# JOB-SHOP WITH GENERIC TIME-LAGS: A HEURISTIC BASED APPROACH

Marie-José Huguet, Philippe Lacomme, Nikolay Tchernev

*9th International Conference of Modeling, Optimization and Simulation - MOSIM'12*
*June 06-08, 2012 – Bordeaux - France*
*"Performance, interoperability and safety for sustainable development"*

# JOB-SHOP WITH GENERIC TIME-LAGS: A HEURISTIC BASED APPROACH

**P. LACOMME, N. TCHERNEV**

Université Blaise Pascal
LIMOS - UMR CNRS 6138
Campus des Cézeaux,
63177 Aubière Cedex
{placomme, tchernev}@isima.fr

**M.J. HUGUET**

CNRS ; LAAS ;
7 avenue du Colonel Roche,
F-31077 Toulouse Cedex 4, France
Université de Toulouse ; UPS, INSA, INP, ISAE ; UT1, UTM, LAAS ; F-31077 Toulouse Cedex 4, France
huguet@laas.fr

**ABSTRACT:** *This paper deals with the job-shop scheduling problem with generic time-lags (JSPGTL). This problem is a generalization of the job-shop scheduling problem where extra (minimal and maximal) delays can appear between any operations. To solve the problem we extend the ARP-MD Deppner's heuristic to tackle this extension providing a new randomized heuristic. And propose a greedy version denoted GREEDY_ARP-MD. The numerical experiments are based on a set of 48 instances including 8 instances based on the flow-shop Carlier's instances and on the well-known 40 Laurence'sjob-shop instances. The numerical experiments proved that ARP-MD heuristic is time consuming and can be used only for small scale instances. Instances with 10 jobs and 10 machines required several hours to obtain a solution. The greedy version is strongly efficient and can be executed thousands of time per second and so gives solutions for a part of the medium and large scale instances. This work is a step into definition of heuristic for the JSPGTL based on the initial Deppner's proposal. This sequel study prove that definition of efficient heuristic for definition of solutions is a challenging problem and would require a considerable amount of attention to obtain time saving approaches*

**KEYWORDS:** *Scheduling, Job-shop problem, Generic time-lags, heuristic*

## 1 INTRODUCTION

The job-shop problem with minimum and maximum time-lags (*JSPGTL*) is a generalization of the job-shop problem, in which there are time relations between the starting times of two successive operations belonging to any two jobs. The *JSPGTL* involves a set of $n$ jobs $(i = 1, 2, \ldots n)$ that have to be processed on set of machines $m$ $(j = 1, 2, \ldots m)$. Each job is fully defined by an ordered sequence of operations that are associated with a particular machine. Therefore, the dimension of the problem is often denoted as $n \times m$. In addition, the process must satisfy other constraints such as: *(i)* no more than one operation of any job can be executed simultaneously; and *(ii)* no machine can process more than one operation at the same time; *(iii)* the job operations must be executed in a predefined sequence and once an operation is started, no preemption is permitted.

A time-lag can be defined between the finish time of a given operation $O_{i,j}$ (denoted by $ft_{o_{i,j}}$) and the start time of another operation $O_{i',j'}$ (denoted by $st_{o_{i',j'}}$) using the following equation:

$$l_{o_{i,j},o_{i',j'}} \leq st_{o_{i',j'}} - ft_{o_{i,j}} \leq L_{o_{i,j},o_{i',j'}} \quad (1)$$

with $L_{o_{i,j},o_{i',j'}} \geq l_{o_{i,j},o_{i',j'}}$.

In this formula $l_{o_{i,j},o_{i',j'}}$ represents the minimal time-lag and $L_{o_{i,j},o_{i',j'}}$ is the maximal time-lag. The first part of this formula $l_{o_{i,j},o_{i',j'}} \leq st_{o_{i',j'}} - ft_{o_{i,j}}$ means that $O_{i',j'}$ cannot start before at least $l_{o_{i,j},o_{i',j'}}$ units after the end of $O_{i,j}$. The second part of the formula $st_{o_{i',j'}} - ft_{o_{i,j}} \leq L_{o_{i,j},o_{i',j'}}$ means that $O_{i',j'}$ cannot be started latter than $L_{o_{i,j},o_{i',j'}}$ units after the end of $O_{i,j}$.

As stressed by Brucker (Brucker *et al.*, 1999), for the *JSPGTL* without preemption and fixed processing times, the time-lags constraints can be formulated with only "*start-start*" relations by using $st_{o_{i,j}} = ft_{o_{i,j}} + p_{o_{i,j}}$.

Then time-lags can be defined by the formula:

$$l_{o_{i,j},o_{i',j'}} + p_{o_{i,j}} \leq st_{o_{i',j'}} - st_{o_{i,j}} \leq L_{o_{i,j},o_{i',j'}} + p_{o_{i,j}} \quad (2)$$

The general time-lags constraints made the problem very hard to solve. For example, the single machine *JSSP* is

polynomial solve for the makespan minimization, since each semi-active schedule is an optimal one. However, the same problem with general time-lags constraint is proven to be *NP-hard* (Wikum *et al.*, 1994). Moreover, even finding a feasible solution is a NP-complete problem.

According to the $\alpha|\beta|\gamma$ notation introduced by (Graham et al. 1976) the problem can be represented by $J\left|l_{o_{i,j},o_{i',j'}}\right|C_{\max}$ (Brucker *et al.*, 1999).

Time-lags between the start and completion times of different activities have to be observed in numerous scheduling problems including the *RCPSP* where resource consumption is addressed (Brucker *et al.*, 1999). They result from technological or organizational constraints in practice. Besides minimum time-lags, maximum time-lags might be given which occurs in chemical industries and food industries.

Recent publications on *RCPSP* focus on *RCPSP* extensions including but not limited to multi-mode/time-lags (Coelho and Vanhoucke, 2011), reactive scheduling in multi-mode (Deblaere et al., 2011). Methods are for numerous proposals, based on heuristic and meta-heuristics. See (Hartmann and Briskorn, 2010) for a survey of RCPSP variants previously addressed in publications.

The classical job-shop problem is a well-addressed problem in the literature but only few articles are concerned with time-lag constraints. (Wikum *et al.*, 1994) study single-machine problems with minimum and/or maximum distances between jobs and state that some particular single-machine problems with time-lags are polynomially solvable, even if the general case is NP-hard. Brucker et al., 1999 show that many scheduling problems (such as multi-processor tasks or multi-purpose machines) can be modeled as single-machine problems with time-lags and propose a branch-and-bound method. A local search approach can be found in (Hurink and Keuchel, 2001).

Furthermore, since the job-shop problem with time-lags can be viewed as a special case of the RCPSP with time-lags, the relevant literature on this problem also applies to the *JSPGTL* (Kolicsh and Padman 2001; Neumann et al., 2002). However, in general, finding a feasible solution with time-lags is an NP-complete problem for the *JSPGTL*.

(Deppner, 2004) proposed heuristics for a general scheduling problem which includes the job-shop problem. In his thesis, minimal and maximal time-lags between every pair of operations are tackled.

(Fondrevelle et al. 2006) investigated permutation flow-shop problems with minimal and/or maximal time lags, where the time lags are defined between couples of successive operations of jobs. They presented theoretical results concerning two-machine cases and proved that the two-machine permutation flowshop with constant maximal time lags is strongly NP-hard. An optimal branch and bound procedure to solve the m-machine permutation flowshop problem with minimal and maximal time lags is developed. Several lower bounds were implemented and tested, and some constructive heuristics were adapted to the particular constraints of the problem, to provide initial upper bounds.

Later (Fondrevelle et al. 2008) succefully adapted the best bound among those proposed (Fondrevelle et al. 2006) and developed a branch and bound procedure to solve the permutation flowshop scheduling problems with time lags with objective to minimize the weighted sum of machine completion times. It is shown that the problem under study generalizes makespan and several complexity results for two- and three-machine problems are derived.

(Zhang and de Velde, 2010) investigated the open shop with generic time-lags. The performance of the greedy algorithm for the on-line two-machine open shop scheduling problem of minimizing makespan is analyzed. It is proven that the competitive ratio for the greedy algorithm is 2, and it can be reduced to 5/3 if the maximum time lag is less than the minimum positive processing time of any operation. They also proved that no on-line non-delay algorithm can have a better competitive ratio.

(Dhouib et al., 2012.) studied the permutation flowshop scheduling problem with sequence dependent setup times and time lags constraints of successive operations of the same job minimizing the number of tardy jobs. Two mathematical programming formulations are proposed for the considered problem. A simulated annealing algorithm is also developed to solve the problem.

(Javadian et al., 2012) introduced meta-heuristic algorithm based on the immune algorithm for hybrid flow shop scheduling problems considering time lags and sequence-dependent setup times to minimize the makespan. A mathematical model is presented which is capable of solving the small size of the considered problem in a reasonable time. Numerical experiments are used to evaluate the performance and effectiveness of the proposed algorithm. Computational results indicate that the proposed algorithm can produce near-optimal solutions in a short computational time. Moreover, it can be applied easily in real factory conditions and for large-sized problems.

Lately some research focus on the job-shop scheduling problem with minimal and maximal time-lags between successive operations of the same job. For this problem it is possible to obtain solution considering scheduling where all operations of a shop are schedule

consecutively giving worst quality solution which is used as initial solution of iterative search process. These solutions are denoted canonical solutions (Caumond *et al.*, 2008). Authors proposed a memetic based approach taking advantages of initial solutions generation by a heuristic which adds each operation iteratively, a powerful local search based on the critical path analysis and on a neighbouring generation system. Their approach is validated solving both flow-shop, job-shop, no-wait and instances with time-lags.

(Artigues et al., 2011) introduced powerful heuristic and generalized resource constraint propagation mechanisms. The proposed approach based on the disjunctive time-bound-on-node (TBON) graph introduced by (Esquirol et al., 1995) is more efficient than the memetic algorithm of (Caumond et al. 2008). The disjunctive TBON representation is equivalent to the disjunctive graph, but it allows the distinct visualization of the different components of the problem: duration time-lags, start and finish times, although it yields a larger number of nodes.

Recently (Lacomme et al. 2011) proposed an integer linear model and some dedicated constraint propagation rules which aim to detect some inconsistencies for the *JSPGTL*. The efficiency of these rules is illustrated on a small example. These propagation rules can be included in a heuristic method for the acceleration of the search for feasible solution thanks to early detection of certain inconsistencies.

## 2 HEURISTIC BASED APPROACH FOR JSPGTL SOLVING

The first heuristic proposed by Deppner (Deppner, 2004) is a priority dispatching rules generation based on a similar approach like the Giffler and Thomson's algorithm (Giffler and Thomson 1960). This is a constructive heuristic which consists in scheduling of one operation per iteration and which includes a backtracking system to avoid trap induced by the maximal time-lags.

This heuristic is known to be inefficient in practice because of too many backtracks. In the following paragraphs, we will propose an improvement of this heuristic.

The problem is modelized as a non-oriented disjunctive graph. Since a job sequence on machines is generated, it is possible to obtain an oriented disjunctive graph. A Bellman like longest path algorithm permits to compute the earliest completion time of the last operation.

### 2.1 Problem modeling and solution representation

#### *2.1.1 Disjunctive graph model*

First introduced by (Roy and Sussmann 1964) the disjunctive graph models each operation by a vertex and precedence constraints between operations of one job are represented by an arc. Disjunctive constraints between operations of two jobs which required the same machine are modeled by an edge.

Let us consider an example of *JSSP* composed of 3 jobs all of them having 3 operations defined in Table 1. For each job, this table gives the set of operations and for each operation the machine needed ($m_1$, $m_2$ or $m_3$) and the processing time on this machine.

| Op. / Jobs | $O_{i1}$ | $O_{i2}$ | $O_{i3}$ |
|---|---|---|---|
| i=1 | ($m_1$, 10) | ($m_2$, 35) | ($m_3$, 25) |
| i=2 | ($m_1$, 15) | ($m_3$, 16) | ($m_2$, 12) |
| i=3 | ($m_3$, 11) | ($m_1$, 12) | ($m_2$, 21) |

Table 1: Example of JSSP

The disjunctive graph illustrating this example is given in figure 1. In this graph, an arc (in full line) between two operations $(O_{i,j}, O_{i,j'})$ represents the routing constraint of this job $i$. It is valuated by the minimal distance between the start times of these two operations:

$$st_{o_{i,j}'} - st_{o_{i,j}} \geq p_{o_{i,j}} \qquad (3)$$

Each pair of disjunctive arcs is represented with a dotted edge and represents the resource constraint between two operations sharing the same resource.

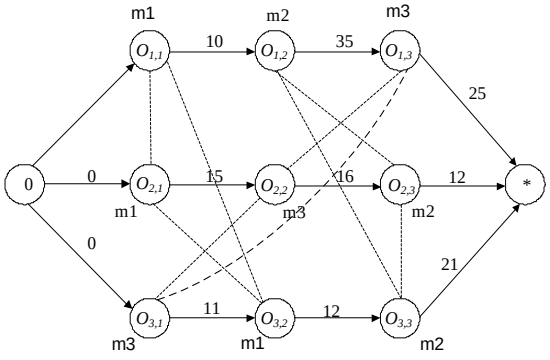


Figure 1: Disjunctive graph of the job-shop (problem modeling).

The time-lags between operations of different jobs are:

$l_{o_{1,1},o_{1,3}} = 36 \qquad L_{o_{1,1},o_{1,3}} = 45$

$l_{o_{1,1},o_{2,2}} = 20 \qquad L_{o_{1,1},o_{2,2}} = 30$

$l_{o_{2,1},o_{3,1}} = 5 \qquad L_{o_{2,1},o_{3,1}} = 20$

To model time-lags constraints on the conjunctive/disjunctive graph, we use the formulation based only on start times of operations (and the processing times of operations). Then these time-lags constraints are modelled by:

- $46 \le st_{o_{1,3}} - st_{o_{1,1}} \le 55$
- $30 \le st_{o_{2,2}} - st_{o_{1,1}} \le 40$
- $20 \le st_{O_{3,1}} - st_{o_{2,1}} \le 35$

The graph of the figure 2 represents this problem. It corresponds to the same graph given in figure 1 for the job-shop with in addition the time-lags constraints. Maximal time-lags constraints are represented by negative arc cost in the disjunctive graph from one operation to the previous one. The negative cost of the arc is equal to the duration of the previous operation plus the maximal time-lag value. The maximal time-lag between $O_{1,1}$ and $O_{1,3}$ operations of the job 1 are represented by one arc which values−55 in the non-oriented disjunctive graph.

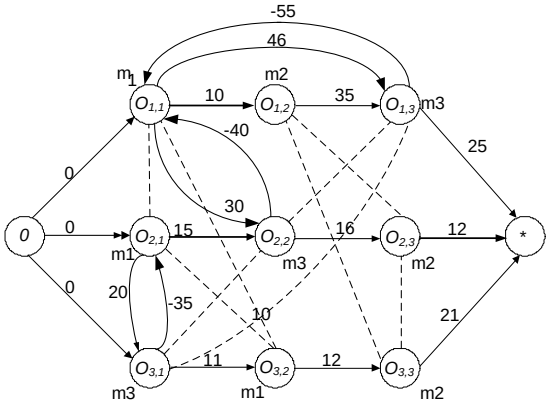


Figure 2: Disjunctive graph of the *JSPGTL* (problem modeling)

Minimal time-lags constraints are represented by arc cost in the disjunctive graph from one operation to the next one. The cost of the arc is equal to the duration of the previous operation plus the minimal time-lag value. The minimal time-lag between $O_{1,1}$ and $O_{1,3}$ operations of the job 1 are represented by one arc which values 46 in the non-oriented disjunctive graph. When no time-lags are specified (for example between $O_{1,2}$ and $O_{1,3}$), it is possible to assume, without loss of generality, to have null minimal time-lags and infinite maximal time-lags. Since there is no interest in considering infinite maximal time-lags, negative arc representing infinite maximal time-lags are ignored in graphs representation in the remainder of this article.

### *2.1.2 Solution representation*

The arcs between operations of jobs which use the same machines define the operations sequence on machines. A solution is a cycle free oriented disjunctive graph which encompassed arcs only. The arcs between operations of jobs which use the same machines define the operations sequence on machines. The graph of figure 3 and the Gantt chart of figure 4 represent a solution in which:

- on machine 1 the sequence is operation $O_{2,1}$, operation $O_{3,2}$ and operation $O_{1,1}$;
- on machine 2 the sequence is operation $O_{1,2}$, operation $O_{3,3}$ and operation $O_{2,3}$;
- on machine 3 the sequence is operation $O_{3,1}$, operation $O_{2,2}$ and operation $O_{1,3}$.

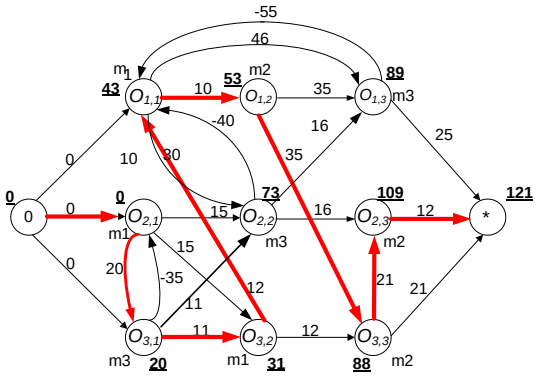


Figure 3: Fully oriented disjunctive graph of the *JSPGTL* (lines in bold give the critical path)

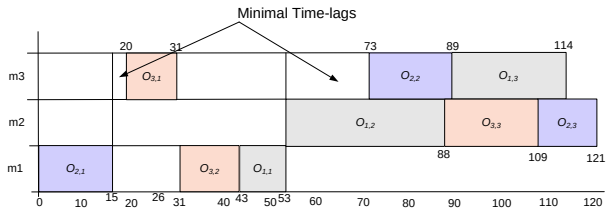


Figure 4: Gantt chart of *JSPGTL*

A strongly efficient strategy has been introduced by (Bierwith 1995) considering that one operation is linked to one job only. For the job-shop problem a Bierwith's sequence does not generate any inconsistent graph. Based on Bierwith's proposal, this solution can be represented by the following Bierwith's vector: 2 3 3 1 1 2 3 1 2 which is also called: sequence with repetition. Let us note, there exist several Bierwirth's sequences which carry out to the same oriented disjunctive graph. Note the $j^{th}$ occurrence of job $i$ represents the operation $(i; j)$. For more details concerning the use of Bierwirth's sequences for solving job-shop with time lags see (Caumond *et al.* 2008; Lacomme *et al.* 2011).

Bierwirth's sequences can be efficiently generated by any greedy algorithm or any iterative method. In this

paper these sequences are managed by heuristics proposed in the next sections.

### 2.2 Heuristic definition: Randomized_ARP_MD

Based on Deppner's proposal (Deppner, 2004) the proposed *Randomized_ARP_MD* works as follow. During the initialization step, the set of unscheduled operations $L$ is initialized to the set of all operations and the set of scheduled operations $S$ is initialized to the empty set. Note that maximal time-lags are not included in the graph during earliest starting time computation. After each evaluation, maximal time-lags are checked to state if they hold or not.

The main loop consists in scheduling of one operation and is ended when the set of unscheduled operations is empty ($L=\varnothing$) or when the maximal number of iterations is reached. The maximal number of iterations prevents excessive computational time.

The heuristic is composed of 5 parts starting with a graph $G$ which encompasses only positive arcs *i.e.* maximal time-lags are not included and where no disjunctions are solve. The heuristic mainspring is given in Algorithm 1.

Part 1 consists in ***identification of eligible operation***. An operation is stated eligible if all predecessors have been previously labeled. Let us note $T$ the set of eligible operations. Note that nodes previously investigated are tagged with the array denoted *Mem* and that the procedure *Eligible* scan only not yet investigated node (*Mem[i]=false*).

Part 2 consists in a ***random selection*** of an operation according to the earliest finishing time $EF_i$. This is achieved with a probability $p$. Let us note $i$ the operation to schedule and $m_i$ the machine required. $T_{m_i}$ is the set of all operations to schedule on the machine $m_i$.

Part 3 consists in ***looking if any eligible operation*** $j$ in $T_{m_i}$ has an earliest starting time $ES_j$ upper bounded by $EF_i$. This step is a careful search for more prior operations on the same machine using earliest starting time. Let us note $k$ the operation of interest either $i$ or $j$.

Part 4 consists in ***managing backtrack*** in the decisional tree if no operation can be identified. This is achieved by the command pop (mem:=Pop(S);) which permits to save the child node previously investigated.

Part 5 consists in ***computing the earliest starting time*** of the operation after insertion of the operation $k$. Two situations can hold. First the graph $G$ is acyclic and the current *Mem* vector is saved on the stack. The process is iterated at step 1 where a new child node will be investigated. Second, the graph is unproductive and no push is achieved. The next iteration consists in computing $T$ thanks to the Eligible procedure.

Algorithm 1. **Randomized_ARP_MD**

```
procedure name
   Randomized_ARP_MD
Input data
   Ω   : set of operations to schedule
   nm  : maximal number of iterations
Output data
   λ   : a Bierwith' sequence
   ES_i : earliest start time of operations
   EF_i : earliest finish time of operations

Local data
   S   : Stack
   Mem : array [1..n] of branching nodes
(boolean)

begin
   L := O        // unscheduled operations
   S := ∅        // scheduled operations
   step := 1;  // save initial state
   for i:=1 to n do
      Mem[i] := false;
   end do
   S.Push(Mem);
  While (S.Empty()=false) do
   S.Pop(Mem);
   Call Evaluate(G)

   // part 1
   // identification of eligible operations

   T := Eligible(L,S,Mem)
   T' := operations of E in
      increasing order of EF_i
   //  part 2 random selection of an operation
   //  according to EF_i
   i:=1; Stop := false;
   while (stop=false) do
      p:=random(100);
      if (p<80) then
         j:=i          // save position in E
         stop:=true
      else
         i:= (i mod |E|)+1
      endif
   end do

   // part 3 looking for a more prior operation

   o := E_i     // operation
   m := m_o     // machine
   j:=1; k:=-1
   while (j<=|E|) and (Stop=true) do
   begin
      oc := E_j   // current operation
      if ( j ≠ i )and ( m_oc = m ) then
       if (Mem[i]=false) then
         if ( ES_oc < EF_o ) then
```

```
            k := j ;
            EF_o := ES_oc
        endif;
      endif;
    endif;
    j := j +1
end;

  // part 4 Backtrack required

if (Stop=false) then    // backtrack
  mem:=Pop(S);
  Step:=Step-1;
Else

  // part 5 insertion of k is investigated

   Mem[k]:=true;
 m_k is the machine of operation k Compute
Prec the previous operation schedule on
machine m_k and assign -1 to Prec if not J_k
is the job of the opearation k
   if ( prec≠−1) then
   Add the disjunctive arc from prec to k
   end;
   Mem[k] := true;
   Call Evaluate(G) to obtain ES_i
   if (G is acyclic) then
     Mem[k] := true;
      S.Push(mem);
      for i:=1 to n do
         Mem[i] := false;
      end do
      λ[Step]:= J_k
   endif
endif
```

Note that the *Evaluate G* routine could not be as basic as the classical job-shop one. When dealing with time-lags, inserting an operation may require alteration of the starting times of previously scheduled operations. Thus, the evaluation of one partial schedule consists in a full run of the longest path algorithm in the disjunctive graph. During this evaluation, a positive length cycle may be detected and the evaluation procedure returns an infinite starting time.

### 2.3 A greedy Heuristic definition: Greedy_Randomized_ARP_MD

The **Randomized_ARP_MD** heuristic cannot be used for instances with up to 20 operations since it is responsible of not acceptable computational time.

The greedy version consists in achieving only one branch into the search tree representing only one decision. The greedy variant can then be re-start many times and due to the random selection several branches can be explore.

This version can be used for medium and large scale instances but provides, depending on the branch in the tree, a non-feasible solution or a solution without any guaranty in the quality.

## 3 COMPUTATIONAL EVALUATION

### 3.1 A new set of instances

To evaluate the proposed heuristic, we consider random generated instances. The benchmark is concerned with instances based on the OR-library[1] for classical shop problems (job-shop and flow-shop).

To include time-lag constraints, a dedicated program randomly generates minimal and maximal time-lags ensuring that one solution exists. Depending on the instances the numbers of time-lags vary from 3 to 13. These instances can be downloaded at:

http://www.isima.fr/~lacomme/GTL/instancesGTL.html

The framework performance is studied over experiments including both flow-shop and job-shop instances with time-lags. For each set of instances, the objective is to underlines, the capabilities of the proposed heuristics to provide new solutions for time-lags instances.

#### *3.1.1 Flow-Shop instances: characteristics*

The Carlier's instances (denoted car1 to car8) is a wide spread set of instances used in a wide majority of publications addressing flow-shop scheduling problem taken from (Carlier 1978).

#### *3.1.2 Job-Shop instances: characteristics*

The Laurence's la01 to la40 instances are wide spread instances with different size ( number of jobs x number of machines) 10x5, 15x5, 20x5, 10x10, 15x10, 20x10, 30x10, and 15x15 (Lawrence 1985).

### 3.2 Numerical experiments

The instances are large scale instances and there is no possibility to fully execute the heuristic which would be responsible of excessive computational time. The maximal number of decision nodes must be upper bounded to avoid time consuming heuristic execution by a generation of partial search tree. This maximal number of nodes during branch and bound is responsible of premature stop, prevents excessive computational time but does not guaranty that a solution is found when the method stops.

The preliminary experiments we carried out, push us into accepting that 100 000 nodes are sufficient enough. Even if the number of nodes is limited, results showed that the

[1] http://people.brunel.ac.uk/~mastjjb/jeb/orlib/files/

*Randomized_ARP_MD* cannot be used for instance with up to 20 operations to schedule due to the excessive computational time.

Considering the conclusion mentioned above we decided to promote the *Greedy_Randomized_ARP_MD* procedure. For each instance the greedy heuristic is restarted 100 000 times and the best found solutions (if a solution has been obtained) are reported in table 3 and 4.

Experiments were achieved on a Pentium IV 2.8 Ghz with 12 Go of Memory and which is about 2800 MFlops.

In the next sections the following notations are used:

*BKS* : Best known solution obtained using integer linear model (Lacomme *et al.* 2011) with one hour time limit. Asterisk denotes optimal solution.
*BFS* : Best found solution during 100 000 executions.
*tt* : Total time to achieved the 100 000 executions (if a solution has been obtained).
*nt* : The number of time-lags introduced.

### *3.2.1 Flow-Shop instances with TL*

Table 2 reports results obtained by the proposed heuristic.

| Instances | *n* | *m* | *BKS* | *nt* | *BFS* | *tt* |
|---|---|---|---|---|---|---|
| car1 | 11 | 5 | 8574* | 9 | 13 788 | 2s |
| car2 | 13 | 4 | 7777 | 7 | / | |
| car3 | 12 | 5 | 9025* | 6 | / | |
| car4 | 14 | 4 | 8787 | 8 | / | |
| car5 | 10 | 6 | 9867* | 7 | 13 597 | <1 |
| car6 | 8 | 9 | 9404* | 10 | / | |
| car7 | 7 | 7 | 8746* | 9 | 10 948 | <1 |
| car8 | 8 | 8 | 11317* | 5 | 16 130 | <1 |

Table 3: Flow-Shop instances with TL

Let us note that for instances car2, car3, car4 and car6 no feasible solution is found after 100 000 executions of the *Greedy_Randomized_ARP_MD*.

### *3.2.2 Job-Shop instances with TL*

The job-shop instances encompass 50 operations of the small instances and more than 200 operations for the larger ones.

Table 3 gives the results for the job-shop instances with time-lags. For these instances it is possible to distinguish:

- Instances for which no solution has been found;
- Solutions for which the computation time remains low (about 1 or 2 seconds);
- Instances for which computational time is greater than 100 seconds (instances la36, la38 for example).

| Instances | *n* | *m* | *BKS* | *nt* | *BFS* | *tt* |
|---|---|---|---|---|---|---|
| la01 | 10 | 5 | 666 * | 5 | 875 | 8s |
| la02 | 10 | 5 | 697 * | 5 | 897 | <1s |
| la03 | 10 | 5 | 636 * | 6 | / | |
| la04 | 10 | 5 | 713 * | 6 | / | |
| la05 | 10 | 5 | 593 * | 6 | 878 | <1s |
| la06 | 15 | 5 | 926 * | 7 | / | |
| la07 | 15 | 5 | 894 | 8 | 1123 | <1s |
| la08 | 15 | 5 | 907 * | 8 | / | |
| la09 | 15 | 5 | 951 | 8 | / | |
| la10 | 15 | 5 | 958 | 6 | / | |
| la11 | 20 | 5 | 1222 | 7 | / | |
| la12 | 20 | 5 | 1039 | 6 | 1575 | 3s |
| la13 | 20 | 5 | 1150 | 7 | / | |
| la14 | 20 | 5 | 1292 | 10 | 1584 | 6s |
| la15 | 20 | 5 | 1207 | 7 | 1593 | <1s |
| la16 | 10 | 10 | 1114 * | 8 | 1599 | <1s |
| la17 | 10 | 10 | 1091 * | 10 | 1292 | <1s |
| la18 | 10 | 10 | 1076 * | 9 | / | |
| la19 | 10 | 10 | 1050 * | 8 | 1403 | 58s |
| la20 | 10 | 10 | 1142 * | 8 | 1635 | <1s |
| la21 | 15 | 10 | 1181 | 13 | 1795 | 151s |
| la22 | 15 | 10 | 1028 | 6 | / | |
| la23 | 15 | 10 | 1054 | 8 | / | |
| la24 | 15 | 10 | 1054 | 6 | / | |
| la25 | 15 | 10 | 1069 | 8 | 1736 | 3s |
| la26 | 20 | 10 | 1306 | 11 | 1877 | <1s |
| la27 | 20 | 10 | 1408 | 7 | / | |
| la28 | 20 | 10 | 1325 | 9 | 1997 | <1s |
| la29 | 20 | 10 | 1308 | 9 | / | |
| la30 | 20 | 10 | 1395 | 8 | / | |
| la31 | 30 | 10 | 1890 | 11 | 2543 | <1s |
| la32 | 30 | 10 | 1986 | 5 | 2500 | <1s |
| la33 | 30 | 10 | 1790 | 6 | / | |
| la34 | 30 | 10 | 1962 | 5 | / | |
| la35 | 30 | 10 | 2128 | 6 | / | |
| la36 | 15 | 15 | 1350 * | 7 | 1747 | 192s |
| la37 | 15 | 15 | 1566 * | 5 | 2452 | <1s |
| la38 | 15 | 15 | 1295 | 8 | 1725 | 281s |
| la39 | 15 | 15 | 1390 | 8 | / | |
| la40 | 15 | 15 | 1320 | 6 | / | |

Table 3: Job-Shop instances with TL

## 4 CONCLUDING REMARKS AND FURTHER RESEARCH

This paper presents the first attempt to solve the job-shop with generic time-lags between some operations of different jobs. In this case, even the computation of a solution is a difficult problem.

Using the original Deppner's proposition (Deppner 2004), we introduce a two randomized heuristics. The first one *Randomized_ARP_MD* can be used

unfortunately only for small sizes instances up to 20 operations which instances are out of interest. This heuristic suffers from many backtracks. The second one *Greedy_Randomized_ARP_MD* which can be used for medium and large size problems is more promising. This heuristic is a greedy variant without backtracks and can be restarted several times to explore the search tree. As far as we know, these heuristics are the first ones for the job-shop problem with time-lags between operations of different jobs which can be used for large size problems.

To evaluate these new heuristics, we propose a new set of instances composed of 8 flow-shop instances based on Carlier's flow-shop instances and 40 job-shop instances based on the Lawrence's instances.

The results are promising. When a solution is found by *Greedy_Randomized_ARP_MD* the results are on average 37% from the solutions obtained using linear programming whit time limit of one hour.

This first study open several research issues. The one of them consists in including some propagation rules into the *Randomized_ARP_MD* which suffers from many backtracks. For instance, constraint propagation dedicated to *JSPGTL* proposed in (Lacomme et al. 2011) or generic time constraint propagation as proposed in (Artigues et al., 2011) can be used to reduce the tree search expansion. Another issue consists in studying the impact of dedicated propagation with regards to general propagations in terms of efficiency and performances to obtain a feasible solution.

Finally, our research will be directed into the definition of GRASP-ELS framework taking advantages of all previous remarks and propositions. The GRASP-ELS is a combination of the GRASP metaheuristic and the ELS metaheuristic combining the positive features of both methods. The GRASP (Greedy Randomized Adaptive Search Procedure) is a multi-start local search metaheuristic. At each iteration, an initial solution must be constructed using a Greedy_Randomized_ARP_MD. It is then improved by a local search and the best solution obtained at the end of each GRASP iteration is kept. The Evolutionary Local Search (ELS) is an extension of the Iterated Local Search (ILS). At each iteration of the ELS, several copies of the current solution are done. Each copy is modified (mutation) before being improved by a local search. The best obtained solution is kept as the new current solution. The purpose of the ELS is to better investigate the neighbourhood of the current local optimum before leaving it, while the GRASP aims at managing the diversity during the solution space exploration.
The framework we promote is a multi-start ELS in which an ELS is applied to the initial solutions generated by greedy randomized heuristics.

**REFERENCES**

Artigues, C., M.J. Huguet and P. Lopez. 2011 Generalized disjunctive constraint propagation for solving the job shop problem with time lags. *Engineering Applications of Artificial Intelligence*, 24, p. 220-231.

Bierwirth C. A., 1995. Generalized permutation approach to jobshop scheduling with genetic algorithms. *OR Spektrum*. 17, 87-92,

Brucker, P., T. Hilbig and J. Hurink, 1999. A branch and bound algorithm for a single machine scheduling with positive and negative time-lags. *Discrete Applied Mathematics*, 94, p. 77–99.

Carlier, J., 1978. Ordonnancements a contraintes disjonctives, RAIRO Recherche operationelle / Operations Research, vol.12, p. 333-351.

Caumond, A., P. Lacomme and N. Tchernev, 2008. A Memetic Algorithm for the Job-Shop with time-lags", *Computers & Operations Research*, vol. 35, p. 2331-2356.

Coelho, J. and M. Vanhoucke, 2011. Multi-mode resource-constrained project scheduling using RCPSP and SAT solvers. *European Journal of Operational Research*, 213, p. 73-82.

Deppner, F. 2004. Ordonnancement d'atelier avec contraintes temporelles entre opérations, PhD thesis in French, LORIA Nancy, France.

Dhouib, E., Teghem, J. and T. Loukil, 2012. Minimizing the number of tardy jobs in a permutation flowshop Scheduling Problem with Setup Times and Time Lags Constraints, *Journal of Mathematical Modelling and Algorithms*, DOI: 10.1007/s10852-012-9180-x.

Esquirol, P., M.J. Huguet and P. Popez, 1995. Modeling and managing disjunctions in scheduling problems", *Journal of Intelligent Manufacturing*, 6, p. 133–144.

Fondrevelle, J., Oulamaraa, A. and M.-C. Portmann, 2006. Permutation flowshop scheduling problems with maximal and minimal time lags, *Computers & Operations Research*, 33, p. 1540–1556.

Fondrevelle, J., Oulamaraa, A. and M.-C. Portmann, 2008. Permutation flowshop scheduling problems with time lags to minimize the weighted sum of machine completion times, *International Journal of Production Economics*, 112, p. 168–176

Giffler B. and J.L. Thompson, 1960. Algorithms for solving production scheduling problems, *Operations Research*, 8, p. 487-503.

Graham, R.L., E.L. Lawler, J.K. Lenstra and A.H.G. Kan Rinnooy, 1979. Optimisation and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics*, 5, pp. 236-287.

Hartmann S. and D. Briskorn 2010. A survey of variants and extensions of the resource-constrained project scheduling problem. *European Journal of Operational Research*, 207, p. 1-14.

Hurink, J. and J. Keuchel, 2001, Local search algorithms for a single-machine scheduling problem with positive and negative time-lags. *Discrete Applied Mathematics*, 112, p. 179-197.

Javadian, N., Fattahi, P., Farahmand-Mehr, M., Mehdi Amiri-Aref, M. and M. Kazemi, 2012. An immune algorithm for hybrid flow shop scheduling problem with time lags and sequence-dependent setup times, *The International Journal of Advanced Manufacturing Technology*, Doi: 10.1007/s00170-012-3911-z.

Kolicsh, R. and R. Padman, 2001. An integrated survey of deterministic project scheduling, *Omega*, 29, p. 249–272.

Lawrence, D., 1985. Job Shop Scheduling with Genetic Algorithms. First International Conference on Genetic Algorithms. Mahwah, New Jersey, p. 136-140.

Deblaere, F., E. Demeulemeester, and W. Herroelen, 2011. Reactive scheduling in the multi-mode RCPSP. *Computers & Operations Research*, 38, p. 63-74.

Lacomme, P., MJ. Huguet and N. Tchernev, 2011. Dedicated constraint propagation for Job-Shop problem with generic time-lags. *16th IEEE conference on Emerging Technologies and Factory Automation* IEEE catalog number: CFP11ETF-USB, ISBN: 978-1-4577-0016-3, *Toulouse, France.*

Neumann K., C. Schwindt and J. Zimmermann, 2002, *Project Scheduling with Time Windows and Scarce Resources*, Springer.

Roy B. and B. Sussmann, 1964. Les problèmes d'ordonnancement avec contraintes disjunctive, In : Note DS N°9 bis, SEMA, Paris, 1964.

Wikum, E. D., D.C. Llewellynnand G.L. Nemhauser, 1994. One-machine generalized precedence constrained scheduling problem, *Operations Research Letters*, 16, p. 87–99

Zhang, X. and S. de Velde, 2010, On-line two-machine open shop scheduling with time lags, *European Journal of Operational Research*, 204, p. 14–19.

.