A Tabu Search Approach for Scheduling Problem Under Labour Constraints

Cristina Célia Barros Cavalcante
Cid Carvalho de Souza

Relatório Técnico IC–97-13

Outubro de 1997

A Tabu Search Approach for Scheduling Problem Under Labour Constraints

Cristina Célia Barros Cavalcante* Cid Carvalho de Souza†

Instituto de Computação
Universidade Estadual de Campinas — UNICAMP
Caixa Postal 6176 - CEP: 13081-970 - Campinas, SP - Brasil
e-mail: {cris,cid}@dcc.unicamp.br

September 1997

Abstract

The Resource Constrained Project Scheduling Problem (RCPS) is a well known difficult combinatorial optimization problem. Many exact and heuristic approaches for this problem have been reported in the literature. Tabu search is a meta-heuristic designed to guide local search methods to escape the trap of local optimality. It has been lagerly used for solving combinatorial problems. In this report we propose a tabu search approach for Scheduling Problem under Labour Constraints (SPLC). Different neighbourhood strategies are discussed and then implemented. Computational experiments on a 25-instance SPLC data set are reported. The results obtained for benchmark instances are compared with those produced by a Constraint Logic Programming algorithm and show that our approach is at least as good as the best heuristics reported in the literature.

Keywords: Scheduling Problem under Labour Constraints, Tabu Search, Combinatorial Optimization

1 Introduction

Combinatorial optimization can be defined [NW88] as the area that deals with problems of maximizing or minimizing a function of many variables subject to (i) inequality and equality constraints and (ii) integrality restrictions on some or all of the variables.

There are a lot of applications of combinatorial optimization, and one of the most common in industrial environment is related to the management and efficient use of scarce resources to increase productivity [BESW93]. The production scheduling problems arise in this context. The Scheduling Problem under Labour Constraints (SPLC) is concerned with

*Research partially suported by CNPq Grant 139077/96-0 and FAPESP Grant 96/10270-8.
†Research partially suported by CNPq Grant 300883/94-3.

the scheduling of a collection of precedence related activities subject to constraints on the available number of workers necessary to execute each of the activities.

SPLC is a typical example of the Resource Constrained Project Scheduling Problem (RCPSP). RCPSP is known to be intractable [BLR83]. Exact and heuristic procedures have been reported in the literature for many different variants of RCPSP [DH92, NWS95, SW93, ZP96]. A constraint programming approach for the special case of SPLC, which is of interest in our work, is presented in [Hei95] and [HC97].

SPLC, as posed here, models a practical problem and, to our knowledge, it was first studied in the PAMIPS Project [PAM]. In this project, the problem was originally formulated as a mixed-integer programming problem.

In this technical report we address the design, implementation and testing of a tabu search procedure for the SPLC. Tabu search is a meta-heuristic for solving optimization problems, designed to guide local search methods to escape the trap of local optimality [Glo90]. The method makes use of a flexible adaptive memory structure and of tabu restrictions in order to drive and constrain the solution search process. Tabu search has been largely used for solving difficult combinatorial problems [Aie96, PR95]. Application of this method for solving RCPSP can be found in [ZP96].

This text is organized in the following way: the next section formalizes the SPLC definition and presents an example that will be used to illustrate our tabu search approach. Section 3 describes briefly the main general aspects of tabu search. Section 4 details the elements of the tabu search for SPLC. Section 5 presents the experimental data and tests, followed by a discussion of the computational results. Finally, Section 6 summarizes the conclusions and discuss some directions for future works.

2 Scheduling Problem under Labour Constraints – SPLC

In this section we formalize the definition of the scheduling problem we want to solve.

The Sheduling Problem under Labour Constraints can be defined as follows:

Let I be a set of orders ($|I| = m$), where each order is preassigned to a machine and consists of n_i identical jobs. Each job is composed by a set of p_j tasks of duration 1. All the tasks of a specific job must be executed one immediately after the other. Each job has a demand for labour specified by the labour profile array $L_j =< \ell_{j1}, \ell_{j2}, ..., \ell_{jp_j} >$, where ℓ_{js} is the labour requirement of the s^{th} task of job j. Workers are necessary to keep the jobs running and there is a limit R_t on the labour capacity available at each period t of the planning horizon. All the jobs of one order must be executed in the same machine (the one to which the order was assigned), and orders cannot be preempted (one order cannot have interrupted its execution to start the execution of another order in the same machine). The precedence constraints between jobs are represented on an acyclic digraph $G = (V, E)$ in which the nodes are jobs and the arcs represent precedence relationships among jobs. Note that, by definition, the subgraph induced in G by the jobs of one specific order is an oriented simple path.

A schedule S for one instance of the SPLC is described by the *starting times* of all its jobs. S is said to be *feasible* if it satisfies the following restrictions:

i	j_F	j_L	p_j	n_i
1	1	3	3	3
2	4	6	4	3
3	7	8	4	2
4	9	12	2	4

Table 1: Job-order relation in a (4 order,12 jobs) instance. j_F and j_L are, respectively, the labels of the first and the last job in order i.

i/t	1	2	3	4
1	6	3	12	
2	18	6	12	3
3	2	2	2	6
4	2	2		

Table 2: Labour requirement for each processing time of a job of an order i.

(i) $\forall\ (i,j) \in E$, job j cannot start before job i has finished;

(ii) the total number of workers required in t cannot exceed the maximal available R_t, for all period t in the planning horizon.

The goal is to find a feasible schedule of minimum makespan, i.e., a feasible schedule that minimizes the completion times of all jobs.

An example of a SPLC instance is depicted in Tables 1,2 and in Figure 2.

SPLC is NP-hard [Hei95]. However, as reported in [LW], there are two cases in which it may be relatively tractable based on the observation that the length of the critical path in G is a trivial lower bound on the minimum makespan. When G is dense, enumerative type procedures become effective because of the relatively small search space and possible effectiveness of the lower bound. Another case in which this lower bound may be tight is when the resource requirement is not too restrictive. Obviously, the problem reduces to a critical path problem if no resource restriction is imposed.

In our study, we focus on instances of SPLC that are not in any of the above cases, i.e., the total labour available in each period is very scarce and the precedence relation digraph is sparse.

3 Tabu Search

This section summarizes the main ingredients of TS which will be adapted later to the SPLC problem.

Tabu search is a meta-heuristic used for guiding local search methods and prevent them from becoming trapped at locally optimal solutions [Glo90]. In this way, TS makes use of mechanisms that continuously allow the exploration of the solution search space, even when

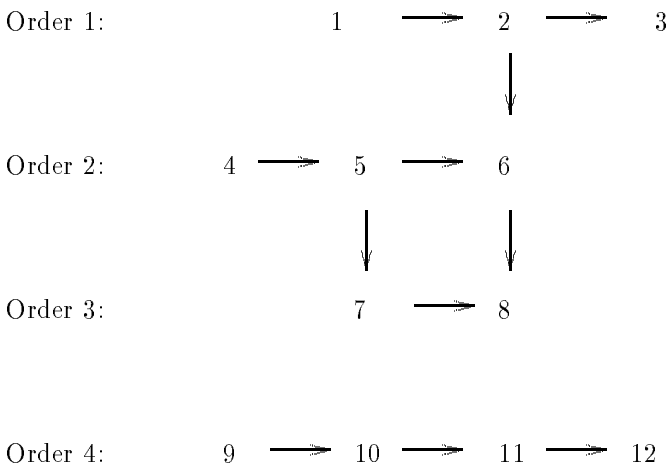

Figure 1: Job precedence relations.

no move is available to improve the current solution. By escaping from local optima, TS eventualy reaches a global optimum solution.

TS can be described as follows: it starts with an initial solution and then at each iteration one move is applied to the current solution, leading to one (desirably the best) of its neighbour solutions. By contrast with the tradicional hill-descending methods, which only accept moves that result in solutions of better (smaller) cost, TS also allows moves to solutions of worse (higher) cost. In this case, tabu restrictions are imposed to the moves that resulted in a deteriorated solution and it is possible to distinguish between admissible and tabu moves. The most common tabu moves are those that attempt to reverse or repeat previous moves. By restricting these moves for a certain number of iterations, called *tabu tenure*, the search process can be driven out from local optimal vales. A tabu restriction can be overrided whenever the corresponding tabu move leads to a better solution. This is called *aspiration criterion*. Finally, the tabu search makes use in the process of three adaptive flexible memory structures:

- *Short term memory*, that keeps track of recently examined solutions, is intended to avoid cycling and interplays between conditions that constrain and free the search process (tabu restrictions and aspiration criterion).
- *Intermediate memory*, that is responsible for intensification strategies to reinforce move combinations and solutions features historically found good.
- *Long term memory*, that drives the search into regions not yet visited, providing a diversification phase in the process.

Tabu search has been largely applied to solve difficult combinatorial optimization problems [Aie96, PR95]. A tabu search approach for Scheduling Resource Constrained Project

with Cash Flows has been recently proposed in [ZP96] and the results confirm the potential of this method in finding good solutions for RCPS problems. All of these have encouraged the design of TS for the SPLC. The details of the algorithm we have proposed are described in the next section.

4 The algorithm TSSPLC - Tabu Search for the Scheduling Problem under Labour Constraints

Initially, let us give a general overview on the TS algorithm that we propose for solving SPLC, denoted by TSSPLC.

In the TSSPLC algorithm, one solution for an instance of the SPLC is represented by a job sequence vector S. Let n be the number of jobs to schedule and $ST_{S[k]}$ be the starting time of job $S_{[k]}$ in the solution represented by S. Then, in this representation, $ST_{S[i]} \leq ST_{S[j]}$, $\forall\ i < j$, $i, j = 1, ..., n$.

The framework we adopted for TSSPLC was the same proposed in [ZP96] and its general structure can be visualized in Figure 2.

The algorithm starts with an initial schedule which is built using one of the heuristics described in Section 4.1. Then it proceeds iteratively, choosing, at each iteration, one (the best) admissible move from the move candidate list. The moves in the candidate list are represented by two operations on jobs (Section 4.2), chosen in a way to lead the current solution to one of its precedence feasible neighbourhood solutions . Tabu restrictions can be overrided everytime the aspiration criterion (Section 4.6) is achieved, leading to a solution with a better makespan. The new schedule obtained at the end of each iteration is then returned to the working memory. The search stops when one of the termination conditions is achieved (Section 4.7). The best solution obtained over all the iterations is returned at the end of the process. Intensification and diversification strategies are not explored in our implementation of TSSPLC.

In the next sections we discuss in detail the main elements of the TSSPLC.

4.1 Initial Solution

Three greedy heuristics were implemented and tested for choosing an initial solution for the algorithm TSSPLC. Basically they work as follows: the schedule is constructed iteratively: at each period, there is a set of jobs available to be scheduled - those jobs whose predecessors have finished. Considering the labour constraints, some of these jobs can be scheduled immediately and others have to wait until there is enough workers available to their execution. The following greedy criteria, that differentiate the three heuristics, were used to choose the next schedulable job to be started:

- Total Order Duration (TOD): Jobs are chosen in descending order of the total duration of the order they belong to.
- Remaining Order Duration (ROD): Jobs are chosen in descending order of the remaining processing time of the order they belong to.

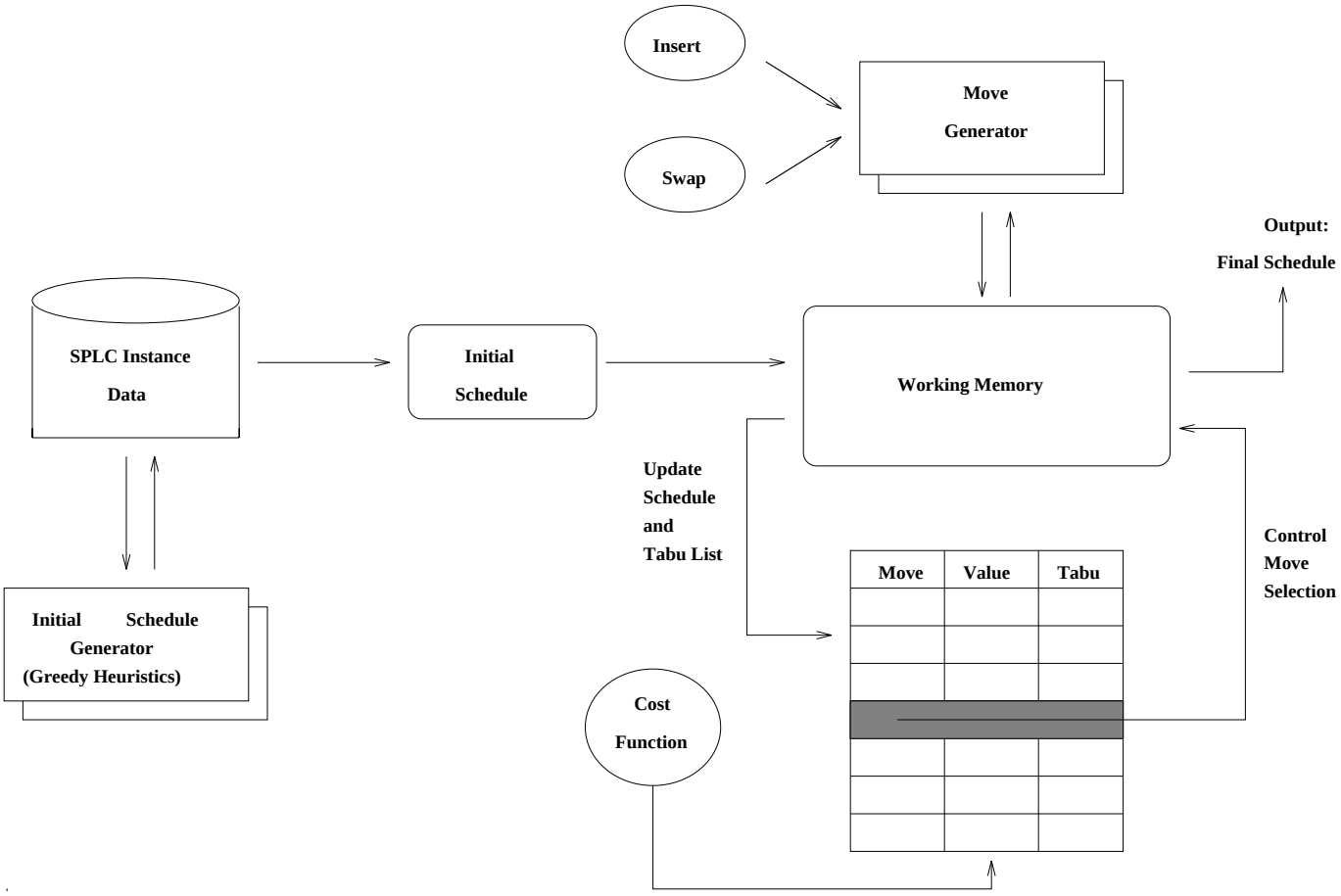

Figure 2: TSSPLC Framework.

- Critical Path (CP): Jobs are chosen in descending order of the length, taken in terms of processing time, of the critical path from them in the precedence relation digraph.

At this point, it is necessary to introduce the concept of *inverse instance*. The instances have, as part of the data describing them, a precedence relation digraph G. In addition, for each job j, there is its corresponding labour profile array $L_j =< \ell_1, \ell_2, ..., \ell_{p_j} >$, where p_j is the job duration. Given any instance I, its inverse image I' is such that all the data are equal to the data of the original (direct) instance, except that:

- If G and G' are the precedence digraphs of I and I', respectively, then G' is the transpose graph of G.
- The labour profile array of each job in I' is the inverse of the correspondending labour profile array in I. So, if in the direct instance I the labour profile array of a job j is given by $L_j =< \ell_1, \ell_2, ..., \ell_{p_j} >$, in the inverse instance I', this labour profile is $L'_j =< \ell_{p_j}, ..., \ell_2, \ell_1 >$.

The rationale behind the use of the inverse instance I' comes from the fact that any solution for I' when read from right to left is a solution for I.

The three greedy heuristics were tested for both direct and inverse instances. Thus, the algorithm TSSPLC can start with a solution built by one of the three heuristics (TOD, ROD or CP) applied to the direct or the inverse instance. A comparison of the initial solutions found by these greedy heuristics is presented in Section 5.2.

4.2 Neighbourhood Strategy

The neighbourhood definition is one of the critical aspects of tabu search. Previous studies show that the choice of a neighbourhood has a significant impact on the solution quality [Glo90, Lag95]. The difficulty lies on determining a simple strategy of move generation that gives the search process the chance to visit new and high quality solutions. By simplicity, it means that the moves must be easy to determine. Remember that, at each iteration of tabu search, a list of candidate moves is constructed. If the move generation requires a huge computational effort, the process becomes slow. Thus the goal is to determine a neighbourhood structure that is simple and that contains good solutions.

In our implementation, we have chosen to use the most common types of moves reported in the scheduling literature [ZP96]. Both of them use the sequence representation of the solution:

- Insert(i, j): Inserts job j immediately in front of job i in the sequence (Figure 3).
- Swap(i, j): Interchanges the positions of jobs i and j (Figure 3).

So, given a solution s, a neighbour solution of s is obtained through the application of a move, Insert or Swap, to a pair of jobs i, j of s. The set of all neighbour solutions of s defines the neighbourhood of s, $N(s)$.

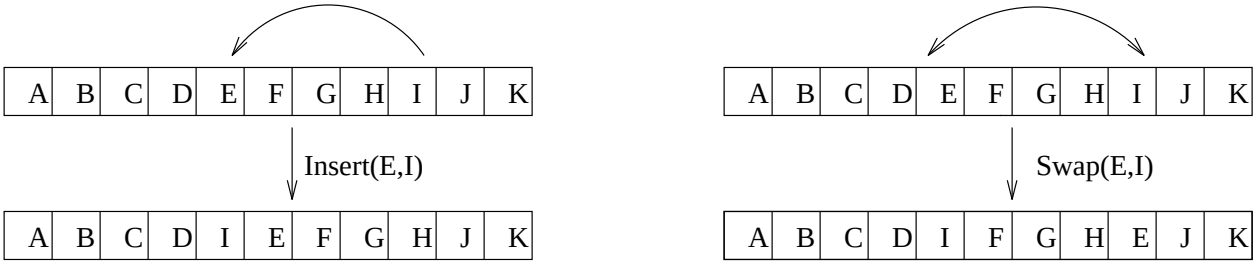

Figure 3: Example of Insert and Swap.

Note that the size of $N(S)$, like defined above, is $O(n^2)$, where n is the total number of jobs in one instance of SPLC. To decrease the number of neighbours to be tested for moving, only moves that results in precedence feasible schedules will be considered as candidate moves. For this reason, in every sequence representing a solution of an instance of SPLC, one job is in the right of all its predecessors.

Yet trying to decrease the size of $N(S)$, the following reduced neighbourhoods were defined:

- $RN1(s)$: Subset of $N(s)$ composed by k solutions each obtained from s through the application of an Insert move to a pair of jobs (i, j) randomly chosen.
- $RN2(s)$: Same as $RN1(s)$ except that the move applied to s is Swap.
- $RN3(s)$: Subset of $N(s)$ composed by the solutions obtained from s through the application of an Insert move to all pairs of jobs (i, j), for one specific j, randomly chosen, and for $i = 1, ..., n$, $i \neq j$.
- $RN4(s)$: Subset of $N(s)$ composed by the solutions obtained from s through the application of an Insert move to all pairs of jobs (i, j), for one specific i, randomly chosen, and for $j = 1, ..., n$, $j \neq i$.

The algorithm TSSPLC was tested with each one of the four reduced neighbourood defined above. Given the list of the alternative solutions delimited by the neigbourhood structure, the new solution chosen is the one with best cost and free of tabu restrictions.

4.3 Cost Function

The goal in SPLC is to find a minimum makespan solution. In this way, the cost function to be minimized used in the TSSPLC is given by:

$$c(s) = makespan$$

where the makespan value of a solution s is calculated as follows:

- Let S be the representative sequence of solution s. Initially, the schedule is empty.

- For $i = 1$ to n, schedule job $S[i]$ in the earliest position that it can be started considering the precedence relations and the labour availability.
- Return the makespan of the schedule obtained.

4.4 Tabu Restrictions

One of the major features of the tabu search procedure is the possibility of forbidden some of the candidate moves for a certain number of iterations. This has two objectives: first, to prevent cycling, which happens when the search process visits always the same subset of solutions; and second, to allow an effective exploration of the solutions space.

The most common tabu restrictions are those recency based [Lag95]. According to this strategy, are labeled tabu-active the attributes of moves that were recently executed and that resulted in solutions that deteriorated the cost function. Moves that contain tabu-active attributes are those that become tabu. A tabu move cannot be executed unless an aspiration criterion (Section 4.6) is achieved.

The following move attributes were defined for the two types of moves, Insert and Swap, adopted in the TSSPLC algorithm:

- $A1$: Job j was moved.
- $A2$: Job j was inserted immediately in front of job i in the sequence.
- $A3$: Jobs i, j were exchanged.

Associated with each one of the above move attributes, three types of tabu restrictions were defined:

- $TR1$: Job j cannot be moved for a certain number of iterations.
- $TR2$: Job j cannot be inserted immediately in front of job i in the sequence for a certain number of iterations.
- $TR3$: Jobs i, j cannot be exchanged for a certain number of iterations.

Note that the tabu restriction $TR1$ is stronger than the other two. This is because $TR1$ forbidds all moves of a job j in the sequence. On the other hand, $TR2$ and $TR3$, only restrict those moves that involve, simultaneously, jobs i and j.

4.5 Tabu Tenure

The tabu tenure specifies the number of iterations for which a particular move is prohibited. In the TSSPLC three tabu tenures were defined, one for each type of tabu restriction:

- $TT1$: 5
- $TT2$: 10

- $TT3$: 7

$TT1$ is smaller than $TT2$ and $TT3$ according to the fact that $TR1$ is more restrictive than $TR2$ and $TR3$. Stronger tabu restrictions should be mantained active for smaller periods of time [Glo89].

4.6 Aspiration Criterion

Tabu restrictions have the purpose of preventing cycling and of making effective the search process. In some situations, however, a temporarilly tabu move can lead to a better solution than the best one currently available. The aspiration criterion appears to solve this problem: it allows to override a tabu restriction whenever the correponding tabu move leads to a solution better than the best known solution at the moment.

In the TSSPLC algorithm, the aspiration criterion is activated everytime a tabu move leads to a solution of smaller makespan than the best makespan current available. In this case, the corresponding tabu restriction is deactivated and the move is executed.

4.7 Termination Condition

Two possible stopping conditions were tested in the TSSPLC algorithm: limit on the maximum number of iterations of the search process and limit on the maximum number of iterations where there is no improvement in the cost of the best known solution.

4.8 An Example

In this section we give an example of the application of the TSSPLC algorithm to the SPLC instance defined in Tables 1,2 and in Figure 2 . Each solution is associated to a sequence vector as defined in Section 4.

Let s be the current schedule at each iteration and let s' be a neighbour schedule of s obtained through the application of an Insert move to a pair of jobs (i, j) of s. For each iteration, the Figures 4,5,6,7,8 show:

- The sequence associated with the current schedule s.
- A table representing the list of candidate moves (only Insert moves that leads to precedence feasible schedules s' are considered) and the makespan (given by the cost function) of the corresponding schedules s'.
- A table T representing the tabu status of the Insert moves. $T[i][j]$ is the number of the iteration from which the move Insert(i, j) is allowed (no number means $T[i][j] = 0$. If the iteration number is smaller then $T[i][j]$, Insert(i, j) is tabu-active.

The move in the candidate list that is not tabu and that leads to the schedule s' with smallest makespan is chosen to be applied to the current solution.

The initial schedule has makespan 23. Initially, no move is tabu. So, the moves in candidate list are evaluated solely based on the makespan of the correponding schedules s'.

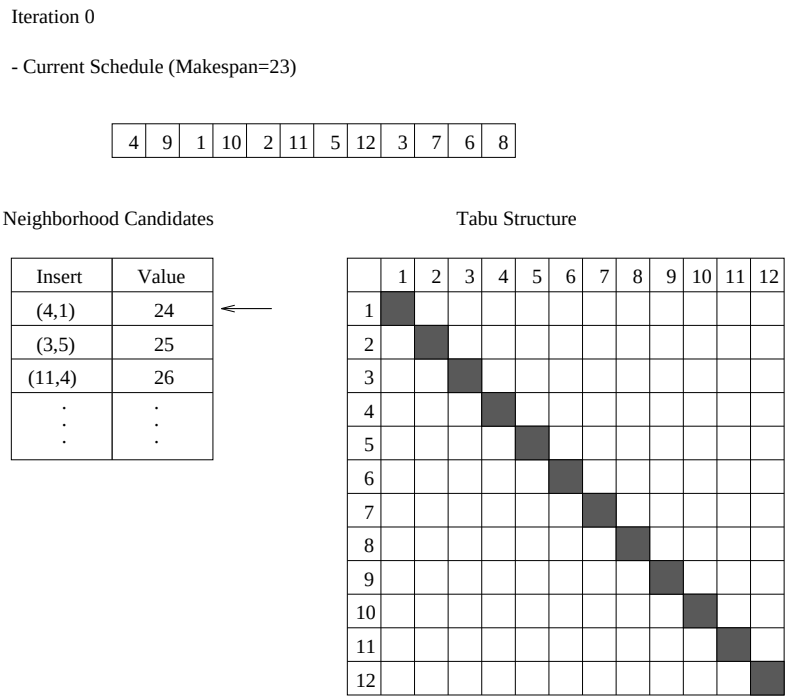

Figure 4: TSSPLC Example - Iteration 0.

The move that leads to a s' with smallest makespan, Insert(4,1), is selected to be applied. Note that this move is chosen even though it produces a worse schedule s' (with a greater makespan of 24). The advantages of this move will be clear in the next iteration when s' will be re-evaluated.

At iteration 1, the schedule obtained by inserting job 1 in front of job 4 becomes the current solution. Since this move resulted in a deteriorated solution, its repetition is prevented for a certain number of iterations (3 in this example) by updating the tabu tenure in the tabu structure table. In this iteration the move chosen to be applied is Insert(3,6). The resulting schedule s' has makespan 21 and is delivered to the next iteration as the current solution. Note that since the new schedule has a better makespan, the repetion of the move Insert(3,6) in the next iterations still allowed.

At iteration 2, the move chosen is Insert(9,1) which leads to an improved schedule of makespan 20.

At iteration 3, a better schedule can be obtained by applying Insert(2,5) to the current solution. Note that although Insert(4,1) is an admissible move in the candidate list, this move is tabu-active in this iteration (and until iteration 4 since $T[4][1] = 4$).

The iterations continue in a similar manner as described above until one termination condition is satisifed. The schedule of smallest makespan found in the whole process is returned at the end.

Iteration 1

- Current Schedule (Makespan=24)

1	4	9	10	2	11	5	12	3	7	6	8

Neighborhood Candidates

Insert	Value
(3,6)	21
(10,5)	23
(5,3)	26
. . .	. . .

Tabu Structure

	1	2	3	4	5	6	7	8	9	10	11	12
1												
2												
3												
4	3											
5												
6												
7												
8												
9												
10												
11												
12												

Figure 5: TSSPLC Example - Iteration 1.

Iteration 2

- Current Schedule (Makespan=21)

1	4	9	10	2	11	5	12	6	3	7	8

Neighborhood Candidates

Insert	Value
(9,1)	20
(7,6)	21
(11,4)	23
. . .	. . .

Tabu Structure

	1	2	3	4	5	6	7	8	9	10	11	12
1												
2												
3												
4	2											
5												
6												
7												
8												
9												
10												
11												
12												

Figure 6: TSSPLC Example - Iteration 2.

Iteration 3

- Current Schedule (Makespan=20)

4	1	9	10	2	11	5	12	6	3	7	8

Neighborhood Candidates

Insert	Value
(2,5)	19
(4,1)	20
(10,4)	21
. . .	. . .

Tabu Structure

	1	2	3	4	5	6	7	8	9	10	11	12
1	■											
2		■										
3			■									
4	1			■								
5					■							
6						■						
7							■					
8								■				
9									■			
10										■		
11											■	
12												■

Figure 7: TSSPLC Example - Iteration 3.

Iteration 4

- Current Schedule (Makespan=19)

4	1	9	10	5	2	11	12	6	3	7	8

Neighborhood Candidates

Insert	Value
. . .	. . .

Tabu Structure

	1	2	3	4	5	6	7	8	9	10	11	12
1	■											
2		■										
3			■									
4				■								
5					■							
6						■						
7							■					
8								■				
9									■			
10										■		
11											■	
12												■

Figure 8: TSSPLC Example - Iteration 4.

5 Data, Experiments and Results

This section describes the instances generated to test the TSSPLC algorithm, the computational experiments carried out and the results obtained.

5.1 Data Sample

From the Project PAMIPS [PAM], two practical instances of SPLC were available. In order to have a larger sample to test TSSPLC, a random generator of SPLC instances was implemented with the following charactheristics:

- INPUT
  - m: total number of orders in the instance;
  - mj: minimum number of jobs in one order of the instance;
  - Mj: maximum number of jobs in one order of the instance;
  - md: minimum duration of a job in one order of the instance;
  - Md: maximum duration of a job in one order of the instance;
  - p: probability of jobs from different orders been related.
- OUTPUT
  - Instance with m orders, where each order has between mj and Mj identical jobs. The jobs in a given order have the same duration, randomly chosen between md and Md. The labour requirement of each task of a job is chosen from the set $\{2, 3, (4), 6, (12), (18)\}$, where numbers in brackets are less frequent. The precedence graph describing the relations between jobs is built in such a way that jobs from different orders are related with probability p if they belong to consecutive orders. Jobs from nonconsecutive orders cannot be related. Precedence relations between jobs in the same order are implicit.

The data set generated contains 23 small, medium and large size instances of SPLC and can be obtained in [Cav97]. The instances Ins_4o_24j_A and Ins_10o_88j_A were obtained from Project PAMIPS [PAM].

5.2 Computational Experiments and Results

The code was written using the C++ language and compiled with the gnu g++ compiler. The tests were done in a SUN SPARCstation 1000.

Three greedy heuristics (Section 4.1) were tested for obtaining an initial solution for the TSSPLC algorithm. The results obtained by each of them when applied to the data set are summarized in Table 3. For each instance, the solutions are displayed in the form (s, s') where s and s' are the solutions obtained with the direct and inverse instances, respectively. As can be seen, none of the heuristics dominates the other two and frequently a better solution is found when the inverse instance is considered.

	TOD	ROD	CP
Ins_4o_21j_A	(102,87)	(87,89)	(92,**86**)
Ins_4o_23j_A	(76,**58**)	(76,**58**)	(67,**58**)
Ins_4o_24j_A	(84,76)	(82,75)	(75,**74**)
Ins_4o_24j_B	(78,89)	(77,85)	(**73**,89)
Ins_4o_27j_A	(71,85)	(71,79)	(**69**,79)
Ins_6o_41j_A	(167,154)	(165,**152**)	(169,155)
Ins_6o_41j_B	(137,120)	(141,122)	(137,**114**)
Ins_6o_41j_C	(161,158)	(168,149)	(171,**140**)
Ins_6o_44j_A	(137,**127**)	(135,138)	(135,128)
Ins_6o_44j_B	(160,180)	(156,177)	(**155**,172)
Ins_8o_63j_A	(307,352)	(293,344)	(**284**,348)
Ins_8o_63j_B	(371,414)	(359,384)	(**357**,398)
Ins_8o_63j_C	(392,384)	(367,386)	(**346**,371)
Ins_8o_65j_A	(479,**433**)	(442,449)	(445,451)
Ins_8o_65j_B	(466,468)	(433,482)	(**419**,487)
Ins_10o_84j_A	(874,885)	(751,798)	(**735**,794)
Ins_10o_84j_B	(749,646)	(684,**605**)	(692,631)
Ins_10o_85j_A	(919,978)	(**910**,1045)	(912,1054)
Ins_10o_87j_A	(656,636)	(650,**611**)	(640,615)
Ins_10o_88j_A	(563,538)	(567,516)	(574,**490**)
Ins_10o_100j_A	(1618,1715)	(1610,1696)	(**1595**,1729)
Ins_10o_102j_A	(1449,1323)	(1516,1337)	(1450,**1239**)
Ins_10o_106j_A	(1248,1230)	(1213,1183)	(1223,**1146**)
Ins_12o_108j_A	(1492,1534)	(1505,**1386**)	(1483,1415)
Ins_12o_109j_A	(1741,1570)	(1582,1500)	(1651,**1476**)

Table 3: Results obtained by the greedy heuristics.

	Neighbourhood Strategy	Tabu Restriction	Tabu Tenure
TSSPLC_1	$RN1$	$TR1$	$TT1$
TSSPLC_2	$RN1$	$TR2$	$TT2$
TSSPLC_3	$RN2$	$TR1$ applied to jobs i and j	$TT1$
TSSPLC_4	$RN2$	$TR3$	$TT3$
TSSPLC_5	$RN3$	$TR1$	$TT1$
TSSPLC_6	$RN3$	$TR2$	$TT2$
TSSPLC_7	$RN4$	$TR1$	$TT1$
TSSPLC_8	$RN4$	$TR2$	$TT2$

Table 4: TSSPLC configurations.

Given the general features of TSSPLC described in section 4, eight types of tabu search were defined by the combination of the different neighbourhood strategies, tabu restrictions and tabu tenure. The resume of the processes configuration can be seen in Table 4. The common aspects in all of the types of TSSPLC are:

- Initial Solution: the best obtained by the greedy heuristics.
- Cost function: as defined in Section 4.3.
- Aspiration Criterion: as defined in Section 4.6.
- Termination Condition: a limit of 500 on the total number of iterations was used.

The makespan of the solutions obtained with the application of all the configurations of TSSPLC in Table 4 to the data set are summarized in Table 5. All the solutions were obtained for a scenario with $R_t = 18$ available workers.

It can be verified from Table 5 that in 23 of the 25 instances used in the tests, the TSSPLC algorithm finished with a solution better than the initial one. It can also be noted that the configuration TSSPLC_2 is the one with best results: it reaches the best solutions in 15 instances, and even when another configuration of TSSPLC generates a solution of better makespan, the solution returned by TSSPLC_2 is at most 1% worse. Surprisingly, naive greedy heuristics have been able to produce schedules with makespan only a few longer than those produced by TS routines.

Figures 9 and 10 show the trace of TSSPLC_2 in a medium and large size instance, respectively. These figures seem to indicate that the best solutions were found only at the final iterations.

Figure 11 illustrates the trace of TSSPLC_2 in the same instance used in Figure 9. The difference here is the use of another termination condition: 500 iterations without improvement in the best known solution. Note that in Figure 11 only after 1250 iterations is that the algorithm reached a solution 2% better (Makespan=304) then the best found in the 500 iterations of Figure 9 (Makespan=310). We think that the additional computational

Table 5: Results obtained by all configurations of TSSPLC.

	Initial Sol	TSSPLC$_1$	TSSPLC$_2$	TSSPLC$_3$	TSSPLC$_4$	TSSPLC$_5$	TSSPLC$_6$	TSSPLC$_7$	TSSPLC$_8$
Ins_4o_21j_A	86	**82**	**82**	**82**	**82**	**82**	**82**	**82**	**82**
Ins_4o_23j_A	**58**	**58**	**58**	**58**	**58**	**58**	**58**	**58**	**58**
Ins_4o_24j_A	74	69	**68**	70	69	70	69	69	69
Ins_4o_24j_B	73	**72**	**72**	**72**	**72**	73	73	73	**72**
Ins_4o_27j_A	69	68	**67**	**67**	**67**	68	69	**67**	68
Ins_6o_41j_A	152	149	147	**146**	147	147	145	**146**	149
Ins_6o_41j_B	**114**	**114**	**114**	**114**	**114**	**114**	**114**	**114**	**114**
Ins_6o_41j_C	140	**129**	130	130	**129**	131	133	131	131
Ins_6o_44j_A	127	**117**	118	119	119	120	127	119	120
Ins_6o_44j_B	155	141	**140**	148	146	149	155	142	**140**
Ins_8o_63j_A	284	263	262	265	268	271	284	**261**	266
Ins_8o_63j_B	357	322	320	**319**	322	328	357	320	325
Ins_8o_63j_C	346	315	**310**	314	316	315	346	316	313
Ins_8o_65j_A	433	414	**405**	407	410	413	433	428	429
Ins_8o_65j_B	419	393	**392**	403	395	405	419	**392**	393
Ins_10o_84j_A	735	**655**	657	685	668	688	689	677	664
Ins_10o_84j_B	605	583	581	592	590	582	585	589	**579**
Ins_10o_85j_A	910	848	**843**	896	892	896	909	858	862
Ins_10o_87j_A	611	601	597	**596**	607	599	605	601	606
Ins_10o_88j_A	490	479	**467**	490	490	490	490	478	482
Ins_10o_100j_A	1595	1541	**1537**	1543	1558	1565	1576	1553	1576
Ins_10o_102j_A	1239	1210	1212	1205	**1202**	1219	1232	1204	1213
Ins_10o_106j_A	1146	1142	**1133**	1146	1146	1146	1146	1140	1144
Ins_12o_108j_A	1386	1348	1349	1341	1339	1345	1356	1360	**1335**
Ins_12o_109j_A	1476	1458	1397	1427	1412	1433	1423	1428	**1382**

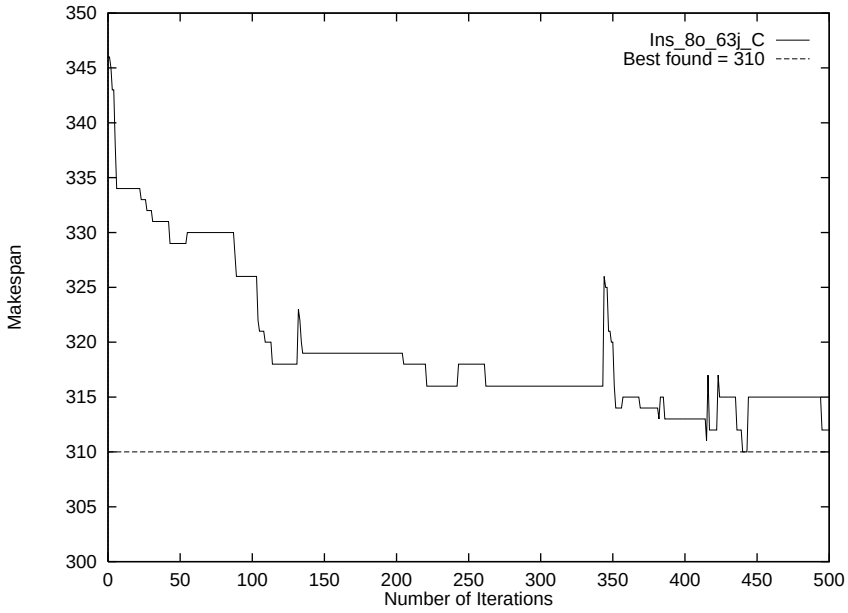

Figure 9: Trace of TSSPLC on Ins_8o_63j_C.

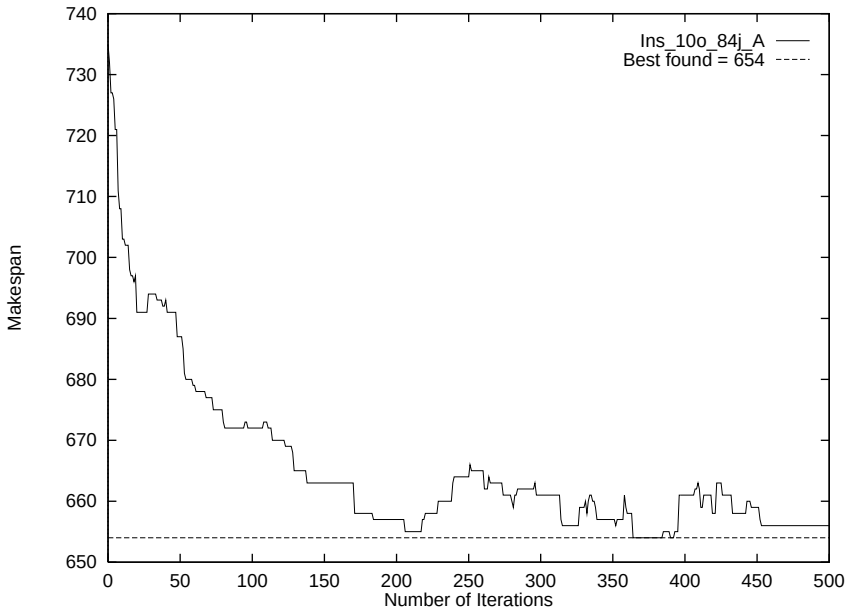

Figure 10: Trace of TSSPLC on Ins_10o_84j_A.

effort is not worthwhile. For this reason, we addopted as termination condition in all the experiments a limit of 500 in the total number of iterations.

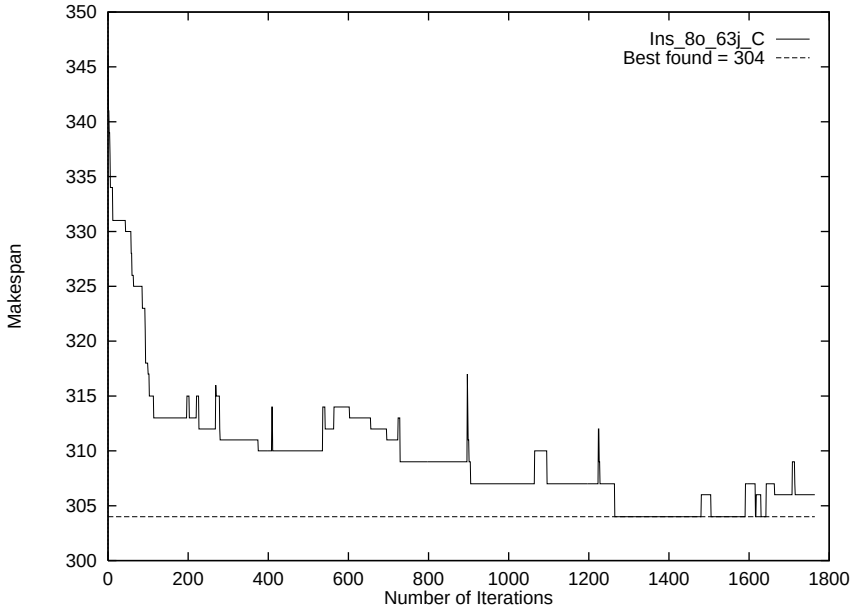

Figure 11: Effect of Different Termination Condition: 500 iterations without improvement in the best known solution.

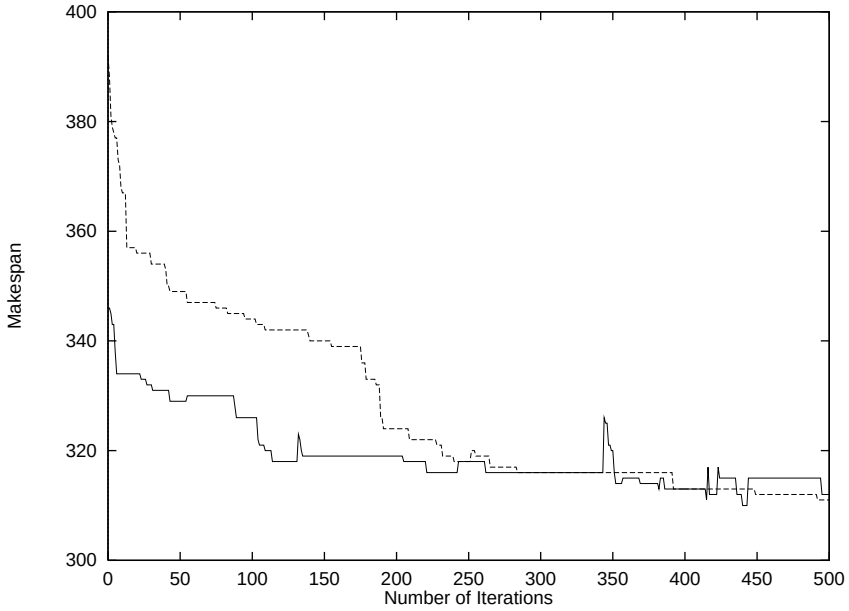

Figure 12: Effect of Different Initial Solutions.

Figure 12 illustrates the behaviour of TSSPLC$_2$ with two different starting solutions.

The dark and dashed lines represent, respectively, the trace of the algorithm when it starts with the best and worst solution found by the greedy heuristics in Table 3 for the Ins_8o_63j_C instance. In the first case, the best solution (Makespan=310) was found in iteration 440. In the second case, the best solution (Makespan=311) was found in iteration 492. This confirms the observation that the initial solution quality has an impact on the performance of tabu search.

The Table 6 presents the comparison of the results obtained in our implementation of TSSPLC with the best ones generated by a Constraint Logic Programming approach to SPLC [HC97].

It can be verified that for all instances the makespan obtained by TSSPLC is less than or equals to the makespan generated with the Constraint Logic Programming algorithm described in [HC97]. However, it is important to note that while CLP best solutions were found in few seconds, TSSPLC sometimes took more than one hour to find its best solutions [Cav97]. Another advantage of CLP is that, at least for small instances, it can prove optimality. In general, unless the makespan is equal to the length of the critical path, TSSPLC cannot prove optimality of a solution. It is worth to note that in 13 of the 25 instances tested, the best solution obtained by the greedy heuristics (Table 3) is better than the corresponding CLP solution. In the remaining 12 instances, the greedy solutions are no more than 5% worse than the CLP solution.

6 Conclusions and Extensions

We presented a tabu search approach for the Scheduling Problem under Labour Constraints (SPLC). The algorithm was tested in a set of 25 small, medium and large size instances of SPLC. The results obtained in the computational experiments show that our tabu search heuristic is comparable with, if not better than, other heuristics reported in the literature. Moreover, it is a simple strategy to implement and, therefore, we can conclude that it is a good alternative to be used when one is seeking for good solutions of SPLC instances.

The TSSPLC algorithm can be further improved by incorporating intermediate and long term memory structure. A dynamic tabu tenure can also be included. The study of different neighbourhood strategies and cost functions is certainly one of the main aspects to be explored in future works.

7 Acknowledgments

We are very grateful to Susanne Heipcke who kindly have tested her own CLP code for the data set used in the computational experiments.

References

[Aie96] R. M. Aiex, *Estratégias Paralelas Assíncronas de Busca Tabu Aplicadas ao Problema de Particionamento de Circuitos*, Msc Dissertation (in portuguese), Dep. de Informática, PUC-RJ, Rio de Janeiro, Brazil, Aug/1996.

R_t	18
Ins_4o_21j_A	(**82**, **82**)
Ins_4o_23j_A	(**58**, **58**)
Ins_4o_24j_A	(**68**,**68**)
Ins_4o_24j_B	(**72**,**72**)
Ins_4o_27j_A	(**67**,**67**)
Ins_6o_41j_A	(**147**,156)
Ins_6o_41j_B	(**114**,125)
Ins_6o_41j_C	(**129**,153)
Ins_6o_44j_A	(**117**,127)
Ins_6o_44j_B	(**140**,149)
Ins_8o_63j_A	(**261**,281)
Ins_8o_63j_B	(**319**,344)
Ins_8o_63j_C	(**310**,344)
Ins_8o_65j_A	(**412**,445)
Ins_8o_65j_B	(**392**,411)
Ins_10o_84j_A	(**654**,730)
Ins_10o_84j_B	(**571**,678)
Ins_10o_85j_A	(**830**,912)
Ins_10o_87j_A	(**597**,638)
Ins_10o_88j_A	(**467**,489)
Ins_10o_100j_A	(**1494**,1587)
Ins_10o_102j_A	(**1221**,1450)
Ins_10o_106j_A	(**1133**,1243)
Ins_12o_108j_A	(**1297**,1483)
Ins_12o_109j_A	(**1415**,1647)

Table 6: Comparison of best results obtained with TSSPLC and Constraint Logic Programming approaches. The results are in the form (TSSPLC solution, CLP solution).

[Ba74] Baker K.R., *Introduction to Sequencing and Scheduling*, John Wiley and Sons, New York, 1974.

[Cav97] http://www.dcc.unicamp.br/~cris/SPLC.html

[BESW93] J. Blazewicz, K. Ecker, G. Schmidt e J. Werglarz, *Scheduling in Computer and Manufacturing Systems*, Springer-Verlag, Berlin, 1993.

[BLR83] J. Blazewicz, J. K. Lenstra and A. H. G. Rinnooy Kan, "Scheduling subject to resource constraints: classification and complexity". *Discrete Applied Mathematics*, 5:11-24, 1983.

[DH92] Demuelemeester E. and W. Herreolen, "A branch-and-bound procedure for the multiple resource constrained project scheduling problem", *Management Science*, 38(12):1803-1818, 1992.

[GJ79] M. R. Garey e D. S. Johnson, *Computer and Intractability: A Guide to the Theory of NP-Completeness*, Freeman, San Francisco, 1979.

[Glo89] F. Glover, "Tabu Search - Part I". *ORSA Journal on Computing*, 1(3):190-206, 1989.

[Glo90] F. Glover, "Tabu Search - Part II". *ORSA Journal on Computing*, 2(3):3-32, 1990.

[HC97] S. Heipcke and Y. Colombani, "A New Constraint Programming Approach to Large Scale Resource Constrained Scheduling".

[Hei95] S. Heipcke, *Resource Constrained Job-Shop Scheduling with Constraint Nets: Two Case Studies*, Diploma-Thesis, Mathematisch Geographische Fakulät, Katholische Universität Eichstätt, Jan/1995.

[Lag95] M. Laguna, *Tabu Search Tutorial*, II Escuela de Verano Latino-Americana de Investigacion Operativa, Mendes, RJ, Brazil, Jan/1995

[LW] L. Wolsey, *Private Communication*

[NW88] G. L. Nemhauser e L. A. Wolsey, *Integer and Combinatorial Optimization*, Jonh Wiley, 1988.

[NWS95] K. Naphade, S. D. Wu and R. H. Storer, "A Problem Space Search Method for the Resource Constrained Project Scheduling Problem". Submitted to *The Annals of Operations Research*. http://www.lehigh.edu/~rhs2/rhs2.html.

[PAM] http://www.iwr.uni-heidelberg.de/iwr/agbock/doc/pamips.html

[PR95] S. C. S. Porto e C. C. Ribeiro, *A Tabu-Search Approach to Task Scheduling on Heterogeneous Processors under Precedence Constraints*, Int. Journal of High-Speed Computing , vol. 7, 1995.

[SW93] Sampson S. E. and E. N. Weiss, "Local Search Techniques for the Generalized Resource Constrained Project Scheduling Problem", *Naval Research Logistics*, 40(5):665-676, 1993.

[ZP96] D. Zhu and R. Padman, "A Tabu Search Approach for Scheduling Resource-Constrained Projects with Cash Flows". Working Paper, The Heinz School, Carnegie Mellon University, Pittsburgh, PA 15213.