# Solving Timetabling Problems by Hybridizing Genetic Algorithms and Tabu Search

Malek Rahoual[1], Rachid Saad[2]

[1] IBISC - FRE 2873 CNRS - Université d'Evry Val d'Essonne,
Tour Evry 2, 523 place des terrasses de l'Agora, 91000 EVRY, France.
mrahoual@lami.univ-evry.fr

[2] Département d'informatique, Université Mohamed Bougara, Boumerdès, Algeria.
rachid_saad2003@yahoo.com

**Abstract.** As demand for Education increases and diversifies, so does the difficulty of designing workable timetables for schools and academic institutions. Besides the intractability of the basic problem, there is an increasing variety of constraints that come into play. In this paper we present a hybrid of two metaheuristics (genetic algorithm and tabu search) to tackle the problem in its most general setting. Promising experimental results are shown.

**Keywords***:* timetables, combinatorial optimization, genetic algorithms, tabu search, computational complexity.

## 1 Introduction

In academic institutions, nested groups of students (comprising streams, sections …) are concerned by a set of subjects. A subject may be a lecture of some specific course or a tutoring or a lab, or any other meeting involving the group on a regular basis. For example, the lecture of the course entitled *MATH3802A* is a subject. Tutoring associated with the same course is another subject. Each subject takes a certain length of time whose unit is referred to as a '*period*'. Each subject may be broken down into a number of meetings to be scheduled.

To solve the timetabling problem (TTP) is to assign a qualified teacher to each subject and a time-slot together with a classroom of a suitable capacity and characteristics to each meeting. The assignment of times-slots, classrooms and teachers is subject to constraints that depend on the nature of the institution and its priorities. These constraints fall into three categories: physical constraints, which provide among others that no student can attend two different meetings at the same time; preference constraints and specification constraints bearing on some particular meetings, which, for example, must be held in some specified time window [3][4][5][8][10].

E. K. Burke, H. Rudová (Eds.): PATAT 2006, pp. 467–472. ISBN 80-210-3726-1.

## 2 Application of genetic algorithms to timetabling problem

A Genetic Algorithm (GA) makes a population of individual solutions evolve under the control of two operators: 'mutation' and 'crossover'. Mutation operates on one or possibly many genes (attributes) of a solution by altering their values. Crossover consists in mating the parental genes pairwise, yielding offspring with new properties. Only the best-fit individuals are likely to survive down the generations. We encoded our timetables as vectors associating 3 genes with each meeting, referring to the period, the classroom and the teacher assigned to the meeting. In a bid to promote a well-spread search of the space of solutions, we allowed our initial generation to be constructed randomly by assigning a random time-slot along with a teacher to each meeting in a "greedy" fashion. We then defined our objective function to be the weighted sum of all the violated constraints, each constraint being associated with a penalty (a weight) in proportion to the importance we ascribe to the constraint. The individuals (timetables) are ranked in the order of descending fitness conditions (as measured by the objective function), the best-fit individual being ranked 0. The probability of replacing an individual $i$ is then defined to be the ratio of $rank_i$ to the sum of all the ranks. In other words, the poorer the fitness condition, the greater the probability of an individual being replaced.

### 2.1 Mutation

The simplest way to define a mutation operator is to select randomly both the gene and the gene value to which mutation is to be applied. For large sized instances, however, this may lead to unacceptable running times because of the poor choice of the genes. In contrast, our mutation operator operates on the most problematic genes. With each meeting m of M, an integer-valued variable is associated representing a violation record. The evaluation process consists in computing the penalty record incurred by the timetable: It adds all the violation records of meetings involved in any constraint of the timetable.

Thus, the violation record of a meeting is highest when the meeting is most problematic. The mutation operator then selects the gene to be muted as a function of the violation record. Likewise, a violation record is computed for each possible allele (gene value). The new value to be assigned to the mutating gene is deduced similarly as the sum of the penalties of all the constraints that will be violated if the change is accepted.

### 2.2 Dedicated mutations

We have designed a mutation operator for each of the three types of constraint. Each mutation operator of a given type maintains a violation record for that type for each meeting in *M*. This proved useful in many respects. First, the fact that only one class of constraint is handled in a mutation results in a significant gain in computing time. Secondly, in keeping separate records on different types of constraints we gain a better insight into which of the types is most violated, and by the same token, a means

is provided by which the sources of the problem can be accessed direct and dealt with. Thirdly, we now have another means (other than penalty value) to distinguish certain types of constraints and single them out as more important than others: it merely suffices to call their corresponding mutation operators more frequently.

### 2.3 Crossover

The role of crossover is to enable the combination of the good properties borne by the so-called parent-individuals. Our crossover operator is based on Abramson's encoding scheme [1][2], where a timetable is represented as a vector of lists, one for each time-slot. Each list brings together the set of meetings assigned to that time-slot. The crossover operator consists then in swapping groups of the two parents.

## 3 Hybridizing genetic algorithm with tabu search

Violation-driven Mutation (VDM) provides an effective means (time-wise) by which to intensify search [6][7]. However, the resulting tendencies to explore vs. exploit the search space often conflict with each other. The reason is that VDM may lead to premature convergence, wherein the search process gets stuck in a local optimum. Moreover, GAs are oblivious to the history of the search process, since only the population of individuals may be regarded as a short-term memory. On the other hand, we feel that Tabu Search (TS), with its numerous built-in strategies and features is better equipped to offset both shortcomings. For one thing, TS distinguishes itself by a greater ability to jump out from local optima thanks to its diversification strategy [6][7]. Furthermore, it makes use of both a short- and a long-term built-in memory to investigate the search space. Tabu Search proceeds by stepwise improvement. Thus, at each step of the algorithm, we move from one solution to another through an operation referred to as a 'move'. Attributes modified during a move become 'tabu' for a certain space of time in terms of the number of iterations. A list called Tabu list contains all tabu-attributes. Several hybridizing schemes are possible [9][12]. For one thing, two mutations on two identical individuals are most likely to perform the same selections of their genes. On the other hand, the mutation of an individual may bring it back to an already known state. Our solution requires that we keep two tabu-lists: a long-term list ("*Long_list*") and a short-term one ("*Short-list*"). "*Long_list*" is dedicated to storing the new values assigned to the genes so as to prevent new mutations from performing the same selection again. "*Short_list*" stores former gene values to prevent future mutations from restoring them.

## 4 Experimental tests

To analyze the performance of our algorithms, which were developed in C, we carried out an array of tests on instances of our own choosing, on benchmarks from literature

[2][11] as well as on a real case instance. We performed the tests using a micro-computer of type Pentium 4 (2.4 GHz, 512 Mo de RAM) operating on Linux 2.0.

The purpose of the first set of tests was to set the parameters of the algorithms. In the absence of relevant theoretical results on the subject, we had to resort to empirical tests to set our parameters. To determine each parameter value, we ran 10 tests per instance. Each such set of tests determined the most appropriate value of a given parameter, all else being unchanged. Crossover and mutation probabilities were set at 0,75 and 0,3 respectively. The number of iterations was set at 20000 and the population size at 300 individuals. As for the sizes of the tabu lists, the sizes of *Long_list* and *Short_list* were 7 and 5 respectively.

### 4.1. Theoretical instances

The tests were performed on 4 types of problems: problem 1 is constituted of 64 subjects with 12 teachers and 16 classrooms. Problem 2 has 100 subjects, 21 teachers and 25 classrooms. Problem 3 is made of 150 subjects, 26 teachers and 31 classrooms. Problem 4 has 200 subjects, 33 teachers and 37 classrooms. For each problem, we allowed the number of periods to vary between 20, 15 and 12 to test the efficiency of the algorithm for increasing period sizes. Run times are given in seconds.

The tests show that the use of a violation-driven mutation speeds up search while compromising the stability of search and the rate of success. The genetic algorithm hybridized with tabu search provides much better results in terms both of the quality of the solution and of the convergence rate, thanks to the tabu component of the algorithm which provides a better spread of the search. This is illustrated in the two charts above featuring a comparison between a hybrid GA and a classic GA; a comparison between a GA that uses plain mutation and a GA that uses a VDM; and a comparison between a GA using a period-based cross-over and one using plain crossover.

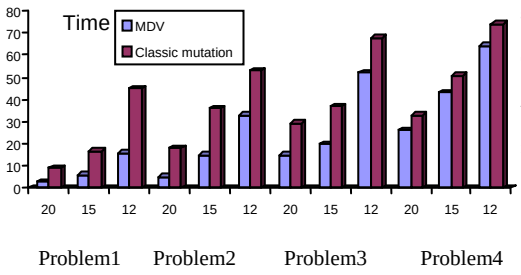


**Fig. 1.** Comparison between a classic GA and a GA with MDV.

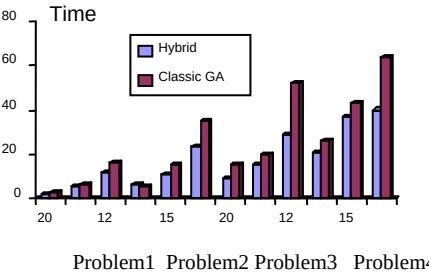


**Fig. 2.** Comparison between a hybrid GA and a classic GA.

### 4.2. Instances from literature

In our set of tests on instances from literature [2][11], we ran our algorithms on the challenging benchmarks Hdtt (*High difficult time-tabling*). These are difficult instances, because all the capacities are stretched thin: every teacher and every classroom must be assigned, and there must be no idle period at the optimum.

Using an IBM SP2, Abramson and Dang [2] have tested these instances with two metaheuristics: simulated annealing and tabu search. We performed 20 runs for each instance. There is a similarity between the results found in literature [2][11] and those provided by our algorithms. For larger instances (Hdtt7 and Hdtt8), more iterations are required in order for the quality of the solutions to be conclusive. A parallel version of the hybrid algorithm would be of much help as it will make it possible to deal with larger populations and more iterations.

### 4.3. Concrete Example

We have applied the hybrid algorithm to the TTP instance of the University of Science and Technology Houari Boumediene USTHB of Algiers, which has 500 groups, 1000 courses, 3000 teachers, about 5000 subjects to be scheduled on six consecutive periods a day for six days. We had two types of rooms to hold lectures and labs: 24 amphitheatres and 181 classrooms. Before the algorithm was applied to the ten faculties of the university, the department of Computer Science had tried it on its own Faculty. The manual designing of timetables in that Faculty used to take a tremendous amount of time (three to four weeks). With our algorithm, that time was reduced to just one hour in average.

## Conclusions

Genetic Algorithms and Tabu Search provide a great flexibility of use when it comes to solving combinatorial problems. We exploited this flexibility to design algorithms capable of generating timetables for any type of academic institution, a problem intractable for approximation. Our algorithms process more than 11 constraints from among the most common ones, with the possible extension to others.

The idea of hybridizing of GA and TS stems from our desire to reap the benefits of both methods: the simplicity of use of GAs on the one hand, and such ability to jump out from local optima through a more diversified and balanced search as provided by TS, on the other hand. This hybridizing is enhanced by a host of ingredients of our own choosing and implementing. One of these is the so-called Violation Driven Mutation, which provides us with an intelligent operator capable of detecting and solving sources of conflicts. The idle time that might be generated in the process is eliminated via tabu-search.

The efficiency of this method makes itself felt on both counts of the convergence rate (running time) and the rate of success, as a result of a good compromise between

expansion and intensification, in terms of exploring new regions of the search space (through tabu-list) versus exploiting the solutions found (through crossover). In view of the variety of settings in which our algorithms fared well in comparison with other algorithms from literature, we conclude that our method together with its ingredients is pertinent to the timetabling problem.

As an extension to this work, we contemplate parallelizing some of the algorithms proposed to enhance the quality of solutions and improve on their running time as well. We also contemplate using a constraint-programming approach that would, in our view, best fit the constrained character of this type of problem. Finally, we consider implementing a metaheuristic cooperation under a Mozart-Oz environment (http://www.mozart-oz.org/) to deal with our problem.

## References

1. Abramson, D.: A parallel genetic algorithm for solving the school time tabling problem. 15 Australian Computer Science Conference, Hobert, Feb. 1992.
2. Abramson, D., Dang, H.:, School Timetables: A Case Study in Simulated Annealing. Applied Simulated Annealing, Lecture Notes in Economics and Mathematics Systems, Springer-Verlag, Ed. V. Vidal , Chapter 5, 103 - 124, 1993.
3. Burke, E.K., Newall, J.P.: Solving examination timetabling problems through adaptation of heuristic orderings. Annals of Operations Research, 129, 107-134, 2004.
4. Burke, E.K., De Werra, D., Kingston, J.: Applications in timetabling. In: Yellen J., Gross J.L. (Eds): Handbook of Graph Theory, Chapman Hall, CRC Press., 445-474, 2003.
5. Geraldo Ribeiro Filho, Luiz Antonio Nogueira Lorena, A Constructive Evolutionary Approach to School Timetabling. Applications of Evolutionary Computing, EvoWorkshops2001: EvoCOP, EvoFlight, EvoIASP, EvoLearn, and EvoSTIM. Proceedings, volume2037, Springer-Verlag, Egbert J. W. Boers and Stefano Cagnoni and Jens Gottlieb and Emma Hart and Pier Luca Lanzi and Gunther Raidl and Robert E. Smith and Harald Tijink (editors), 130-139, 2001.
6. Glover, F., Taillard, E., De Werra, D.: A user's guide to Tabu search. Annals of Operation Research, Volume 41, 3-28, 1993.
7. Glover F., Laguna M., Tabu Search, Kluwer, Boston, MA 1998.
8. Merlot, L.T.G., Boland, N, Hughes, B.D.: A Hybrid Algorithm for the Examination Timetabling Problem. E. Burke and P. De Causmaecker (Eds.): PATAT 2002, LNCS 2740, 207–231, 2003.
9. Kragelund, L.V.: Solving a timetabling problem using hybrid genetic algorithms. Software Practice and Experience, 27(10): 1121-1134, Oct 1997.
10. Petrovic, S., Burke, E.K.: University Timetabling. Ch. 45 in the Handbook of Scheduling: Algorithms, Models, and Performance Analysis (ed. J. Leung), Chapman and Hall, CRC Press, 2004.
11. Reeves, C.R.: Modern heuristic techniques for combinatorial problems. Black Scientific Publication, 1993.
12. Ross, P., Corne, D., Fang, H.: Improving Evolutionary Timetabling with Delta Evaluation and Directed Mutation. In: Schwefel, H.P., Davidor, Y., Manner, R. (Eds.): Parallel Problem Solving from Nature (PPSN) III, LNCS, Vol. 866, 560–565, 1994.