Flexible solutions in disjunctive scheduling: general formulation and study of the flow-shop case

Mohamed Ali Aloulou, Christian Artigues

Flexible solutions in disjunctive scheduling: general formulation and study of the flow-shop case

Mohamed Ali Aloulou[1] Christian Artigues[2]

[1]LAMSADE – Université Paris Dauphine
Place du Maréchal de Lattre de Tassigny, 75775 Paris Cedex 16, France
aloulou@lamsade.dauphine.fr,

[2]Université de Toulouse, LAAS-CNRS, 7 avenue du Colonel Roche, 31077 Toulouse, France
artigues@laas.fr

Abstract

We consider the context of decision support for schedule modification after the computation off-line of a predictive optimal (or near optimal) schedule. The purpose of this work is to provide the decision-maker a characterization of possible modifications of the predictive schedule while preserving optimality. In the context of machine scheduling, the anticipated modifications are changes in the predictive order of operations on the machines. To achieve this goal, a flexible solution feasible w.r.t to operations deadlines, is provided instead of a single predictive schedule. A flexible solution represents a set of semi-active schedules and is characterized by a partial order on each machine, so that the total order can be set on-line, as required by the decision maker. A flexible solution is feasible if all the complete schedules that can be obtained by extension are also feasible.

In this paper we develop two main issues. The first one concerns the evaluation of a flexible solution in the worst case allowing to certify if the solution is feasible. The second issue is the computation of feasible (w.r.t deadlines) flexible solutions of maximal flexibility imposed by the decision maker. Under an *epsilon*-constraint framework, solving this problem allows to find compromise solutions for the flexibility criterion and any minmax regular scheduling criterion. The special case of the flow-shop scheduling problem is studied and computational experiments are carried out.

1 Introduction

We consider a general non preemptive disjunctive scheduling problem in which a set of operations has to be scheduled on a set of machines, each operation requiring a fixed single machine during its execution and each machine being able to process only one operation simultaneously. The operations are linked by precedence constraints such that if operation i precedes operation j then the start time of j must be greater than the completion time of i. Such a model encompasses the standard flow-shop and job-shop models.

An important issue in scheduling concerns the support provided to the end-user(s) for on-line schedule execution after the off-line scheduling phase, which consists in providing an optimal or suboptimal schedule.

We consider in this paper a bicriteria disjunctive scheduling problem. Assuming that a non decreasing function is associated with each job completion time, the first considered objective function is defined as the maximum of these functions and has to be minimized. Being itself

non decreasing in the job completion times, this objective function is called a regular minmax objective function. In scheduling, the most studied regular minmax objective functions are the makespan and the maximum lateness. The second criterion involves the maximization of a flexibility objective function, for on-line decision support.

As soon as a regular minmax objective function is considered, the support for on-line scheduling most often lies in representing the solution as a m-vector of sequences of operations (total operation orders), where m is the number of machines, and for each operation an earliest and a latest start time yielding operation slacks. The sequences and the time windows are such that scheduling the operations in the predetermined order and inside their time windows is feasible and keeps the objective function under an acceptable threshold defined by the latest completion times. Such a flexibility provided to the end-user is referred to as time flexibility. This paper addresses the problem of providing more flexibility than the classical time one in disjunctive scheduling problems.

As already considered in previous studies [1, 2, 3, 4, 5, 6, 7], this can be achieved by providing sequential flexibility. Sequential flexibility lies in defining only a partial order of the operations on each machine, leaving to the end-user the possibility to make the remaining sequencing decisions. We define a flexible solution of a disjunctive problem as a m-vector of partial operation orders. This is the principle of the groups of permutable operations model that has been studied by several authors [2, 3, 5, 6], also called ordered assignment model by [7] and presented in more details in Section 3. However, the group model sets restrictions on the proposed partial orders that we relax in this paper. Indeed we represent the partial orders through additional precedence constraints between operations of the same machine, which allows representing any partial order. The second objective function, measuring the provided sequential flexibility could be defined as the number of feasible total order vectors extending the partial order vector of the flexible solution. However, the problem of computing this number is #P-complete [8]. Consequently we will propose in this paper another flexibility measure linked to the disjunctive graph representation.

A flexible solution allows to postpone some sequencing decisions and to hedge against some small to medium disruptions (raw material unavailability, small breakdowns, ...) with a minimum effort of computations [1, 9, 7]. It also permits to take into account some decision maker preferences that cannot be simply modeled or that may render the problem difficult to solve. For example, if two jobs can be executed in any order, without any influence on the first objective function, then the decision maker may prefer to sequence first the job that can be processed rapidly to avoid the starvation of the next machine, or the job with a maximum number of successors to maintain enough flexibility in the future. He may also favor a job belonging to a privileged client or a job that has to be sent to a downstream shop for further processing.

Providing a flexible solution through partial orders is useful in practice only if the problem of evaluating the complete solutions that can be obtained by extension is tractable. More precisely, given a reasonable decision policy followed by the decision maker, the following questions have to be answered. Do there remain decisions (following the decision policy) leading to a feasible schedule ? What is the worst first objective function value reachable by the remaining set of decisions ? Answering these questions provides a performance guarantee for the first criterion if the given on-line policy is followed.

The main problem considered in this paper, denoted (P), is the bicriteria problem of finding a compromise flexible solution between the first (regular minmax) criterion and the second (flexibility) criterion. To solve (P) we need to answer the above-defined questions by solving the following subproblem, denoted (SP), which concerns the evaluation of a flexible solution with respect to the problem constraints and the first objective function. Given an on-line decision policy and a flexible solution, problem (SP) is to decide whether all total order vectors reachable

by the decision policy are feasible w.r.t. problem constraints and, in the case of a "yes" answer, to compute the maximal value of the first objective function among all reachable total orders.

The paper is organized as follows. In Section 2, we define formally problems (SP) and (P), for a general disjunctive scheduling problem and the semi-active on-line scheduling policy. In Section 3, we discuss the previous related work and show that our study provides a general framework to the majority of previous studies on representation and evaluation of flexible solutions in disjunctive problems [10, 2, 3, 4, 5, 6, 7]. In Section 4, we reformulate problem (SP) as a elementary constrained longest path problem. Using previous results on the group of permutable operations model, we provide sufficient conditions on the precedence constraints under which (SP) can be solved in polynomial time. In Section 5, we provide a polynomial time dynamic programming algorithm to solve (SP) if other conditions on the precedence constraints, not implied by the preceding ones, are met, which is the case for the flow-shop precedence model. In Section 6, we propose an $\epsilon-$constraint method [6, 11, 12, 13] to solve (P) for the flow-shop case. The method is optimal for the regular minmax criterion but tackles heuristically the flexibility criterion. Computational results are provided on standard flow-shop instances issued from the literature [14].

2 Problem and subproblem setting

We consider the following disjunctive scheduling problem. There is a set $N = \{1, \ldots, n\}$ of operations to be scheduled on m machines. m_i, p_i and r_i denote the machine, processing time and release date of operation i, respectively. The release date is the earliest time when the operation processing can start. Each operation is associated with a non-decreasing cost function $f_i(C_i)$ of its completion time C_i. We introduce two dummy operations 0 and $n+1$ such that $p_0 = p_{n+1} = 0$. This problem is represented by a disjunctive graph $\mathcal{G} = (V, C, D)$ [15]. V is the set of vertices corresponding to operations $i \in N$ and the two dummy operations 0 and $n+1$. C is the set of conjunctive arcs representing the precedence constraints between the operations. Each conjunctive arc (i, j) is valuated by p_i. D is a set of pairs of disjunctive arcs $\{(i, j), (j, i)\}$ for each pair of operations $i, j \in N$ requiring the same machine for their execution ($m_i = m_j$). We have $D = \{\{(i, j), (j, i)\} | i \neq j \text{ and } m_i = m_j\}$. Arc (i, j) represents the decision to sequence i before j, whereas arc (j, i) represents the decision to sequence j before i on the machine. In the remaining a pair of disjunctive arcs $\{(i, j), (j, i)\}$ is called a disjunction and denoted by e_{ij} or e_{ji}.

Let $\mathcal{D}$ denote the set of all disjunctive arcs, i.e. $\mathcal{D} = \{(i, j) | e_{ij} \in D\}$. A selection π is a (possibly empty) set of arcs such that $\pi \subseteq \mathcal{D}$ and $|\pi \cap e_{ij}| \leq 1$, for all $e_{ij} \in D$. Let $D(\pi) = \{e_{ij} \in D | e_{ij} \cap \pi = \emptyset\}$. A selection is complete if $D(\pi) = \emptyset$, otherwise it is partial. The disjunctive graph issued from a complete or partial selection π is denoted by $\mathcal{G}(\pi) = (V, C \cup \pi, D(\pi))$. Given a set of arcs E, let $G(E)$ denote graph $(V, C \cup E)$. A complete selection π is feasible if the graph $G(\pi) = (V, C \cup \pi)$ is acyclic. The completion time $C_i(\pi)$ of any operation $i \in N$ in the semi-active schedule derived from the complete feasible selection π is equal to the length of the longest path in $G(\pi)$ from 0 to i plus p_i. Let $\Pi(D)$ denote the set of feasible complete selections given a set D of disjunctions. The objective of the classical scheduling problem is to find a complete feasible selection $\pi \in \Pi$ such that a *regular* minmax objective function $F(C_1(\pi), \ldots, C_n(\pi)) = \max_{i=1,\ldots,n} f_i(C_i(\pi))$ is minimized.

Here, we assume the decision maker makes on-line the remaining sequencing decisions on each machine following a semi-active policy until obtaining a complete selection π. A feasible semi-active schedule can be obtained by a list scheduling algorithm as soon as C is acyclic: sort the operations in a non decreasing order of their level in $G = (V, C)$ then sequence as soon as possible on its machine each operation according to this order. The first problem tackled in this

paper is the following maximization problem (SP) : Given a disjunctive graph $\mathcal{G} = (V, C, D)$, what is the worst case objective function value over all feasible semi-active schedules, i.e compute $\max_{\pi \in \Pi(D)} F(C_1(\pi), \ldots, C_n(\pi))$?

To illustrate this problem, consider the following 2-machine and 4-job flow-shop problem. This gives 8 operations and the flow-shop context sets $m_1 = m_3 = m_5 = m_7 = 1$ and $m_2 = m_4 = m_6 = m_8 = 2$. Furthermore, we have $p_1 = 1$, $p_2 = 6$, $p_3 = 2$, $p_4 = 5$, $p_5 = 4$, $p_6 = 6$, $p_7 = 6$ and $p_8 = 1$. All release dates are equal to 0 except for $r_5 = 2$. The considered criterion is the makespan. Let us consider additional precedence constraints $\{(1,3),(1,5),(1,7),(3,7),(5,7),(2,4),(2,6),(6,8),(2,8)\}$. We obtain the disjunctive graph $\mathcal{G}$ displayed in Figure 1.

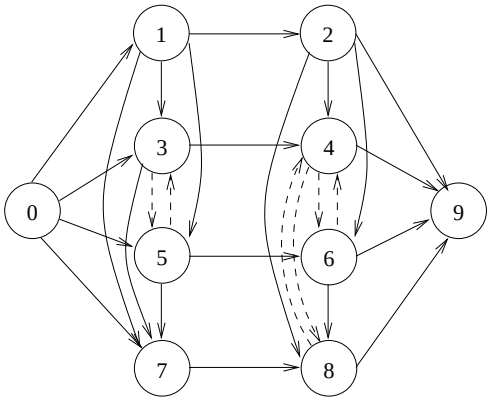

Figure 1: The disjunctive graph for a partial selection

The precedence constraints restrict the possible sequences to $(1,3,5,7)$ and $(1,5,3,7)$ on machine 1 and $(2,4,6,8)$, $(2,6,4,8)$ and $(2,6,8,4)$ on machine 2. We obtain the 6 schedules displayed in Figure 2. The solution of problem (SP) is 20 which is the worst-case makespan value of the 6 semi-active schedules. Note that the optimal makespan of the flow-shop problem is 19.

The second problem considered in this paper (P) consists in maximizing the flexibility of the solution represented by a partial selection π subject to operation deadlines. Let π denote a partial selection. π is worst-case feasible if its associated disjunctive graph $\mathcal{G}(\pi) = (V, C \cup \pi, D(\pi))$ is such that for each operation i, $\max_{\pi' \in \Pi(D(\pi))} C_i(\pi') \leq \tilde{d}_i$ where $\tilde{d}_i$ denotes the deadline of operation i.

In order to select among feasible flexible solutions, we have to propose a measure of flexibility. The number of characterized semi-active schedules is naturally a good measure. However, as already underlined, the problem of computing this number is #P-complete [8]. For this reason, we measure the flexibility of a solution by the number of unselected disjunctions $|D|$. Indeed, this quantity represents the amount of remaining decisions that can be managed by the decision maker. Hence problem (P) can be stated as the problem of finding a worst-case feasible selection π such that $\max_{\pi' \in \Pi(D(\pi))} C_i(\pi') \leq \tilde{d}_i$ and $D(\pi)$ is maximized.

Note that solving (P) may require to solve a series of problems (SP) to evaluate the worst-case feasibility of the tentative selections. (P) can be seen as an $\epsilon-$constraint problem where objective $\min \max_{i=1,\ldots,n} f_i(C_i(\pi))$ appears as a constraint through deadlines $\tilde{d}_i$ while the second objective is the maximization of flexibility through the number of unselected disjunction $|D(\pi)|$. Hence solving (P) in an $\epsilon-$constraint framework allows finding a compromise between flexibility and performance [6].

In the above example, if we consider a deadline equal to 20, then the flexible solution rep-

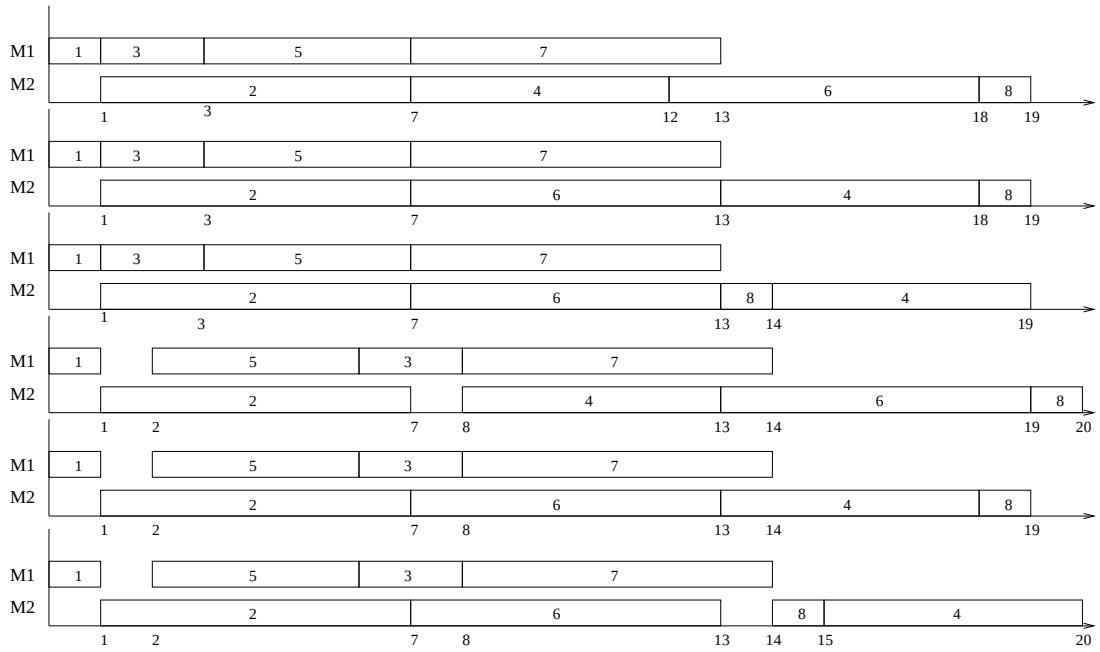

Figure 2: Sequences and semi-active schedules compatible with the partial selection

resented in Figure 1 is worst-case feasible and represents the 6 schedules displayed in Figure 2. The number of unselected disjunctions in this solution is $|D| = 3$.

3 Related work

Problem (SP) can be viewed as a maximization problem of an objective function that is naturally minimized. Several works have already been proposed in this domain. In particular, Aloulou, Kovalyov and Portmann [10] adapt the traditional three-field notation $\alpha|\beta|\gamma$ to this class of problems. They denote this family of considered maximization problems as $\alpha(sa)|\beta|(f \rightarrow \max)$. The first field α provides the shop environment. Here $\alpha \in \{1, F, J\}$ for respectively single machine (1), flow-shop (F) and job shop (J) problems. sa indicates that we search for the worst schedule among all semi-active schedules, which corresponds to the considered on-line scheduling policy. The second field gives additional constraints on operations. The third field contains information about the criterion to maximize.

To the best of our knowledge, the first related results we are aware are due to Posner [16]. Posner studied reducibility among single machine weighted completion time scheduling problems including minimization as well as maximization problems. In these problems, the jobs may have release dates and deadlines but there are no precedence constraints between the jobs. Besides, inserting idle times between the jobs is allowed.

Aloulou *et al* [10] studied several maximization versions in a single machine environment. They examined problems $1(sa)|\beta|(\gamma \rightarrow \max)$, where $\beta \subseteq \{r_i, prec\}$ and $\gamma \in \{f_{\max}, C_{\max}, L_{\max}, T_{\max}, \sum(w_i)C_i, \sum(w_i)U_i, \sum(w_i)T_i\}$. They showed that these problems are at least as easy as their minimization counterparts, except for problems $1(sa)||(\sum w_i T_i \rightarrow \max)$ and $1(sa)|r_i|(\sum w_i T_i \rightarrow \max)$, which are still open. In particular, problems $1(sa)|r_i, prec|(L_{\max} \rightarrow \max)$ and $1(sa)|r_i, prec|(T_{\max} \rightarrow \max)$ can be solved in $O(n^3)$ times while the minimization counterparts are strongly NP-hard, even if $prec = \emptyset$ [17].

This work is closely related to the work presented in the first part of the paper. Indeed, prob-

lem (SP) can be denoted, in the Aloulou *et al* notation, as $\alpha(sa)|r_i, prec|(f_{\max} \rightarrow \max)$. Besides, we propose in section 5 an algorithm solving the flow-shop problem $F(sa)|r_i, prec_k|(f_{\max} \rightarrow \max)$, generalizing the algorithm of Aloulou *et al* for problem $1(sa)|r_i, prec|(f_{\max} \rightarrow \max)$ [10]. Here $prec_k$ denotes precedence constraints appearing only between operations scheduled on the same machine, besides the classical flow-shop precedence constraints.

Another class of related work for both problems (SP) and (P) in the context of flexibility generation for on-line scheduling is linked to the concept of groups of permutable operations [3, 5], also called ordered (group) assignment [2, 7]. A group of permutable operations is a restriction of the sequential flexibility considered here in such a way that each operation is assigned to a group and there is a complete order between the groups of operations performed on the same machine. There are no precedence constraints between the operations of the same group. A pioneering work for the definition of the groups of permutable operations concept and the generation of flexible solutions has been achieved by Erschler and Roubellat [5] in the context of a job-shop problem with due dates. However no computational experiments were given to validate the practical interest of the approach. This has been achieved later but independently by Wu *et al* [7] who define the identical ordered assignment representation and propose an approach that computes an ordered assignment in a job-shop (i.e. ordered groups of permutable operations on each machine), focusing first on resolving a critical subset of scheduling decisions. As in the work of Erschler and Roubellat [5], the principle is to allow the remaining scheduling decisions to be made dynamically in the presence of disturbances. They show through numerical experiments on the weighted tardiness job-shop that this approach is superior to the one that generated a complete solution, in the presence of small to medium disturbances.

Other heuristics have been designed to generate groups of permutable operations for general disjunctive problems [3] and multiobjective methods have been designed to find a compromise between flexibility and performance in the two-machine flow-shop [6]. Artigues *et al* [2] establish the correspondence between the groups of Erschler and Roubellat [5] and the ordered assignment of Wu *et al* [7] and make a synthesis by calling this representation the ordered group assignment. They also propose a polynomial time algorithm to perform the exact worst-case evaluation of an ordered group assignment, i.e. to solve the corresponding problem (SP). This method is based on longest path computations in a so-called worst-case graph, derived from the considered ordered group assignment. This method solves problem (SP) for general disjunctive problems (e.g. job-shop) where the disjunctions appear only between operations of the same group (inside each group the graph of disjunctions e_{ij} is a clique) and when the precedence constraints are defined between operations of different groups. The flexibility of a solution is measured by the numbers of groups it contains. They propose a heuristic to minimize this number in the job-shop problem with deadlines, which corresponds to problem (P).

As an illustration of the differences between our model and the groups of permutable operations, the selection π proposed for the flow-shop example yielding the 6 feasible schedules of Figure 2 with a worst-case makespan of 20 cannot be represented by the groups of permutable operations formalism. More precisely, on machine 1 the considered partial order is a group partial order: operation 1 is in the first group, operations 3 and 5 are in the second group, and operation 7 is in the third and last group (there is a total order between the groups). However, there is no group partial order representing the situation on machine 2.

Recently Briand *et al* [4] have proposed to characterize a set of optimal schedules for the two-machine permutation flow-shop $F2|prmu|C_{\max}$ by means of interval structures. The interval structure provides a partial order which does not involve the restrictions of the concept of groups of permutable operations. Aloulou and Portmann [1] consider the single-machine scheduling problem with dynamic job arrival and total weighted tardiness and makespan as objective functions. They propose a multiobjective genetic algorithm to compute partial order

flexible solutions. The flexibility is measured by the number of unselected disjunctions.

In this paper we provide a general framework for these previous works by defining formally the problem of computing the worst-case completion times of the operations in the set of semi-active schedules compatible with a given operations partial orders. We show that this problem is polynomially solvable for the nonpermutation flow-shop and we provide a dynamic programming algorithm to solve it. We also integrate this algorithm in a method allowing to compute feasible flexible solutions.

4 A longest path formulation of the maximization problem (SP)

Let $\hat{C}_i$ denote the worst case completion time of operation i, i.e. $\hat{C}_i = \max_{\pi \in \Pi} C_i(\pi)$. Computing $\hat{C}_i$, for each $i \in N$ solves problem (SP) since the objective function is a minmax function of non decreasing functions of the completion times.

In any semi-active schedule represented by a feasible complete selection π, the completion time of an operation i is equal to the length of the longest path from 0 to i in the acyclic directed graph associated with π. Consequently, to compute the worst case completion times $\hat{C}_i$, a basic algorithm is to find for each operation i the feasible selection π with the longest path length between 0 and i. We show below that we can reduce the search to a partial selection.

Recall that $\mathcal{D}$ is the set of all disjunctive arcs. Let us consider the following Constrained Longest Path problem associated to operation i ($CLP(i)$).

Definition 1 *Given a disjunctive graph $\mathcal{G} = (V, C, D)$ and an operation i, problem $CLP(i)$ consists in computing the longest elementary path $L^*(0, i)$ from 0 to i in $G(\mathcal{D}) = (V, C \cup \mathcal{D})$ such that $G(L^*) = (V, C \cup L^*(0, i))$ is acyclic.*

We show now that solving ($CLP(i)$), $\forall i \in N$ solves problem (SP).

Theorem 1 *The worst case completion time $\hat{C}_i$ is equal to the length of path $L^*(0, i)$ solution of problem $CLP(i)$.*

Proof. We first show that (a) $\hat{C}_i$ is the length of an elementary path l from 0 to i in $G(\mathcal{D})$ and that $G(l)$ is acyclic. Let $\pi \in \Pi$ such that we have $\hat{C}_i = C_i(\pi)$. π is the complete selection such that $\hat{C}_i$ is the length of a longest path l from 0 to i in $G(\pi)$. Since $G(\pi)$ is acyclic, l is elementary and since $l \subseteq \pi \cup C$, $G(l)$ is also acyclic. Since $\pi \subset \mathcal{D}$, l is also an elementary path in $G(\mathcal{D}) = (V, C \cup \mathcal{D})$.

Let us show that (b) any elementary path L from 0 to i in $G(\mathcal{D})$ verifying $G(L)$ is acyclic is such that there exists a feasible complete selection $\pi \in \Pi$ verifying $C \cup L \subseteq C \cup \pi$. Suppose that L includes only conjunctive arcs. Then $L \subseteq C$ and (b) is verified. Suppose now that L includes also disjunctive arcs. Since L is elementary, we have $|L \cap e_{ij}| \leq 1$ for each disjunction e_{ij}. Hence $L \setminus (L \cap C)$ is a partial selection. Furthermore, since $G(L) = G(V, C \cup L)$ is acyclic $C \cup L$ defines a new acyclic precedence constraints graph and the disjunctive problem defined by $(V, C \cup L, D(L))$ is feasible. Hence $L \setminus (L \cap C)$ is included in a feasible complete selection.

From (b) it follows that the length of any elementary path L from 0 to i, verifying $G(L)$ is acyclic, is less than or equal to $\hat{C}_i$. We proved in (a) that there exists an elementary path l, such that $G(l)$ is acyclic, with a length equal to $\hat{C}_i$. Hence, $\hat{C}_i$ is the length of $CLP(i)$-solution. ■

We conjecture that $CLP(i)$ is not easy to solve in the general case. It admits as a particular case the search for the longest elementary path in a graph with positive length cycles. This problem is known to be NP-hard for general graphs [18]. Taking account of the acyclicity

constraint, the problem can be seen as a constrained longest path problem which is also NP-hard in a general case [19].

In Figure 3, we illustrate the problem and the necessity of the no-cycling condition in definition 1 for a job-shop with 2 machines, 3 jobs and no release dates. The processing time of each operation is equal to one. Operations 1, 4 and 5 are assigned to the first machine and operations 2, 3 and 6 are assigned to the second machine. Structural precedence constraints are $(1, 2)$, $(3, 4)$ and $(5, 6)$. $(3, 2)$ and $(1, 4)$ are additional precedence constraints.

The longest elementary path from 0 to 7 is displayed in bold. Its length is equal to 6. Such a path is infeasible since it induces a cycle with precedence constraint $(3, 4)$. Hence, it is not a solution of $CLP(7)$.

This can be interpreted by considering the operation sequences represented by the disjunctive graph of Figure 3 on each machine. On Machine 1, the represented sequences are $(1, 4, 5)$, $(1, 5, 4)$, and $(5, 1, 4)$ while on Machine 2 the represented sequences are $(3, 2, 6)$, $(3, 6, 2)$ and $(6, 3, 2)$. However there are only 8 feasible semi-active schedule since the combination of sequences $(1, 4, 5)$ and $(6, 3, 2)$ is inconsistent with the precedence constraints. The worst schedules have a duration of 5 and are given either by $\{(1, 4, 5), (3, 6, 2)\}$ or $\{(1, 5, 4), (6, 3, 2)\}$.

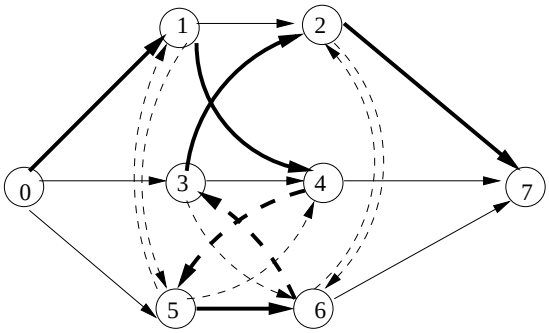

Figure 3: A job-shop example and an infeasible elementary longest path

The preceding studies on the group of permutable operations model allow us to define sufficient conditions that render (SP) solvable in polynomial time.

Theorem 2 *If the disjunctive arc pairs form a binary transitive relation on each set of operations assigned to the same machine, (SP) is solvable in polynomial time*

Proof. The model based on groups of permutable operations corresponds to the case where the absence of precedence constraint between two operations assigned to the same machine (i.e. the presence of a pair of disjunctive arcs between these operations), defines a binary transitive relation. Indeed the operations of a group are totally permutable while there is a precedence constraint between any operation of a group and any operation of the consecutive group on the considered machine. Conversely, any feasible partial selection such that disjunctive arc pairs form a binary transitive relation on each set of operations assigned to the same machine can be clearly represented by a group partial order on each machine. In [2], a polynomial time algorithm is precisely provided to solve problem (SP). This proves the theorem. ∎

Below we give a condition on the precedence constraints which do not restrict the possible partial orders under which the no cycling condition of each problems $CLP(i)$ is always verified by any longest elementary path.

Theorem 3 *Given a disjunctive graph $\mathcal{G}(V, C, D)$ such that $\mathcal{G}(V, C)$ is acyclic, if the machines can be renumbered so that $(i, j) \in C \implies m_i \leq m_j$ then any elementary path L starting with node 0 in $G(\mathcal{D}) = (V, C \cup \mathcal{D})$ is such that $G(L) = (V, C \cup L)$ is acyclic.*

Proof. A cycle can be generated with a precedence constraints only inside a strongly connected component of $G(\mathcal{D})$ (i.e. a set of nodes pairwise connected by a path). Due to the structure of the precedence constraints given by the condition, the strongly connected components of $G(\mathcal{D})$ include only operations assigned to the same machine. Hence any elementary cycle of $G(\mathcal{D})$ involves only operations assigned to the same machine. Let L denote an elementary path in $G(\mathcal{D})$. By definition L is acyclic and no cycle can be created by adding precedence constraints to L. ∎

We show in the next section that problem (SP) is polynomially solvable in the flow-shop context, for which the condition of theorem 3 holds.

5 A polynomial algorithm for solving (SP) in the nonpermutation flow-shop case

In this section, we consider the nonpermutation flow-shop problem with operation release dates and additional precedence constraints appearing only between operations scheduled on the same machine, as in the example presented is section 2. In this case, any sequence of operations compatible with the precedence constraints of machines yields a feasible complete selection as stated by theorem 3. For each operation j, let j^- denote its job predecessor. We assume that if j is the first operation of its job, then j^- is a dummy operation denoted j^0. Let Γ_j^- (resp. Γ_j^+) denote the set of operations that must be scheduled before (resp. after) j on machine m_j. Let I_j denote the set of operations of machine m_j that are not linked to j with any precedence constraint. We have

$$\Gamma_j^- = \{i \neq j | m_i = m_j \text{ and there is a path from } i \text{ to } j \text{ in } G = (V, C)\} \quad (1)$$

$$\Gamma_j^+ = \{i \neq j | m_i = m_j \text{ and there is a path from } j \text{ to } i \text{ in } G = (V, C)\} \quad (2)$$

$$I_j = \{i \neq j | m_i = m_j, i \notin \Gamma_j^- \text{ and } j \notin \Gamma_i^-\} \quad (3)$$

Let us define $\hat{C}_{j^0} = r_j$. We have the following result.

Lemma 1 *The worst case completion time of any operation j is given by*

$$\hat{C}_j = p_j + \max \begin{cases} r_j, & (a) \\ \hat{C}_{j^-}, & (b) \\ \max\limits_{i \in I_j \cup \Gamma_j^-} \{\max(r_i, \hat{C}_{i^-}) + \sum\limits_{x \in I_j \cup \Gamma_j^- \setminus \Gamma_i^-} p_x\} & (c) \end{cases}$$

Proof.

Consider machine 1 and an operation j to be executed on this machine. We have $\hat{C}_{j^-} = \hat{C}_{j^0} = r_j$, hence terms (a) and (b) are redundant. Denote by S the semi-active schedule in which $C_j(S) = \hat{C}_j$ and the block B of consecutive operations on machine 1, ending with j, is such that there is no idle time between any two consecutive operations in B and B is of maximal size. B always exists since we have at least $j \in B$. If $B = \{j\}$, then the starting time

of j, S_j, is such that $S_j = r_j$ and (a) is verified.

If $|B| > 1$, then we have $S_j \geq r_j$. Let i be the first operation of block B. i is not a successor of j on the machine and $i \in I_j \cup \Gamma_j^-$. Similarly, by definition all operations inside B, except j itself, belong to $I_j \cup \Gamma_j^- \setminus \Gamma_i^-$ (they cannot be predecessors of i on the machine). Let us now consider the operations scheduled before i on machine 1. Let x denote the operation scheduled at the largest position before i such that $x \notin \Gamma_i^-$. This operation could be inserted right after i. The obtained schedule is semi-active and the completion time of j increases, which contradicts the maximality of $C_j(S)$. Hence all operations scheduled before i are in Γ_i^-. This implies that all operations of $I_i \cup \Gamma_j^- \setminus \Gamma_i^-$ are scheduled after i. Conversely, suppose that x is the operation scheduled at the smallest position after j such that x is not a successor of j, i.e. $x \notin \Gamma_j^+$. This operation could be inserted right before j providing a new semi-active schedule in which the start time of j increases, which contradicts the maximality of $C_j(S)$. Hence all operations scheduled after j are successors of j. It follows that if $S_j > r_i$ then

$$\hat{C}_j \leq \max_{i \in I_j \cup \Gamma_j^-} \{r_i + \sum_{x \in I_j \cup \Gamma_j^- \setminus \Gamma_i^-} p_x\} + p_j. \tag{4}$$

Can we have $i \in I_j \cup \Gamma_j^-$ such that $\hat{C}_j < r_i + \sum_{x \in I_j \cup \Gamma_j^- \setminus \Gamma_i^-} p_x + p_j$?
Suppose that i is such an operation. It is possible to build a feasible semi-active schedule in which all the operations before i are machine predecessors of i and the operations after j are only machine successors of j. This can be made by scheduling the operations of Γ_i^- in an order compatible with the precedence constraints within this set, then i, then the operations of $I_i \cup \Gamma_j^+$ in an order compatible with the precedence constraints within this set, then operation j, then the operations of Γ_j^+ in an order compatible with the precedence constraints within this set. The operations on machine 2 can be scheduled in any order compatible with the precedence constraints of machine 2, and so on. The obtained schedule S is semi-active and we have $C_j \geq r_i + \sum_{x \in I_j \cup \Gamma_j^- \setminus \Gamma_i^-} p_x + p_j$. Hence (4) is verified to equality.

We can use the similar arguments to prove the result for any machine $k > 1$.

Consider a machine k and an operation to be executed on this machine. Let S be a semi-active schedule in which $C_j(S) = \hat{C}_j$ and the block B of operations consecutive on machine k, ending with j, is such that there is no idle time between any two consecutive operations in B and B is of maximal size. If $|B| = 1$ then we have either $C_j(S) = r_j + p_j$ or $C_j(S)$ is set by C_{j^-}. To maximize this value we have $C_j(S) = \hat{C}_{j^-} + p_j$.

If $|B| > 1$ we can also state that an operation $i \in I_j \cup \Gamma_j^-$ starts the block with $S_i = r_i$ or $S_i = C_{i^-}$. With similar arguments as for machine 1, we prove that all operations of the set $I_j \cup \Gamma_j^- \setminus \Gamma_i^-$ are in the block. Furthermore if $S_i > r_i$ then we have $S_j = C_{i^-} = \hat{C}_{i^-}$ to have $\hat{C}_j$ maximal. Last we can show that for any operation $i \in I_j \cup \Gamma_j^-$, we can build a feasible semi-active schedule in which all the operations before i are machine predecessors of i and the operations after j are machine successors of j and $S_i = \max(r_i, \hat{C}_{i^-})$. This achieves the proof. ∎

Let $\nu = n/m$ be the number of jobs. Due to lemma 1, we have the following result.

Theorem 4 *Problem $F(sa)|r_i, prec_k|(f_{\max} \rightarrow \max)$ can be solved in $O(m\nu^3)$ times if each function f_i is computable in $O(1)$ time.*

Proof. Once sets Γ_j^- and I_j are built for each operation j, all worst-case completion times can be computed trivially via the proposed recursion by dynamic programming in $O(m\nu^3)$. ∎

In the illustrative example of section 2, the worst case completion times are given (in the order of their computation) by $\hat{C}_1 = r_1+p_1 = 1$ (a), $\hat{C}_3 = r_5+p_5+p_3 = 8$ (c), $\hat{C}_5 = r_1+p_1+p_3+p_5 = 7$ (c), $\hat{C}_7 = r_5 + p_5 + p_3 + p_7 = 14$ (c), $\hat{C}_2 = \hat{C}_1 + p_2 = 7$ (b), $\hat{C}_4 = \hat{C}_7 + p_8 + p_4 = 20$ (c), $\hat{C}_6 = \hat{C}_3 + p_4 + p_6 = 19$ (c), $\hat{C}_8 = \hat{C}_3 + p_4 + p_6 + p_8 = 20$ (c).

6 A ϵ-constraint heuristic for solving the minimal makespan, maximal flexibility flow-shop problem

In this Section, we show how feasible flexible solutions can be computed for a flow-shop problem where the first objective function is the makespan and the second objective function is the number of unselected disjunctions. Contrarily to most references encountered in the literature we do not restrict the set of schedules to permutation schedules, where the order of the jobs has to be identical on all machines [20, 21, 22].

Under the ϵ-constraint framework, the method we propose is a heuristic aiming at maximizing the number of unselected disjunctions while all jobs have a common deadline corresponding to the desired makespan value.

In Section 6.1, we present a simple elementary heuristic that computes a feasible flexible solution. Taking a worst-case feasible selection as input, this heuristic issues a worst-case feasible selection with a non-larger number of unselected disjunctions. In Section 6.2, we present a branch-and-bound method whose aim is twofold. First, the method ensure the deadline constraint is satisfied and so works as an exact method for the first criterion. Second, the branch-and-bound scheme is used as a heuristic for the second criterion to compute several input worst-case selections allowing to apply WCH several times. Last, Section 6.3 provides computational experiments on standard flow-shop instances and illustrates the ability of the method to find a good compromise between flexibility and performance.

6.1 WCH: A simple heuristic to compute a flexible solution

Let π denote a worst-case feasible selection and consider its induced disjunctive graph $\mathcal{G}(\pi) = (V, C \cup \pi, D(\pi))$. Recall π is worst-case feasible if and only if the worst case completion time $\hat{C}_i$ of each operation verifies $\hat{C}_i \leq d_i$. The heuristic aims at finding a flexible solution with a maximal number of unselected disjunctions $|D|$ which represents the greatest amount of decisions let to the decision maker.

To this end, WCH modifies the disjunctive-graph so that the corresponding selection becomes minimally worst-case feasible, i.e. it does not include any orientated disjunctive arc (i, j) such that $\pi \setminus \{(i, j)\}$ is worst case feasible. The heuristic traverses all arcs (i, j) of π and checks whether the latter property is verified for (i, j). If such an arc (i, j) is found, it is removed from π and e_{ij} is consequently added to $D(\pi)$ which increases the objective function. The process is iterated until π becomes minimally worst-case feasible. Only those arcs (i, j) such that there is no other path from i to j (belonging to the transitive reduction of π) are considered for being removed. Otherwise, removing (i, j) does not remove any precedence constraint.

The order in which arcs are selected for being removed from π determines the resulting disjunctive graph. Hence several orders may result in different minimally worst-case feasible graphs. We call the above procedure with 100 randomly generated arc orders. The solution with the largest number of unselected disjunctions is returned.

6.2 Computing several worst-case feasible selections through branch-and-bound

Any exact or heuristic solution method to $F|\tilde{d}_i = d|-$ could be used to generate an input for WCH. To obtain several worst-case feasible selections (and so apply WCH several times), we solve problem $F|\tilde{d}_i = d|-$ through branch-and-bound and we call WCH each time a node corresponds to a worst-case feasible selection.

In this section, we briefly give the elements of the branch-and-bound, all borrowed from previous studies: the branching scheme (Section 6.2.1), the constraint propagation algorithms used at each node to sharpen the operation time windows (Section 6.2.2), the heuristic used at each node to try to find a feasible solution (Section 6.2.3). In Section 6.2.4 we explain how the WCH heuristic has been integrated in the branch-and-bound scheme.

6.2.1 Branching Scheme

The proposed branching scheme is based on the disjunctive graph. At each node the disjunctive graph is updated through the last branching decisions. The branching rules are based on the relative ordering of the operations assigned to the same machines. At each node a machine is selected and a child node is generated for each operation candidate for being scheduled next on the machine. The machine is selected as the one on which the operation with the smallest release date i^* is assigned. The candidates for being scheduled first on this machine are the operations with a release date not greater than the earliest completion time of i^*. All disjunctive arcs issued from the selected operation are then orientated as outgoing arcs for this operation and included in the selection π. Hence at each node, a disjunctive graph $\mathcal{G}(\pi) = (V, C \cup \pi, D(\pi))$. The tree is explored by depth-first search.

6.2.2 Constraint propagation

The common deadline d allows to compute a time window $[r_i, d_i]$ for each operation $i \in N$. Constraint propagation algorithms are used at each node to maintain the time window as tight as possible, to detect implied precedence constraints and to prune the node if inconsistency is proven. The release times r_i and the deadlines d_i are first computed with forward and backward longest path computations in $(V, C \cup \pi)$. Time window tightening, precedence constraint detection and consistency checking are performed by the disjunctive constraint propagation and edge-finding techniques. For a precise description of these techniques, we refer to [23, 24].

Furthermore at the root node initial time windows are computed by the shaving technique [25, 26] which consists in running the above-referred constraint propagation algorithms after setting the start time of an operation to its release date (or to its deadline). If inconsistency is proven, the tentative value can be removed from the time window. Such a technique has been proven very useful for flow-shop problems [27].

6.2.3 Heuristic

In the case where the current node has not been pruned by constraint propagation, a heuristic is used to find a feasible solution. The heuristic is simply based on the application of the well-known priority-rule based active and non-delay constructive algorithms [28]. At each node, we apply 4 times the non-delay scheduling algorithms and 4 times the active scheduling algorithm. The 4 used priority rules are the minimal earliest possible start (r_i), the minimal lastest start ($d_i - p_i$) and the randomized version of these rules where another operation than the one determined by the rule is selected with a low probability.

When no feasible solution has been found, the solution with the lowest makespan is kept. Each time one of the 8 constructive methods improves the best known solution in terms of makespan, an intensification phase is applied by running $H1$ times the randomized version of the priority rule that yielded the improvement with both active and non-delay algorithms.

6.2.4 Integration of worst-case completion times

Each time a node corresponds to a worst-case feasible graph selection π, WCH is called to increase $|D(\pi)|$. Although at this time, problem $F|\tilde{d}_i = d|-$ is solved, the search process continues to find further flexible solutions and improves the flexibility criterion. The branch-and-bound stops when this process has been applied 100 times, which corresponds to a total of 10000 calls to the WCH procedure, or when there is no more node to develop. Throughout this process the flexible solution with the largest $|D|$ is stored. Recall that the branch-and-bound is an exact method w.r.t problem $F|\tilde{d}_i = d|-$ and serves as a generator of worst-case feasible selections for WCH. The number of unselected disjunctions of the flexible solutions is consequently heuristically maximized only.

6.3 Experimental results

In this section we give the performance of the branch-and-bound on flow-shop instances issued from the literature, in terms of issued flexibility.

Because of the difficulty of the non permutation flow-shop problem problem, we have modified the smallest instances designed by Taillard [14], which originally comprise 10 problems with 20 jobs and 5 machines, to keep only the 10 first jobs in each. All programs have been coded in C++ and run on an AMD64 archicture under Linux. Cplex 9.0 was used to solve the LP relaxations and the MILP problems. Parameters H1 was set to 50000 iterations.

We have made 2 series of experiments, one with the common deadline of each instance set to the optimal makespan and the other one with the common deadline set to the optimal makespan augmented by 5%. The results are given in Table 1 for the instances with the tight common deadline and in Table 2 for the instances with the loose common deadline. In both tables, we give for each instance the number of jobs, the number of machines, the minimal makespan, the common deadline, the largest obtained number of unselected disjunctions of the flexible solutions (also expressed in percentage of the total number of disjunctions, equal to 225). We also provide the numbers of nodes and the CPU times in seconds needed to obtain the first feasible solution and the numbers of nodes and CPU times needed to obtain the flexible solution. The results show that in both cases, flexible solutions are exhibited with a reasonable amount of additional CPU time. As expected the flexibility is higher for the instances with a loose common deadline but significant flexibility is also generally obtained for the instances with the tight deadline.

7 Conclusion

In this paper, we proposed a longest path formulation of the problem of evaluating the worst case performance of flexible solutions in disjunctive scheduling with any minmax regular objective function. A flexible solution is defined by an operation partial order on each machine. We proved that this problem is polynomially solvable in the special case of the flow-shop problem with release dates and additional precedence constraints between operations scheduled on the same machine. We used the worst case computation method to generate flexible solutions for a flow-shop problem with a common deadline. We show that flexible solutions with a significant

Table 1: Results on instances with a tight common deadline

P	#jobs	m	$C^*_{\max}$	d	$\lvert D\rvert$	#nodes 1st	CPU 1st	#nodes	CPU
1	10	5	767	767	3 (1.3%)	5286	18	6614	21
2	10	5	763	763	6 (2.6%)	7	0	1660	40
3	10	5	691	691	7 (3.1%)	12	0	837	52
4	10	5	813	813	3 (1.3%)	1	0	33	11
5	10	5	731	731	8 (3.5%)	4666	12	7295	118
6	10	5	749	749	9 (4%)	267	0	6373	20
7	10	5	741	741	11(4.8%)	1676	10	3254	16
8	10	5	717	717	4 (1.7%)	58	6	1026	12
9	10	5	687	687	7 (3.1%)	1	0	62	27
10	10	5	762	762	17(7.5%)	14	0	499	150

Table 2: Results on instances with a loose common deadline

P	#jobs	m	$C^*_{\max}$	d	$\lvert D\rvert$	#nodes 1st	CPU 1st	#nodes	CPU
1	10	5	767	805	14 (6.2%)	180	0	3500	146
2	10	5	763	801	9 (4%)	1	0	758	70
3	10	5	691	725	11 (4.8%)	1	0	2778	105
4	10	5	813	853	10 (4.4%)	1	0	4805	109
5	10	5	731	767	11 (4.8%)	2	0	34104	177
6	10	5	749	786	16 (7.1%)	1	0	3708	162
7	10	5	741	778	22 (9.7%)	21	0	13748	220
8	10	5	717	752	9 (4%)	16	0	704	90
9	10	5	687	721	21 (9.3%)	4	0	7807	186
10	10	5	762	800	19 (8.44%)	1	0	515	193

flexibility are obtained with a reasonable computational overhead with partial solutions leaving unselected up to 10% of the disjunctions when the deadline is loose and up to 7% of the disjunctions when the deadline is tight.

Besides their interest for on-line decision support, flexible solutions could also be used in bicriteria scheduling as a support to $\epsilon-$constraint methods with other criteria than flexibility maximization. This has been underlined by Gupta and Stafford in [29], about the work of Briand *et al* [4]. Indeed, once a set of schedules achieving a required worst-case value on the first (regular minmax) criterion, the optimal solution for the second criterion can be searched on this set without considering any constraint on the first criterion.

Another work of interest would be to focus on extensions of the worst-case performance evaluation procedure to more general problems. Unfortunately, extending this approach to the job shop is not trivial. A way to solve it is to study the complexity of the problem of finding the constrained longest elementary path in the disjunctive graph of the job-shop problem. It is also of great interest to investigate maximization problems with total (weighted) flow time as objective function. Indeed, the flow-shop problem $F(sa)|r_j|(\sum C_j \rightarrow \max)$ is open whereas the single machine $1(sa)|r_j|(\sum w_j C_j \rightarrow \max)$ is polynomially solvable [10].

Last, we have to underline, as already stated in [2] among others, the fact that maximizing flexibility remains an indirect way for schedule robustness under uncertainty. An important open issue is to first quantify and second maximize the ability of a flexible solution to absorb different classes of disruptions.

References

[1] M.A. Aloulou and M.-C. Portmann. An efficient proactive-reactive scheduling approach to hedge against shop floor disturbance. In G. Kendall, E.K. Burke, S. Petrovic, and M. Gendreau, editors, *Multidisciplinary Scheduling: Theory and Applications 1st International Conference, MISTA '03 Nottingham, UK, 13-15 August 2003. Selected Papers*, pages 223–246. Elsevier, 2005.

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

[4] C. Briand, H.T. La, and J. Erschler. A new sufficient condition of optimality for the two-machine flowshop problem. *European Journal of Operational Research*, 169(3):712–722, 2006.

[5] J. Erschler and F. Roubellat. An approach for real time scheduling for activities with time and resource constraints. In R. Slowinski and J. Weglarz, editors, *Advances in project scheduling*. Elsevier, 1989.

[6] C. Esswein, J.C. Billaut, and V. Strusevich. Two-machine shop scheduling: Compromise between flexibility and makespan value. *European Journal of Operational Research*, 167(3):796–809, 2005.

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

[8] G. Brightwell and P. Winkler. Counting linear extensions. *Order*, 8:225–242, 1991.

[9] W. Herroelen and R. Leus. Project scheduling under uncertainty: Survey and research potentials. *European Journal of Operational Research*, 165(2):289–306, 2005.

[10] M. Aloulou, M. Kovalyov, and M.C. Portmann. Maximization in single machine scheduling. *Annals of Operations Research*, 129:21–32, 2004.

[11] V. Chankong and Y. Haimes. *Multiobjective decision making theory and methodology*. Elsevier, 1983.

[12] H. Hoogeveen. Multicriteria scheduling. *European Journal of Operational Research*, 167(3):592–623, 2005.

[13] V. T'Kindt and J.-C. Billaut. *Multicriteria Scheduling*. Springer, 2 edition, 2006.

[14] E.D. Taillard. Benchmarks for basic scheduling problems. *European Journal of Operational Research*, 64:278–285, 1993. available at http://ina2.eivd.ch/collaborateurs/etd/problemes.dir/ordonnancement.dir/ordonnancement.html (visited on April, 3 2006).

[15] B. Roy and B. Sussmann. Les problèmes d'ordonnancement avec contraintes disjonctives, 1964. D.S. vol. 9, SEMA, Paris, France.

[16] M. E. Posner. Reducibility among wighted completion time scheduling problems. *Annals of Operations Research*, pages 91–101, 1990.

[17] J. K. Lenstra, A. H. G. Rinnooy Kan, and P. Brucker. Complexity of machine scheduling problems. *Annals of Discrete Mathematics*, 1, 1977.

[18] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman, 1979.

[19] G. Y. Handler and I. Zang. A dual algorithm for the constrained shortest path problem. *Networks*, 10(4):293–310, 1980.

[20] J. Lemesre, C. Dhaenens, and E.G. Talbi. An exact parallel method for a bi-objective permutation flowshop problem. *European Journal of Operational Research*, 177(3):1641–1655, 2007.

[21] M. F. Tasgetiren, Y.-C. Liang, M. Sevkli, and G. Gencyilmaz. A particle swarm optimization algorithm for makespan and total flowtime minimization in the permutation flowshop sequencing problem. *European Journal of Operational Research*, in press.

[22] V. T'Kindt, J.N.D. Gupta, and J.-C. Billaut. Two-machine flowshop scheduling with a secondary criterion. *Computers and Operations Research*, 30:505–526, 2003.

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

[24] Ph. Baptiste, C. Le Pape, and W. Nuijten. *Constraint-Based Scheduling*. Kluwer Academic Publishers, 2001.

[25] P. Martin and D.B. Shmoys. A new approach to computing optimal schedules for the job-shop scheduling problem. In *5th International IPCO Conference*, pages 389–403, 1996.

[26] J. Carlier and E. Pinson. Adjustment of heads and tails for the job-shop problem. *European Journal of Operational Research*, 78:146161, 1994.

[27] L. Peridy, E. Pinson, and D. Rivreau. Enhanced disjunctive elimination rules for the flow-shop and permutation flow-shop problems. In *6th International Workshop on Project Management and Scheduling*, Istanbul, 1998.

[28] K. R. Baker. *Introduction to sequencing and scheduling.* Wiley, 1974.

[29] J.N.D. Gupta and E. F. Stafford. Flowshop scheduling research after five decades. *European Journal of Operational Research*, 169(3):699–711, 2006.