solution based on genetic algorithm and local search was proposed. By increasing the mutation and crossover rates, and sending the better half of the population directly to the selection pool, plus the semi-random nature of our initialization step, we avoided an early convergence, and some improvements were observed. Figure.3 shows a comparison between a basic genetic algorithm and the algorithm after the changes above.

In the basic genetic algorithm, there is a risk of losing a good chromosome by applying mutation or crossover operation to it, thus the curve for this type of genetic algorithm is not necessarily ascending, while in our improved genetic algorithm the curve is ascending.

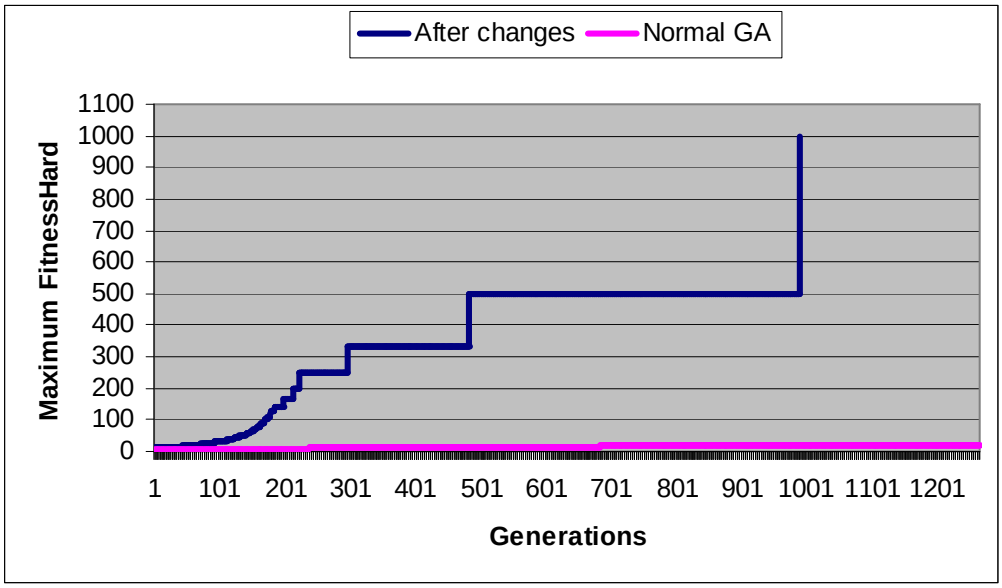

Fig.3 Basic GA and the improved GA

Then a local search was added to the algorithm as we described earlier, and Great improvements were observed. In Figure.4 the time it took for the first algorithm (the algorithm with no local search) to find a feasible timetable was 786 seconds, while by using the local search and running the program with the same random seed, it was decreased to 99 seconds. The fitness values on the vertical axis in figures 3 and 4 are the fitness values calculated in the first stage, thus they are not the actual fitness values, and are based only on the number of violations of hard constraints.

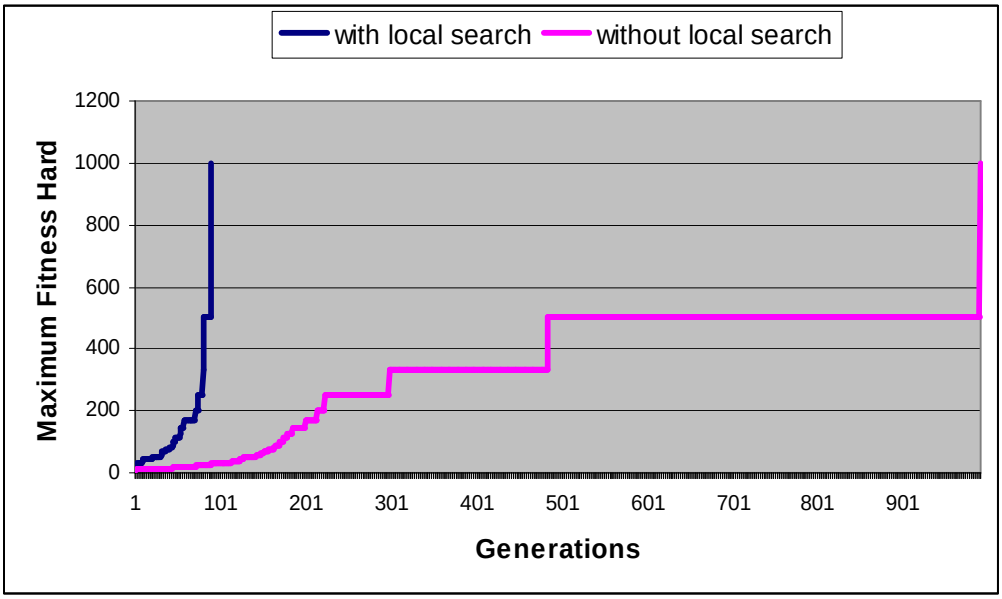

Fig.4 The effect of the local search on the first stage

Figure.5 shows result for the same run of the algorithm as Figure.4, after a feasible timetable is found. The fitness values on the vertical axis in figure 5 are the actual fitness values which are calculated in the second stage.
It is obvious in this figure that the local search speeds up the algorithm in improving the maximum fitness of the population.

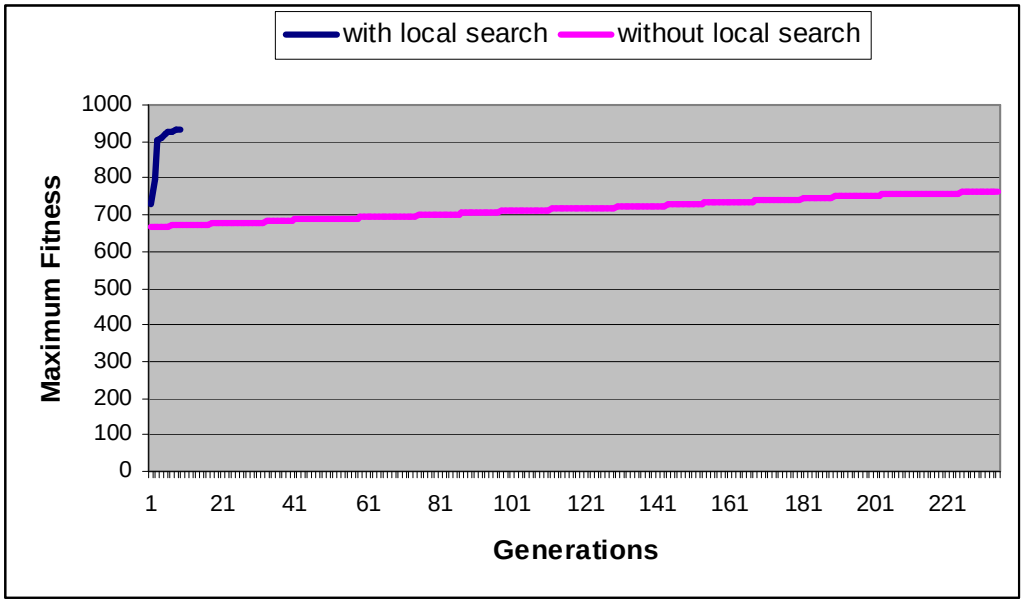

Fig.5 The effect of the local search on the second stage

References:

[1] http://www.cs.qub.ac.uk/itc2007

[2] Whitely, D.: A Genetic Algorithm Tutorial, Computer Science department, Colorado State University, 2003.

[3] Erben, W.;Keppler, J. : A Genetic Algorithm Solving a Weekly Course-timetabling Problem. Department of Computer Science, Fachhochschule Konstanz, D-78462 Konstanz, 1995.

[4] Goldberg,D.E. :Genetic Algorithms, In Search, Optimization and Machine Learning,1998,Addison-Welsy Publishing Co.

[5] Rossi-Doria, O.; Paechter, B., A Memetic Algorithm for University Timetabling. School of Computing, Napier University.

[6] W.Carter, M.: A Comprehensive Course Timetabling and Student Scheduling System at the University of Waterloo. Mechanical and Industrial Engineering, University of Toronto.

[7] Abramson, D.; Abela, J.: A Parallel Genetic Algorithm for Solving the School Timetabling Problem. High Performance Computation Project Division of Information Technilogy. Appeared in 15 Australian Computer Science Conference, Hobart, Feb 1992, pp1-1.

[8] Burke, E.; Elliman, D., Weare, R.: A Genetic Algorithm for University Timetabling. Department of Computer Science, University of Nottingham, NG7 2RD.

[9] Arntzen, H.; Lokketangen, A.: A Local Search Heuristic for a University Timetabling Problem.

[10] Abdullah, S.; K.Burk, E., McCollum, B.: A Hybrid Approach to the University Course Timetabling Problem.

[11] B. Cooper, T.; H. Kingston, J.: The Complexity of Timetable Construction Problems. Technical Report Number 495, February 1995, Basser Department of Computer Science University of Sydney NSW 2006