Preprints of the 13th IFAC Symposium on Information Control Problems in Manufacturing, Moscow, Russia, June 3 - 5, 2009

An Exact Method for the Best Case in a Group Sequence: Application to a General Shop Problem

Guillaume Pinot Nasser Mebarki

IRCCyN — UMR CNRS 6597, 1 rue de la Noé
BP 92101, 44321 Nantes Cedex 3, France
(e-mail: firstname.lastname@irccyn.ec-nantes.fr)

Abstract: Group sequencing is a well-studied scheduling method for the job shop problem. The goal of this method is to have a sequential flexibility during the execution of the schedule and to guarantee a minimal quality corresponding to the worst case. But the best case quality of a group sequence should also be interesting. This article presents a new method to evaluate the best-case quality. This method is a branch and bound algorithm to find the optimal solution for any regular objective. Experiments show the efficiency and the limits of the exact method.

Keywords: Scheduling, Flexibility, Branch and bound, Combinatorial optimization, Flexible manufacturing systems.

1. INTRODUCTION

The job shop problem with multiple precedence constraints ($J|r_i, \text{prec}|f$ from the classification described in Graham et al. (1979)) is an optimization problem composed of resources, operations, and constraints. Operations (O_i) are executed on resources (m_i denotes the resource M_ℓ used for operation O_i) during a processing time p_i with precedence constraints (predecessors (resp. successors) of O_i are given by $\Gamma^-(i)$ (resp. $\Gamma^+(i)$)). A resource, also named machine, can execute only one operation at a time. An operation O_i has a release date r_i, its starting time is denoted by t_i and its completion time is denoted by C_i. Generally, the job shop problem uses a regular objective function f that is an increasing function of the C_i. The goal is to minimize this objective function. The makespan, denoted by $C_{\max}$, calculated $\max_i C_i$, that corresponds to the total time of execution of the schedule, is a classical regular objective.

Actually, manufacturing problems are not deterministic. This is why group sequencing was created by Erschler and Roubellat (1989). This method aims at solving the job-shop problem by proposing not only one schedule but a set of different schedules in order to delay decisions to take into account uncertainties. This set of schedules is presented through groups of permutable operations. Group sequencing evaluates a group sequence according to the worst case quality in the set of feasible schedules.

The best case quality of a group sequence can also be interesting for different reasons. It gives information on the schedule before the execution: for a given group sequence, with an evaluation of the best case quality and the worst case quality, it will give us the minimal and the maximal quality of the final schedule. It could also be helpful to evaluate a decision during the execution of the schedule: it will allow to know, during the execution of the schedule, if there is at least one schedule in the group sequence that has no late jobs.

In this article, we present a new exact method to evaluate the best case quality of a group sequence. First, we will present group sequencing. Second, we will present lower bounds for the best-case of a group sequence. Third, we will present a branch and bound method to find the optimal solution according to a regular objective function in a group sequence. This exact method uses the lower bounds previously introduced. Finally, we will present experiments of this method on well-known instances of the job shop problem.

2. GROUPS OF PERMUTABLE OPERATIONS

Group of permutable operations was first introduced in Erschler and Roubellat (1989). The goal of this method is to provide a sequential flexibility during the execution of the schedule and to guarantee a minimal quality corresponding to the worst case. This method has been developed in the last twenty years, in particular in Erschler and Roubellat (1989); Billaut and Roubellat (1996); Wu et al. (1999); Artigues et al. (2005). For a theoretical description of the method, see Artigues et al. (2005).

In the rest of this article, the index i will be used for the operations and the index ℓ will be used for the resources.

A group of permutable operations is a set of operations to be performed on a given resource M_ℓ in an arbitrary order. It is named $g_{\ell,k}$. The group containing the operation O_i is denoted by $g(i)$. The index k will be used for the sequence number of a group.

A group sequence is defined as a sequence of v_ℓ groups (of permutable operations) on each machine M_ℓ: $g_{\ell,1}, \ldots, g_{\ell,v_\ell}$, performed in this particular order. On a given machine, the group after (resp. before) $g(i)$ is denoted by $g^+(i)$ (resp. $g^-(i)$).

Table 1. A job shop problem

i	1	2	3	4	5	6	7	8	9
$\Gamma^-(i)$	$\varnothing$	{1}	{2}	$\varnothing$	{4}	{5}	$\varnothing$	{7}	{8}
m_i	M_1	M_2	M_3	M_2	M_3	M_1	M_3	M_1	M_2
p_i	3	3	3	4	3	1	2	2	2

Fig. 1. A group sequence solving the problem describe in Tab. 1

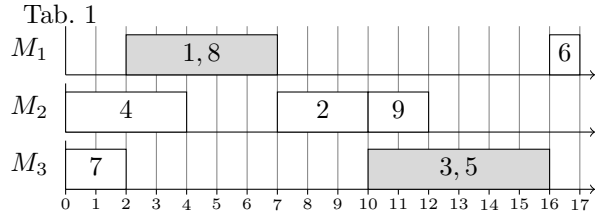
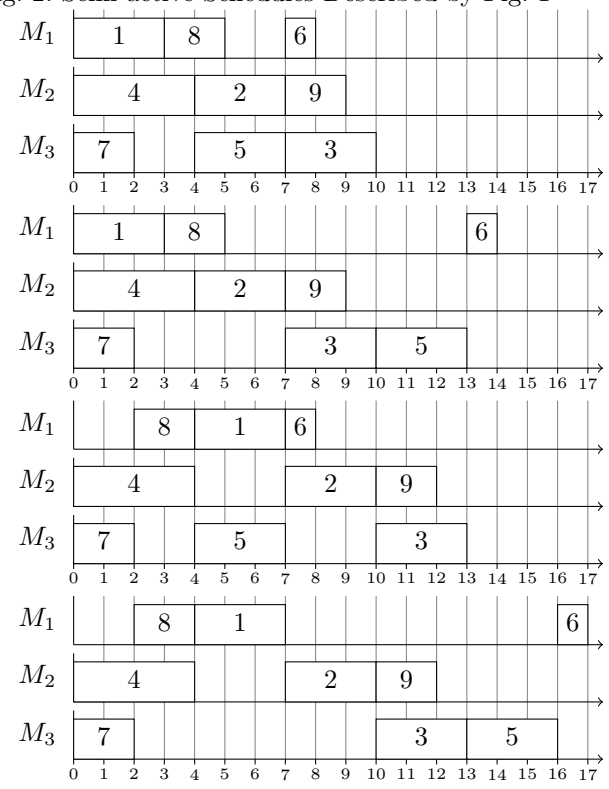


Fig. 2. Semi-active Schedules Described by Fig. 1



A group sequence is feasible if for each group, all the permutations among all the operations of the same group gives a feasible schedule (*i.e.* a schedule which satisfies all the constraints of the problem). As a matter of fact, a group sequence describes a set of valid schedules, without enumerating them.

The quality of a group sequence is expressed in the same way as that a classical schedule. It is measured as the quality of the worst semi-active schedule [1] found in the group sequence, as defined in Artigues et al. (2005).

To illustrate these definitions, let us study an example.

Tab. 1 presents a job shop problem with three machines and three jobs, while Fig. 1 presents a feasible group sequence solving this problem. This group sequence is made of seven groups: two groups of two operations and five groups of one operation. This group sequence describes four different semi-active schedules shown in Fig. 2. Note that these schedules do not always have the same makespan: the best case quality is with $C_{\max} = 10$ and the worst case quality is with $C_{\max} = 17$.

The execution of a group sequence consists in choosing a particular schedule among the different possibilities described by the group sequence. It can be viewed as a sequence of decisions: each decision consists in choosing an operation to execute in a group when this group is composed of two or more operations. For instance, for the group sequence described on Fig. 1, two decisions have to be done: on M_1, at the beginning of the scheduling, either operation O_1 or O_8 has to be executed. Let us suppose the decision taken is to schedule O_1 first. On M_3, after the execution of O_7, there is another decision: scheduling operation O_3 or O_5 first. If the decision is to execute O_5 first, the first schedule of Fig. 2 is realized.

Group sequencing has an interesting property: the quality of a group sequence in the worst case can be computed in polynomial time for minmax regular objective functions like makespan (see Artigues et al. (2005) for the description of the algorithm). Thus, it is possible to compute the worst case quality for large scheduling problems. Consequently, this method can be used to compute the worst case quality in real time during the execution of the schedule. Due to this property, it is possible to use group sequencing in a decision support system in real time during the execution of the scheduling process.

This method enables the description of a set of schedules in an implicit manner (*i.e.* without enumerating the schedules) and guarantees a minimal performance. Actually, as it proposes a group of permutable operations, one can choose inside a group the operation that best fits the real state of the system.

Furthermore, the flexibility added to the schedule should permit to handle uncertainties. Three studies have tried to evaluate this property. Wu et al. (1999) study the impact of disturbed processing times on the weighted total tardiness in comparison with static and dynamic heuristics. When processing times are not greatly disturbed, they observe that group sequencing obtain better performances. Esswein (2003) studies the impact of disturbed processing times, due dates and release dates on a one machine problem and compares its results with a static heuristic method. Average performances are better with group sequencing than with the static method. Pinot et al. (2007) studies the impact of non-modeled transportation time between two operations. The group sequencing method is always better than a static scheduling method and better than a reactive scheduling method as long as transportation times are less or equal to processing times.

3. LOWER BOUNDS

In this section, we present an algorithm for finding a lower bound of the best case completion time of an operation. This algorithm will be the core of different lower bounds described later in the section. A more complete description of these bounds can be found in Pinot and Mebarki (2008).

[1] A semi-active schedule is a feasible schedule in which no local left-shift of an operation leads to another feasible schedule.

3.1 Finding the best case completion time of an operation

In Artigues et al. (2005), the worst case completion time of each operation is computed in polynomial time using dynamic programming. Our main goal in this section is to compute the best case completion time, that corresponds to the smallest value of C_i in every semi-active schedule described by a group sequence. As this problem is NP-hard,[2] it would be very useful to compute a lower bound in polynomial time on these best case completion times.

We can easily compute such a lower bound of our problem using a relaxation on the resources by making the assumption that each resource has an infinite capacity. However this bound can be improved using the property of group-sequencing: an operation in a given group cannot be executed before all the operations of its previous group have been executed. As a consequence, an operation can only begin after the optimal makespan of the previous group.

In this case, the best case lower bound for starting time of an operation (θ_i) is computed as the maximum of the best case (lower bound for) completion time (χ_j) of its predecessors ($\Gamma^-(i)$) and the (lower bound for the) completion time of the previous group ($\gamma_{g^-(i)}$).

Thus, it needs the computation of the optimal makespan of a group ($\gamma_{g_{\ell,k}}$). The best case lower bound for starting time of an operation (θ_i) can be viewed as a release date, so we can generate a $1|r_i|C_{\max}$ problem instance that corresponds to our problem, with $r_i = \theta_i$. This problem is polynomially solvable by ordering the operations in ascending release date (Brucker and Knust, 2007; Lawler, 1973).

Thus, the lower bounds are computed as follows:

$$\begin{cases} \theta_i = \max\left(r_i, \gamma_{g^-(i)}, \max\limits_{j\in\Gamma^-(i)} \chi_j\right) \\ \chi_i = \theta_i + p_i \\ \gamma_{g_{\ell,k}} = C_{\max} \text{ of } 1|r_i|C_{\max}, \forall O_i \in g_{\ell,k}, r_i = \theta_i \end{cases} \quad (1)$$

3.2 Lower bounds for regular objectives

Because regular objective functions are increasing functions of C_i by definition, a lower bound of C_i enables to compute a lower bound for any regular objective function. For this, we can use χ_i as a lower bound of C_i.

For example, we can compute a lower bound of the total tardiness (ΣT_i):

$$\mathrm{LB}(\Sigma T_i) = \sum_i T_i(\chi_i) = \sum_i (\max(\chi_i - d_i, 0))$$

The makespan, denoted by $C_{\max}$, is the most studied regular objective. It represents the total time span of the schedule.

A lower bound can be computed as:

$$\mathrm{LB}(C_{\max}) = \max_{\ell,k} \gamma_{g_{\ell,k}}$$

[2] Because the reduction between $F||C_{\max}$ and the best $C_{\max}$ in a group sequence is trivial, and that $F||C_{\max}$ is NP-hard, then the best case in a group sequence is NP-hard.

This makespan lower bound is improved by Pinot and Mebarki (2008) adapting the classical job-shop lower bound (Carlier and Pinson, 1989) to group sequencing using (1).

The lower bounds presented in this section can be used in an exact method. Next section describes such a method.

4. EXACT METHOD

In this section, we will describe an exact method to find the best schedule in a group sequence. This algorithm is a branch and bound algorithm that can be used for any regular objective function: only the lower bound needs to be adapted to the objective function.

A branch and bound algorithm is made of two procedures: a branching procedure and a bounding procedure. The branching procedure takes a (partially resolved) problem (called a node) and generates several subproblems (also called nodes) with the union of these problems covering the original problem. Solving all the nodes solves the global problem. The bounding procedure uses two bounds: the upper bound, that is the quality of the best solution found so far, and the lower bound, that gives a lower bound of the best quality possible to find at a given node. If the lower bound of a given node is greater or equal than the upper bound, all the solutions of the subproblem of this node will be equal or worst than the better solution found. So, this node can be discarded.

4.1 Branching

The branching procedure is based on active schedules. In an active schedule, it is impossible to schedule an operation earlier without delaying another operation. In the set of active schedules, it exists an optimal schedule for any regular objective, so we can use this branching procedure to find the optimal value for any regular objective.

We do not use a classical method to enumerate all active schedules, but we use the properties of group sequencing to simplify this enumeration. For a feasible group sequence, the groups are partially ordered according to the operation's precedence constraints and the group order constraint (*i.e.*, on a given machine, a group cannot begin before the preceding group on the same machine is not completed). So, sequencing the groups according to this partial order guarantees that all the predecessor operations (according to precedence constraints and group ordering) of the operations in the group are scheduled. It is possible to enumerate every active schedule by scheduling entirely each group according to the partial order of the groups.

A node on the search space will be represented as a group sequence, and an ordered list of groups to be sequenced. A solution is a group schedule with only one operation per group (and an empty list of groups).

In practice, at the beginning of the algorithm, the groups are ordered according to their partial order. Then, the groups with one operation are removed, because they are already sequenced. This ordered list of groups represent the groups to be sequenced in this particular order.

The node generation will be as follows:

- The earliest starting times (r_i) of each operation in the first group of the list are calculated.
- The smallest completion time is calculated : $C_{\min} = \min_{i|O_i \in g_{k,\ell}} r_i + p_i$.
- All operations with a starting time smaller than the smallest completion time ($t_i < C_{\min}$) are selected.
- For each of these operations, a new node is generated: the operation is sequenced (a group with the operation is created before the current group and the operation is removed from the current group) and then, if the current group has only one operation remaining, the group is removed from the list.

As an illustration of the method, let us enumerate all the nodes of the group sequence described on Fig. 1. First, we order all the groups. A solution is

$$[\{O_4\}, \{O_7\}, \{O_1, O_8\}, \{O_2\}, \{O_9\}, \{O_3, O_5\}, \{O_6\}],$$

We can see that this order is not unique: for example, the two first groups (*i.e.*, $\{O_4\}$ and $\{O_7\}$) can be permuted, but $\{O_7\}$ must precede $\{O_1, O_8\}$ because of the precedence constraint between O_7 and O_8. Then, we remove all the groups with only one operation per group. So, we have $[\{O_1, O_8\}, \{O_3, O_5\}]$ remaining in the list.

Fig 3 shows the different nodes generated. The global node that describes the problem is composed of the original group sequence and the group list found previously. The branching procedure is executed on this node, with the current group $\{O_1, O_8\}$:

- $r_1 = 0, r_8 = 2$ because of the precedence constraint between O_7 and O_8.
- $C_{\min} = \min(C_1, C_8) = \min(3, 4) = 3$.
- Because $r_1 < r_8 < C_{\min}$, the two operation of this group are selected.
- Two nodes are generated: node 1 with O_1 sequenced first and node 2 with O_8 sequenced first. Because the current group has now only one operation for node 1 and 2, the current group is removed.

Then, the branching procedure is executed on node 1, with a new current group $\{O_3, O_5\}$:

- $r_3 = 7, r_5 = 4$.
- $C_{\min} = \min(C_3, C_5) = \min(10, 7) = 7$.
- Because $r_5 < C_{\min} \leq r_3$, only O_5 is selected.
- Only node 1,1 is generated with O_5 sequenced before. Each group of this node has only one operation per group, so this node is a solution.

Then, the branching procedure is executed on node 2. As for node 1,1, only node 2,1 is generated. This node is a solution of the problem. The optimal solution of any regular objective is the schedule in node 1,1 or in node 2,1.

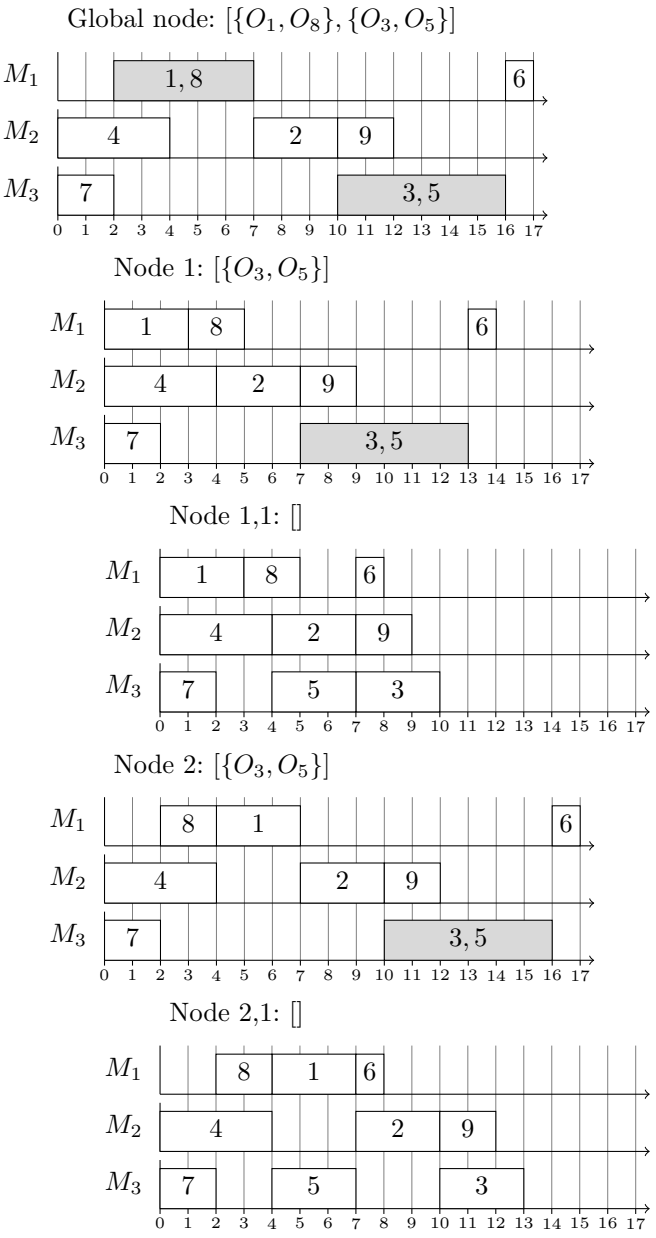
4.2 Reducing the search space

To reduce the search space, we propose a sufficient condition for the complete sequencing of the first group in the group list without losing the optimal solution.

The completion time of an operation interfere with the objective function in two ways:

- the completion time itself, because the objective function is a function of the completion times;

Fig. 3. Search space



- by interfering with the completion time of the other operations, because of precedence constraints or resource constraints.

So, a sufficient condition for the sequencing of an entire group (according that the group is the current group in the node structure) without losing the optimal solution is:

- the sequencing does not degrade the objective function;
- the sequencing does not interfere on the earliest starting time of the operations with successor constraints and resource constraints.

The last condition can be translated to a one machine problem using θ_i computed with (1): the operations need to be completed before the earliest starting time of the successor operations and before the smallest earliest starting time of the operation in the successor group. So, we

have a one machine problem with release dates and strict due dates, the due date of an operation as the minimum between the earliest starting time of the successors of the operation and the earliest starting time of the operation in the successor group:

$$1|r_i, \tilde{d}_i|-, \forall O_i \in g_{\ell,k}, \tilde{d}_i = \min_{j|O_j \in \Gamma^+(O_i) \cup g_{\ell,k+1}} \theta_j$$

with r_i the earliest starting time of the operation O_i in the group sequence.

If a sequence of the current group is a solution of this problem and the completion times of the operations sequenced do not degrade the objective function, then this group can be sequenced without losing the optimal solution.

Before processing a node, a sequence that satisfies the sufficient condition is searched for the first group of the list. If it is found, then the group is sequenced, the group is removed from the list, and this new node is added to the list of nodes.

Due to this sufficient condition, we can avoid the generation of some nodes, and so, reduce the size of the search space.

Let us illustrate this improvement on the global node of Fig. 3. First, we generate the one machine problem corresponding to the group $\{O_1, O_8\}$ using (1):

$$r_1 = 0, \tilde{d}_1 = \min(\theta_2, \theta_6) = \min(4, 7) = 4$$
$$r_8 = 2, \tilde{d}_8 = \min(\theta_9, \theta_6) = \min(7, 7) = 7$$

The sequence O_1, O_8 solves this problem. If C_1 and C_8 are not involved in the computation of the objective function, then, this group can be sequenced, corresponding to node 1. For example, for the makespan, C_1 and C_8 are not involved because C_6 will always be greater than C_1 and C_8. Due to the sufficient condition, it is possible to discard Node 2 (and Node 2,1 because it is generated from Node 2) and, thus, reducing the search space.

4.3 Searching strategies

The branching procedure generates nodes, but the order of exploring the nodes affect the performances of the algorithm: if the best solution is found sooner, the upper bound will be better, and then more nodes will be discarded.

There are two main techniques to search in a branch and bound method: deep search and best-bound search.

Deep search goes directly to a solution: the nodes are processed in a last-in-first-out order. In this mode, it is very important to order the nodes correctly when a list of nodes is added. It would allow to find good solutions earlier. In this mode, we order the generated nodes as follow: the generated nodes are ordered in increasing order of the lower bound.

Another technique to search the tree is to process first the nodes with the best bound: best-bound first. In this mode, only the needed node will be processed, except a few nodes that have a lower bound equal to the optimal quality but which are not solutions The drawback of this mode is that the stored nodes can take a lot of memory. That is why this mode is generally used with branching procedure that generates only two nodes. Our branching procedure generates more than two nodes, so we need to limit the number of nodes stored to control the memory used. First, we use a best-bounded first search, *i.e.* we process first the nodes with the lowest lower bound, and then, the ties are broken according to the number of operations sequenced (the nodes close to a solution are favored). If the number of stored nodes is greater than a given number, the best-bound node is processed entirely using the deep search.

5. EXPERIMENTS

This section presents some experiments of the exact method for the makespan objective function. Different variants will be tested in order to analyze the impact of each component of the exact method.

5.1 Protocol

We took a well known set of benchmark instances called `la01` to `la40` from Lawrence (1984). These instances are widely used in the job shop literature. These are classical job shop instances, with m operations on each job (m as the number of machine), each operation of a job executed on a different machine. It is composed of 40 instances of different sizes (5 instances for each size). The objective is to minimize the makespan.

For each instance, we generated group sequences with known optimal value and very high flexibility. To generate these group sequences, we used a greedy algorithm that merged two successive groups according to different criteria until no group merging is possible. This algorithm begins with a one-operation-per-group sequence computed by the optimal algorithm described in Brucker et al. (1994) (so, by construction, the optimal makespan of these group schedules is the makespan of the one-operation-per-group sequence). The greedy algorithm is described in Esswein (2003). The experiments are executed on an Intel Pentium Xeon 2.60GHz.

We try to solve these problems with three different variants of our exact method:

- Default: the makespan lower bound presented in section 3.2. is used, the sufficient condition is used, and best-bound search is used until 1000 nodes are stored (the program always uses less than 100MB of memory with these settings);
- Deep search: same as Default with deep search;
- No sufficient condition: same as Default without using the sufficient condition of section 4.2.

5.2 Results

The results of these experiments are exposed on Tab. 2. For each instance set of the same size and each variant of the exact method, we give the execution time. The last line of the table gives the average gap.

Results for instances not resolved by Default after 24 hours are shown on Tab. 3. The variant used is Default, except 10000 nodes can be stored before using deep search. For each instance, the optimal value is shown (except for instance la27, the exact algorithm for the job shop problem

Table 2. Exact Method Results

The size is noted as $n \times m$ with n the number of jobs and m the number of machines.

Size	Nb. inst.	Average time (s)		
		Default	Deep search	No suff. cond.
10×5	5	0.057	0.127	0.063
15×5	5	0.112	0.285	0.148
20×5	5	0.028	0.028	0.028
10×10	5	18.382	8.902	116.610
15×10	5	165.416	1044.473	827.296
20×10	2	31.722	1946.364	343.394
30×10	5	3.380	8.816	4.215
15×15	3	4429.970	95597.361	8206.857
Average Gap		–	18.936	1.716

Table 3. Results for hard instances after 24h

LB: lower bound, UB: upper bound

Size	Inst.	Opt.	LB	UB
20×10	la27	1252*	1235	1279
20×10	la29	1202	1202	1221
20×10	la30	1355	1355	1359
15×15	la37	1397	1397	1412

has only proposed an upper bound of the optimum). Then, the lower bound and upper bound after 24 hours is shown.

The exact method finds the optimal solution for most of the instances in less than a day of computation. The Default variant is the most effective variant on these experiments: it outperforms the other variants on almost every instances.

First, we can see that instances with 5 machines are resolved very fast: less than one second. Our algorithm is very effective for small size problems. For small manufacturing systems, it could be used in real time during the execution of the schedule. The results for the instances of size 30×10 are also very good. This can be explained by the fact that there are lots of optimal schedules in these problems (finding an optimal solution is easier), and that the lower bound is very accurate on long instances (there are less nodes to explore in order to prove the optimality).

Let us study the impact of the searching strategy. The average time gap is about 19, *i.e.* using deep search is 20 times longer than using best-bound search. Best-bound search minimizes the number of unnecessary nodes explored. Deep search outperforms best-bound search only for la16 and la18. In these cases, the optimal solution is found very early, and then, only necessary nodes need to be explored.

The sufficient condition reduces the search space by definition. The overhead is profitable: the average gap time is about 1.7. In some cases, the gap can be very high: for la17, it is more than 27.

Now let us study the results on Tab. 3. This table gives results for the instances that cannot be solved within a day. Instances la30 and la37 have an optimal lower bound. The difficulty for these instances is to find the optimal solution. Instance la27 is different. The lower bound and the upper bound are disposed around the best upper bound found.

6. CONCLUSION

In this article, we study the best-case in a group sequence. An exact method solving this problem is proposed. This method is a branch and bound algorithm that can solve the problem for any regular objective. The enumeration is based on the enumeration of the active schedules, with a dedicated method to reduce the search space.

The experiments show that our exact method is really fast for small instances, but some instances with 200 operations could not be solved within a day.

REFERENCES

Artigues, C., Billaut, J.C., and Esswein, C. (2005). Maximization of solution flexibility for robust shop scheduling. *European Journal of Operational Research*, 165(2), 314–328.

Billaut, J.C. and Roubellat, F. (1996). A new method for workshop real-time scheduling. *International Journal of Production Research*, 34(6), 1555–1579.

Brucker, P., Jurisch, B., and Sievers, B. (1994). A branch and bound algorithm for the job-shop scheduling problem. *Discrete Applied Mathematics*, 49(1-3), 107–127.

Brucker, P. and Knust, S. (2007). Complexity results for scheduling problems. `http://www.mathematik.uni-osnabrueck.de/research/OR/class/`. [online; retrieved on 2008-06-18].

Carlier, J. and Pinson, E. (1989). An algorithm for solving the job-shop problem. *Management Science*, 35(2), 164–176.

Erschler, J. and Roubellat, F. (1989). An approach for real time scheduling for activities with time and resource constraints. In R. Slowinski and J. Weglarz (eds.), *Advances in project scheduling*. Elsevier.

Esswein, C. (2003). *Un apport de flexibilité séquentielle pour l'ordonnancement robuste*. Thèse de doctorat, Université François Rabelais Tours.

Graham, R.L., Lawler, E.L., Lenstra, J.K., and Rinnooy Kan, A.G.H. (1979). Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics*, 5, 287–326.

Lawler, E.L. (1973). Optimal sequencing of a single machine subject to precedence constraints. *Management Science*, 19(5), 544–546.

Lawrence, S. (1984). Resource constrained project scheduling: an experimental investigation of heuristic scheduling techniques (supplement). Technical report, Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, Pennsylvania.

Pinot, G., Cardin, O., and Mebarki, N. (2007). A study on the group sequencing method in regards with transportation in an industrial FMS. In *Proceedings of the IEEE SMC 2007 International Conference*.

Pinot, G. and Mebarki, N. (2008). Best-case lower bounds in a group sequence for the job shop problem. In *Proceedings of the 17th IFAC World Congress*.

Wu, S.D., Byeon, E.S., and Storer, R.H. (1999). A graph-theoretic decomposition of the job shop scheduling problem to achieve scheduling robustness. *Operations Research*, 47(1), 113–124.