JOB-SHOP WITH GENERIC TIME LAGS: AN HEURISTIC BASED APPROACH

Huguet Marie Josée
Lacomme Philippe1 Nikolay Tchernev1
Research Report LIMOS/RR-12-06
Juin 2012

1 Université Blaise Pascal
Laboratoire d’Informatique (LIMOS) UMR CNRS 6158,
Campus des Cèzeaux, 63177 Aubiere Cedex
placomme@isima.fr, tchernev@isima.fr

2 LAAS-CNRS
7 avenue du colonel Roche
31077 Toulouse Cedex 4
huguet@laas.fr
JOB-SHOP WITH GENERIC TIME LAGS: AN HEURISTIC BASED APPROACH

MARIE-JOSE 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

PHILIPPE LACOMME
Blaise Pascal University, Computer Science Laboratory (UMR CNRS 6158), 63177 Aubière (France)

NIKOLAY TCHERNEV
Blaise Pascal University, Computer Science Laboratory (UMR CNRS 6158), 63177 Aubière (France)
# Table of contents

**Job-Shop with generic time LAGS: an heuristic based approach**

1. **Job-Shop**
   - Problem definition: 5
   - Graph modeling: 6
   - Job-Shop Assumptions: 7
   - Job-Shop key points in efficient resolution: 7

2. **Job-Shop with time-lags**
   - Time-lags definition: 8
   - Time-lags for modelization of real-life constraints: 8
   - Graph modeling: 9
   - Previous published works on RCPSP with time-lags: 9
   - Previous published works on Job-Shop with time-lags: 9
   - Linear formulation with generic time lags: 10

3. **Heuristic based approach for JSPTL solving**
   - Bierwith vector for the JSPTL: 11
   - Heuristic definition: Randomized_ARP_MD: 11
   - Heuristic example: 12
   - A greedy Heuristic definition: Greedy_Randomized_Deppner Heuristic: 17

4. **A new set of instances**
   - Flow-Shop instances: characteristics: 17
   - Job-Shop instances: characteristics: 17

5. **Numerical experiments**
   - Flow-Shop instances with TL using the Greedy Randomized Heuristic: 18
   - Job-Shop instances with TL: 18
   - Flow-Shop instances with TL: linear resolution with CPLEX: 19
   - Job-Shop instances with TL: using CPLEX: 19

6. **Concluding remarks**

7. **References**
Index of figures

FIGURE 1. DISJUNCTIVE GRAPH OF THE JOB-SHOP (PROBLEM MODELLING) ................................................. 6
FIGURE 2. ACYCLIC CONJUNCTIVE GRAPH OF THE JOB-SHOP (SOLUTION MODELLING) ...................... 6
FIGURE 3. THE MAPPING FROM OBJECT TO SOLUTION [6] ................................................................. 7
FIGURE 4. EXAMPLE OF MAXIMAL TIME-LAGS END USE. ................................................................. 8
FIGURE 5 JSPTL EXAMPLE .................................................. 9
FIGURE 6. INITIAL GRAPH $G^+$ AFTER EVALUATION ..................................................................... 12
FIGURE 7. PART 3 WHERE OPERATION $O_{21}$ IS SCHEDULED ......................................................... 13
FIGURE 8. PART 3 WHERE OPERATION $O_{11}$ IS SCHEDULED ......................................................... 13
FIGURE 9. PART 3 WHERE OPERATION $O_{31}$ IS SCHEDULED ......................................................... 13
FIGURE 10. PART 3 WHERE OPERATION $O_{12}$ IS SCHEDULED .................................................... 14
FIGURE 11. PART 3 WHERE OPERATION $O_{13}$ IS SCHEDULED .................................................... 14
FIGURE 12. PART 3 WHERE OPERATION $O_{22}$ IS SCHEDULED .................................................... 14
FIGURE 13. FIRST DEVELOPED BRANCH IN THE SEARCH TREE .................................................. 14
FIGURE 14. SECOND BACKTRACK IN SCENARIO ........................................................................... 15
FIGURE 15. BACKTRACK IN SCENARIOS UNTIL DECISION NODE 4.1 .......................................... 15
FIGURE 16. PART 3 WHERE OPERATION $O_{12}$ IS SCHEDULED .................................................... 15
FIGURE 18. PART 3 WHERE OPERATION $O_{12}$ IS SCHEDULED .................................................... 16
FIGURE 19. PART 3 WHERE OPERATION $O_{22}$ IS SCHEDULED .................................................... 16
FIGURE 20. PART 3 WHERE OPERATION $O_{23}$ IS SCHEDULED .................................................... 16
FIGURE 21. PART 3 WHERE OPERATION $O_{23}$ IS SCHEDULED .................................................... 16
FIGURE 22. PART 3 WHERE OPERATION $O_{23}$ IS SCHEDULED .................................................... 17
JOB-SHOP WITH GENERIC TIME LAGS: A HEURISTIC BASED APPROACH

MARIE-JOSE 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 Cedex4, France

PHILIPPE LACOMME
Blaise Pascal University, Computer Science Laboratory (UMR CNRS 6158), F-63177 Aubière, France

NIKOLAY TCHERNEV
Blaise Pascal University, Computer Science Laboratory (UMR CNRS 6158), F-63177 Aubière, France

This paper deals with the job-shop scheduling problem with generic time-lags (JSPTL). This problem is a generalization of the job-shop scheduling problem where extra (minimal and maximal) delays can be introduced between any operations. In this study, the considered objective is to obtain a solution that minimizes the total duration of the schedule (or makespan). For the JSPTL with the minimization of makespan, in general, even finding a feasible solution is NP-hard.

In this report, first a state of the art on the job-shop scheduling with time lags is provided. Second we extend the ARP-MD Deppner’s heuristic providing a new randomized heuristic for Bierwith’ sequences generation. A greedy version is then introduced.

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’s instances. The numerical experiments proved that original 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 proposed greedy version of ARP-MD heuristic 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.

To conclude this work is a first step into the definition of heuristics for the JS with generic time lag 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.

1 JOB-SHOP

1.1 Problem definition

The Job-Shop Scheduling Problem (JSSP) is a well-known optimization problem often used in practical scheduling applications in the manufacturing sector. The JSSP can be formulated as follows: a set of \( n \) jobs \((inde x i = 1,2,\ldots n)\) has to be processed on a set of \( m \) machines \((index j = 1,2,\ldots m)\). Each job is fully defined by an ordered (linear) 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 JSSP 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.
Each operation $O_{i,j}$ is associated with a particular job $i$ and machine $j$ and has a duration $p_{i,j}$. The objective is to schedule each operation on the machines, taking the precedence constraints into account such that the total makespan ($C_{\text{max}}$) is minimized. According to the $\alpha/\beta$ notation introduced by [1] the problem can be represented by $J|\alpha/\beta|C_{\text{max}}$ and is known to be $\text{NP-hard}$ [2]. JSSP is widely considered as one of the most difficult problem over the last few decades. A helpful problem representation is the disjunctive graph model due to Roy and Sussmann [3].

Using the disjunctive graph model any job shop problem instance can be visualized by a directed graph $G = (V, A, E)$, where $V$ represents the set of nodes, $A$ the set of conjunctive arcs and $E$ the set of pairs of disjunctive arcs. The set of nodes contains one element for each operation $O_{i,j}$, a source node, denoted by 0 connected to the first operation of each job and a sink node, denoted by $\ast$, linked with the last operation of each job. Conjunctive arcs are used to represent the routings of the different operations of the jobs and connect each pair of consecutive operations of the same job. Each pair of disjunctive arcs connects two operations, belonging to different jobs, which are to be processed on the same machine. A feasible solution corresponds to an acyclic subgraph that contains all conjunctive arcs and that contains exactly one disjunctive arc for each pair of disjunctive arcs. An optimal solution corresponds to the feasible subgraph with the minimal makespan $C_{\text{max}}$.

### 1.2 Graph modeling

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.

<table>
<thead>
<tr>
<th>Jobs</th>
<th>$O_{i1}$</th>
<th>$O_{i2}$</th>
<th>$O_{i3}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>i=1</td>
<td>(m_1, 10)</td>
<td>(m_2, 35)</td>
<td>(m_3, 25)</td>
</tr>
<tr>
<td>i=2</td>
<td>(m_1, 15)</td>
<td>(m_3, 16)</td>
<td>(m_2, 12)</td>
</tr>
<tr>
<td>i=3</td>
<td>(m_3, 11)</td>
<td>(m_1, 12)</td>
<td>(m_2, 21)</td>
</tr>
</tbody>
</table>

**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 successive operations $(O_{i,j}, O_{i,j+1})$ represents the routing constraint of this job $i$. It is valued by the minimal distance between the start times of these two operations: $st_{O_{i,j+1}} - st_{O_{i,j}} \geq p_{O_{i,j}}$.

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

![Disjunctive graph of the job-shop (problem modelling).](image1)

The acyclic conjunctive graph of the solution modelling is given in figure 2. In this graph, all pairs of disjunctive arcs are reduced to one arc and represent the sequencing of operations processed on the same machine.

![Acyclic conjunctive graph of the job-shop (solution modelling).](image2)

A comprehensive survey of scheduling techniques can be found in, for example, [4] and [5]. Due to the complexity of the problem, a wide number of approaches are based on heuristics and meta-heuristics and provide reasonable good rather than optimal solutions within reasonable computational effort.
1.3 Job-Shop Assumptions

The standard job-shop scheduling problem makes the following classical assumptions:

- Each job consists of a finite number of operations.
- The processing time for each operation using a particular machine is defined.
- There is a sequence of operations that has to be achieved to complete each job.
- Delivery times of the products are undefined.
- There is no setup or tardiness cost.
- A machine can process only one job at a time.
- Each job is performed on each machine only once.
- The system cannot be interrupted until each operation of each job is finished.
- Preemption is not allowed.

1.4 Job-Shop key points in efficient resolution

The key points are the following:

- A **Quasi-Direct Representation** of Solution
- An **efficient local search** taking advantages of the longest path analysis

The approaches must guarantee an efficient exploration of the solution search space avoiding premature convergence and trying to favor non visited search space area (detection of clones, memory management avoiding analogous curves in the search space…).

The importance of definition of an ad hoc **quasi direct representation** of solution has been highlighted for years. Let us note the sequel publication of [6] where authors clearly defined that a quasi-direct representation permits to have: (1) a coding space and (2) a solution space. According to [6] an efficient mapping function must assign to any object of the coding space a solution i.e. must satisfy the feasibility constraint. Authors also note that there are three types of mapping function (Figure 3):

- 1-to-1 mapping function where one and only one object of the coding space is linked to one and only one solution;
- n-to-1 mapping function where several objects in the coding space can be linked to the same solution;
- 1-to-n mapping function where one object in the coding space can be linked to several solutions.

Whatever the mapping function, a job-shop solution is a fully oriented disjunctive graph [3] where a longest path computation permits to obtain the earliest starting time of operations defining a semi-active solution.

![Diagram](image)

Figure 3. The mapping from object to solution [6]

Numerous proposals have been introduced during the last decade including the Bierwith’s proposition which takes advantages of a vector with repetition where each job number appear several time for each operation of the sequence. This vector permits to define a n-to-1 mapping function [7]. The main advantages of Bierwith’s proposal concern the local search which could be achieved not in the solution space but in the coding space by definition of basic but strongly efficient modification of the object in the coding space.

Efficient **local searches** take advantages of the longest path defining strongly efficient modification of the graph. Propositions include (but are not limited to):

- the neighborhood system of [8] which concern to consecutive operation on the longest path;
- the neighborhood system of [9] who define a machine operation block;
- the neighborhood system of [10] who propose permutation of operation at the extremities of the Grabowski’s machine block.

The next section is devoted to the job-shop with time-lags which is an extension of the classical job-shop. The job-shop with time-lags resolution implies of course, a careful definition of the coding space, the solution space, efficient local search, and global search space framework. The next section is dedicated to the JS with time-lags definition, the presentation of the disjunctive graph and Deppner’s heuristic.
2 JOB-SHOP WITH TIME-LAGS

The job-shop problem with minimum and maximum time-lags (JSPTL) is a generalization of the JSSP, in which there are time relations between the starting times of two successive operations belonging to any two jobs. In this section is first formalized the general time lag constraints and then related literature review is given.

2.1 Time-lags definition [11]

A time-lag can be define between the finish time of a given operation \( O_{i,j} \) (denoted by \( ft_{i,j} \)) and the start time of another operation \( O_{i,j'} \) (denoted by \( st_{i,j'} \)):

\[
I_{o_{i,j}, o_{i,j'}} \leq st_{i,j'} - ft_{i,j} \leq L_{o_{i,j}, o_{i,j'}}
\]

with \( L_{o_{i,j}, o_{i,j'}} \geq I_{o_{i,j}, o_{i,j'}} \).

In this formula: \( I_{o_{i,j}, o_{i,j'}} \) represents the minimal time-lag and \( L_{o_{i,j}, o_{i,j'}} \) is the maximal time-lag. \( O_{i,j} \). The first part of this formula \( I_{o_{i,j}, o_{i,j'}} \leq st_{i,j'} - ft_{i,j} \) means that \( O_{i,j'} \) cannot start before at least \( I_{o_{i,j}, o_{i,j'}} \) units after the end of \( O_{i,j} \).

The second part of the formula \( st_{i,j'} - ft_{i,j} \leq L_{o_{i,j}, o_{i,j'}} \) 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 [11] the JSPTL without preemption and when processing times are fixed, time-lags constraints can be formulated with only “start-start” relations by using

\[
st_{i,j'} = ft_{i,j} + p_{i,j}.
\]

Then time-lags can be define by the formula

\[
I_{o_{i,j}, o_{i,j'}} + p_{i,j} \leq st_{i,j'} - ft_{i,j} \leq L_{o_{i,j}, o_{i,j'}} + p_{i,j}.
\]

The relation presented above is very general time-lags constraints. 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 [12]. Moreover, even finding a feasible solution is a NP-complete problem.

According to the \( d[t]\) notation introduced by [11] the problem can be represented by \( J\{i, gi, fi\}, C_{max} \). [11].

As stressed by Brucker [11], scheduling problems with general time-lags have mainly been discussed in connection with project scheduling [13], [14].

2.2 Time-lags for modelization of real-life constraints

The time-lags constraints find a broad spectrum of industrial and design applications whenever uncertainty about durations is an issue and there deadlines to be mandatory met. The use of time lags in scheduling problems permits to model general timing relations between jobs like: start-start relations between jobs; starting interval of a job; release times and deadlines of jobs. Positive and negative time-lags are also used to describe timing restrictions for industrial production processes (e.g. in chemical production or the steel industry). Let \( O_{i,j} \) be the \( j^{th} \) operation of job \( i \), a minimal time lag between two operations can be used to model a transportation time from operation \( O_{i,j} \) to \( O_{i,j'} \) since a time lag max models a maximum delay of resource utilization. Let us consider a working-man required for two operations \( O_{i,j} \) and \( O_{i,j'} \) with durations \( d_{i,j} \) and \( d_{i,j'} \). If the labour regulation limits the working time to \( d_{max} \) its imply that the maximal amount of time between operation \( i \) and \( j \) (i.e. maximal time lag) is upper bounded by \( d_{max} - (d_{i,j} + d_{i,j}') \). Such situation is highlighted in Figure 4.

![Figure 4. Example of maximal time-lags end use.](image-url)
Job-Shop with Generic time-lags

2.3 Graph modeling

Let us consider the same example of JSPP (made of 3 jobs and 3 machines) given in table 1 in which we add 3 time-lags constraints:

- $36 \leq s_{O,3} - f_{O,1} \leq 45$
- $20 \leq s_{O,2} - f_{O,1} \leq 30$
- $5 \leq s_{O,1} - f_{O,2} \leq 20$

To model time-lags constraints on the conjunctive/disjunctive graphe, 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:

- $46 \leq s_{O,3} - s_{O,1} \leq 55$
- $30 \leq s_{O,2} - s_{O,1} \leq 40$
- $20 \leq s_{O,1} - f_{O,2} \leq 35$

The graph of the Figure 5 represents this problem. It corresponds to the same graph given in figure 2 for the JSP with in addition the time-lags constraints.

Figure 5 JSPTL example

2.4 Previous published works on RCPSP with time-lags

In the resource-constrained project scheduling problem (RCPSP) activities (machine operation) are interrelated by two kinds of constraints: precedence constraints and limitation of resources available to achievement of operations. 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. 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 [15], reactive scheduling in multi modes [16] since methods are for numerous proposals, based on heuristic and meta-heuristics. See [17] for a survey of RCPSP variants previously addressed in publications.

2.5 Previous published works on Job-Shop with time-lags

Since the classical job-shop problem (JSSP) is a well-addressed problem, in the literature only few articles are concerned with time-lag constraints.

The single machine problems with time-lags constraints were first addressed by Wikum et al. [12]. They have state that some particular single-machine problems with time-lags are polynomially solvable, even if the general case is NP-hard. Brucker et al. [11] 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 [18].

Caumond et al. [19] focused on the JSPTL with minimal and maximal time-lags between successive operations of the same job. Contrary to problems with time-lags between any operations, finding a feasible solution is polynomial problem. It can be done by scheduling consecutively all of the jobs but it leads to worst quality solutions (for the makespan minimization). But this solution may initialize an iterative search process. These solutions are denoted canonical solutions by [19]. The authors proposed an heuristic for finding a feasible solution with better quality. This heuristic considers sequentially the operations of the jobs and tries to extend a partial schedule with a new operation. When an unfeasibility arises, a backtrack process is done. Moreover, the authors promoted a memetic based approach taking advantages of heuristics for initial solutions generation adding operation iteratively, a powerful local search based on the critical path analysis and on a neighbouring generation system. Note that authors validated
their approach solving both flow-shop, job-shop, no-wait and instances with time-lags.

Artigues et al. [20] consider the same problem (JSPTL with time-lags between consecutive operations). They introduced powerful heuristic and generalized resource constraint propagation mechanisms. Their heuristic considers sequentially the jobs of the problem and try to add all the operations of the selected job in the partial schedule. As previously, when an unfeasibility arises, some backtrack mechanisms are used.

The authors developed a branch-and-bound method including their heuristic and their generalized constraint propagation. This approach is more efficient than the memetic algorithm of Caumond et al. [19] especially for small size instances. The propagation mechanisms used are based on the disjunctive time-bound-on-node (TBON) graph introduced by [21]. The disjunctive TBON representation is a generalization of the disjunctive graph in which there are two nodes per operation (one node for the start time and one node for the finish-time). It allows then the distinct visualization of the different constraints of the problem: duration, time-lags, earliest and latest start and finish times, although it yields a larger number of nodes.

In 2004, in his thesis, Deppner [22] considered job problems with time-lags constraints. Minimal and maximal time-lags between every pair of operations are tackled. He proposed several heuristics for these problems, neighborhood search and memetic algorithm. This works considers general shop problems with time-lags but focuses on flow-shop with time-lags.

### 2.6 Linear formulation with generic time lags [21]

The model uses one type of continuous decision variables \( st_o \) and one type of binary variables \( a_{o,o'} \). The continuous variables \( st_o \) are related to the starting times of operations \( o \) and the binary variables \( a_{o,o'} \) are used to represent disjunction between two operations sharing the same machine.

Notations:

- \( M \) set of machines
- \( \Omega \) set of operations to schedule

The objective consists in minimizing \( Z \) the completion time of the last operation. Constraints (1) (2) represent machine disjunctions. Let us note that as regards Manne’s model, two extra constraints (constraints (3) and (4)) have been added. These constraints (3) and (4) represent time-lags constraints.

If no minimal time-lags exist between \( o \) and \( o' \), one can assume that \( t_{o,o'}^{\min} = 0 \) and \( t_{o,o'}^{\max} \) are a large positive values.

\[
\begin{align*}
Z & \quad \text{completion time of the last operation} \\
E_k & \quad \text{set of operations processed on machine } k \\
p_o & \quad \text{processing time of operation } o \\
t_{o,o'}^{\min} & \quad \text{minimal time-lag between } o \text{ and } o' \\
t_{o,o'}^{\max} & \quad \text{maximal time-lag between } o \text{ and } o' \\
H & \quad \text{a large positive number} \\
st_o & \quad \text{starting time of operation } o \\
a_{o,o'} & = \begin{cases} 
1 & \text{if operation } o' \text{ is processed before } o \\
0 & \text{otherwise}
\end{cases}
\end{align*}
\]
3 HEURISTIC BASED APPROACH FOR JSPTL SOLVING

3.1 Bierwirth vector for the JSPTL

Since the final objective consists in defining a global iterative framework for the JSPTL, we propose to update the Deppner’s proposal introduced hereafter, including a Bierwirth’s vector design.

3.2 Heuristic definition: Randomized_ARP_MD

The first heuristic proposed by Deppner is a priority dispatching rules generation based on a similar approach like the Giffler and Thomson’s algorithm [22]. 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 paragraph, we will propose an improvement of this heuristic.

The Deppner’ heuristic 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 ($L = \emptyset$) 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 solved.

**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 $Eligible$ can 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. $E_{m_i}$ is the set of all operations to schedule on the machine $m_i$.

**Part 3** consists in looking if any eligible operation $j$ in $E_{m_i}$ 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_Deppner Heuristic

```
procedure name
Randomized_ARP_MD

Input data
\[ \Omega \]: set of operations to schedule
\[ nm \]: maximal number of iterations

Output data
\[ \lambda \]: 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
\[ L := \Omega \] // unscheduled operations
\[ S := \emptyset \] // scheduled operations
\[ step := 1 \] // save initial state
for $i := 1$ to $n$ do
\[ Mem[i] := false; \]
end do
S.Push(Mem);
Call Evaluate($G$)
```
While (S.Empty()=false) do
  S.Pop(Mem);
  // part 1
  // identification of eligible operations
  T := Eligible(L, S, Mem)
  T' := operations of E in increasing order of EF
  // part 2
  // random selection of an operation
  // according to EF
  i:=1; Stop := false;
  while (stop=false) and (|T|>0) 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:=T_i // operation
  m:=m_o // machine
  j:=1; k:=i
  while (j<|E|) and (Stop=true) do
    oc:=E_j // current operation
    if (j!=i) and (m_oc = m) then
      if (ES_oc < EF_i) then
        k:=j;
        EF_k := ES_oc
      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
  endif
end;
Call Evaluate(G) to obtain ES
if (G is acyclic) then
  Mem[k] := true;
  S.Push(mem);
  for i:=1 to n do
    Mem[i] := false;
  end do
end
j[Step]:=J_k
end

3.3 Heuristic example

Job-Shop disjunctive graph for problem modeling

Let us restart with the example of JSPTL illustrated in figure 5.

First the graph G^+ which encompasses only conjunctive arcs is evaluated to obtain the earliest starting time of operations. This graph first evaluated permits to obtain the earliest starting time of operation as stressed in Figure 6.

Figure 6. Initial graph G^+ after evaluation

At the beginning the set of unscheduled operations L encompasses all operations since T, the set of eligible operations, is composed of only operations O_{11} and O_{21} which are operations with all predecessor previously labeled.

The current state is:
T = \{(1:1)(2:1)\}

The earliest finished time is 15 for O_{21} and 10 for O_{11}.  
First Branch:

Assume that operation $O_{21}$ is randomly selected at step (2) of the algorithm. This operation is concerned by machine $m_1$. Since there is no operation in $T$ which required this machine, the part 3 of the algorithm is unprofitable. Operation $O_{21}$ is scheduled and the earliest starting time of both $O_{21}$ and the successors including the next operation of the job and the minimal time lag operation are computed (see Figure 7).

![Figure 7. Part 3 where operation $O_{21}$ is scheduled](image1)

In the next iteration, one has to address:

$L = \{(1;1),(1;2),(1;3),(2;2),(2;3),(3;1),(3;2),(3;3)\}$

$T = \{(1;1),(3;1)\}$

The partial Bierwith’s vector under construction is composed of the number 2 in the first position:

$$m_1 \begin{array}{cccccc} \text{2} & \text{1} & \text{1} & \text{1} & \text{1} & \text{1} \end{array}$$

Only operations $O_{11}$ and $O_{31}$ can be schedule and $O_{11}$ is randomly selected. This operation is the second operation scheduled on the machine $m_1$ and its earliest starting time is 15. The successor $O_{12}$ and $O_{22}$ are labeled to 25 and 45 as stressed in Figure 8.

The partial Bierwith’s vector under construction is composed of the number 2 in the second position:

$$m_1 \begin{array}{cccccc} \text{2} & \text{1} & \text{1} & \text{1} & \text{1} & \text{1} \end{array}$$

Next $T = \{(3;1),(1;2),(2;2)\}$ and suppose that the scheduled operation is $O_{31}$ which leads to the graph of the Figure 9.

Note that the maximal time lag which value 35 from $O_{21}$ to $O_{31}$ holds.

![Figure 8. Part 3 where operation $O_{11}$ is scheduled](image2)

![Figure 9. Part 3 where operation $O_{31}$ is scheduled](image3)

The partial Bierwith’s vector under construction is composed of the number 3 in the second position:

$$m_1 \begin{array}{cccccc} \text{2} & \text{1} & \text{3} & \text{1} & \text{1} & \text{1} \end{array}$$

In the next iteration, one has to address:

$T = \{(1;2),(2;2),(3;2)\}$

$O_{12}$ is randomly selected leading to the graph of Figure 10.
The partial Bierwirth’s vector under construction is composed of the number 3 in the second position:

$$m_1 \ m_1 \ m_3 \ 2 \ 1 \ 3 \ 1$$

In the next iteration, one has to address: $T = \langle 2,2 \rangle \langle 3,2 \rangle \langle 1,3 \rangle$. $O_{13}$ is randomly selected leading to the graph of Figure 11. Since $O_{31}$ has been previously scheduled on machine $m_3$, a disjunctive arc is added from $O_{31}$ to $O_{13}$ as stressed in the graph of Figure 11.

In the next iteration, one has to address: $T = \langle 2,2 \rangle \langle 3,2 \rangle \langle 1,3 \rangle$. $O_{13}$ is the operation previously scheduled on the machine $m_3$, leading to the earliest starting time $ES_{22} = 85$.

Figure 10. Part 3 where operation $O_{12}$ is scheduled

Figure 12. Part 3 where operation $O_{22}$ is scheduled

Note that the maximal time lag which value 40 from $O_{14}$ to $O_{22}$ does not hold. Such situation required a backtrack since the scenario of decision leads to an unprofitable graph where time lag constraints cannot be achieved.

In the search tree, previous taken decisions develop a branch which leads to an unfeasibility where a maximal time lag does not hold. This scenario is introduced in the Figure 13 where selected operations are represented by red arcs and where nodes are labeled by the decision level. At node (6.1) there is some unfeasibility.

Figure 11. Part 3 where operation $O_{13}$ is scheduled

Figure 13. First developed branch in the search tree

14
At the decision node (4.1), the current graph is the graph of Figure 16 with $T = (2;2), (3;2), (1;3)$. Operation $O_{13}$ is removed from $T$ and $T = (2;2), (3;2)$.

The current state addresses $T = (2;2), (3;2), (1;3)$ where $O_{13}$ has been previously investigated. So $T = (2;2), (3;2)$ and $O_{22}$ is randomly selected leading to the graph of Figure 16. A new disjunctive arc is added from $O_{31}$ to $O_{22}$ leading to the graph of Figure 17. Note the maximal time lag of operation $O_{22}$ holds.

Figure 17. Part 3 where operation $O_{32}$ is scheduled

The current Bierwith’ sequence is as follow:

<table>
<thead>
<tr>
<th>m1</th>
<th>m1</th>
<th>m3</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

m2

Figure 16. Part 3 where operation $O_{12}$ is scheduled

The partial Bierwith’s vector under construction is composed of four operation previously schedule.

<table>
<thead>
<tr>
<th>m1</th>
<th>m1</th>
<th>m3</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

m2

This second backtrack leads to scheduling $O_{33}$ which induces $T = (2,2)$ at the decision node number 7.2. Since $O_{22}$ cannot be scheduled the process consists in backtracking until 6.2, 5.1 and lately 4.1 as stressed on the Figure 15.

Figure 15. Backtrack in scenarios until decision node 4.1

Decision (6.1) is removed and the method go back to the node (5.1) with $T = (2;2), (3;2)$ and then choose operation $O_{32}$ which leads to node (6.2) leading to $T = (2;2), (3;3)$. The next decision leads to schedule $O_{22}$ which is not possible as regards the maximal time lag of $O_{22}$. Then a second backtrack is applied and go back to $T = (2;2), (3;3)$ and choose to schedule $O_{32}$ as illustrated in Figure 14.

Figure 14. Second backtrack in scenario
The set of eligible operation is \( T = \{(3;2),(1;3),(2;3)\} \). \( O_{12} \) is randomly selected and scheduled inducing a new disjunctive arc from \( O_{11} \) to \( O_{12} \) as stressed on Figure 18 and leading to the following vector:

\[
\begin{array}{cccc}
\text{m1} & \text{m1} & \text{m3} & \text{m3} \\
2 & 1 & 3 & 1 \\
\end{array}
\]

Figure 18. Part 3 where operation \( O_{12} \) is scheduled

The set of eligible operation is \( T = \{(1;3),(2;3),(3;3)\} \). \( O_{13} \) is randomly selected and scheduled inducing a new disjunctive arc from \( O_{22} \) to \( O_{13} \) as stressed on Figure 19. Part 3 where operation \( O_{22} \) is scheduled and leading to the following vector:

\[
\begin{array}{cccc}
\text{m1} & \text{m1} & \text{m3} & \text{m3} & \text{m3} \\
2 & 1 & 3 & 1 & 2 \\
\end{array}
\]

Figure 19. Part 3 where operation \( O_{22} \) is scheduled

The last decision consists in scheduling \( O_{13} \) as stressed on Figure 20 leading to the following vector:

\[
\begin{array}{cccc}
\text{m1} & \text{m1} & \text{m3} & \text{m3} & \text{m3} \\
2 & 1 & 3 & 1 & 2 \\
\end{array}
\]

Figure 20. Part 3 where operation \( O_{23} \) is scheduled

Since a feasible solution has been generated, the heuristic ended.

Part of the search tree which lead to a solution

Figure 22 gives the branch which leads to a solution. To each decision point, the scheduling decision is on the arc and the set \( T \) gives the set of eligible operations.
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

### 4.1 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 [23].

Table 1 and 2 gives size instances and use the following notations:

- \(n\) : Number of jobs
- \(m\) : Number of machines
- \(BKS\) : Best known solution found with the by competitive meta-heuristic search procedures or linear programming from literature

Note that \(BKS\) is the best known solution i.e. a lower bound of the problem of interest.

#### Table 2. Flow-Shop instances

<table>
<thead>
<tr>
<th>Instances</th>
<th>(n)</th>
<th>(m)</th>
<th>(BKS)</th>
</tr>
</thead>
<tbody>
<tr>
<td>car1</td>
<td>11</td>
<td>5</td>
<td>7038</td>
</tr>
<tr>
<td>car2</td>
<td>13</td>
<td>4</td>
<td>7166</td>
</tr>
<tr>
<td>car3</td>
<td>12</td>
<td>5</td>
<td>7312</td>
</tr>
<tr>
<td>car4</td>
<td>14</td>
<td>4</td>
<td>8003</td>
</tr>
<tr>
<td>car5</td>
<td>10</td>
<td>6</td>
<td>7702</td>
</tr>
<tr>
<td>car6</td>
<td>8</td>
<td>9</td>
<td>8313</td>
</tr>
<tr>
<td>car7</td>
<td>7</td>
<td>7</td>
<td>6558</td>
</tr>
<tr>
<td>car8</td>
<td>8</td>
<td>8</td>
<td>8264</td>
</tr>
</tbody>
</table>

### 4.2 Job-Shop instances: characteristics

The Laurence’s la01 to la40 instances are wide spread instances with different size as detailed in Table 3 (10x5, 15x5, 20x5, 10x10,15x10,20,10,30x10, et 15x15) **Erreur**!

Source du renvoi introuvable..

---

\(1\) \url{http://people.brunel.ac.uk/~mastijb/elib/orlib/files/}
5 NUMERICAL EXPERIMENTS

Trying to evaluate the performances of the heuristic we have achieved 25 executions to determine the best solution. Let us note that 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 experiment we carried out push us into accepting that 100 000 nodes are sufficient enough. But with this limit, the randomize Deppner heuristic cannot be used for instance with up to 20 operations to schedule. So we promote the Greedy_Randomized_Deppner heuristic procedure which is executed 100 000 times.

Experiments were achieved on a Pentium IV 2.8 Ghz with 12 Mo of Memory and which is about 14 000 MFlops under Windows 7 / 64 bits

5.1 Flow-Shop instances with TL using the Greedy Randomized Heuristic

Table 4 report results obtained by the proposed heuristic. In this table:

- nt : The number of time lags introduced
- LB : Lower Bound (LB is the optimal solution of the flow-shop instances without time-lag)
- BFS : Best found solution during 100 000 executions
- tt : Total time to achieved the 100 000 executions (if a solution has been obtained)

Table 4. Flow-Shop instances with TL

<table>
<thead>
<tr>
<th>Instances</th>
<th>n</th>
<th>m</th>
<th>LB</th>
<th>nt</th>
<th>BFS</th>
</tr>
</thead>
<tbody>
<tr>
<td>car1</td>
<td>11</td>
<td>5</td>
<td>7038</td>
<td>9</td>
<td>13</td>
</tr>
<tr>
<td>car2</td>
<td>13</td>
<td>4</td>
<td>7166</td>
<td>7</td>
<td>/</td>
</tr>
<tr>
<td>car3</td>
<td>12</td>
<td>5</td>
<td>7312</td>
<td>6</td>
<td>/</td>
</tr>
<tr>
<td>car4</td>
<td>14</td>
<td>4</td>
<td>8003</td>
<td>8</td>
<td>/</td>
</tr>
<tr>
<td>car5</td>
<td>10</td>
<td>6</td>
<td>7702</td>
<td>7</td>
<td>13</td>
</tr>
<tr>
<td>car6</td>
<td>8</td>
<td>9</td>
<td>8313</td>
<td>10</td>
<td>/</td>
</tr>
<tr>
<td>car7</td>
<td>7</td>
<td>7</td>
<td>6558</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>car8</td>
<td>8</td>
<td>8</td>
<td>8264</td>
<td>5</td>
<td>16</td>
</tr>
</tbody>
</table>

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

5.2 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 5 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).
The next sections present results obtained using the linear programming model presented in section 2.6. All experiments were carried out using CPLEX 12 solver with a time limit upper bounded by 3600s. The default CPLEX configuration is used for all parameters including the mip strategy branch.

### 5.3 Flow-Shop instances with TL: linear resolution with CPLEX

Table 6 gives results of the linear model. Columns PL sol. is the best found solution and Time is the total computational time (in seconds). Asterisks denote proven optima. Note that all instances are solved to optimality except Car02 and Car04.

### 5.4 Job-Shop instances with TL: using CPLEX

Table 7 reports the results for the job-shop instances with time-lags. For 14 instances the optimal solutions were obtained in acceptable time less 200 seconds.

6 CONCLUDING REMARKS

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

First using the original Deppner’s proposition, we introduce a new randomized heuristics. We propose a greedy variant of the Deppner heuristic which can be restarted several time to explore the search tree.

To evaluate this new heuristic, 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.
Moreover, some of the optimal solution can be found with Cplex.

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 [21] or generic time constraint propagation as proposed in [20] 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.

7 REFERENCES


