

# A Tabu Search Strategy to Solve Scheduling Problems with Deadlines and Complex Metric Constraints

Angelo Oddi \* and Amedeo Cesta

IP-CNR

National Research Council of Italy

Viale Marx 15, I-00137 Rome, Italy

{`oddi,amedeo`}@`pscs2.irmkant.rm.cnr.it`

**Abstract.** In this paper a *Relaxable Metric Scheduling Problem (RMSP)* is defined that extends the classical job shop scheduling problem with the use of complex temporal metric constraints and with the possibility of making the distinction between *relaxable* and *not-relaxable* constraints explicit. The paper proposes and experimentally evaluates a *tabu search* procedure to improve a previously created heuristic solution to the *RMSP*. The tabu procedure uses the idea of relaxing some temporal constraints to “navigate” the search space and to find a solution. In particular, it tries to produce either a solution where there are no relaxations or a solution where only the constraints classified as relaxable are violated. Experimental results on scheduling problems of increasing size demonstrate the usefulness of the proposed approach.

## 1 Introduction

Scheduling problems are an important class of constraint satisfaction problems which have many practical applications [14]. A classical formalization of the problem, generated in the manufacturing domain, is the *Job Shop Scheduling Problem (JSSP)* [6], which involves synchronizing the production of  $n$  jobs in a facility with  $m$  resources. The production of a given job requires the execution of a sequence of operations (or activities). Each operation has a specific processing time and its execution requires the exclusive use of a designated resource. Each job has associated a ready time and a deadline, and its production must be accomplished within this interval. It is possible to formulate two versions of this problem with respect to the fact that deadlines and ready times are relaxable or not. The relaxable version of the problem models many practical applications where, rather than finding an exact solution (where all the constraints are satisfied), a solution which represents an agreement between conflicting constraints is searched. There are several motivations to choose this approach to solve a

---

\* Part of this work was developed during the author’s Ph.D. program in Medical Computer Science at the Department of Computer and System Science of the University of Rome “La Sapienza”.

scheduling problem. Firstly, greedy heuristic methods are not infallible, and the possibility of search failure increases when the problem grows in size. Chronological backtracking is one possibility to find a solution in those cases, but its systematicity generally implies high computational cost. Secondly, it may be the case that there is no solution which satisfies all the original constraints, but at the same time some constraints are relaxable and the only possible solution is to find an agreement on the operated violations.

In the scheduling literature, two type of constraints are distinguished: *hard* and *soft* or not-relaxable and relaxable. Examples of relaxable time constraints are the deadlines of set of jobs in a factory domain or staff turns in a medical-resource domain. This type of constraints are generally relaxable because the people or things involved in the scheduling process can accept the violations at an additional cost, that is, the schedule is physically possible even if relaxations are not desirable. Other type of constraints cannot be relaxed. For example, in the HSTS domain (a scheduling problem for the Hubble Space Telescope [9]) the constraints imposed by the visibility of a star, in order to take its picture, cannot be relaxed for the simple motivation that the star could not be visible anymore. Another important issue is that sometimes JSSP is not enough to represent physical constraints in a work environment. For example, let us suppose to work in a medical domain, where jobs represent care protocols and activities are care units. In the hypothesis to consider a simple care protocol which represents two basic actions in order for a patient to take an X-ray: 1) to swallow a preparation to make some organs evident; 2) take an X-ray on the machine. A simple precedence constraint is not enough to represent the real constraints between the two previous actions. In fact, the preparation will be effective after a minimum time and it will become ineffective after a maximum time. A way to enrich the JSSP model is to introduce constraints on the jobs' activities where it is possible to impose that the difference in time between the end-time of an activity and the start-time of the next must be greater than (or equal to) a minimum value and less than (or equal to) a maximum value. Generally this type of constraints represents scheduling constraints which cannot be relaxed as in the example of the medical protocol.

In this paper, we propose and evaluate the use of a *tabu search* strategy for solving job shop scheduling problems with deadlines and complex metric constraints. The problem is also used in [3] where the authors developed highly effective, greedy heuristics for this class of problems based on simple measures of temporal sequencing flexibility, and in [12] where a set of stochastic procedures were proposed which are a randomized counterpart of the deterministic ones tested in [3]. The aim of this paper is to propose a method which is able to manage scheduling problems where it is unlikely to find a solution where all the constraints are satisfied. The tabu search algorithm will be used to "navigate" in the space of the relaxed solutions from an initial one, in order to find a further solution where the total amount of violations operated on the constraints is sensibly reduced and in particular all the violations on the hard constraints are removed.

The paper is organized as follows: Section 2 introduces the scheduling problem and reminds the basic temporal representation for a solution. Section 3 briefly introduces the greedy algorithm that builds an initial solution. Section 4 illustrates the basic idea to explore the space of the relaxed solution and gives the tabu search algorithm used to improve a solution. Section 5 proposes some experiments which show the usefulness of the search strategy introduced. Section 6 draws some conclusions.

## 2 Problem Definition

As we said in the previous Section, the problem solved in this paper was considered in [3, 12] in a version where constraints can not be violated. Here we consider a new definition of the problem called *Relaxable Metric Scheduling Problem* (RMSP) which involves synchronizing the use of a set of resources  $R = \{r_1 \dots r_m\}$  to perform a set of jobs  $J = \{j_1 \dots j_n\}$  over time. The processing of a job  $j_i$  requires the execution of a sequence of  $n_i$  activities  $\{a_{i1} \dots a_{in_i}\}$ , and the execution of each activity  $a_{ij}$  is subject to a set of constraints that can be violated only in the case it is specifically indicated. The type of constraints are the following:

- *resource availability* - each  $a_{ij}$  requires exclusive use of a single resource  $r_{a_{ij}}$  for its entire duration.
- *processing time constraints* - each  $a_{ij}$  has a minimum and maximum processing time,  $proc_{ij}^{min}$  and  $proc_{ij}^{max}$ , such that  $proc_{ij}^{min} \leq e(a_{ij}) - s(a_{ij}) \leq proc_{ij}^{max}$ , where the variables  $s(a_{ij})$  and  $e(a_{ij})$  represent the start and end times respectively of  $a_{ij}$ .
- *separation constraints* - for each pair of successive activities  $a_{ij}$  and  $a_{i(j+1)}$ ,  $j = 1 \dots (n_i - 1)$ , in job  $j_i$ , there is a minimum and maximum separation time,  $sep_{ik}^{min}$  and  $sep_{ik}^{max}$ , such that  $\{sep_{ik}^{min} \leq s(a_{i(k+1)}) - e(a_{ik}) \leq sep_{ik}^{max} : k = 1 \dots (n_i - 1)\}$ .
- *job release and due dates* - Every job  $j_i$  has a release date  $rd_i$ , which specifies the earliest time that any  $a_{ij}$  can be started, and a due date  $dd_i$ , which designates the time by which all  $a_{ij}$  must be completed. We suppose that this type of constraint can be violated in the final solution.

There are different ways to formulate this problem as a *Constraint Satisfaction Problem* (CSP). In [3, 12], the problem is treated as one of establishing precedence constraints between pairs of activities that require the same resource, so as to eliminate all possible conflicts in resource use. In CSP terms, a decision variable  $O_{ijr}$  is defined for each pair of activities  $a_i$  and  $a_j$  requiring resource  $r$ , which can take one of two values:  $a_j\{before\}a_i$  or  $a_i\{before\}a_j$ . As we said above, in order to get a solution we can violate some constraints. With *violation* we indicate a tuple  $(c, \delta_{lb}, \delta_{ub})$  where  $c$  is a constraint of the form  $a \leq tp_j - tp_i \leq b$  and  $\delta_{lb}, \delta_{ub}$  are two values greater or equal to zero, such that,  $0 \leq a - \delta_{lb} \leq b + \delta_{ub}$ , which represent the amount of violation operated on the constraint  $c$ . We define as a *violated solution* a couple  $\langle A, V \rangle$ , where  $A = \{o_{ijr}\}$  is a set of assignments

to the decision variables  $O_{ijr}$  consistent with all the time constraints (below we will give an exact definition of this concept) and  $V = \{(c, \delta_{lb}, \delta_{ub})\}$  is a set of violations operated on the time constraints which make time-feasible the solution obtained. In other words, without the violations it should not be possible to order every set of activities which request the same resource. The problem is to determine a *feasible solution*, that is, a relaxed solution such that it satisfies all constraints which cannot be violated and the total amount of violation (the sum of the values  $\delta_{lb}$  and  $\delta_{ub}$ ) on the other constraints is as low as possible.

## 2.1 Managing Temporal Information

To support the search for a consistent assignment to the set of decision variables  $O_{ijr}$  introduced above, we can define for any *RMSP* a directed graph  $TM(TP, E)$  called *time map* [4], where the set of nodes  $TP$  represents time-points or temporal variables (i.e., the origin point, the horizon point and the start and end time points,  $s(a_i)$  and  $e(a_i)$ , of each activity  $a_i$ ) and the set of edges  $E$  represents temporal distance constraints between couple of time-points (that is, the time constraints listed in the itemization of Section 2). Every temporal constraint has the general form  $a \leq tp_j - tp_i \leq b$  and is represented in the graph  $TM(TP, E)$  as a direct edge  $(tp_i, tp_j)$  with label  $[a, b]$ . Each time point  $tp_i \in TP$  has associated an interval  $[lb_i, ub_i]$  of the possible time instants, or temporal values, where the event associated to the time-points may happen. The time-point  $tp_1$ , the origin point, has associated the constant interval  $[0, 0]$ .

The graph  $TM(TP, E)$  corresponds to a *Simple Temporal Problem* (STP) [5], the computation of the intervals  $[lb_i, ub_i]$  and the check for problem's consistency (a problem is inconsistent when there exists at least an empty interval  $[lb_i, ub_i]$ ) can be polynomially determined via shortest path computations on a directed graph  $G_d(V_d, E_d)$  called *distance graph* [5]. The graph  $G_d$  is obtained from the time map  $TM(TP, E)$  as follows: (a) the set of nodes  $V_d = TP$ ; (b) the set of edges  $E_d$  is built from the set  $E$  considering for each constraint  $a \leq tp_j - tp_i \leq b \in E$  two weighted edges in the set  $E_d$ : the first one directed from  $tp_i$  to  $tp_j$  with weight  $b$ , the second one directed from  $tp_j$  to  $tp_i$  with weight  $-a$ . In  $G_d$  the usual definitions of path and path's length on a weighted graph are assumed: a path is sequence of consecutive edges  $(tp_1, tp_2), (tp_2, tp_3) \dots (tp_{n-1}, tp_n)$ ; the length of a path is the sum of the weights associated to the sequence of edges.  $d(i, 1)$  is the length of the shortest path on  $G_d$  from the time point  $tp_i$  to the origin point  $tp_1$  and  $d(1, i)$  is the length of the shortest path from the origin point  $tp_1$  to the time point  $tp_i$ . A negative cycle is a closed path with negative length. As shown in [5], the interval  $[lb_i, ub_i]$  of time values associated to the generic time variable  $tp_i$  is computed on the graph  $G_d$  as the interval  $[-d(i, 1), d(1, i)]$ . In [5] is also shown that an STP is consistent iff there are no negative cycles in its graph  $G_d$ .

A search for a solution to *RMSP* can proceed by repeatedly adding new precedence constraints into  $TM$  in order to resolve conflicts in the use of resources and efficiently recomputing [1, 2, 11] shortest path lengths to confirm that  $TM$  (and  $G_d$ ) remains consistent.

### 3 Finding an Initial Solution by Relaxing Constraints

A local search method works starting from an initial solution built with another method. The algorithm currently used to create the initial solution is a greedy procedure largely inspired by [3]. The algorithm iteratively selects decision variables  $O_{ijr}$  (which represent conflicts) and assigns them values on the basis of a measure of temporal flexibility leaved in the domain after the value of a variable is assigned. After a decision is taken, the temporal constraints are propagated and a new conflict is considered until all the decision variables have a fixed value. During the search process, it might happen that the domain of a decision variable becomes empty. This means that the insertion of a precedence constraint to resolve the conflict has induced temporal inconsistencies (a set of negative cycles on the graph  $G_d$ ). The original algorithm described in [3] stops, because a *dead-end* is reached. On the contrary, the present greedy algorithm looks for a *violated solution*. It detects negative cycles in the distance graph, chooses one cycle and fixes it by relaxing a constraint on the cycle. When inserting relaxations the heuristic tries to maintain the solution as safe as possible. There are two mechanisms whose aim is to get an acceptable violated solution: the first one tries to minimize the amount of violations of the time constraints; the second one, uses a simple priority heuristic, which, in order to reduce the probability to get a violated solution, selects first relaxable time constraints, and in the case there is no one of the previous type, selects not-relaxable time constraints. For lack of space we are not able to insert a complete description of the algorithm which is included in [11].

### 4 A Tabu Search Approach to Improve a Solution

*Tabu search* [7, 8] is a local search approach recently applied with success to a large set of combinatorial optimization problems. In this Section we introduce a tabu search approach to solve the *RMSP*.

The tabu meta-heuristic is based of the notion of *move*. A move is a function which transforms one solution into another. For any solution  $S$ , a subset of moves applied to  $S$  is computed. The subset of moves produces a subset of solutions called the *neighborhood of  $S$* . A tabu search algorithm starts from an initial solution  $S_0$ , and at each step  $i$  searches the neighborhood of the current solution in order to find a new solution (a neighbor)  $S_i$  that has the best value of a given objective function. At this point the neighbor  $S_i$  becomes the current solution and the search process is iterated to find a new best neighbor  $S_{i+1}$ . In order to prevent cycling, it is not allowed to turn back to solutions visited in the previous *MaxSt* steps. Where *MaxSt* is the max length of the so-called *tabu list*, a list of forbidden moves. Whenever a move from a current solution to its neighbor is made, the inverted move is inserted in the tail of the *tabu list* whose length is kept less or equal to *MaxSt* by removing its head. During the search process, in order to find a near-optimal solution for the problem, at each step a “global” best solution  $S^*$  is updated. A tabu move might happen to be interesting with

respect to the current solution. In order to perform this move, an *aspiration function* is defined which, under certain conditions, accepts a forbidden move. Generally, the definition of aspiration function is such that a forbidden move is accepted when it generates a neighbor which improve the solution  $S^*$ . The search process is performed until at least one of the following conditions becomes true: 1) the objective function of the solution is close to a known lower bound; 2) the algorithm performs *MaxIter* steps without improving  $S^*$ ; 3) a time limit is reached. To apply *tabu search* to a particular problem, the definition of structural elements, such as move, neighborhood, tabu list, aspiration function, etc., is needed. With respect to the classical job shop scheduling problem, the *RMSP* uses a more sophisticated time model (the STP [5]). A tabu approach to *RMSP* needs an extension of the algorithms presented in [10, 13] to deal with temporal information. Our tabu procedure extensively uses algorithms to modify the time map. To perform a move it is needed to remove some temporal constraints and add other ones. This is achieved by using incremental algorithms [2, 11] which work on the STP model and quite efficiently can add and remove temporal constraints in incremental style. At each modification they update only the part of the time map really affected by the change.

#### 4.1 A Running Example

To explain the way our tabu procedure works we introduce an example. Let us consider a simple problem with two resources and two jobs, where each job has two activities. Figure 1 contains a representation of a violated solution for the problem but it is also useful to understand the problem formulation. To maintain the figure simple we have represented the time map instead of the graph  $G_d$ .

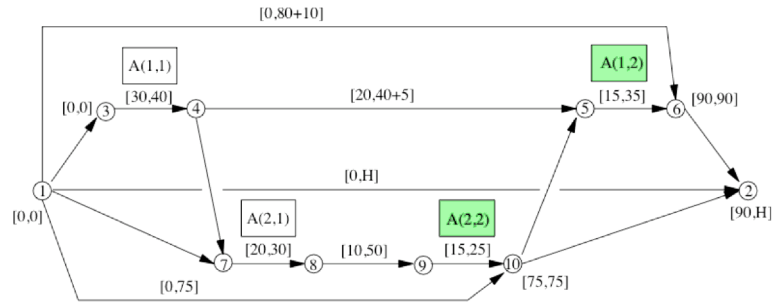


Fig. 1. A initial violated solution for the example

$A(i, j)$  means that activity  $A$  belongs to job  $job_i$  and is executed in position  $j$ . In Figure 1, white activities request the resource  $r_1$ , and grey activities request  $r_2$ . Time-points  $tp_1$  and  $tp_2$  represent the start and end of the time line (the origin point and horizon point). The edge  $(tp_1, tp_2)$  is the *horizon constraint* which

constrains all the activities in a fixed time horizon of amplitude  $H$  (the exact value of  $H$  is not relevant here, we can suppose a value much greater than any deadline in the problem). The four activities  $A(1, 1)$ ,  $A(1, 2)$ ,  $A(2, 1)$  and  $A(2, 2)$  are depicted “anchored” to the *time map*. Each of them has a start and end time (for example,  $tp_3$  and  $tp_4$  are the start-time and end-time of  $A(1, 1)$ ). Edges  $(tp_3, tp_4)$ ,  $(tp_5, tp_6)$ ,  $(tp_7, tp_8)$  and  $(tp_9, tp_{10})$  are the *time processing constraints*. Edges  $(tp_4, tp_5)$  and  $(tp_8, tp_9)$  are the *separation constraints*. Edges  $(tp_1, tp_3)$ ,  $(tp_1, tp_6)$ ,  $(tp_1, tp_7)$  and  $(tp_1, tp_{10})$  are the *job release and due dates constraints*. All edges are labeled with intervals  $[a, b]$ , when these are not explicitly shown the default label  $[0, H]$  should be assumed. Edges  $(tp_4, tp_7)$  and  $(tp_{10}, tp_5)$  included in Figure 1 are not part of the problem formulation but are inserted to resolve the conflicts in the use of resources. We can suppose to build a solution to the problem by inserting first the constraint  $(tp_4, tp_7)$  (without generating time inconsistencies) and second the constraint  $(tp_{10}, tp_5)$ . In this latter case two negative cycles are induced on the graph  $G_d$  derived from the time map. The first one is  $1 \rightarrow 6 \rightarrow 5 \rightarrow 10 \rightarrow 9 \rightarrow 8 \rightarrow 7 \rightarrow 4 \rightarrow 3 \rightarrow 1$  with length  $-10$  can be canceled by changing the label 80 of  $(tp_1, tp_6)$  to the value  $80 + 10$ . The second one  $4 \rightarrow 5 \rightarrow 10 \rightarrow 9 \rightarrow 8 \rightarrow 7 \rightarrow 4$  with length  $-5$  can be removed by modifying the label 40 of the edge  $(tp_4, tp_5)$  to  $40 + 5$ . In this way we obtain the particular conflict-free *violated solution* in the figure.

#### 4.2 Defining a Move in the *RMSP*

Similarly to [10, 13], a move in the *RMSP* is a transformation of a solution  $S$  by swapping a couple of activities  $(a_i, a_j)$  which require the same resource.

It is worth observing that the swap of two activities can generally create a state of temporal inconsistency in the time map. There are at least two ways to resolve this problem: either to only consider the subset of moves which do not produce time inconsistency, or to introduce some violations between the time constraints and try to repair such violations during the tabu search process to arrive again to a feasible solution. This second choice has the advantage of exploring more deeply the search space and will be used in the following, but at the same time, it is computationally more expensive, as can be seen observing the tabu algorithm below. The neighborhood of a solution  $S$  is a set of new solutions which may involve violations of time constraints. In the following we consider moves that swap activities  $(a_i, a_j)$  when the following conditions hold:

1.  $(a_i, a_j)$  are consecutive on the resource  $r$ ;
2.  $(a_i, a_j)$  are on a shortest path in the graph  $G_d$ .

It is worth reminding that two activities are on a shortest path if considered the time points  $e(a_i)$  and  $s(a_j)$  or  $e(a_j)$  and  $s(a_i)$  on the graph  $G_d$  an edge exists which connects  $e(a_i)$  and  $s(a_j)$  or  $e(a_j)$  and  $s(a_i)$  and belongs to the shortest path either from or to the time origin  $tp_1$ .

A violation (Section 2) is a tuple  $(c, \delta_{lb}, \delta_{ub})$ , where  $c$  is a temporal constraint in the time map,  $\delta_{lb}$  and  $\delta_{ub}$  are the violations operated respectively on the lower

and upper bound of the constraint's label  $[a, b]$ . Under the restrictions stated by Conditions 1 and 2 it is possible to prove [11] that the time inconsistencies can be removed by violating only the upper bounds of either a separation or a deadline constraints. In other words, we can consider only violations of the form  $(c, 0, \delta_{ub})$  that can be written as  $(c, \delta)$ . In a similar way, a *restoration* is a couple  $(c, \delta)$ , where  $c$  is a temporal constraints and  $\delta$  is the reduction value for the upper bound constraint to restore the original value.

With the previous assumptions the definition of *move* is the following:

**Definition 1.** Given a violated solution, a move  $mv$  on a resource  $r$  is defined as the tuple  $(r, acts, rlx, res)$ , where  $acts$  is a couple of activities  $(a_i, a_j)$  which satisfies Condition 1 and 2,  $rlx = \{(c_i, \delta_i)\}$  is a set of violations on the time constraints and  $res = \{(c_j, \delta_j)\}$  is a set of possible restorations operated after the swap of the couple of activities.

The definition does not specify a method to violate and restore constraints, this is a matter of the specific heuristics adopted. The previous definition is general enough to be applied with the STP temporal model.

### 4.3 The Tabu Search Algorithm

In Figure 2 the tabu search algorithm for the *RMSP* is shown that manages an initial solution  $S_0$ . As input the algorithm is given the initial solution  $S_0$ , the parameters *MaxIter* and *MaxSt*, and the objective functions  $f_{obj}$ . The algorithm uses two variables  $S^*$  and  $S_{min}$ , which represent respectively the current optimal solution and the best solution found in the exploration of a neighborhood. In the case the algorithm explores a sequence of *MaxIter* neighborhood without an improvement of the solution  $S^*$ , then the algorithm stops (Step 2). A neighborhood is searched between Steps 4 and 10, where is found the solution  $S_{min}$ . The idea of *aspiration function* is implemented at Step 8 where a tabu move is accepted in the case it improves  $S^*$  and the current  $S_{min}$ . Finally, when a solution  $S_{min}$  is found, the move  $move_{min}$  is added to the *tabu-list* (Step 13) and the variable  $S^*$  is updated.

The core of the algorithm consists of the two sub-primitives *Get-Neigh-Moves* and *Make-Move*.

The *Get-Neigh-Moves* implements the definition of neighborhood. By using Definition 1 two different neighborhoods are introduced: (a) the set of moves is computed for each resource. This choice analyzes deeply the search space, but at the same time is quite expensive; (b) the set of moves is computed only for a randomly selected resource .

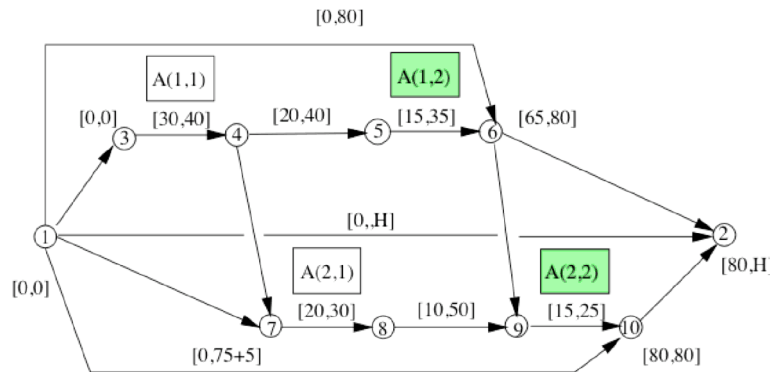
The procedure *Make-Move* makes the swap of the couple of activities  $(a_i, a_j)$  specified by the move  $mv$ . It is essentially composed by a sequence of application of three primitives that work on the time-map: *Remove*, *Insert-with-Relax* and *Restore-Heuristic*. To explain how the procedure works we use Figure 3 in which a swap is performed between activities  $A(1, 2)$  and  $A(2, 2)$  starting from the solution of Figure 1.

The *Make-Move(mv)* performs the following steps:

**TABU-STP**( $S_0, MaxIter, MaxSt, f_{obj}$ )

1.  $S^* ::= S_0; S_{min} ::= S_0;$
2. **while not**(Last-Iteration( $MaxIter$ )) **do begin**
3.   Get-Neigh-Moves( $S_{min}$ );
4.   **while not**(Neighborhood-Empty) **do begin**
5.      $move ::=$  Select-Move( $S_{min}$ );
6.      $S_{curr} ::=$  Make-Move( $move$ );
7.     **if** Tabu-Move( $move$ )
8.       **then if** Aspiration-Func( $S_{curr}, S^*, S_{min}, f_{obj}$ ) **then**  $S_{min} ::= S_{curr}$
9.       **else if** Improves( $S_{curr}, S_{min}, f_{obj}$ ) **then**  $S_{min} ::= S_{curr}$
10.   **end**
11. **if** Improves( $S_{min}, S^*, f_{obj}$ )
12.   **then begin**
13.     Update-Tabu-List( $move_{min}, MaxSt$ )
14.      $S^* ::= S_{min}$
15.   **end**
16. **end**

**Fig. 2.** TABU-STP algorithm



**Fig. 3.** Feasible solution obtained after the swap of the activities  $A(1,2)$  and  $A(2,2)$

1. *Remove*: it incrementally deletes time constraints from the time map. In the example the constraint  $(tp_{10}, tp_5)$  is removed.
2. *Insert-with-Relax*: it tries to insert a precedence constraint between a couple of time-points. If the insertion gives a consistent situation it stops, otherwise (when a time inconsistency is detected) the procedure iteratively: (1) gets one of the negative cycle created on the graph  $G_d$ ; (2) selects a constraint on the negative cycle and relaxes it; (3) propagates the effect of the relaxation in the graph  $G_d$ . The previous sequence of steps continues until all the negative cycles induced by the insertion on the precedence constraints are deleted

and a time-feasible solution is reached. In the example, the new constraint  $(tp_6, tp_9)$  is inserted and the effects are propagated in the time map. The new insertion generates a time inconsistency. In fact, the negative cycle  $1 \rightarrow 10 \rightarrow 9 \rightarrow 6 \rightarrow 5 \rightarrow 4 \rightarrow 3 \rightarrow 1$  is induced whose length is  $-5$ . This inconsistency can be resolved by relaxing one or more constraints along the cycle. The same criterion explained in Section 3 is used, the algorithm selects the label 75 on the constraint  $(tp_1, tp_{10})$  and relaxes it to the value  $75 + 5$ .

3. *Restore-Heuristic*: it tries to restore the original constraint by scanning the set of violations  $V$  operated on the solution. For each element  $(c, \delta)$ , where  $c$  has the form  $a \leq tp_j - tp_i \leq b$ , it tries to insert the new constraint  $a \leq tp_j - tp_i \leq b - \delta$ . If the insertion is consistent the element  $(c, \delta)$  is deleted from the set  $V$ , otherwise the method tries to repair the next violation. In the example, the labels on constraints  $(tp_1, tp_6)$  and  $(tp_4, tp_5)$  are restored to the original values, in fact the modification of the network removes both the negative cycles mentioned in Section 4.1.

The solution obtained in the example is now feasible, because as a consequence of the move also the separation constraint  $(tp_4, tp_5)$  is restored to the original value. The only relaxed constraint in the new solution is a deadline constraint that is feasible according to the definition of *RMSP*.

## 5 Experimental Evaluation

In this Section, we evaluate the tabu search procedure on a set of randomly generated scheduling problems. The main goal of the experiments is to demonstrate the usefulness of the tabu search procedure in the restoration of temporal constraints. We try to resolve scheduling problems which have a low probability to have a solution without violations of temporal constraints.

We consider scheduling problems of size  $n \times m$ , whose structure is sketched in the following. There are  $n$  jobs to be scheduled. Each job requires operations to be performed on each of the  $m$  different resources, and the order in which each job must visit each resource is randomly decided. Experiments concern problem sets of 50 instances at each of three different sizes:  $8 \times 3$ ,  $12 \times 3$  and  $12 \times 4$ . The problem sets are generated following the generation scheme used in [3] adapted to generate particularly hard instances. We indicate with  $U[x, y]$  a random number in the interval  $[x, y]$  generated with a uniform distribution. The minimum processing time of activities is chosen as  $U[10, 50]$ , and the maximum processing time is generated by multiplying the minimum processing time by the value  $(1+p)$ , where  $p = U[0, 0.4]$ . Separation constraints  $[a, b]$  between every two consecutive operations in a job are generated with  $a = U[0, 10]$  and  $b = U[40, 50]$ . All the ready times  $rd_i$  of the jobs are fixed to 0 and the deadlines  $dd_i$  are fixed to a common value  $M$ . The value of  $M$  is calculated by the follow formula  $M = (n-1)p_{bk} + \sum_{i=1}^m p_i$ , where  $p_{bk}$  is the average minimum processing time of operation on the bottleneck resource, and  $p_i$  is the average minimum processing time of operation on resource  $r_i$ . The bottleneck resource is the resource with

maximum value of the sum of the minimum processing time of the activities which requests the resource. In order to generate hard scheduling problems with a low probability to have an “exact solution”, we have introduced a parameter called *Slack* which controls the amplitude of the allocation windows of the jobs (see Section 2) [ $rd_i, dd_i$ ]. In practice, the real windows for a job  $j_i$  is [ $Slack * rd_i, Slack * dd_i$ ]. For *Slack* we have used the value 0.8. This is used to shrink the original time window to make the satisfaction of the constraints harder.

In the experiments we use the following objective function:

$$f_{obj}(S) = \alpha_{sep} v_{sep}(S) + \alpha_{dl} v_{dl}(S)$$

where  $v_{sep}(S)$  represents the sum of the violation values on the separation constraints and  $v_{dl}(S)$  represents the sum of the violation values on the deadline constraints.  $\alpha_{sep}$  and  $\alpha_{dl}$  are numeric coefficients whose aim is to focus the tabu search on a specific type of violation. In these experiments we fixed  $\alpha_{sep} = 8$  e  $\alpha_{dl} = 1$ , in order to focus the procedure on the restoration of separation constraints. Finally, we complete the setting of tabu parameters as follows. For each class of problem *MaxSt* is 9. For problems  $8 \times 3$  *MaxIter* is 9; for problems  $12 \times 3$  *MaxIter* is 36 and for problems  $12 \times 4$ : *MaxIter* is 72.

As said in Section 4.3 two different definitions of neighborhood are used. For problems  $8 \times 3$ , the neighborhood of a solution is the set of moves that satisfy Definition 1. As we have said, this choice analyzes deeply the search space, but at the same time it is time expensive and we can use it only on small problems. For problems  $12 \times 3$  and  $12 \times 4$ , the neighborhood is built by considering only the moves on a randomly chosen resource.

The procedure and the experimental setting were implemented in *Allegro Common Lisp* on a SUN Sparc 10 workstation. Table 1 shows the results obtained for the three different classes of problem. For each class results are shown relative to: (a) the application of the heuristic method described in Section 3 (lines labeled HEU); (b) the application of the tabu procedure to the previous solution (lines labeled TABU). Lines labeled with  $\Delta\%$  simply show the percent variations of the results obtained before and after the application of the tabu procedure. Each line labeled with HEU or TABU reports the following data:  $N_r$  denotes the number of instances which have violations on the time constraints;  $N_{sep}$  denotes the number of instances which have violations on the separation constraints;  $N_{dl}$  denotes the number of instances which have violations on the deadline constraints;  $R_{sep}$  denotes the average amount of violations (in time units) operated on the separation constraints;  $R_{dl}$  denotes the average amount of violations (in time units) operated on the deadline constraints; *CPU-time* denotes, in seconds, the average cpu-time needed to produce the solution.

The lines labeled with  $\Delta\%$  show the usefulness of the tabu procedure on highly constrained scheduling problems. In particular, column  $N_{sep}$  shows that the tabu procedure makes respectively feasible 90%, 82% and 64% of the unacceptable initial solutions. Moreover, even if all the problems presents violations on the deadlines constraints, in the case of problems  $8 \times 3$  and  $12 \times 3$ , the value  $R_{dl}$  is reduced.

**Table 1.** *TABU-STP* 's performance on  $8 \times 3$ ,  $12 \times 3$  and  $12 \times 4$  problem sets

Problem	Strategy	$N_r$	$N_{sep}$	$N_{dl}$	$R_{sep}$	$R_{dl}$	CPU-time(sec)
$8 \times 3$	HEU	50	39	50	54.3	77.9	5.8
	TABU	50	4	50	21.7	55.8	44.4
	$\Delta\%$	0	90.0	0	60.0	28.0	-
$12 \times 3$	HEU	50	40	50	58.5	162.9	27.5
	TABU	50	7	50	5.0	152.3	142.9
	$\Delta\%$	0	82.0	0	91.0	6.0	-
$12 \times 4$	HEU	50	44	50	101.5	210.1	62.9
	TABU	50	16	50	16.6	221.4	498.6
	$\Delta\%$	0	64	0	84.0	-5.0	-

Observing the cpu-times shown in Table 1, it is quite evident that the algorithms are computationally expensive. This is mainly due to the explicit consideration of metric constraints. In the heuristic method HEU, the main source of time complexity is the update of the array of distances  $d(i, j)$ . This array is extensively used to build an effective method [3, 11] to solve scheduling problems with metric constraints. Every time a new conflict is resolved,  $d(i, j)$  is updated, and in the case some constraints are violated, this is done by a sequence of operations of constraints detection, relaxation and updating of the array  $d(i, j)$ . The last sequence of operations is quite expensive with respect to the situation where no violation is allowed. For example, in the case of instances of problems  $12 \times 4$  which does not need violations the average solution time should be about 10-15 seconds instead of 62 seconds.

As far as the tabu search algorithm is concerned we observe that these results are not definitive, they represent an intermediate step towards a more elaborate tabu search strategy to effectively scale on larger problems. Here we have introduced a methodology to cope with the problem and demonstrated the usefulness of the definition of *move* given in an STP framework.

## 6 Conclusions

The aim of this paper is to present the application of a tabu strategy to solve scheduling problems which need the STP temporal model. The defined tabu search procedure uses a definition of *move* based on the violation and restoration of temporal constraints. It can be implemented by using some efficient algorithms, described in [2, 11], that dynamically manage temporal constraints. The idea of relaxing and restoring temporal constraints allows to work with temporarily violated solutions and can be used "to navigate" the search space in order to find a feasible solution from a violated one.

In an experimental study on randomly generated scheduling problems of increasing scale, the local search procedure was found to significantly improve

a start solution obtained with a heuristic strategy. The particular use of tabu search is an interesting step in solving scheduling problems with relaxable deadlines and complex metric constraints which have a low probability to have an “exact solution”.

## Acknowledgments

We would like to thank the ECP anonymous reviewers for their useful comments. Authors’ work was supported by Italian Space Agency, CNR Committee 12 on Information Technology (Project SCI\*SIA), CNR Committee 04 on Biology and Medicine, and Italian Ministry of Scientific Research. Angelo Oddi is currently supported by a scholarship from CNR Committee 12 on Information Technology.

## References

1. G. Ausiello, G.F. Italiano, A. Marchetti Spaccamela, and U. Nanni. Incremental Algorithms for Minimal Length Paths. *Journal of Algorithms*, 12:615–638, 1991.
2. A. Cesta and A. Oddi. Gaining Efficiency and Flexibility in the Simple Temporal Problem. In *Proceedings of the Third International Conference on Temporal Representation and Reasoning (TIME-96)*. IEEE Computer Society Press, 1996.
3. C. Cheng and S.F. Smith. Generating Feasible Schedules under Complex Metric Constraints. In *Proceedings 12th National Conference on AI (AAAI-94)*, 1994.
4. T.L. Dean and D.V. McDermott. Temporal Data Base Management. *Artificial Intelligence*, 32:1–55, 1987.
5. R. Dechter, I. Meiri, and J. Pearl. Temporal Constraint Networks. *Artificial Intelligence*, 49:61–95, 1991.
6. S. French. *Sequencing and Scheduling: an Introduction to the Mathematics of the Job-Shop*. Hellis Horwood Lim., 1982.
7. F. Glover. Tabu Search – Part I. *ORSA Journal of Computing*, 1:190–206, 1989.
8. F. Glover. Tabu Search – Part II. *ORSA Journal of Computing*, 2:4–32, 1990.
9. N. Muscettola. HSTS: Integrating Planning and Scheduling. In M. Zweben and M.S. Fox, editors, *Intelligent Scheduling*. Morgan Kaufmann, 1994.
10. E. Nowicki and C. Smutnicki. A Fast Taboo Search Algorithm for the Job Shop Problem. *Management Science*, 42:797–813, 1996.
11. A. Oddi. *Sequencing Methods and Temporal Algorithms with Application in the Management of Medical Resources*. PhD thesis, Department of Computer and System Science, University of Rome “La Sapienza”, 1997.
12. A. Oddi and S.F. Smith. Stochastic Procedures for Generating Feasible Schedules. In *Proceedings 14th National Conference on AI (AAAI-97)*, 1997.
13. P.J.M Van Laarhoven, E.H.L. Aarts, and J.K. Lenstra. Job Shop Scheduling by Simulated Annealing. *Operations Research*, 40:113–125, 1992.
14. M. Zweben and M.S. Fox, editors. *Intelligent Scheduling*. Morgan Kaufmann, 1994.