Two neighbourhood approaches to the timetabling problem

Fernando Melício [1,3], João P. Caldeira [2,3], Agostinho Rosa [3]

fmelicio@deea.isel.pt, caldeira@isr.ist.utl.pt, acrosa@isr.ist.utl.pt

[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: It is well known that any search algorithm needs for certain parts to be problem specific. It is very important the way these parts are implemented. A fine tuning of parameters will never balance a bad definition of the solution set, of the neighbourhood or the cost function. In this paper we try to compare two well known neighbourhood operators applied to the timetabling problem. Tests were made using real data from three Portuguese schools of different size and complexity. We then observed that the application of the correct neighbourhood operator is essential to the success of the search algorithm.

Keywords: timetabling, combinatorial optimization problems, local search, neighbourhood operators.

1 INTRODUCTION

Educational timetabling problems are known to be difficult real world problems that have been studied in some detail over the last few decades or so. This problem is NP-Complete mainly due to the associated constraints [5][10]. We tried to define a general model that would fit the majority of the Portuguese education system. As any local search method needs a neighbourhood operator, we focused our attention in the implementation of the neighbourhood operator.

The neighbourhood operators that we studied can have many names depending mainly on the domain where they are referred. Here we are going to study the *single move* and the *double move*.

Single move means that we exchange the value of just one variable, while the double move signifies that the values of two variables are exchanged. These operators can have other designations, like insertion move or pairwise interchange depending mainly on the domain where they are applied.

We begin by describing our timetabling problem in section 2. In section 3 and 4 we review some important aspects of any combinatorial optimization problem. In section 5 we explain the neighbourhood operators. In section 6 the cost function used is explained and also the way it is computed. Finally in section 7 we see some results that were made with real data of three Portuguese schools of different size and complexity.

2 PROBLEM DESCRIPTION

The timetabling problem [7] consists of assigning a set of lessons to time slots within a time period (typically a week), satisfying a set of constraints of various types.

It is widely accepted that the timetabling problem can be divided in three main categories [2],[19]:

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 all schools even in universities students are grouped in classes with common subjects. Nevertheless, we tried to formulate the timetabling problem in a general way in order to take into account all the particular requirements of every school in Portugal. Thus, we have the following data sets:

- A set $T=\{t_1,\ldots,t_m\}$ of teachers.
- A set $C=\{c_1,\ldots,c_n\}$ of classes. A *class* is a group of students having the same curriculum.
- A set $S=\{s_1,\ldots,s_s\}$ of subjects.
- A set $R=\{r_1,\ldots,r_r\}$ 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 $H = \{h_1, \dots, h_p\}$ of time slots. Time slots are distributed in d week days and h daily periods $|H| = d \times h$. Each period has the same duration. There can be two types of periods. Periods with or without teaching activities.
– A set $A = \{a_1, \dots, a_l\}$ of lessons. A lesson is the teaching unit. It is an instance of a list of teachers, a list of classes, a list of subjects and a list of rooms. Each lesson has a duration expressed in time slots.

There are two types of lessons:

1. *Simple lesson*. A lesson identified only by one class and one subject, it can have more than one teacher.
2. *Compound lesson*. A Lesson specified by a set of classes and/or a set of subjects.

In general, a compound lesson is the way where several classes can group together to attend a certain subject or the way a class can be subdivided into subgroups to attend special subjects, like laboratories, etc.

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

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 type 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. We also have defined the concept of *flexible constraint*, i.e., the user may choose to which category each constraint belongs.

As it can be easily seen, to get a complete solution for a particular timetabling problem with all of the constraints satisfied is very difficult, possibly even impossible to accomplish.

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*.

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

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 metaheuristic.

4 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 [12]. For this problem we have the set $H = \{h_1, \dots, h_p\}$ 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* $S = H_1 \times \dots \times H_l$. *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.

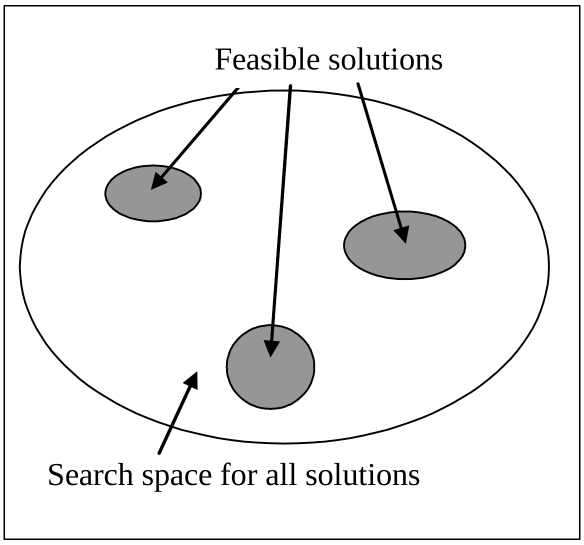

Figure 1 Search space of all solutions for the timetabling problem.

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.

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.

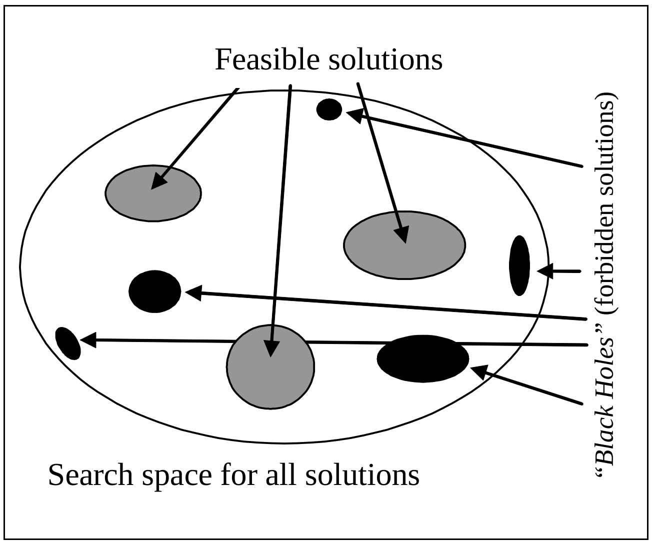
Feasible solutions
"Black Holes" (forbidden solutions)
Search space for all solutions

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.

5 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 [13]. 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.

5.1 Single move

Due to its simplicity this kind of move was considered initially and consists in the algorithm shown in Figure 3.

```
1. Select a lesson i randomly
2. Select randomly a new time slot from the set $H_i$
3. Verify the hard constraints:
   3.1. If it satisfies the hard constraints put the lesson in the new time slot
   3.2. .If it doesn't satisfy the hard constraints leave the lesson outside the timetable (time slot $h_0$)
```

Figure 3 Single move algorithm.

This kind of move is identical to several moves described in literature [9][18][20]. In spite of its simplicity as it is referred in [1] this move allows a well balanced mix of lessons among all time slots. The size of this neighbourhood is,

$$|N(s)| = l \times (p-1) \tag{3}$$

Where l is the number of lessons and p is the number of time slots.

5.2 Single move with heuristic improvement

```
1. Execute a single move with lesson i (Figure 3)
   1.1. If lesson i is scheduled select randomly other lesson j from the same time slot $j \neq i$
   1.2. If lesson i isn't scheduled select randomly other lesson $j \neq i$ from the complete set of lessons.
2. Schedule lesson j in its "best" time slot
Obs: "best" in the sense of the cost function value.
```

Figure 4 Single move with heuristic improvement.

As many authors suggest [14][15][16] it would probably be better to walk through the search space only among local optimum. Nevertheless, since it is applied many times, this heuristic improvement should be easy to compute. With this in mind we define the algorithm based on the single move with a simple heuristic improvement (Figure 4).

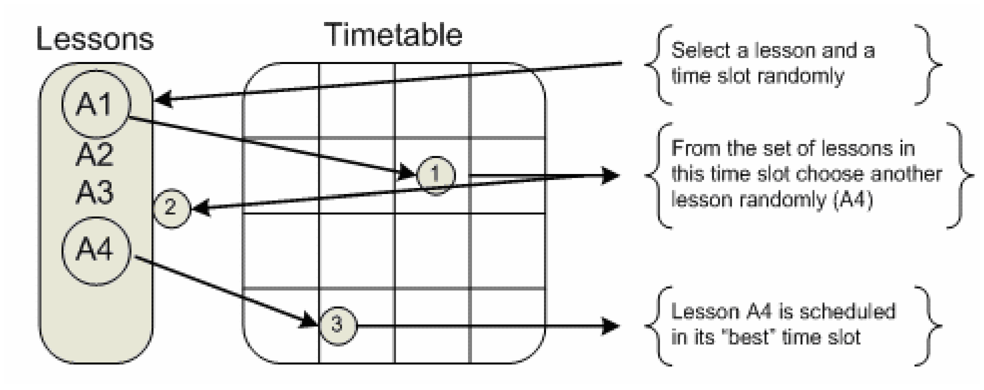
Lessons
Timetable
A1
A2
A3
A4
1
2
3
Select a lesson and a time slot randomly
From the set of lessons in this time slot choose another lesson randomly (A4)
Lesson A4 is scheduled in its "best" time slot

Figure 5 Single move with heuristic improvement schematic diagram.

5.3 Double move

This kind of move, also called *pairwise* interchange, is used in many types of combinatorial problems. It can be considered as an extension of the first one. It is identical to some other implementations of a neighbourhood operator [6].

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 6 Double move adapted to the timetabling problem.

The size of this neighborhood is

$$N(s) = \frac{l(l-1)}{2} \tag{4}$$

Where l is the number of lessons. Usually the number of time slots is much smaller than the number of lessons. The size of this neighbourhood is then greater than the first one and so better results one should expect to get with this kind of move.

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,

$$N_c(s) = \sum_{c=1}^{|C|} \frac{l_c\left(l_c - 1\right)}{2} \tag{5}$$

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,

$$l = \sum_{c=1}^{|C|} l_c \tag{6}$$

And if each class have the same number of lessons expression (6) will become $l = |C| \times l_c$. The ratio between these two neighbourhoods will be equal to,

$$\frac{|N(s)|}{|N_c(s)|} = \frac{\left(|C| \cdot l_c - 1\right)}{\left(l_c - 1\right)} = |C| + \frac{|C| - 1}{l_c - 1} \approx |C|\left(1 + \frac{1}{l_c}\right) \approx |C| \tag{7}$$

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 (7) the ratio between the sizes of these two neighbourhoods is approximately proportional to the number of classes.

6 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:

$$f(s) = \sum_k w_k C_k \tag{8}$$

Where $s \in S$ is a point 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 $f(s)$.

Constraint	*Description*
C_0	Duration of lessons that aren't yet scheduled in time slots.
$C_{1,2}$	Number of overlaps in time slots. (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}	Duration of lessons without a room assigned.
$C_{13,14,15}$	Number of time slots that are forbidden that 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 per day.
C_{18}	Number of time slots that doesn't satisfy the predefined space between lessons.

Table 1 Constraint set.

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 [4] 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:

$$w_0 = \sum_{k=1}^{K} p_k w_k \tag{9}$$

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 all the move operators defined above the cost function is computed incrementally, i.e., it is only computed the change in cost between the new solution and the old one.

7 COMPUTATIONAL RESULTS

The objective of this work it was to compare the behaviour of two well known types of moves applied to the timetabling problem. In order to do this we have applied a simple Hill-Climbing algorithm to each of the neighbourhood operators.

We have tested in three typical Portuguese schools chosen among almost 100 schools of different sizes and systems. 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 (#Duration in 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.

It is hard to tell, just by looking to the values of Table 2 which is the most difficult problem. If we were only concerned with dimension then clearly C1, C3, C2 would be the order of increasing difficulty.

However, the constraints associated with each problem can modify this order. We have defined a ratio identical to the one expressed in [8][9], in order to quantify the difficulty of each problem.

$$\eta = \frac{D_{lessons}}{T_{free}} \tag{10}$$

There is a coefficient per classes, teachers, subjects and rooms and $D_{lessons}$ represents the total of time slots needed for the lessons for each class, teacher, subject and room. T_{free} is the number of free time slots for scheduling the related lessons, i.e., is equal to $\left|\bigcup H_i\right|$ of the related lessons (belonging to a class, teacher, subject, etc.). η is a real value that must be in the interval $[0,1]$. 0 means that there isn't any lesson to schedule or they are all predefined. 1 means that the problem in hand is very *"tight"*. If this value rises above 1 it means that the problem in matter does not have a single feasible solution.

	Classes			***Teachers***			***Subjects***			***Avg.***
	Total	*Avg.*	*Worse*	*Total*	*Avg.*	*Worse*	*Total*	*Avg.*	*Worse*	
C1	0,844	0,859	1,00	0,171	0,177	0,37	0,085	0,089	0,36	0,375
C2	0,740	0,747	0,88	0,185	0,197	0,50	0,076	0,079	0,50	0,341
C3	0,636	0,635	0,81	0,455	0,458	0,94	0,138	0,145	0,57	0,413

Table 3 Density ratio η for classes, teachers and subjects.

In Table 3 is shown the total, average and worse values for classes, teachers and subjects (the room constraints were relaxed) of η. The last column represents the average of these values.

We executed 1000 times Hill-Climbing algorithm for each neighbourhood move. For C1 we run with the steepest descent Hill-Climbing, for the other two we run with random descent because of the neighbourhood's size.

		SMHS	***DMI***
SM	C1	20,9%	56,7%
	C2	4,7%	15,3%
	C3	7,4%	22,5%
SMHS	C1	—	45,6%
	C2	—	8,9%
	C3	—	16,5%

Table 4 Improvement of Single Move Heuristic Search (SMHS) over Single Move and Double Move *Intraclass* over the other two.

The results obtained can be summarized in Table 4, where we see that the double move *intraclass* is clearly the best one. But as important as to see what is the best neighbourhood operator is also to check in what time can we get this result.

Figure 7 shows the best run of each of these operators for C1. We see that every operator takes almost the same time to converge and the double move *intraclass* shows the best value.

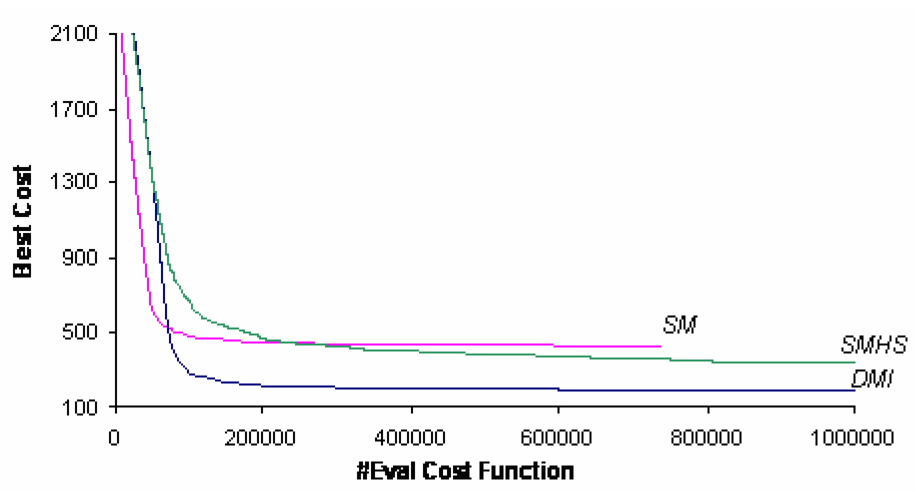

Figure 7 Best cost *vs*. #evaluations of cost function for each operator move for C1 school.

8 CONCLUSIONS

As Burke et. al. [3] refer 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.

We also think that good choices of specific parts of each problem are fundamental for the success of any search algorithm. As [21] 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 work we analyze two well known neighborhood operators adapted to the timetabling problem. We verify for our problem that the double move *intraclass* always showed a better performance than single move even if this is improved with a heuristic method. The tests were made using real data from three different Portuguese schools.

In this work we didn't emphasis on the computation of the cost function: But this is also a key issue for any iterative algorithm, as it is done a lot of times it should be done efficiently. The move operators that we implemented here also permit that the cost function should be computed in an incremental fashion.

It is beyond the scope of this paper but we have also tried these neighborhood operators with significant success with a metaheuristic described in [17].

REFERENCES

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

[2] 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.

[3] 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.

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

[5] 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.

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

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

[8] 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

[9] 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.

[10] 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.

[11] Fonlupt, C., Robilliard, D., Preux, P., Talbi, E.G., “Fitness Landscapes and Performance of Meta-Heuristics”. In Voss, S., Martello, S., Osman, I.H., (eds.), *Meta-heuristics: Advances and Trends in Local Search Paradigms for Optimization*, pp. 257-268, Kluwer, 1999.

[12] 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.

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

[14] Lourenço, H.R., Martin,O., Stützle, T. “A Beginner’s Introduction to Iterated Local Search”. *Proceedings 4th Metaheuristics International Conference*, pp. 1-6, 2001.

[15] Lourenço, H.R., Martin, O., Stützle, T. “Iterated Local Search”. *Handbook on Metaheuristics*. Glover, F., Kochenberger G., (eds.), Kluwer Academic Press, 2003.

[16] Martin, O., Otto, S.W., “Combining simulated annealing with local search heuristics”. *Annals of Operations Research*, v. 63, pp. 57-75, 1996.

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

[18] Reeves, C.R., “Landscapes, operators and heuristics search”. *Annals of Operations Research*, v. 86, pp. 473-490, 1999.

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

[20] Thompson, J., Dowsland, K.A. “Variants of simulated annealing for the examination timetabling problem”. *Annals of Operations Research*, v. 63, pp. 105-128, 1996.

[21] 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.