Complete Characterization of Near-Optimal Sequences for the Two-Machine Flow Shop Scheduling Problem

Jean-Charles Billaut, Emmanuel Hébrard, Pierre Lopez

Complete Characterization of Near-Optimal Sequences for the Two-Machine Flow Shop Scheduling Problem

Jean-Charles Billaut[1], Emmanuel Hebrard[2], and Pierre Lopez[2]

[1] Université François-Rabelais Tours
Laboratoire d'Informatique
64 avenue Jean Portalis, 37200 Tours, France
jean-charles.billaut@univ-tours.fr
[2] CNRS ; LAAS ; 7 avenue du colonel Roche, 31077 Toulouse, France
Université de Toulouse ; UPS, INSA, INP, ISAE, UT1, UTM ; LAAS ; 31077 Toulouse, France
{hebrard,lopez}@laas.fr

Abstract. In a two-machine flow shop scheduling problem, the set of ϵ-approximate sequences (*i.e.*, solutions within a factor $1+\epsilon$ of the optimal) can be mapped to the vertices of a permutation lattice.
We introduce two approaches, based on properties derived from the analysis of permutation lattices, for characterizing large sets of near-optimal solutions. In the first approach, we look for a sequence of minimum level in the lattice, since this solution is likely to cover many optimal or near-optimal solutions. In the second approach, we look for all sequences of minimal level, thus covering all ϵ-approximate sequences.
Integer linear programming and constraint programming models are first proposed to solve the former problem. For the latter problem, a direct exploration of the lattice, traversing it by a simple tree search procedure, is proposed. Computational experiments are given to evaluate these methods and to illustrate the interest and the limits of such approaches.

1 Introduction

Computing the set of near-optimal solutions of a combinatorial problem has many applications. It is for instance the case when there is some uncertainty in the problem definition. For example, production systems are subject to numerous disturbances deriving mainly from data uncertainty and unexpected events. These disturbances result in lack of raw materials, tardiness, tool failure, etc., and may make the initial planning infeasible. In particular, this motivates research works on flexibility and robustness in scheduling [9]. This is why dynamic or reactive approaches have been developed [10, 24, 25]. Among the possible dynamic approaches, some of them are based on the characterization of sets of solutions. In this context, the set of solutions can be used in an interactive and/or dynamic environment to guide the decision making from an optimal solution to another one when an unexpected event occurs, or on a user request.

Another important reason to obtain the characteristics of optimal solutions is to solve multiobjective scheduling problems [27]. These characteristics could be incorporated into branch-and-bound procedures, in order to prune nodes more efficiently, hopefully.

In [2, 3, 10], the set of solutions or schedules is given by a sequence of groups of jobs on each machine, where the sequence of jobs inside each group is not fixed (the authors talk about 'groups of permutable jobs'). All the characterized solutions ensure a performance guarantee on a given quality measure. This methodology has been implemented in a software and has been used in practice by several companies [23]. In [1], a set of semi-active schedules is characterized by a partial order of jobs on each machine. The authors propose a method for computing the best case and the worst case performances. In [15], the authors propose a method for characterizing a large set of optimal solutions, based on the analysis of interval structures and on a theorem established in [17]. The method is applied to a single-machine problem, where jobs have release dates and due dates. All these methods allow the characterization of a subset of the set of optimal or approximate solutions. Another way for this characterization is to provide constraint propagation techniques or dominance properties, in order to maintain the tightest set of remaining consistent decisions [14].

It is well known that there potentially exist several optimal solutions to a given scheduling problem. They may even have a huge number of different optimal solutions [7, 26], for example several hundreds of thousands for mid-size academic instances. Moreover, it is well known as well that it is often not easy to find an optimal solution to a scheduling problem because of its NP-completeness in the general case. However, some scheduling problems can also be solved in polynomial time using specific methods, which generally consist in sorting the jobs according to a simple priority rule.

In this paper, we investigate the possibility to characterize the whole set of optimal solutions of the two-machine flow shop scheduling problem, *i.e.*, to give the analytical characteristics of these solutions. Obviously, the aim is not to enumerate these solutions, but to describe their characteristics. In our approach, this is achieved by the knowledge of dominance rules for the problem under consideration and through the study of the various optimal solutions associated with the vertices of a lattice. Since a realistic goal (for an enterprise for example) is not necessarily to focus on optimal solutions, note that the same methods apply for characterizing the whole set of ϵ-approximate solutions, *i.e.*, the set of solutions in the lattice with a performance not worse than a given distance function of ϵ from the value of the optimal solution.

The rest of the paper is organized as follows. Section 2 provides the necessary notations, definitions, and properties concerning the main mathematical object used in this work: the "lattice of permutations". Then in Section 3, we recall how all optimal solutions of a class of scheduling problems can be characterized by a subset of vertices of minimal level in the permutohedron. Then we present the problem addressed in this paper, namely: finding a minimum vertex, and finding all minimal vertices. In Section 4 we propose an Integer Linear Programming

(ILP) approach as well as a Constraint Programming (CP) approach for the former problem, whereas Section 5 describes an algorithm to solve the latter problem. Section 6 presents the results obtained from computational experiments and Section 7 gives a conclusion and some future research directions.

2 The Lattice of Permutations: Definitions and Properties

We consider the set $\{1, 2, \ldots, n\}$ of integers and S_n the group of all permutations on $\{1, 2, \ldots, n\}$. We represent the members of S_n by strings of integers. As an example, consider $n = 4$; $\sigma = 4213$ denotes a permutation σ where $\sigma(1) = 4$, $\sigma(2) = 2$, $\sigma(3) = 1$, and $\sigma(4) = 3$. Using the same notations as in [22], we denote by $index(\sigma, i)$ the position of integer i in permutation σ. For the previous example, we have $index(\sigma, 1) = 3$.

With the elements of S_n we define a directed graph where the nodes are the elements of S_n. In this digraph, there exists an edge between nodes σ and σ' if and only if $\sigma = \alpha ij\beta$ with α and β two partial orders, $i, j \in \{1, 2, \ldots, n\}$ with $index(\sigma, j) = index(\sigma, i) + 1$, $\sigma' = \alpha ji\beta$, and $i < j$. In other words, there is an edge between σ and σ' if these permutations are the same, except that there exist i and j, two consecutive jobs with $i < j$, that are in the reverse order in σ'.

This digraph is a lattice, called the *lattice of permutations* or *permutohedron* [13]. Figure 1 gives the lattices of permutations for $n = 3$ and $n = 4$.

To each permutation in the lattice can be associated a level. There are at most $\frac{n(n-1)}{2} + 1$ levels. By convention we say that permutation $(n, n-1, n-2, \ldots, 1)$ is at level 0 and that permutation $(1, 2, 3, \ldots, n)$ is at level $n(n-1)/2$. We denote by $\kappa(\sigma)$ the level of permutation σ. For a given permutation σ, we denote by $\Gamma(\sigma)$ the set of couples defined as follows:

$$\Gamma(\sigma) = \{(i, j) \in \{1, 2, \ldots, n\}^2 \mid i < j \text{ and } index(\sigma, i) < index(\sigma, j)\}$$

For example, in permutation $\sigma = 4132$, we have $\Gamma(\sigma) = \{(1, 3), (1, 2)\}$.

We now report some properties associated to the lattice of permutations previously defined.

Property 1. [6]: For any permutation σ, the level of σ is exactly its number of inversions from permutation at level 0, *i.e.*, the number of times we have $i < j$ and $index(\sigma, i) < index(\sigma, j)$:

$$\kappa(\sigma) = |\Gamma(\sigma)|$$

Property 2. [6]: Let consider a permutation σ. Any predecessor π of σ in the digraph is such that:

$$\Gamma(\sigma) \subset \Gamma(\pi)$$

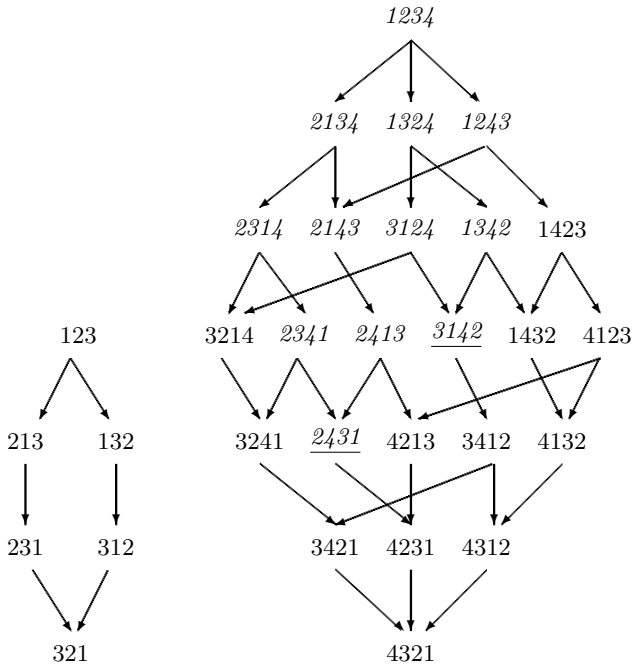

Fig. 1: Lattices of permutations for $n = 3$ (left) and $n = 4$ (right)

If we consider the elements of $\Gamma(\sigma)$ as a set of constraints associated to σ, we can say that all the predecessors of σ have to satisfy at least the same constraints as σ. We then claim that $\Gamma(\sigma)$ gives the characteristics of all the predecessors of permutation σ in the digraph.

3 Characteristic Solutions

In many scheduling problems, a set of dominant solutions (*e.g.*, left-shifted schedules) can be mapped with the set of permutations of jobs and consequently to the vertices of a permutohedron. Moreover, in a previous paper [8], we showed that, for a number of classes of such problems, by carefully choosing the labeling of the jobs, one can design a permutohedron with a very desirable property: none of its children can have a strictly better objective value than its own. Thus, the set of optimal solutions with minimal level in the permutohedron is sufficient to characterize *all* optimal solutions. Indeed, a solution is optimal if and only if it belongs to this set, or is an ancestor of an element in this set.

The problems for which this method holds are those where a simple rule (checkable in $O(1)$) exists guaranteeing that given two consecutive jobs in a sequence, it is possible to say which job it is always better to put first. Such a rule allows the building of optimal algorithms in $O(n \log n)$ time complexity, after sorting the jobs according to this rule. They are therefore all easy to solve.

However, finding all solutions, or a large number of solutions to such problems may still be difficult.

In terms of scheduling, a permutation is a sequence of jobs. Let $f(\sigma)$ denote the objective value (to minimize) of a sequence σ. We say that an ordering relation $\mathcal{R}$ is an *improving* relation iff for any sequence $\alpha ji\beta$ (with α and β two arbitrary subsequences and i and j two consecutive jobs) we have:

$$i\mathcal{R}j \Rightarrow f(\alpha ij\beta) \leq f(\alpha ji\beta)$$

This property is often referred to as "pairwise interchange" mechanism, and it is frequently used for proving properties of scheduling algorithms. Indeed it is easy to see that the sequence σ built by sorting all jobs according to an improving order relation $\mathcal{R}$ is optimal. Starting from an arbitrary sequence, one can obtain σ by iteratively swapping consecutive elements while never degrading the objective value.

Assume that sequence $(1, 2, \ldots, n)$ is an optimal sequence given by a sorting algorithm based on an improving relation $\mathcal{R}$ and consider the permutohedron rooted in $(1, 2, \ldots, n)$. By definition, given any sequence σ and one of its successor σ', we have $f(\sigma) \leq f(\sigma')$. Therefore, if a sequence σ is optimal, all the predecessors of σ are optimal sequences. Moreover, we say that a sequence σ is *minimal* with respect to a permutohedron, iff it is optimal and none of its children are, *i.e.*, any further swap of two consecutive jobs leads to a sub-optimal solution. Notice that the method that we introduce in this paper applies to any problem for which an improving relation exists (this is true for many scheduling problems).

In this paper, we are concerned with finding all minimal sequences in such permutohedra and, in particular, the minimal sequence with minimum level in the lattice (so-called *minimum sequence*). Intuitively, the optimal sequence with minimum level covers many optimal solutions. It can therefore be seen as robust in the sense that, when for some reason this solution is no longer valid, one can easily generate optimal solutions by swapping jobs in order to climb up the lattice, with the hope that at least one of these solutions will still be valid. Another possibility is to compute all minimal sequences. Indeed, this is a relatively concise way of storing all the optimal solutions of the problem.

We shall introduce several methods to address the two following problems:

- PB1, finding an optimal sequence with minimum level in the lattice;
- PB2, finding all minimal sequences in the lattice.

Example 1. For instance, suppose that the sequence (1234) in Figure 1 is given by a sorting algorithm with respect to an improving relation $\mathcal{R}$ and that the sequences (3142) and (2431) are the only minimal sequences (they are underlined in Figure 1). Then we know that the set of optimal solutions is exactly the set composed of themselves and of their ancestors: {(1234), (2134), (1324), (1243), (2314), (2143), (3124), (1342), (2341), (2413), (3142), (2431)} (marked in italic). Observe that the minimum sequence is (2431) and covers 8 optimal solutions (including itself), whereas (3142) covers only 5.

We consider a set of n jobs to schedule in a two-machine flow shop, where the two machines are denoted by M_1 and M_2 and we denote by $p_{j,1}$ and $p_{j,2}$, the processing time of job j on machine M_1 and on machine M_2, respectively, $1 \leq j \leq n$. C_j denotes the completion time of job j on machine M_2. The criterion under study is the makespan $C_{\max} = \max_{1 \leq j \leq n} C_j$. The problem that we consider is denoted by $F2||C_{\max}$ and is solved to optimality by ordering the jobs following Johnson's dominance condition [20].

Considering the makespan minimization ($\min C_{\max}$ in our case), we define an ϵ-approximate sequence ($\epsilon \in \mathbb{R}^+$) as a sequence $\bar{\sigma}$ such that:

$$C^*_{\max} \leq C_{\max}(\bar{\sigma}) \leq (1+\epsilon) \times C^*_{\max}$$

In the following expressions, it is sufficient to replace $C^*_{\max}$ by $(1+\epsilon)C^*_{\max}$ for characterizing the ϵ-approximate solutions.

4 Finding the Minimum Sequence

In order to find a schedule of minimum level in the permutohedron we solve the scheduling instance with some slight modifications.

We assume that the root sequence σ^* of the lattice was obtained using Johnson's algorithm, and such that jobs are renumbered with respect to this first optimal sequence. Then we re-solve the initial instance with the following changes:

- The objective function $C_{\max}$ to minimize is replaced by the constraint stating that the expression $C_{\max}$ should be less than $(1+\epsilon)C_{\max}(\sigma^*)$.
- We use a new objective function: minimize the level in the lattice.

The complexity of the modified problem is not known. However, there are examples of tractable problems that become NP-hard when adding a similar objective function. For instance `monotone-2SAT` (the satisfiability of a 2-CNF formula with only positive literals) is NP-hard if the number of atoms set to true is to be minimized [4].

4.1 Integer linear programming approaches

We first propose an integer linear programming model with the variables defined as follows: for all $1 \leq i < j \leq n$, $y_{i,j}$ is a binary variable (0–1) equal to 0 if job i precedes job j in the sequence and 1 otherwise. We also introduce continuous variables: $t_{j,1}$ and $t_{j,2}$ are the start times of job j on machine M_1 and on machine M_2, respectively. HV stands for High Value, and can be set for example to $\sum_{j=1}^{n} \sum_{k=1}^{2} p_{j,k}$.

In addition, it is assumed that the binary variables verify a kind of triangle inequality:

$$y_{i,k} \leq y_{i,j} + y_{j,k}, \forall i,j,k \in \{1,\ldots,n\}, i \neq j \neq k$$

We have:

$$t_{j,k} \geq t_{i,k} + p_{i,k} - HV y_{i,j}, \tag{4.1}$$

$$t_{i,k} \geq t_{j,k} + p_{j,k} - HV(1 - y_{i,j}), \tag{4.2}$$

$$(4.1) \text{ and } (4.2), \forall k \in \{1,2\}, \forall i,j \in \{1,\ldots,n\}, i \neq j$$

$$t_{j,2} \geq t_{j,1} + p_{j,1}, \forall j \in \{1,\ldots,n\}$$

To take account of the objective function, let $C^*_{\max}$ denote the optimal value – supposed to be known – of the makespan.

We post the following constraints:

$$t_{j,2} + p_{j,2} \leq C^*_{\max}, \forall j \in \{1,2,\ldots,n\} \tag{4.3}$$

Another way to express Property 1 is to say that the level in the lattice is equal to the cardinality of the set of permutations preserved from the root sequence σ^*. Notice that the characteristic function of the set $\Gamma(\sigma)$ is given by the set of variables $y_{i,j}$ (with the correspondence $y_{i,j} = 0 \Leftrightarrow (i,j) \in \Gamma(\sigma)$). Finding the sequence with minimum level can therefore be expressed by the following objective function:

$$\text{MAX} \sum_{i=1}^{n} \sum_{j=i+1}^{n} y_{i,j}$$

4.2 Constraint programming approach

We propose a constraint programming model similar to the integer linear programming model described in Section 4.1, however solved using a different approach.

As in the ILP model, we introduce a binary variable $y_{i,j}$ for each pair of jobs, taking the value 0 if job i precedes job j and 1 otherwise. We also introduce integer variables $t_{j,1}$ and $t_{j,2}$ for the start times of job j on machine M_1 and on machine M_2, respectively.

We post precedence constraints between the two activities of each job as follows:

$$y_{i,j} \Rightarrow t_{j,k} \geq t_{i,k} + p_{i,k}, \tag{4.4}$$

$$\neg y_{i,j} \Rightarrow t_{i,k} \geq t_{j,k} + p_{j,k}, \tag{4.5}$$

$$(4.4) \text{ and } (4.5), \forall k \in \{1,2\}, \forall i,j \in \{1,\ldots,n\}, i \neq j$$

$$t_{j,2} \geq t_{j,1} + p_{j,1}, \forall j \in \{1,\ldots,n\}$$

Note that we do not post constraints on triplets of binary variables to exclude cycles. Bound consistency is enforced on these constraints, *i.e.*, when the value of the binary variable $y_{i,j}$ is set, the bounds of the integer variables $t_{i,k}$ and $t_{j,k}$ are updated with respect to the constraint $t_{i,k} \geq t_{j,k} + p_{j,k}$, or $t_{j,k} \geq t_{i,k} + p_{i,k}$ according to $y_{i,j}$'s value. Conversely, if one of the two precedence constraints

becomes entailed or disentailed because of the current domain of $t_{i,k}$ and $t_{j,k}$, then the value of $y_{i,j}$ is set accordingly.

Exactly as in the ILP model, we set a maximum value to the objective function using the same constraints. Moreover, finding the sequence with minimum level is expressed by the same objective function:

$$\text{MAX} \sum_{i=1}^{n} \sum_{j=i+1}^{n} y_{i,j}$$

The method that we used to solve this constraint programming model is essentially the same used for several variants of the job-shop scheduling problem in [18, 19].

Search method: The problem is solved using a dichotomic search on the objective, yielding a satisfaction problem at each step. Each of these steps is bounded by the number of nodes of the search tree that can be expended during search. If the dichotomic search is not conclusive, a branch-and-bound procedure is used starting from the bounds computed during the dichotomic phase.

Variable selection heuristic. We use a slightly modified version of the *domain over weighted-degree* heuristic [12] to select the next variable to branch on. Observe that we branch only on the binary variables ($y_{i,j}$). Let $w(t_i)$ be the number of times search failed while propagating any constraint involving job i, and let $min(t_{i,k})$ and $max(t_{i,k})$ be, respectively, the minimum and maximum starting time of $t_{i,k}$ at any point during search. The next variable $y_{i,j}$ to branch on is the one minimizing the value of:

$$(max(t_{i,k}) + max(t_{j,k}) - min(t_{i,k}) - min(t_{j,k}) + 2)/(w(t_{i,k}) + w(t_{j,k}))$$

Value selection heuristic. When branching on the variable $y_{i,j}$ we try first the value assigned to this variable in the best feasible solution found so far. If no solution has been found, the value 0 is tried first. This idea is a simplified version of the solution guided approach (SGMPCS) proposed by Beck for job-shop scheduling problems [5].

Restarts with Nogoods. We use a geometric restarting strategy [28]. When a given number of nodes have been explored, we stop the current exploration and start again from scratch. The limit in number of nodes grows geometrically: it is of the form $s, sr, sr^2, sr^3, \ldots$ where s is the base and r is the multiplicative factor. In our experiments the base was 256 failures and the multiplicative factor was 1.3. Moreover, after each restart, the dead ends of the previous explorations are stored as clausal nogoods [21].

5 Finding all Minimal Sequences

Observe that it is possible to solve this problem using iteratively the method described in the previous section, and avoiding rediscovery of previous solutions with nogoods.

Suppose that a first sequence σ_0 of minimum level has been computed, and suppose that it corresponds to the set of precedences $index(\sigma_0, a_1) < index(\sigma_0, b_1)$ and $index(\sigma_0, a_2) < index(\sigma_0, b_2)$ and ... and $index(\sigma_0, a_{\nu_0}) < index(\sigma_0, b_{\nu_0})$. The clause $index(\sigma_0, b_1) < index(\sigma_0, a_1)$ or $index(\sigma_0, b_2) < index(\sigma_0, a_2)$ or ... or $index(\sigma_0, b_{\nu_0}) < index(\sigma_0, a_{\nu_0})$ can be added to the model in order to avoid finding σ_0 again.

For the CP formulation we can add:

$$y_{a_1,b_1} \vee y_{a_2,b_2} \vee \ldots \vee y_{a_{\nu_0},b_{\nu_0}} \tag{5.1}$$

And for the ILP formulation:

$$y_{a_1,b_1} + y_{a_2,b_2} + \ldots + y_{a_{\nu_0},b_{\nu_0}} \geq 1 \tag{5.2}$$

At each iteration, only one additional constraint of this type is produced and no additional variable is generated.

However, we shall see that such an approach is not efficient, hence we propose an algorithm that directly explores the lattice in depth first order and finds all minimal sequences. The only difficulty is to avoid exploring twice the nodes of the lattice, given that we cannot store it explicitly. As in the previously discussed models, the method starts from a schedule σ^* given by rule $\mathcal{R}$, then the jobs are renumbered with respect to this first optimal solution. However, instead of solving the scheduling problem, we explore the lattice depth first in Algorithm 1. In other words, a move occurs on the lattice by swapping elements of the optimal order only if the move corresponds to a downward edge in the lattice, and if it does not degrade the objective value below $(1 + \epsilon) \times C_{\max}(\sigma^*)$.

Let σ be a permutation on $\{1, \ldots, n\}$. The only operations that we use to move on the lattice is to swap two consecutive elements. We denote by $\texttt{swap}(\sigma,(a,b))$ the permutation σ' equal to σ except that a and b are swapped. For instance, $\texttt{swap}((41325),(1,3)) = (43125)$. We denote by $\texttt{opt}(\sigma)$ the fact that the objective value of the schedule σ is within the tolerated interval.

Algorithm 1 `Explore-Lattice`

Data: σ, $\Gamma(\sigma)$, *explored*
minimal $\leftarrow$ `True`;
foreach $k \in \{1, \ldots, n-1\}$ **do**
  $a \leftarrow \sigma(k)$;
  $b \leftarrow \sigma(k+1)$;
  if $a < b$ *and* $(a,b) \in \Gamma(\sigma)$ *and* $\texttt{opt}(\texttt{swap}(\sigma,(a,b)))$ **then**
    minimal $\leftarrow$ `False`;
    if $(a,b) \notin$ *explored* **then**
      `Explore-Lattice`$(\texttt{swap}(\sigma,(a,b)), \Gamma(\sigma) \setminus \{(a,b)\}, explored)$;
  explored $\leftarrow$ *explored* $\cup \{(a,b)\}$;
if *minimal* **then** $\texttt{print}(\sigma)$;

Algorithm 1 is initially called with $\sigma = (123..n)$, $\Gamma(\sigma) = \{(a,b)/1 \leq a < b \leq n\}$ and $explored = \emptyset$. It explores the optimal part of the lattice depth first. The current permutation is kept in the variable σ, whenever we reach a local optimum, *i.e.*, a permutation σ such that any swap increases the objective value above the acceptable threshold, we print it (Line 4).

We keep track of the current value of $\Gamma(\sigma)$ using a set. Last, we use another set denoted *explored* to store the elements of $\Gamma(\sigma)$ that we already explored in previous branches, to avoid visiting twice the same vertex of the lattice.

The first loop (in Line 1) goes over all the possible swaps in the sequence σ. The swaps that satisfy the conditions in Line 2 are actually edges in the lattice leading to an ϵ-approximate solution. If such an edge exists, the current node is not a local optimum. Then, the condition in Line 3 ensures that the successor has not yet been explored.

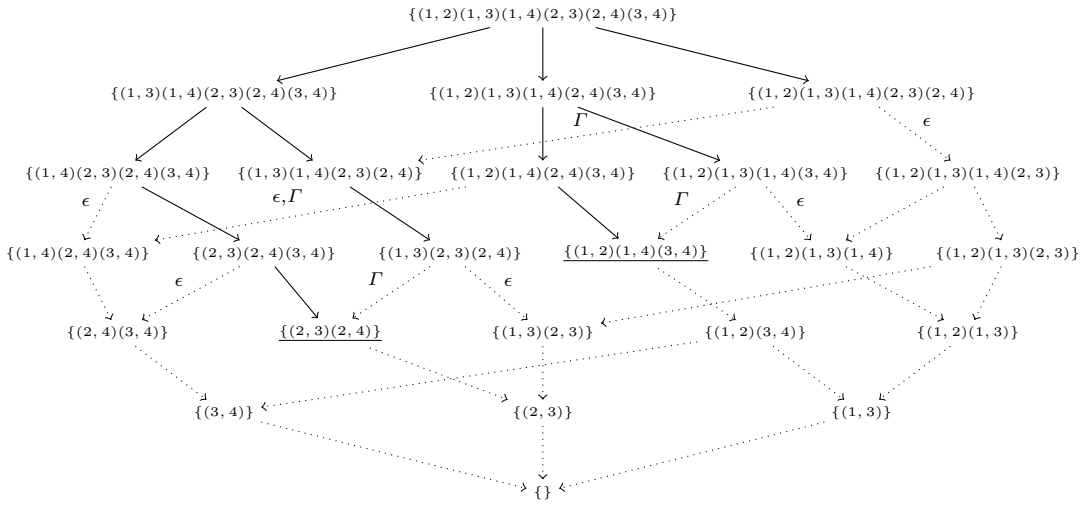

Fig. 2: Elements of $\Gamma(\sigma)$ in the lattice of permutations ($n = 4$)

Example 2. To illustrate the exploration of the lattice, take again the example depicted in Figure 1 for $n = 4$. Figure 2 now displays the corresponding sets $\Gamma(\sigma)$. Algorithm 1 explores all the solid edges of this digraph. For each dotted edge adjacent to a solid edge, we mark the reason why it is not explored with the following notation:

- ϵ: The child has an objective value strictly greater than $(1 + \epsilon) \times f(\sigma^*)$.
- Γ: The successor edge corresponds to swapping a for b, however $(a, b) \in explored$.

Theorem 1. *Algorithm 1 is correct and explores exactly one node for each ϵ-approximate solution in the instance.*

Proof. The correctness of the algorithm is entailed by the properties of the permutation lattice described in Section 3. We merely show here that branches of the search tree pruned because of the set *explored* do not lead to unseen minimal solutions. A swap (a, b) is added to *explored* iff it has been previously tried in the same or in an earlier recursive call. In other words, let the current sequence be σ with characteristic set $\Gamma(\sigma)$. If $(a, b) \in$ *explored*, then a node π with characteristic set $\Gamma(\pi) \supseteq \Gamma(\sigma) \setminus \{(a, b)\}$ and all its successors have been explored. Now suppose that there exists a successor ω or σ that is not a successor of π and such that $(a, b) \notin \Gamma(\omega)$. This implies that $\Gamma(\omega) \subset \Gamma(\sigma)$ and $\Gamma(\omega) \not\subset \Gamma(\pi)$. However, since $\Gamma(\pi) \supseteq \Gamma(\sigma) \setminus \{(a, b)\}$, we necessarily have $(a, b) \in \Gamma(\omega)$, contradicting our hypothesis.

Algorithm 1 never explores a sub-optimal node, that is, a node with objective value strictly greater than $(1 + \epsilon) \times f(\sigma^*)$. Indeed all predecessors of an ϵ-approximate node are ϵ-approximate, and a sub-optimal successor is not visited. Moreover, no node is explored twice. Indeed, suppose that a node ω has already been explored, and that the current node is a direct predecessor σ of ω. Let consider the recursive call at which ω was first explored. This call was at the same level in the recursive tree, and when exiting the branch starting with ω a swap $(a, b) \in \Gamma(\omega)$ is added to *explored*. Therefore, the branch leading to ω from σ will not be explored, since $(a, b) \in \Gamma(\omega)$ and $(a, b) \in$ *explored*. Therefore every ϵ-approximate node is explored at most once (in fact exactly once since there is no other pruning condition). □

6 Computational Experiments

We generated 30 random instances of two-machine flowshop problems ($F2||C_{\max}$) for 3 sizes: 8, 10 and 12 jobs. Then for each of them we computed an optimal sequence by sorting the jobs according to Johnson's rule. We give results for each size class, and for 5 values of ϵ, average across the 30 instances. The experiments were run on a MacBook pro dual core 2.3 GHz (no parallelization was used) using Mistral solver, except for the ILP model that was run on a PC clocked at 2.26 GHz with 3.45 GB RAM and solved using IBM ILOG CPLEX.

6.1 PB1: Finding a sequence of minimum level

In Table 1, we compare the results of the ILP and CP approaches to solve PB1, *i.e.*, find a solution of minimum level. The first column indicates the number of jobs n, and the value of ϵ. A time limit of 10 minutes was imposed on both approaches. For each method, we report the average minimum level of the sequence, the ratio of proven results within the time cutoff, the average CPU time in seconds, and the number of nodes explored.

Table 1: Minimum level: ILP *vs.* CP

Instance	CP				ILP			
	Level	Opt.	Time (s)	Nodes	Level	Opt.	Time (s)	Nodes
$\epsilon = 0$	14.40	1.00	0.05	5085	14.40	1.00	0.38	2048
$\epsilon = 0.05$	9.80	1.00	0.04	3945	9.80	1.00	0.32	1340
$n = 8$ $\epsilon = 0.1$	7.06	1.00	0.02	2597	7.06	1.00	0.24	823
$\epsilon = 0.15$	4.96	1.00	0.01	1510	4.96	1.00	0.30	392
$\epsilon = 0.2$	3.36	1.00	0.01	896	3.36	1.00	0.11	228
$\epsilon = 0$	23.36	1.00	3.27	249195	23.36	1.00	7.57	43679
$\epsilon = 0.05$	16.00	1.00	2.05	170201	16.00	1.00	4.65	28641
$n = 10$ $\epsilon = 0.1$	12.16	1.00	1.15	91903	12.16	1.00	3.28	19410
$\epsilon = 0.15$	9.06	1.00	0.59	46386	9.06	1.00	2.34	9300
$\epsilon = 0.2$	6.46	1.00	0.30	23350	6.46	1.00	0.92	4086
$\epsilon = 0$	30.26	0.66	72.64	3.83M	30.26	0.70	329.02	942822
$\epsilon = 0.05$	19.63	0.90	57.50	3.42M	19.63	0.93	141.71	506002
$n = 12$ $\epsilon = 0.1$	14.36	0.96	35.34	2.07M	14.36	1.00	56.85	281196
$\epsilon = 0.15$	10.46	1.00	18.10	1.06M	10.46	1.00	20.78	100744
$\epsilon = 0.2$	7.23	1.00	6.50	0.37M	7.23	1.00	6.53	27201

We observe that despite the modest size of the instances, the problem is very hard to solve for the model we used. We believe that the objective function makes the problem extremely difficult to solve to optimality. These models can be efficient on much larger job-shop problems, however with more standard objective functions (such as $C_{\max}$ or $L_{\max}$).

We were unfortunately unable to run the two approaches on identical hardware, however, the two machines were of the same generation and the data we report here is sufficient to conclude that:

- the CP approach is slightly faster than the ILP model on smaller instances.
- however, when the size grows, and not all instances are solved to optimality within the time cutoff, CPLEX is able to prove optimality more often than Mistral on the CP model.

Another interesting observation is that despite the fact that the proposed approaches cannot prove it for all instances, they both find a sequence of minimum level. The fact that they both find the exact same objective value (level) is already a strong argument, and in fact we were able to verify it in the following set of experiments.

6.2 PB2: Finding all minimal sequences

In Table 2, we compare our lattice exploration algorithm with a simple enumeration of all the solutions of the problem using the CP model above. The first column still refers to the number of jobs n, and the value of ϵ. The number

Table 2: All characteristic solutions *vs.* number of optimal solutions

Instance	Characteristic solutions		All solutions		Sol. Ratio
	CPU Time (s)	Count	CPU Time (s)	Count	
$\epsilon = 0$	0.00	28	0.04	1549	55
$\epsilon = 0.05$	0.00	79	0.10	7127	90
$n = 8$ $\epsilon = 0.1$	0.00	119	0.19	16143	135
$\epsilon = 0.15$	0.00	107	0.27	25310	236
$\epsilon = 0.2$	0.00	78	0.37	31951	409
$\epsilon = 0$	0.05	413	1.63	63020	152
$\epsilon = 0.05$	0.17	1696	7.43	442323	260
$n = 10$ $\epsilon = 0.1$	0.39	2522	18.10	1126764	446
$\epsilon = 0.15$	0.62	2182	34.18	2101697	963
$\epsilon = 0.2$	0.71	1345	47.06	2864340	2129
$\epsilon = 0$	2.40	7383	281.90	7476041	1012
$\epsilon = 0.05$	21.41	32015	1821.84	77899794	2433
$n = 12$ $\epsilon = 0.1$	46.24	47703	–	–	–
$\epsilon = 0.15$	71.69	32881	–	–	–
$\epsilon = 0.2$	82.10	15384	–	–	–

of characteristic solutions (total number of minimal sequences) is reported in column 3 while the CPU time in seconds to compute them all using the lattice exploration algorithm is in column 2. Then, in columns 5 and 4 we give the total number of solutions (total number of ϵ-approximate solutions) and the time it takes to list them all using the CP model. Last, we give the ratio of the total number of solutions over the number of characteristic solutions in column 6.

We first observe that the ratio between the size of the whole set of ϵ-approximate solutions and the number of minimal solutions required to characterize them seems to grow geometrically both with n and with ϵ. It suggests that the benefit of this approach grows with the size of the instances. For $n = 12$ and $\epsilon > 0.05$, the enumeration took longer than the one hour and a half time cutoff that we imposed in every case.

The CPU time needed to compute characteristic solutions with the lattice exploration algorithm is of course much smaller than that of enumerating all solutions. However the factor is relatively constant when the size augments. This is not surprising since the complexity of the lattice exploration algorithm is very much linked to the total number of optimal solutions.

Moreover, we observe that finding all minimal solutions using Algorithm 1 is often faster than finding the solution of minimum level with either the CP or ILP model. Clearly, since the minimum solution is minimal, it suggests that the CP and ILP models are sub-optimal. However, as ϵ grows, the runtime required for both the CP and the ILP models for solving PB1 becomes lower than that of Algorithm 1 for finding all minimal solutions. Indeed, the complexity of solving these models does not depend directly on ϵ. In fact, we observe empirically that

both CPLEX and Mistral are faster when ϵ grows. Therefore, the two approaches are complementary.

7 Conclusions and Further Research Directions

In this paper, we propose to characterize the whole set of ϵ-approximate schedules for the two-machine flow shop scheduling problem. The main concept to set up our reasoning is the lattice of permutations (total orders of the jobs). To treat the issue, two optimization problems are addressed: (1) finding an optimal sequence with minimum level (maximum depth) in the lattice; (2) finding all minimal sequences. The complexity of these problems remains open.

We propose three different approaches operating the concept of lattice of permutations, namely an integer linear programming formulation, a constraint programming approach, and a direct exploration of the lattice based on depth-first search.

Computational experiments show that the CP approach with Mistral is faster than the ILP with CPLEX for small instances, but CPLEX can prove optimality more often than Mistral.

This approach can be used for characterizing the set of ϵ-approximate schedules for other scheduling problems, if an ordering relation exists between two consecutive jobs. It is the case for some single-machine problems ($1||L_{max}$, $1||\sum C_j$, ...) and for other two-machine flow shop scheduling problems [11].

A future research direction is to go further in the complexity study of the problems and to succeed to prove that they are actually NP-hard for the $F2||C_{max}$.

It is unlikely, however, that this method will be helpful for counting the number of optimal solutions. Indeed, counting the number of predecessors of a sequence in the lattice is equivalent to counting the linear extensions of a partial order, which is #P-complete [16]. Furthermore, one needs to take into account the intersection of sets of predecessors, which makes the problem even more difficult.

References

1. M.-A. Aloulou and C. Artigues. Worst-case evaluation of flexible solutions in disjunctive scheduling problems. In *International Conference on Computational Science and its Applications (ICCSA 2007)*, Kuala Lumpur, Malaysia, 2007.
2. C. Artigues, J.-C. Billaut, and C. Esswein. Maximization of solution flexibility for robust shop scheduling. *European Journal of Operational Research*, 165(2):314–328, 2005.
3. C. Artigues, F. Roubellat, and J.-C.. Billaut. Characterization of a set of schedules in a resource-constrained multi-project scheduling problem with multiple modes. *International Journal of Industrial Engineering*, 6(2):112–122, 1999.
4. O. Bailleux and P. Marquis. DISTANCE-SAT: Complexity and Algorithms. In *AAAI*, pages 642–647, 1999.
5. J.C. Beck. Solution-Guided Multi-Point Constructive Search for Job Shop Scheduling. *JAIR*, 29:49–77, 2007.

6. M. Bennett and G. Birkhoff. Two families of Newman lattices. *Algebra Universalis*, 32(1):115–144, 1994.
7. J.-C. Billaut and P. Lopez. Enumeration of all optimal sequences in the two-machine flowshop. In *Computational Engineering in Systems Applications (CESA'98), Symposium on industrial and manufacturing systems, IMACS / IEEE-SMC*, pages 378–382, Hammamet, Tunisie, April 1998.
8. J.-C. Billaut and P. Lopez. Characterization of all rho-approximated sequences for some scheduling problems. In *Emerging Technologies and Factory Automation (ETFA'2011)*, Toulouse, September 2011.
9. J.-C. Billaut, A. Moukrim, and E. Sanlaville, editors. *Scheduling with Flexibility and Robustness*. ISTE Ltd, Wiley, London, 2008.
10. J.-C. Billaut and F. Roubellat. A new method for workshop real time scheduling. *International Journal of Production Research*, 34(6):1555–1579, 1996.
11. J-L. Bouquard, C. Lenté, and J-C. Billaut. Application of an optimization problem in max-plus algebra to scheduling problems. *Discrete Applied Mathematics*, 154(15):2041–2238, 2006.
12. F. Boussemart, F. Hemery, C. Lecoutre, and L. Sais. Boosting Systematic Search by Weighting Constraints. In *ECAI*, pages 482–486, 2004.
13. V. Bowman. Permutation polyhedra. *SIAM Journal on Applied Mathematics*, 22(4):580–589, 1972.
14. C. Briand, M.-J. Huguet, H.T. La, and P. Lopez. Constraint-based approaches for robust scheduling. In *Scheduling with Flexibility and Robustness*, pages 199–226. ISTE Ltd, Wiley, 2008.
15. C. Briand, H.T. La, and J. Erschler. A robust approach for the single machine scheduling problem. *Journal of Scheduling*, 10(3):209–221, 2007.
16. G. Brightwell and P. Winkler. Counting linear extensions. *Order*, 8:225–242, 1991.
17. J. Erschler, G. Fontan, C. Mercé, and F. Roubellat. A new dominance concept in scheduling n jobs on a single machine with ready times and due dates. *Operations Research*, 31:114–127, 1983.
18. D. Grimes and E. Hebrard. Job Shop Scheduling with Setup Times and Maximal Time-Lags: A Simple Constraint Programming Approach. In *CPAIOR*, pages 147–161, 2010.
19. D. Grimes, E. Hebrard, and A. Malapert. Closing the Open Shop: Contradicting Conventional Wisdom. In *CP*, pages 400–408, 2009.
20. S. M. Johnson. Optimal two- and three-stage production with setup times included. *Naval Research Quarterly*, 1:61–68, 1954.
21. C. Lecoutre, L. Sais, S. Tabary, and V. Vidal. Nogood Recording from Restarts. In *IJCAI*, pages 131–136, 2007.
22. G. Markowsky. Permutation lattices revised. *Mathematical Social Sciences*, 27(1):59–72, 1994.
23. F. Roubellat, J.-C. Billaut, and M. Villaumié. Ordonnancement d'ateliers : d'Orabaid à Ordo (in French). *Revue d'Automatique et de Productique Appliquées*, 8(5):683–713, 1995.
24. I. Sabuncuoğlu and M. Bayiz. Analysis of reactive scheduling problems in a job shop environment. *European Journal of Operational Research*, 126(3):567–586, 2000.
25. I. Sabuncuoğlu and O. Kizilisik. Reactive scheduling in a dynamic and stochastic fms environment. *International Journal of Production Research*, 41(17):4211–4231, 2003.

26. S. Sevastyanov and B.M.T. Lin. Efficient enumeration of optimal and approximate solutions of the two-machine flow-shop problem. In *10th Workshop on Models and Algorithms for Planning and Scheduling Problems (MAPSP'11)*, Nymburk, Czech Republic, June 2011.
27. V. T'kindt, J.N.D. Gupta, and J.-C. Billaut. Two-machine flowshop scheduling with a secondary criterion. *Computers and Operations Research*, 30(4):505–526, 2003.
28. T. Walsh. Search in a Small World. In *IJCAI*, pages 1172–1177, 1999.