Solving real school timetabling problems with meta-heuristics

F. MELÍCIO[1,3], P. CALDEIRA[2,3], A. ROSA[3]
[1] ISEL, R. Conselheiro Emídio Navarro, 1900 Lisboa, Portugal
[2] EST-IPS, R. Vale de Chaves-Estefanilha, 2810 Setúbal, Portugal
[3] LaSEEB-ISR-IST Av. Rovisco Pais, 1, TN 6.21, 1049-100 Lisboa, Portugal

Abstract: - School timetabling is a classical combinatorial optimization problem, which consists in assigning lessons to time slots, satisfying a set of constraints of various kinds. Due mostly to the constraints this problem falls in the category of NP-Complete problems. In this paper we try to show an implementation of a decision support system that solves real timetabling problems from various schools in Portugal. This implementation is based on the Simulated Annealing meta-heuristic. The constraints we use were obtained after inquiries made to several schools in Portugal. We show the results on three schools from different levels of teaching.

Key-Words: - Timetabling, simulated annealing, meta-heuristics, combinatorial optimization problems, local search methods.

1 Introduction

Educational timetabling problems are known to be difficult real world problems that have been studied in some detail over the years. This problem is NP-Complete mainly due to the associated constraints [7], [12]. In order to implement a system that would fit the majority of the Portuguese education system, we define a set of general constraints. In this paper we are going to explain a decision support system that it is used by more than 100 schools with significant success in their timetabling elaboration.

Meta-heuristics algorithms, like Simulated Annealing, Tabu Search, etc., have been applied with significant success to different combinatorial optimization problems.

In this paper we explain one specific implementation of a Simulated Annealing algorithm adapted to the timetabling problem.

2 Problem Formulation

The timetabling problem [9] consists of assigning a set of lessons to time slots within a time period (typically a week), satisfying a set of constraints of various kinds. It is widely accepted that the timetabling problem can be divided in three main categories [4], [21]:

1. Class/Teacher timetabling. The weekly scheduling of all classes, avoiding teachers meeting two classes in the same time and vice-versa.
2. Course timetabling. The weekly scheduling for all lessons of a set of courses, minimizing the overlaps of lessons of courses having common students.
3. Examination timetabling. The scheduling for the exams of a set of courses, avoiding overlapping exams of courses having common students, and spreading the exams for the students as much as possible.

In this work we are mainly concerned with the classification class/teacher, because in Portugal almost every school even in universities students are firstly joined together in groups with common subjects. Nevertheless, we tried to formulate the timetabling problem in a general way in order to take into account all the requirements of every school in Portugal. Thus, we have the following data sets:

- A set $T = \{t_1, \cdots, t_m\}$ of teachers.
- A set $C = \{c_1, \cdots, c_n\}$ of classes. A class is a group of students that attend the same subjects.
- A set $S = \{s_1, \cdots, s_s\}$ of subjects.
- A set $R = \{r_1, \cdots, r_r\}$ of rooms. Rooms are first grouped in subsets of the same kind, i.e., with the same resources. Each subject has associated at least one type of room.
- A set $H = \{h_1, \ldots, h_p\}$ of time slots. The number of time slots is equal to the number of days, times the number of daily periods. Each period has the same duration and there can be two types of periods. Periods with or without teaching activities.
- A set $A = \{a_1, \ldots, a_l\}$ of lessons. A lesson is the

teaching unit. It is characterized by the following tuple $\langle T^*, C^*, S^* \rangle$. Where T^* is a subset of the teachers set, C^* is a subset of the classes set and S^* is a subset of the subjects set. Each lesson has a duration expressed in time slots.

There are two types of lessons:

1. *Simple lesson*. Where $|C^*| = 1$ and $|S^*| = 1$.
2. *Compound lesson*. Where $|C^*| \geq 1$ and/or $|S^*| \geq 1$.

In general, a *compound lesson* means that we have several classes joined together to attend a certain subject or it means that a class can be subdivided into subgroups to attend special subjects, like laboratories, etc.

It is associated with each subject the kind of room it must have, i.e., the resources that there must exist in the room for a lesson of that subject should happen.

2.1 Constraints

As it was stated in the beginning of this section, a set of constraints must be satisfied in order to have a valid timetable. The number and the kind of constraints vary from school to school, even within the same school system. Nevertheless there are only two categories of constraints:

- **Hard constraints** are constraints that physically cannot be violated. There are also other constraints in spite of not being any physical constraint they fall into this category because of several reasons, for instance, because they are governmental ruled.
- **Soft constraints** are in general preferences and they do not represent a physical conflict.

By hard constraints, we mean the following:

- A teacher cannot teach different lessons at the same time.
- A class cannot have different lessons at the same time.
- Different classes cannot be held in the same room at the same time.
- Class unavailabilities.
- Teacher unavailabilities.
- Etc.

As soft constraints are mainly preferences they vary a lot among schools some examples are:

- Teachers may prefer specific time slots.
- Teachers may prefer specific rooms.
- Certain kind of subjects should not be in contiguous time slots.

For a complete description of the set of constraints we have used, see Table 1.

Constraint	*Description*
C_0	Number of time slots of lessons that aren't yet scheduled
$C_{1,2}$	Number of time slots of overlapped lessons (1-classes; 2-teachers)
$C_{3,4}$	Number of time slots exceeding the maximum allowed per day (3-classes; 4-teachers)
$C_{5,6}$	Number of time slots exceeding the maximum consecutive time slots allowed (5-classes; 6-teachers).
$C_{7,8,9}$	Number of preferable time slots filled (7-classes; 8-teachers; 9-subjects).
$C_{10,11}$	Number of idle time slots (10-classes; 11-teachers)
C_{12}	Number of time slots of lessons without a room assigned.
$C_{13,14,15}$	Number of time slots that are forbidden and are filled with lessons (13-classes; 14-teachers; 15-subjects)
C_{16}	Total number of teaching days for teachers
C_{17}	Number of repetitions of lessons of the same subject in the same class per day
C_{18}	Number of time slots that doesn't satisfy the predefined space between lessons.

Table 1 Constraint set.

We also have introduced the concept of *flexible constraint*. Which means that a user may choose to which category each constraint belongs.

Therefore the main objective of any Decision Support System for this kind of problem should be solving the hard constraints and minimizing the soft constraints. Even if it is impossible to find any feasible solution, it is better to give an approximate solution than none at all.

3 Combinatorial Optimization Problem (COP)

Any timetabling problem belongs to the class of combinatorial optimization problem. In general a combinatorial optimization problem has a discrete finite search space *S*, and a function *f*, that measures the quality of each solution in *S*.

$$f : S \rightarrow \mathbb{R} \tag{1}$$

The problem is to find

$$s^* = \arg\min_{s \in S} f(s) \tag{2}$$

Where s is a vector of *decision variables* and f is the *cost function*. The vector $s*$ is a global optimum. The neighbourhood $N(s)$ of a solution s in S is defined as the set of solutions which can be obtained from s by a *move*. Each solution $s' \in N(s)$ is called a *neighbour* of s.

For each s the set $N(s)$ doesn't need to be listed explicitly, in general it is implicitly defined by referring to a set of possible moves. Moves are usually defined as local modifications of some part of s. The "locality" of moves (under a correspondingly appropriate definition of distance between solutions) is one of the key ingredients of local search. Nevertheless, from the definition above there is no implication that there exist "closeness" in some sense among neighbours, and actually complex neighbourhood definitions can be used as well. This operator can be quite complicated it might even be a meta-heuristic.

3.1 Search Space

When working with discrete domains it is possible to define the search space in terms of the possible values that each variable can have [13]. For this problem we have the set $H = \{h_1, \ldots, h_p\}$ as the set of possible values that each lesson can have.

Definition *Search space: The complete set of solutions that belongs to the search space is defined by* $S = H_1 \times \ldots \times H_l$. *If all* H_i *are equal then* $S = H^l$ *and* $|S| = |H|^l = p^l$.

This value is an extreme case. For instance, there are 10^{10} possible solutions if there are 10 lessons and 10 time slots. Even if we restrict each lesson to a different time slot there will be $10! = 3{,}628{,}800$ possible assignments. As it can easily be verified the search space for this kind of problem is very large. However not all solutions are feasible, i.e., a feasible solution has to have its lessons all scheduled and satisfying a certain number of constraints (hard constraints). A possible search space for this kind of problem could be similar to the one shown in Figure 1.

For certain problems it is very difficult to know if there exists at least one feasible solution before starting any search algorithm. Thus any search algorithm should be able to walk across the search space even inside infeasible regions. One of the most common ways to do that is to penalize constraints that are not satisfied and mixing them together in a cost function.

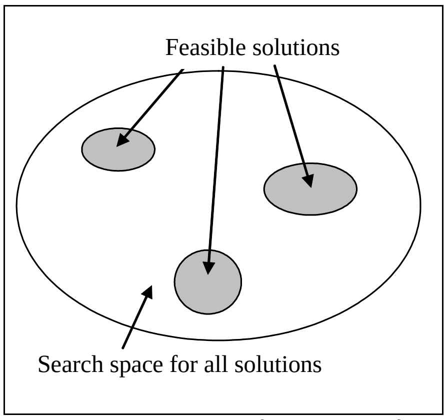
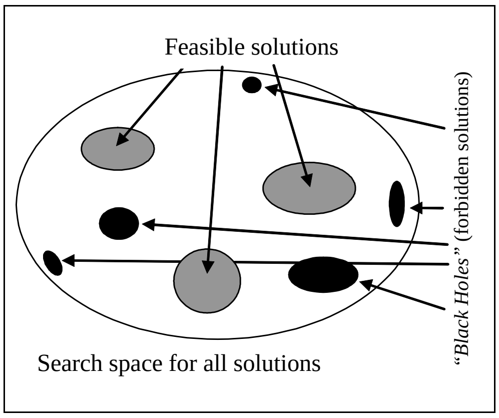


Figure 1 Search space of all solutions for the timetabling problem.

In our problem we did more or less the same thing with the main difference that we didn't relax the hard constraints (user defined). Instead, we will allow partial solutions to belong to the search space. We have a partial solution when there is at least one lesson that is not scheduled. Mathematically this can be represented by augmenting each set H_i with one more time slot, h_0. From a technical point of view, we will assume that the search space (with partial solutions) satisfies the following properties:

1. The empty solution is in the search space $\varnothing \in S$
2. There is a path from any partial solution leading to other partial solution along which the lessons are scheduled one after the other.
3. All complete solutions in the search space satisfy the hard constraints.

In an attempt to limit the search space it is possible to define at the beginning regions of the search space that are forbidden, *black holes*. This can be accomplished by defining a bipartite graph $G = (V_1, V_2, E)$, where every lesson belongs to V_1 and every time slot belongs to the other vertex set V_2 of this bipartite graph. The edge $(i, j) \in E$ means that lesson i can be given in time slot j. This graph only takes into account the static constraints, i.e., class unavailabilities, teacher unavailabilities, subject unavailabilities, etc. It is then possible to define the set H_i for each lesson. The search space thus formed could be like the one shown in Figure 2.


Feasible solutions
"Black Holes" (forbidden solutions)
Search space for all solutions
Figure 2 Search space with forbidden regions.

Nevertheless, the number of possible solutions is still a huge value, so any attempt to check them all would be impossible for most real problems instances.

3.2 Local search methods

Any local search algorithm starts off with an initial solution and then continually tries to find better solutions by searching the neighbourhood of the current solution. A local process can be viewed as a walk in a graph $G=(S,E)$ where the vertex set is the set of solutions S and there is an edge (s,s') in E if and only if s and s' are neighbours $s'\in N(s)$. The efficiency of any local search method depends on the modelling [14]. A fine tuning of parameters will never balance a bad definition of the solution set, of the neighbourhood or the cost function. In general the following rules apply for any local search methods:

1. It should be easy to generate solutions in S.
2. For each solution $s\in S$, there should be a path linking to an optimal solution.
3. The solutions in the neighbourhood of s should be in some sense *close* to s (strongly correlated to s).

It is important to define neighbourhoods in which it is possible to determine the best solution within a reasonable small amount of time.

3.2.1 Double move

In our system we implemented this kind of move, also called *pairwise* interchange. It is identical to some other implementations of a neighbourhood operator [8].

```
1. Select randomly two lessons $i \neq j$
2. Exchange the time slots of each lesson
3. If any of the two lessons isn't scheduled choose
   the "best" time slot for the other
```

Figure 3 Double move adapted to the timetabling problem.

The size of this neighborhood is given by,

$$N(s)=\frac{l(l-1)}{2} \tag{3}$$

Where l is the number of lessons. Usually the number of time slots is much smaller than the number of lessons.

Nevertheless for real problems the size of this neighbourhood is quite large, for instance, if there are 1000 lessons, there will be 499500 neighbours for each solution. However it is known that only a fragment of this number of neighbours can actually improve the quality of a solution. Normally, the exchange of two lessons of two different classes deteriorates the quality of the solution.

So, if we limit the exchanges of lessons belonging to the same class the number of neighbours would be,

$$N_c(s)=\sum_{c=1}^{|C|}\frac{l_c\left(l_c-1\right)}{2} \tag{4}$$

Where l_c is the number of lessons belonging to class *c*. In the above example, if we have 50 classes and each class have 20 lessons, which makes a total of 1000 lessons (same number as above). The size of this reduced neighbourhood would be of 9500 neighbours.

Without considering compound lessons the total number of lessons is given by,

$$l=\sum_{c=1}^{|C|}l_c \tag{5}$$

And if each class has the same number of lessons expression (5) will become $l=|C|\times l_c$. The ratio between these two neighbourhoods will be equal to,

$$\frac{|N(s)|}{|N_c(s)|}=\frac{\left(|C|\cdot l_c-1\right)}{\left(l_c-1\right)}\approx|C|\left(1+\frac{1}{l_c}\right)\approx|C| \tag{6}$$

Where $|N(s)|$ is the neighbourhood size associated with the double move and $|N_c(s)|$ is the neighbourhood size related with the *double move intraclasses*. As it can be seen by equation (6) the ratio between the sizes of these two neighbourhoods is approximately proportional to the number of classes. Hence, for bigger problems better results one would expect to obtain with this reduced neighbourhood. The neighbourhood has a major

However this move has a major drawback. As it is explained in [2], any kind of move should attempt to visit all possible time slots and in this case this would never happen after an initial solution. So in our implementation we made a the following change from the described technique,

```
1   Select randomly two lessons $i \neq j$ from the same class
2   $d_i \leftarrow$ day of lesson $i$
3   $d_j \leftarrow$ day of lesson $j$
4   Unschedule lessons $i$ and $j$
5   Find the best time slot in $d_j$ for lesson $i$
6   Find the best time slot in $d_i$ for lesson $j$
```

Figure 4 Double move with heuristic improvement

4 Cost Function

The cost function plays a key role in any optimization problem. It is through its calculation that one can measure the quality of any solution. Hence its correct definition is essential for the behaviour of any search algorithm. Our cost function is given by the following expression:

$$f(s) = \sum_k w_k C_k \qquad (7)$$

Where $s \in S$ is a solution in the search space (can be a partial solution) and the values C_k represent each of the objectives that we are trying to optimize, weighted by a factor w_k. This weight translates the relative importance of the related constraint. The objective of any search algorithm will be to find an optimum solution s^* that minimize $f(s)$.

In our problem there is one objective that is clearly much more important than the rest which is the scheduling of all lessons. As we have stated before partial solutions make part of the search space. In order to do that and following the same idea expressed in [6] it is defined a constraint C_0 that represents the sum in time slots of all unscheduled lessons. The weight that affects this constraint is the only one that the user can't modify and it is computed as follows:

$$w_0 = \sum_{k=1}^{K} p_k w_k \qquad (8)$$

Where K is the number of constraints defined to the specific problem and p_k is the maximum value that one can violate constraint k if a lesson is scheduled in a given time slot. For the move operator defined in the last section the cost function is computed incrementally, i.e., it is only computed the change in cost between the new solution and the old one.

5 Implementation

The implemented system is based on a modular implementation [19].

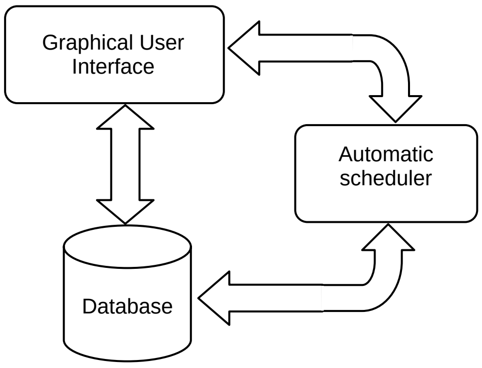


Figure 5 Block diagram of the implemented system.

It was developed in C++ using an object oriented technique. It runs on Microsoft Windows ® and the database is implemented in Microsoft Access®.

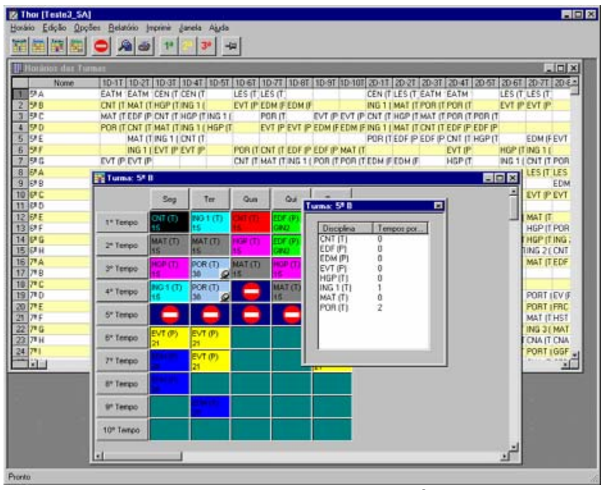


Figure 6 Graphical user interface.

The Automatic scheduler contains two different and complementary algorithms.

1. An iterative algorithm based on Simulated Annealing
2. A heuristic constructive algorithm.

The iterative algorithm is based on the original Simulated Annealing algorithm [18].

Simulated Annealing has been applied with significant success to different combinatorial optimization problems. One of the reasons why SA has been widely used is mostly because it is an algorithm very easy to implement and it doesn't depend on the problem structure or any solution strategy. The basic algorithm is described in Figure 7.

```
Generate initial solution $i$
Compute $cost(i)$
Compute initial temperature $t_0$
$t = t_0$
while stop criteria is not satisfied
{
  while equilibrium is not reached
  {
     Generate new solution $j$
     $\Delta c = cost(i) - cost(j)$
     if $((\Delta c \le 0) \,||\, (\text{random}() < e^{-\Delta c/t}))$
     {
          $i = j$
     }
  }
  Update temperature $t$
}
```

Figure 7 Simulated Annealing Algorithm

The design of a good annealing algorithm is nontrivial [10], it generally comprises four parts:

1. Search space.
2. Neighbourhood set.
3. Cost function.
4. Annealing schedule.

We have explained the first three items in the previous sections we are going now to explain with a little more detail the fourth item.

It is widely accepted that this algorithm is very time consuming, i.e., it needs a great amount of time to get to a near optimum solution.

Therefore our efforts were made in order to speed up the algorithm without loss of quality.

As it can be seen in Figure 7 the algorithm starts with a parameter called temperature with a high value, which is then lowered during the annealing process generally by the following expression [3], [11], [22],

$$t_k = \alpha \cdot t_{k-1} \qquad (9)$$

Where α is designated the cooling factor. Typically, the value of α is chosen in the range 0.90 to 0.99.

In the beginning of the algorithm's execution, as the temperature is high all new possible solutions have a big probability of being accepted ($p > e^{-\Delta c/t}$), and we can travel around the search space without getting much information from it. It seems in this phase much like a random algorithm.

In order to avoid the time spent in this initial stage, we choose to start with a lower temperature but in a good region of the search space.

The problem is then how to choose the temperature value to start with the annealing algorithm.

We implement an adaptive technique [17] which depends on the quality of the starting solution.

$$t_k(i) \approx \frac{\sigma_\infty^2}{E_\infty - f(i) - \gamma_\infty \sigma_\infty} \qquad (10)$$

Where E_∞ and σ_∞ are respectively the expected cost and standard deviation of cost over the entire search space, $f(i)$ the cost of the initial heuristic solution and γ_k represents the number of standard deviation units the expected cost is greater than the minimum cost at temperature t_k,

$$E_k = f(i_{k\min}) + \gamma_k \sigma_k \qquad (11)$$

Assuming a normal distribution of the solutions for almost all temperatures [1], we expect the offset, γ_k, to remain approximately constant over all temperatures except near the temperature corresponding to the cost function optimal value where it converges rapidly towards zero.

Equation (10) tell us that the starting temperature for the annealing phase depends on how good the initial solution is, regarding the cost function.

To get an initial solution we developed a heuristic algorithm based on the following ideas,

1. Sort all lessons unscheduled based on its *urgency*.
2. Schedule each unscheduled lesson in its *best* free time slot.

This notion of urgency is similar to the one used by several authors [15], [24], is given by,

$$u(i) = \frac{1}{|H_i|} \qquad (12)$$

Where $u(i)$ is the urgency coefficient for lesson i and $|H_i|$ is the number of free time slots for lesson i taking into account all the hard constraints (section 4). $0 < u(i) \leq 1$ for all lessons. When $|H_i| = 0$ means that lesson i is impossible to schedule and it is withdrawn from the lessons set.

The results obtained from this hybrid algorithm were compared with other techniques and were very good [20].

6 RESULTS

This system is now used in several schools from the Portuguese education system. We show the results in three typical schools. The data is shown in Table 2.

Data	***ISEL (DEEA) C1***	***EST C2***	***Escola Sec. Fernando Namora C3***
Classes	21	124	54
Teachers	79	208	112
Subjects	250	1345	492
Rooms	21	90	50
#Lessons (#Time slots)	359 (1135)	1908 (3175)	1357 (1660)

Table 2 Three Portuguese schools.

Every test was made with real data, this means that, every constraint and the correspondingly weight was defined by each school.

We used the real timetable for comparison and the results are summarized in Table 3.

	C1	***C2***	***C3***
Real Timetable	479,1	2138,2	481,6
Best result from this algorithm	141,0	1998,1	277,0
Gain in percentage	70 %	7 %	57 %

Table 3 Results from the 3 schools.

The difference in case C2 is due to the fact that the real timetable was made with a previous version of this system while the other two were hand made.

It is also important to mention that the tests were made in a Pentium IV, 1,6MHz and could last from 20 minutes for C1 until 2 hours for C2.

Nevertheless, the gain obtained in using such a system is much greater than just the one expressed in Table 3. Before using this system each school needed several weeks and a group of people to do it. Now, after the introduction of such a system real timetables are made in just a day or two with one or two persons attached to it.

7 Conclusion

As it is well recognized people in general are not interested in solving their optimization problems to optimality or even close to optimality. They are more often interested in *"good enough – soon enough – cheap enough"* solutions to their problems [5].

We also think that good choices of specific parts of each problem are fundamental for the success of any search algorithm. As [23] showed there are no algorithms either deterministic or stochastic behaving the same on the total set of search and optimization problems defined on a finite and discrete domain.

In this paper we illustrate the implementation of customized simulated annealing algorithm for the timetabling problem. It is worthwhile to say that by now more than 100 schools from the Portuguese education system uses this system.

Our main conclusion from this work is that we can solve a very difficult scheduling problem with simulated annealing and be time competitive, but we must be very careful in the way we choose to implement specific parts of the problem.

References:

[1] Aarts, E. H. L., Van Laarhoven, P. J. M., Korst, J. H. M., "Simulated annealing", *Local Search in Combinatorial Optmization*, E. H. L. Aarts and J. K. Lenstra (eds.), John Wiley & Sons, 1997.

[2] Abramson, D. "Constructing school timetables using simulated annealing: sequential and parallel algorithms". *Management Science*, v. 37, pp. 98-113, 1991.

[3] Abramson, D., Dang, H., Krishnamoorthy, M., "An Emprirical Study of Simulated Annealing Cooling Schedules", *Griffith Univ. report*, Nathan, Qld, Aus. 1994.

[4] Bardadym, V.A. "Computer-Aided School and University Timetabling: The New Wave". In Burke, E.K., Ross, P. (eds), *Practice and Theory of Automated Timetabling*, v. 1153, Lecture Notes in Computer Science, pp. 22-45. Springer-Verlag, Berlin, 1996.

[5] Burke, E., Hart, E., Kendall, G., Newall, J., Ross, P., Schulenburg, S., "*Hyper-Heuristics: An Emerging Direction in Modern Search Technology*". In Handbook of Meta-Heuristics, Glover, F., Kochenberger, G., (eds.), pp. 457-474, Kluwer, 2003.

[6] Catoni, O., "Solving Scheduling Problems by Simulated Annealing", *SIAM Journal of Control Optimization*, v. 36, No. 5, pp. 1539-1575, 1998.

[7] Cooper, T.B., Kingston, J.H. "The Complexity of Timetable Construction Problems". In Burke, E.K., Ross, P. (eds), *Practice and Theory of Automated Timetabling*, v. 1153, Lecture Notes in Computer Science, pp. 283-295. Springer-Verlag, Berlin, 1996.

[8] Costa, D. "A tabu search algorithm for computing an operational timetable". *European Journal of Operational Research Society*, v. 76, pp. 98-110, 1994.

[9] de Werra, D. "An introduction to timetabling". *European Journal of Operational Research Society*, v. 19, pp. 151-162, 1985.

[10] Dowsland, K.A., "Off-the-peg or made-to measure? Timetabling and Scheduling with SA and TS". *In Proceedings of the Second International Conference on the Practice and Theory of Automated Timetabling*, pp. 7-26, 1997

[11] Elmohamed, S., Coddington, P., Fox, G. "A Comparison of Annealing Techniques for Academic Course Scheduling". *Proc. 2nd Intl. Conf. On the Pratice and Theory of Automated Timetabling*, pp. 146-166, 1997.

[12] Even, S., Itai, A., Shamir, A. "On the complexity of timetabling and multicommodity flow problems". *SIAM Journal of Computation*, v. 5, pp. 691-703, 1976.

[13] Hemert, J., Back, T., "Measuring the Searched Space to Guide Efficiency: The Principle and Evidence on Constraint Satisfaction", *Proceedings of the Seventh International Conference on PPSN*, Lecture Notes in Computer Science 2439, Springer-Verlag, pp.23-43, 2002.

[14] Hertz, A., Widmer, M., "Guidelines for the use of meta-heuristics in combinatorial optimization", *European Journal of Operational*, vol. 151, pp. 247-252, 2003.

[15] Hilbert H., "High School Timetabling in Germany-Can it be done with MIP?". *In Proceedings of the Second International Conference on the Practice and Theory of Automated Timetabling*, pp. 325-327, 1997.

[16] Huang, M. D., Romeo, F., Sangiovanni-Vincetelli, A., "An Efficient General Cooling Schedule for Simulated-Annealing", *Proc. IEEE-ICCAD*, pp. 381-384, 1986.

[17] Varanelli, J. M., Cohoon, J. P., "A Fast Method for Generalized Starting Temperature Determination in Monotonically Cooling Two-Stage Simulated Annealing Systems", *Report CS-9508, Dep. Computer Science*, University of Virginia, 1995.

[18] Kirkpatrick, S., Gellati, C. D. , Vecchi, M. , "Optimization by Simulated Annealing", *Science*, vol. 220, pp. 671-680, 1983.

[19] Melício, F., "THOR: Uma ferramenta para elaboração de horários duma escola", *Proc. 3º Meeting OE*, pp. 77-82, Porto, Jun 1997.

[20] Melício, F., Caldeira, P., Rosa, A., "Solving Timetabling Problem with Simulated Annealing". Filipe, J. (ed.), Kluwer Academic Press, pp.171-178, 2000.

[21] Schaefer, A. "A survey of automated timetabling". *Artificial Intelligence Review*, v. 13, pp. 87-127, 1999.

[22] Thompson, J., Dowsland, K. A., "General Cooling Schedules for a Simulated Annealing Based Timetabling System", *Proc. 1st Intl. Conf. on the Pratice and Theory of Automated Timetabling*, pp. 345-363, 1995.

[23] Wolpert, D.H., Macready, W.G., "No Free Lunch Theorems for Optimization", *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 67-82, 1997.

[24] Wright, M. "School Timetabling Using Heuristic Search". *Journal of the Operational Research Society*, v. 47, pp. 347-357, 1996.