

AN EFFICIENT PROACTIVE REACTIVE SCHEDULING APPROACH TO HEDGE AGAINST SHOP FLOOR DISTURBANCES

Mohamed Ali Aloulou

Laboratoire d'Informatique de Paris 6

8, rue du Capitaine Scott

75015 Paris, France

aloulou@lip6.fr

Marie-Claude Portmann

MACSI team of INRIA-Lorraine and LORIA-UHP-INPL

Ecole des Mines de Nancy Parc de Saurupt,

54042 Nancy Cedex, France

portmann@loria.fr

Abstract We consider the single machine scheduling problem with dynamic job arrival and total weighted tardiness and makespan as objective functions. The machine is subject to disruptions related to late raw material arrival and machine breakdowns. We propose a proactive-reactive approach to deal with possible perturbations. In the proactive phase, instead of providing only one schedule to the decision maker, we present a set of predictive schedules. This set is characterized by a partial order of jobs and a type of associated schedules, here semi-active schedules. This allows to dispose of some flexibility in job sequencing and flexibility in time that can be used on-line by the reactive algorithm to hedge against unforeseen disruptions. We conduct computational experiments which prove that our approach outperforms a predictive reactive approach particularly for disruptions with low to medium amplitude.

Keywords: scheduling, single machine, flexibility, robustness, total weighted tardiness, genetic algorithms.

1. Introduction

Scheduling is an important element of production systems because it allows improving the performance of the system and serves as an overall plan on which many other shop activities are based. Several techniques have been proposed to generate, for a given problem, a unique schedule satisfying the shop constraints and providing optimal or near optimal performance. However, when this pre-computed or predictive schedule is released for execution, continual adapting is required to take into account the presence of uncertainties. These uncertainties are related for example to machine breakdowns, staffing problems, unexpected arrival of new orders, early or late arrival of raw material and uncertainties in the duration of processing times [9]. When the first perturbations arise, the schedule is slightly modified and the performance is a little bit affected. But when there are more important perturbations, the performance of the final schedule becomes generally much worse than the initial one. Besides, the predictive schedule is also used as a basis for planning activities, such as raw material procurement, preventive maintenance and delivery of orders to external or internal customers [16]. Consequently, if the obtained schedule deviates considerably from the initial predictive schedule, then this may delay the execution of many activities related to internal customers. It may also add some costs due to early procurement of raw material from suppliers or late delivery of finished products to external customers. Therefore, we are interested in this paper in developing an approach that builds predictive schedules providing a sufficiently detailed sketch of the schedule to serve as basis for other planning requirements and retain enough flexibility in order to hedge against perturbations that may occur on-line and/or minimize their effects on planned activities.

Many approaches of scheduling and rescheduling have been proposed in literature to take into account the presence of uncertainties in the shop floor. They can be classified in four categories : completely reactive scheduling, predictive-reactive scheduling, proactive scheduling and proactive-reactive scheduling, see for example [9] and [11].

“Completely reactive approaches are based on up-to-date information regarding the state of the system” [9]. No predictive schedule is given to the shop floor and the decisions are made locally in real time by using priority-dispatching rules. These approaches are used when the level of disturbances is always important or when the data are known very late making impossible the computation of predictive schedules. In predictive-reactive approaches, a predictive schedule is generated without considering possible perturbations. Then, a reactive algorithm is

used to maintain the feasibility of the schedule and/or improve its performances, see for example [8] and [19]. The goal of proactive or robust scheduling is to take into account possible disruptions while constructing the original predictive schedule. This allows to make the predictive schedule more robust. A robust schedule is defined by Leon *et al* [15] as a schedule that is insensitive to unforeseen shop floor disturbances given an assumed control policy. This control policy is generally simple. Clearly, robust scheduling is appropriate only if, while generating the predictive schedule, the uncertainty is known or at least some suspicions about future are given thanks to the experience of the decision maker. If the uncertainty is completely unknown, a reactive scheduling approach is more suitable.

“A scheduling system that is able to deal with uncertainty is very likely to employ both proactive and reactive scheduling” [9]. Indeed, it is very difficult to take into account all unexpected events while constructing the predictive schedule. Consequently, a reactive algorithm more elaborate than those used in proactive or robust approaches should be used to take into account these improbable events. However, due to the constraints on the response time of the reactive algorithm, one cannot expect for an optimal or near-optimal decision. That is why, it is interesting that the proactive algorithm provides solutions containing some built-in flexibility in order to minimize the need of complex search procedures for the reactive algorithm. Several approaches that more explicitly use both off-line (proactive) and on-line (reactive) scheduling were developed in literature, see for example [4], [6] and [20].

The approach we present in this paper is a proactive reactive approach. In the first step, we build a set of schedules restricted to follow a partial order of jobs, which allows to introduce some flexibility in the obtained solution. Then, this flexibility is used on-line to hedge against some changes in the shop environment. It can also be used by the decision maker to take into account some preferences or non-modeled constraints.

In this paper, we consider the problem of scheduling a set $N = \{1, \dots, n\}$ of jobs on a single machine. Each job $j \in N$ has a processing time $p_j > 0$, a release time $r_j \geq 0$, a due date $d_j \geq 0$ and a weight $w_j > 0$. The shop environment is subject to perturbations that can be modeled by the increase of the release dates of some jobs and by machine breakdowns. The objective functions considered here are the total weighted tardiness $\bar{T}_w = \sum_{j=1}^n w_j T_j$ and the makespan $C_{\max} = \max\{C_j\}$, where C_j and $T_j = \max\{0, C_j - d_j\}$ are, respectively, the job completion time and the tardiness of job j , $j = 1, \dots, n$, of a

given schedule. The preemption of job processing is allowed only if a machine breakdown occurs.

The rest of the paper is organized as follows. Sections 2 and 3 are dedicated, respectively, to the proactive algorithm and the reactive algorithm. Before concluding, our approach is compared, in section 4, to a predictive reactive approach in stochastic shop conditions.

2. The proactive algorithm

In this section, we define a solution to the single machine scheduling problem presented in the introduction and explain its interest. Then, we provide different measures that evaluate the performance and the flexibility of a solution a priori. Finally, we present the genetic algorithm used to compute solutions providing a good tradeoff between the defined quality measures.

2.1 Definition of a solution to the problem

As noticed in the surveys of Davenport and Beck [9] and Herroleen and Leus [11] on scheduling with uncertainty, introducing flexibility in the solutions computed off-line allows to increase the robustness of the system. Hence, we consider that a solution to the problem is not a particular schedule but a set of schedules characterized by a structure defined by a partial order of jobs and a type of schedules. The schedule type can be semi-active, active or non-delay, see for example [5]. For a semi-active schedule, a job cannot be shifted to start earlier without changing the job sequence or violating the feasibility (here precedence constraints and release dates). An active schedule is a schedule where no job can be shifted to start earlier without increasing the completion time of another job or violating the feasibility. A schedule is called non-delay if the machine does not stand idle at any time when there is a job available at this time. Observe that the set of non-delay schedules is a subset of the set of active schedules which is in turn a subset of the set of semi-active schedules.

Consider the four job problem shown in table 1 and the partial order where the only restrictions are that job 1 precedes jobs 3 and 4 and job 2 precedes 4. We obtain two non-delay schedules S_1 and S_2 , three active schedules S_1 , S_2 and S_3 and five semi-active schedules S_1, \dots, S_5 , see figure 1. The corresponding objective function values are also given in the same figure.

It is clear that proposing several good schedules is more interesting than proposing only one. The decision maker can consequently choose the schedule that better respond to his preferences and possibly to non-

Table 1. A numerical example

Job j	r_j	p_j	d_j	w_j
1	0	3	6	1
2	1	3	8	1
3	0	2	9	1
4	9	2	12	1

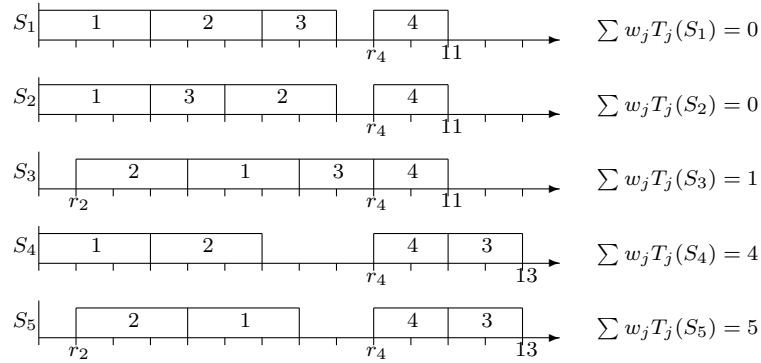


Figure 1. Represented non-delay, active and semi-active schedules.

modeled constraints. It is even more interesting if some proposed schedules have common characteristics allowing to switch from a schedule to another easily. In this case, the decision maker can postpone the choice of the schedule to be executed. It is only on-line that he makes this decision taking into account up-to-date information regarding the state of the system.

The structure defined by the couple (partial order of jobs, type of schedules) allows to represent several schedules having common precedence constraints between jobs. In order to switch from a schedule to another, or from a subset of schedules to another subset, one or several edges in the disjunctive graph representing the partial order should be oriented without creating cycles in the obtained graph. Furthermore, a partial order is a natural structure used in iterative methods. In such approaches, one or several arcs are added in each step. The use of transitive graphs allows to guarantee the feasibility of the obtained schedule.

2.2 Definition of the quality of a solution

The example discussed in subsection 2.1 shows that the quality a priori of a partial order depends on the type of schedules considered. Indeed, for semi-active schedules, there are less restrictions and this allows to obtain five represented schedules instead of two and three for, respectively, non-delay and active schedules. The number of represented schedules can measure the flexibility in job sequencing of a solution. Thus, considering semi-active schedules allows to improve the flexibility of a solution according to the proposed measure. However, the performance of the represented schedules is worse than when considering active or non-delay schedules (see the performance of schedules S_4 and S_5).

In our first implementation, we consider that the schedules are restricted to be semi-active. According to this choice, we define in the following the different quality measures of a solution.

2.2.1 A. Performance of a solution and the associated measures.

A.1. Definition of the performance. Let S be a solution to the problem. It is a partial order that represents a set of semi-active schedules o_S . To each schedule $o_S \in S$ is associated a vector $\Gamma(o_S) = (\Gamma_1(o_S), \Gamma_2(o_S))$, where $\Gamma_1(o_S)$ is the makespan of o_S and $\Gamma_2(o_S)$ its total weighted tardiness. Hence, a solution S is represented by the set of vectors $\Gamma(o_S)$. Figure 2 represents the vectors $\Gamma(o_S)$ in the (C_{\max}, \bar{T}_w) space.

Figure 2. The representation of a solution S in (C_{\max}, \bar{T}_w) space.

The performance of solution S is closely related to the objective function values of all represented schedules o_S . Since the number of these schedules can be rather big, it is natural to consider only their most important representatives. We suggest to use only the best and the worst

case performances to evaluate the performance of a flexible schedule. The best case performance provides a lower bound of the performance when following the partial order. Whereas, the worst case performance gives a guarantee about how poorly the schedules may perform when following the partial order. They can be obtained by solving corresponding minimization and maximization problems. Hence, we have to determine the coordinates of points A , B , C and D shown in figure 2. The rectangle formed by these points represents the range of the objective function values of all the schedules following the considered solution.

We denote by *goal point*, the point in (C_{\max}, \bar{T}_w) space whose coordinates are respectively the best makespan C_{\max}^* and the best total weighted tardiness \bar{T}_w^* of the problem (without the additional precedence constraints brought by the considered partial order), see figure 2.

A solution S is all the more efficient that the distance between the goal point to each point A , B , C and D is small. Hence, a performance measure of a solution S can be defined as a linear combination D_S of the distances between the goal point and the four points A , B , C and D (see figure 2). These points are determined by computing the best and worst makespan, denoted by BC_{\max} and WC_{\max} , and best and worst total weighted tardiness, denoted by $B\bar{T}_w$ and $W\bar{T}_w$. We have $D_S = \alpha D_S^1 + (1 - \alpha) D_S^2$, where $\alpha \in [0, 1]$ and

$$D_S^1 = \beta \frac{BC_{\max} - C_{\max}^*}{C_{\max}^*} + (1 - \beta) \frac{WC_{\max} - C_{\max}^*}{C_{\max}^*}, \beta \in [0, 1],$$

$$D_S^2 = \gamma \frac{B\bar{T}_w - \bar{T}_w^*}{\bar{T}_w^* + 1} + (1 - \gamma) \frac{W\bar{T}_w - \bar{T}_w^*}{\bar{T}_w^* + 1}, \gamma \in [0, 1].$$

The lower is D_S , the higher the performance of solution S is.

A.2. Computation of the performance. The objective functions considered are the total weighted tardiness \bar{T}_w^* and the makespan C_{\max} . It is proved that the problem of minimizing the makespan with respect to precedence constraints, denoted by $1|prec, r_j|C_{\max}$, can be solved in $O(n^2)$ times [13]. The problem of minimizing the total weighted tardiness, denoted by $1|prec, r_j|\sum w_j T_j$ is NP-hard in the strong sense, see for example [14]. Consequently, for this latter problem we implemented a genetic algorithm based heuristic and made comparison with known dynamic dispatching rules like ATC, X-RM and KZRM [17]. The results showed that genetic algorithm we implemented is efficient but is time consuming for a large number of jobs, see [3].

In order to compute the worst makespan and the worst total weighted tardiness which can be reached when considering a partial order S , we

introduced in [2] new optimization problems. These are maximization problems denoted by $1(sa)|prec, r_j|(F \rightarrow \max)$, where F is the objective function to be maximized and notation sa means that semi-active schedules are considered.

We proved that the problem $1(sa)|prec, r_j|(C_{\max} \rightarrow \max)$, can be solved in $O(n^2)$ times and the problem $1(sa)|prec, r_j|(\sum w_j T_j \rightarrow \max)$, is NP-hard in the strong sense [2]. We developed several heuristics based on genetic algorithms and dispatching rules. We obtained the same conclusions as for minimization problems, see [3].

As a result, for a given solution (a set of schedules w.r.t. a partial order), we use polynomial time algorithms to compute the exact minimal and maximal makespan and approximate algorithms based on dispatching rules for estimated minimal and maximal total weighted tardiness.

2.2.2 B. Definition of the flexibility of a solution and the associated measures.

One of the main characteristics of a solution to the problem is its flexibility. According to the example presented in the previous subsection, a solution is all the more flexible that the number of represented schedules is great. Indeed, when we have many schedules, it would be possible to dispose on-line, every time a decision has to be taken, of more than one alternative. This could allow to hedge against some perturbations such that raw material availability. This flexibility, called *flexibility in job sequencing*, can be measured by the number of different schedules feasible with respect to the partial order. However, the problem of calculating this number is $\#P$ -complete [7]. Thus, we propose another measure $Flex_{seq}$ equal to the number of non-oriented edges in the transitive graph representing the partial order. If this number is great, then the associated solution is flexible. In our proactive algorithm, we use a transformation of this number into a qualitative scale on several flexibility levels. Each level is defined by an interval $[nbArcsMin, nbArcsMax]$ giving the minimal and maximal number of arcs that the solution contains.

Furthermore, another type of flexibility can be provided according to the time window in which the jobs are executed. This flexibility is called *flexibility in time*. A measure of this flexibility $Flex_{time}$ can be given by the ratio between the time window in which the jobs can be executed and the total processing time of the jobs:

$$Flex_{time} = \frac{WC_{\max} - P}{P}$$

where WC_{\max} is the worst makespan of the considered solution and P is the total processing time of all the jobs.

2.3 A genetic algorithm to achieve scheduling flexibility

In this section, we present a genetic algorithm used to build-up off-line the set of Pareto solutions conciliating good shop performances and flexibility presence. These solutions are determined by an exploration of the whole solution space. In practice, this exploration may be limited to a sub-space of solutions satisfying possible preferences of the decision maker. These preferences are accounted of thanks to an interactive support decision tool which is not presented in this paper.

In the following, we present the general scheme of the genetic algorithm we implemented. Then, we describe the selection and reproduction strategies, the encoding used to represent a solution to the problem (partial order) and the genetic operators, crossover and mutation, that generate new solutions.

2.3.1 A. General scheme of the genetic algorithm.

The aim of this algorithm is to generate for a given problem a set of solutions, which yield a good compromise between flexibility and performance. According to the previous section, a solution S is all the more good that the distance $D_s(S)$ is minimal, the flexibility in job sequencing $flex_{seq}(S)$ is maximal, the flexibility in time $flex_{time}(S)$ is maximal.

The principle of the algorithm is to work on different populations of solutions that belong to a same level of flexibility in job sequencing. Recall that a level of flexibility is defined by an interval $[nbArcsMin, nbArcsMax]$ giving the minimal and maximal number of arcs that a solution can contain. The objective is to find, in the space of solutions that belong to the same level of the flexibility in job sequencing, the solutions realizing a good compromise between the performance (measured by D_s) and the flexibility in time (measured by $Flex_{time}$). We use a linear combination of these two criteria to compute the fitness of a chromosome representing a solution S .

$$Fitness(S) = \theta D_s(S) - (1 - \theta) Flex_{time}(S), \text{ where } \theta \in [0, 1].$$

For a fixed parameter θ , the algorithm works as described in figure 3. In this algorithm, we use the following notations :

- $nbGen$ is the number of generations ;
- P_i is the population of individuals in iteration $i = 1, \dots, nbGen$;
- $nbLevels$ is the number of levels of flexibility in job sequencing ;

- $Elite_{L^l, \theta}$, is the elite population obtained at the end of the algorithm for a fixed value $\theta \in [0, 1]$ and for the level of flexibility in job sequencing $L^l, l = 1, \dots, nbLevels$.

```

For  $l = 1$  to  $nbLevels$  do
  Generate a population of solutions  $P_0$  belonging to a level of flexibility
  in job sequencing  $L^l$ ;
  Evaluate the chromosomes in  $P_0$  and initialize  $Elite_{L^l, \theta}$ ;
  For  $i = 0$  to  $nbGen$  do
    Select  $N_{pop}$  couples of chromosomes for reproduction;
    Cross the selected couples of chromosomes;
    Evaluate the generated children and update  $Elite_{L^l, \theta}$ ;
    Mutate with a small probability  $\pi_{mut}$  the generated children;
    Evaluate the mutated children and update  $Elite_{L^l, \theta}$ ;
    Select  $N_{pop}$  chromosomes from  $P_i$  and the generated children;
  EndFor
EndFor

```

Figure 3. The genetic algorithm for a fixed value of θ

In order to find the Pareto optimal solution set of this multi-criteria problem, we vary the value of θ between 0 and 1 and apply the previous algorithm for each value of θ .

2.3.2 B. Selection and reproduction strategy.

In each iteration of the genetic algorithm, we use two selection procedures. The first selects N_{pop} couples of chromosomes for reproduction. The second selection procedure selects, among the generated children and the old population, N_{pop} chromosomes that will survive and form the new population. Our implementation uses the roulette selection by rank for the two previous procedures. Denote by N_{popIn} the number of chromosomes of the initial population from which the procedure selects N_{popOut} chromosomes for reproduction or survival. The best chromosome is affected a new fitness equal to N_{popIn} , the second best chromosome a fitness equal to $N_{popIn} - 1$ and the last one a fitness equal to 1. The chromosomes are selected with a chance proportional to their rank. The roulette selection works as described in figure 4.

2.3.3 C. Encoding.

In order to encode a given solution S (partial order), we use a ternary precedence constraint oriented matrix A , which represents the set of

```

Sort the initial population of  $N_{popIn}$  chromosomes in the non decreasing
order of their fitness;
set  $f_0 = 0$ ;
For  $i = 1$  to  $N_{popIn}$  do
    set  $f_i = f_{i-1} + \frac{2(N_{popIn}-i)}{N_{popIn}(N_{popIn}+1)}$ ;
EndFor
For  $i = 1$  to  $N_{popOut}$  do
    Choose randomly a real  $\pi \in [0, 1[$ ;
    Select the chromosome  $i^*$  such that  $f_{i^*-1} \leq \pi < f_{i^*}$ ;
EndFor

```

Figure 4. The roulette selection by rank

precedence constraints contained in the transitive graph representing the partial order. This matrix $A = (a_{ij})_{1 \leq i, j \leq n}$ is defined as follows:

$$a_{ij} = \begin{cases} 1 & \text{if job } i \text{ precedes job } j, \\ -1 & \text{if job } j \text{ precedes job } i, \\ 0 & \text{if } i = j \text{ or } i \text{ and } j \text{ are permutable.} \end{cases}$$

This matrix is transitive and anti-symmetric. Only $n(n-1)/2$ elements are kept inside the computer memory, but the complete matrix is used here for clarity sake.

It is a direct encoding because there is a one-to-one correspondence between the ternary matrix space and the solution space [10, 18].

Example 1. Consider a five job problem. A solution to this problem is given by the partial order represented in figure 5. In this solution, the only restrictions are that job 2 precedes jobs 3,4 and 5; and job 4 precedes job 3. This solution represents 15 schedules. It is encoded by the ternary matrix given in figure 5.

Figure 5. A solution and the corresponding encoding

2.3.4 D. Modified MT3 crossover.

In order to generate two children from two mates, we use an adaptation of the crossover MT3 proposed by Djerid and Portmann [10], which was used for job shop scheduling problem. Consider two mates (Mate 1 and Mate2) represented respectively by matrix A^1 and matrix A^2 . The algorithm corresponding to the crossover that generates the first child, represented by matrix F , works as described in figure 6.

Step 1. Compute $F = \frac{A^1 + A^2}{2}$; //integer division: the value is truncated to 0 when obtaining $\frac{1}{2}$ or $-\frac{1}{2}$
Update $nbArcs(F)$;

Step 2. Make a decision on the value of one f_{ij} to ensure, if possible, that the child will be different from both mates ;
Update $nbArcs(F)$;

Step 3. While $nbArcs(F) < nbArcs(A^1)$ do
 Select randomly mate 1 with probability π or mate 2 with probability $1 - \pi$;
 Select randomly two jobs i and j such that $f_{ij} = 0$ and $a_{ij} \neq 0$; // A is the matrix of the selected mate
 set $f_{ij} = a_{ij}$;
 set $f_{ji} = -f_{ij}$;
 Compute the transitive closure of F and update $nbArcs(F)$;
EndWhile

Step 4. if $nbArcs(F) \leq nbArcsMax$ then F is kept ;
else F is discarded ;

Figure 6. The modified MT3 crossover

Step 1 allows to keep the common precedence constraints in the two parents. In step 3, we add 1 and -1 (precedence constraints) in the matrix of the child according to the parents until reaching $nbArcs(A^1)$ (or $nbArcs(A^2)$). The probability π , which we choose equal to 0.6 in the implementation of the algorithm, allows that child 1 (resp. child 2) resembles more to mate 1 (resp. mate 2). In figure 7, we present an example that illustrates the proposed crossover.

Remarks.

- The crossover presented below guarantees an interesting property that is: *if i precedes j in both mate 1 and mate 2, then i precedes j*

Figure 7. MT3 crossover - example

in the generated offspring. It allows to keep the important properties of a good solution. Furthermore, it guarantees that imperative precedence constraints, if they exist, are usually respected.

- If the obtained offspring F is such that $nbArcs(F) > nbArcsMax$, then we can withdraw the obtained solution and reiterate or try to eliminate some arcs in order to obtain $nbArcs(F)$ almost equal to $nbArcsMax$.
- Unlike the crossover proposed by Djerid and Portmann [10], the presented crossover stops adding 1 and -1 when the number of arcs in the generated offspring reaches the number of arcs of the corresponding mate. Besides, step 2 is used to ensure that the generated child is different from his parents. This step is motivated by the chosen scheme of the genetic algorithm: Davis reproduction.

2.3.5 E. MUT3 Mutation.

The mutation operator is used to guarantee the diversity of the population of chromosomes. The mutation we propose consists in changing the order of at least two jobs. It is described in figure 8, where A is the matrix of the considered mate.

```

Step 1. Initialize  $F = F_0$ ; /*  $F_0$  is the matrix representing the imposed
precedence constraints */

Step 2. Select randomly two jobs  $i$  and  $j$  such that  $a_{ij} \neq 0$ ;
set  $f_{ij} = -a_{ij}$ ;
set  $f_{ji} = -f_{ij}$ ;
While  $nbArc(F) < nbArc(A)$ 
    Select randomly two jobs  $i$  and  $j$  such that  $f_{ij} = 0$  and  $a_{ij} \neq 0$ ;
    set  $f_{ij} = a_{ij}$ ;
    set  $f_{ji} = -f_{ij}$ ;
    Compute the transitive closure of  $F$ ;
EndWhile

Step 3. if  $nbArcs(F) \leq nbArcsMax$  then  $F$  is kept ;
else  $F$  is discarded ;

```

Figure 8. The mutation MUT3

3. The reactive algorithm

The reactive algorithm has three important roles. The first is to make the remaining scheduling decisions on-line w.r.t the precedence constraints imposed by the chosen partial order. The second role is to react when a perturbation occurs. The third role is to detect when the solution in use is infeasible with respect to the decision maker objectives.

A reactive algorithm is efficient if it allows

- 1 to offer if possible, every time a decision has to be made, more than one alternative with respect to the partial order and consequently be able to absorb possible late arrival of raw material,
- 2 to exploit the flexibility in time when a machine breaks down, and
- 3 to obtain good performance for the realized schedule.

Clearly, it is impossible to satisfy these objectives simultaneously.

We designed different procedures to achieve an acceptable compromise between the aforementioned objectives. In the remainder of the paper, the following notations are used.

- For a set of jobs X , X^+ is the subset of available jobs. A job is available if all its predecessors (w.r.t. the partial order) are scheduled and if it satisfies the restrictions on the constructed schedules (semi-active, active or non-delay schedules).
- $PRIOR_1(i, t)$ and $PRIOR_2(i, t)$ are two functions that give the priority of a job i at time t .

The general scheme of the reactive algorithms we propose is given in figure 9. At a given time t , the algorithms schedule the job i^* that maximizes $PRIOR_1(i, t)$ among the available jobs i , i.e., $i \in X^+$. When two or more jobs are in competition, the job that maximizes $PRIOR_2(i, t)$ is selected.

```

Set  $X = N$  and  $t = 0$ ;
While  $X \neq \phi$  do
  Determine  $X^+$ ;
  Determine the set  $Y_1 = \{i \in X^+, PRIOR_1(i, t) =$ 
     $\max_{j \in X^+} \{PRIOR_1(j, t)\}\}$ ;
  Select a job  $i^* \in Y_1$  such that  $PRIOR_2(i^*, t) =$ 
     $\max_{j \in Y_1} \{PRIOR_2(j, t)\}$ ;
  Set  $X = X \setminus \{i^*\}$ ;
  Set  $t = \max\{t, r_{i^*}\} + p_{i^*}$ ;
endWhile

```

Figure 9. The general scheme of the proposed algorithms

Clearly, the algorithms depend on the priority functions $PRIOR_1(i, t)$ and $PRIOR_2(i, t)$. They also depend on the definition of the job availability. Indeed, even though the partial order was computed off-line while considering that the represented schedules are semi-active, we may decide on-line to construct active or non-delay schedules for a performance improving purpose.

When a disruption occurs, an analysis module is used. In the case of a breakdown, this module uses some results developed in [1] to compute an upper bound of the increase on the worst total weighted tardiness and to possibly propose one or several decisions to minimize it. The increase on the worst makespan can be computed with a polynomial time algorithm [2]. We consider now the second type of disruptions : late

job arrival. Suppose that, at time t , the algorithm chooses to schedule a disrupted job j such that $r_j^m > t$, where r_j^m is the new release date of the job. If there exists an available non-disrupted job i , it can be scheduled without any increase on the worst case objective function's values. If such a job does not exist, then this perturbation can be considered as a breakdown and an upper bound of the increase on the worst total weighted tardiness can be computed. When the loss of performance is too important, we can either restrain the partial order in order to have an acceptable increase or recalculate a new solution (partial order) for the remaining non scheduled jobs.

4. Computational results

We conducted several experiments to evaluate the proactive algorithm and the use of the proposed approach both in deterministic and stochastic shop conditions.

The first type of experiments concerns the proactive algorithm and its ability to construct solutions providing a good tradeoff between the measures of flexibility and performance. The experiments proved that the genetic algorithm allows to provide an acceptable trade-off between flexibility and performance when the dispersion of job arrival is not very high. Otherwise, the job permutation is penalizing [1]. In the second type of experiments, we tested the application of the approach in deterministic shop conditions. We proved that the best two algorithms in deterministic shop conditions are *Perf_ND* and *Flex1_ND*. *Perf_ND* constructs non-delay (ND) schedules using ATC heuristic as the first priority function ($PRIOR_1(i, t)$) and the non-decreasing order of release dates as the second priority function ($PRIOR_2(i, t)$). *Flex1_ND* tries to maximize the use of the flexibility in job sequencing contained in the considered solution by maximizing the number of possible choices in the following step. The second priority rule is ATC [1]. In this paper, only experiments in stochastic shop conditions case are detailed.

4.1 Experimentations in stochastic shop conditions

In order to evaluate the proposed proactive reactive approach, we compare it to a predictive reactive approach. In this predictive reactive approach, a predictive schedule is computed using KZRM heuristic without taking into account the presence of future disruptions. This schedule is then proposed to the shop for execution. When a disruption occurs, a reactive algorithm, based on ATC heuristic, is used to compute a new schedule with respect to the current state of the system.

Theoretically, we can expect that the predictive reactive approach outperforms our proactive reactive approach. Indeed, we use a simple reactive algorithm which allows only small modifications on the partial order characterizing the retained solution. Besides, it was proved that ATC heuristic constructs good schedules compared to optimal schedules [12]. However, as it is shown in the following subsections, our proactive reactive approach has a better behavior when disruptions have low or medium amplitude.

4.1.1 Description. The context of our experimentations is the following. We consider that, in order to produce a final product, two components are needed: a principal component and a secondary component.

- The principal components are delivered by an upstream shop. The release dates r_j of the jobs, used by the proactive and predictive algorithms, are equal to the arrival dates of the corresponding components.
- The secondary components are bought and used for the transformation of the principal components. We suppose that two different principal components do not need the same secondary components.

After acquisition, the secondary components are stored in the shop until their use. We associate to each secondary component a storage cost proportional to its storage duration. The obtained products are then delivered to a downstream shop. A penalty is associated to each job if it is tardy with respect to the due dates d_j given by the downstream shop.

Two solutions S^{pro} and S^{pred} are computed off-line using, respectively, the proactive algorithm and the predictive algorithm. The computation of these solutions takes into account the delivery time of principal components r_j and the due dates d_j .

In order to minimize the cost of secondary component storage, the purchase date of these components is a function of the starting time of the corresponding principal products in S^{pro} or S^{pred} . The finishing time of the principal component can determine their new delivery dates that can be communicated to the downstream shop. A penalty is also associated to each job if it is tardy with respect to the new delivery dates.

The following notations are used (see figure 10). For each job j , we denote by:

- p_j its processing time;

- $t_{j_{pred}}$ its starting time in S^{pred} ;
- $t_{j_{pro}}^1$ its earliest starting time with respect to the partial order defining S^{pro} ;
- $t_{j_{pro}}^2$ its latest starting time with respect to the partial order defining S^{pro} (j is placed just before its successors [2]);
- $t_{j_{eff}}$ its effective starting time in the realized schedule S_r ;
- s_j the acquisition date of the secondary component used to produce j . It is given by the following:

$$s_i = \begin{cases} t_{i_{pred}} - \theta_1 & \text{for the predictive reactive approach} \\ t_{i_{pro}}^1 & \text{for the proactive reactive approach} \end{cases} \quad (1)$$

where θ_1 is a constant (see figure 10).

- d_j its due date asked by the downstream shop;
- δ_i its new delivery date given by the following:

$$\delta_i = \begin{cases} t_{i_{pred}} + p_i + \theta_2 = \delta_i^{pred} & \text{for the predictive reactive approach} \\ t_{i_{pro}}^2 + p_i = \delta_i^{pro} & \text{for the proactive reactive approach} \end{cases} \quad (2)$$

where θ_2 is a constant (see figure 10).

- T_j (resp. T_j^δ) is the tardiness of j with respect to d_j (resp. δ_j).

The performance of the realized schedule S_r is function of three measures: the storage cost $storageCost(S_r)$ and the tardiness $WT(S_r)$ with respect to the due dates d_j and the tardiness $WT^\delta(S_r)$ with respect to the delivery dates δ_j . These measures are given by the following equations :

$$storageCost(S_r) = \sum_{i \in N} c(t_{i_{eff}} - s_i), \quad (3)$$

where c is the unitary storage cost.

$$WT(S_r) = \sum_{i \in N} w_i T_i = \sum_{i \in N} w_i \max(0, t_{i_{eff}} + p_i - d_i). \quad (4)$$

$$WT^\delta(S_r) = \sum_{i \in N} w_i T_i^\delta = \sum_{i \in N} w_i \max(0, t_{i_{eff}} + p_i - \delta_i), \quad (5)$$

Figure 10. The different parameters for a job i in the proactive solution, the predictive schedule and the realized schedule

To be fair to the predictive approach, constants θ_1 and θ_2 are chosen in such a way that the predictive schedule S^{pred} is given a flexibility equivalent to the flexibility contained in the proactive solution S^{pro} . Constant θ_1 is such that the obtained storage costs are equivalent when considering S^{pro} or S^{pred} . The value of θ_2 is such that

$$\sum_{i \in N} (\delta_i^{pro} - d_i) \approx \sum_{i \in N} (\delta_i^{pred} - d_i) \quad (6)$$

4.1.2 Problem's generation scheme. The generation scheme is the one used by Mehta and Uzsoy [16]. The job's number is equal to 40. The processing times p_j are generated from a discrete uniform distribution $Uniform(p_{min}, p_{max})$. The job release times r_j are generated by a discrete uniform distribution between 0 and $\rho n p_{av}$, where p_{av} is the expected processing time. The parameter ρ controls the rate of job arrivals. The job due dates d_j are generated as $d_j = r_j + \gamma p_{av}$, γ is generated from a continuous distribution $Uniform[a, b]$. The job weights w_j are either all equal to 1 or generated by a discrete uniform distribution between 1 and 10.

4.1.3 Disruption's generation scheme. Two types of disruptions are considered: machine breakdowns and late arrival of principal components (increase of some r_j). The breakdowns are characterized by their occurrence time and their duration. The number of breakdowns is denoted by $nbBreak$. The scheduling horizon is divided into

equal $nbBreak$ intervals such that in each interval a breakdown occurs. The breakdown durations are generated from a uniform distribution $Uniform(durMin, durMax)$.

The disruptions related to late arrival of the principal components are generated according to the considered predictive schedule or proactive solution. When using a predictive schedule S^{pred} , a perturbation is generated by associating a new earliest starting time $r_j^m = t_{j_{pred}} + aug$, where $t_{j_{pred}}$ is the starting time of job j in S^{pred} and aug is generated from a uniform distribution $Uniform(augMin, augMax)$. When considering a proactive solution S^{pro} , $r_j^m = t_{j_{pro}}^1 + aug$, where $t_{j_{pro}}^1$ is the earliest starting time of job j with respect to the partial order defining S^{pro} .

For each problem, we consider a proactive solution and a predictive schedule to be compared. About 50% of disruptions related to late raw material arrival are generated according to the predictive schedule and 50% according to the proactive solution. The number of delayed jobs is denoted $nbDelay$.

4.2 Analysis of the obtained results

We implement two heuristics of the reactive algorithm in the predictive reactive approach : ATC_d_i and $ATC_δ_i$. These are ATC based heuristics that construct non-delay schedules. The first takes into account the due dates d_i and the second the delivery dates $δ_i$. For the proactive reactive approach, we use heuristics $Perf_ND$ and $Flex1_ND$ presented in section 3. The comparison is based on the criteria WT and $WT^δ$ given respectively by equations (4) and (5). The performance is measured by evaluating $WT + WT^δ$.

We consider the following parameters to generate problems and disruptions.

- $(p_{min}, p_{max}) = (1, 11)$,
- $ρ = 0.5, 1$: grouped or dispersed job arrivals,
- $(a, b) = (1, 3)$ or $(2, 5)$ to generate the due dates.

For the four parameter combinations, we generate five problems. For each problem, we use KZRM heuristic to generate a predictive schedule and the proactive algorithm to compute a flexible solution belonging to the flexibility level characterized by $nbArcs \in [620, 700]$. Then, these solutions are executed subject to disruptions characterized by the following parameters.

- The number of breakdowns $nbBreak \in \{0, 1, 2, 3\}$;

- $[durMin, durMax] = [3, 6]$ for breakdown durations;
- $[augMin, augMax] \in \{[1, 6], [6, 12], [12, 24], [24, 36], [36, 48], [60, 90]\}$;
- $nbDelay \in \{4, 6, 8\}$

For each combination of these parameters, a couple (predictive schedule, proactive solution) is experimented in 1000 different scenarii. The performance of each reactive algorithm $Flex1_ND$, $Perf_ND$, ATC_d_i and $ATC_δ_i$ in terms of total weighted tardiness is evaluated for each scenarii. The average value is then computed and associated to each algorithm. The reactive algorithm with best performance is assigned a performance equal to 100%. Any of the three remaining algorithms is assigned the ratio between the best performance and its performance.

Figures 11, 12, 13 et 14 sum up the obtained results for the family of problems corresponding to $\rho = 0.50$, $(a, b) = (1, 3)$, $nbDelay = 8$ and $nbBreak = 0, 1, 2, 3$ and the proactive solution is characterized $nbArcs = 680$.

We can notice that algorithms ATC_d_i and $ATC_δ_i$ are dominated by algorithms $Flex1_ND$ and $Perf_ND$ for late arrival disruptions characterized by $[augMin, augMax] \in \{[1, 6], [6, 12], [12, 24], [24, 36]\}$. The superiority of $Flex1_ND$ and $Perf_ND$ is very clear when $nbBreak = 2$. For important late arrival disruptions, notably when $[augMin, augMax] = [60, 90]$, the performance of $Flex1_ND$ and $Perf_ND$ become less than 86%. Besides, in all cases $Perf_ND$ outperforms $Flex1_ND$. This is also the case for $ATC_δ_i$ when compared to ATC_d_i . We obtain the same conclusions when $(a, b) = (2, 5)$.

We applied the same experimentations for more flexible solutions characterized by $nbArcs \approx 540$. We remarked that algorithms $Flex1_ND$ and $Perf_ND$ becomes rapidly dominated by algorithms $ATC_δ_i$ and ATC_d_i .

When the dispersion of job arrival is high ($\rho = 1.00$), our algorithms $Flex1_ND$ and $Perf_ND$ outperform the other algorithms when the level of disruptions on job arrival is low ($[augMin, augMax] \leq [12, 24]$) and $nbBreak = 0$ or 1 . When the number of breakdowns is greater, notably for $nbBreak = 2$, the performance of $ATC_δ_i$ and ATC_d_i are less than 50%. Besides, in all cases $Perf_ND$ outperforms $Flex1_ND$. This is also the case for $ATC_δ_i$ when compared to ATC_d_i .

To summarize, when the job arrivals are grouped ($\rho = 0.50$), our approach is usually superior to the predictive reactive approach when the amplitudes of disruptions on job arrival are such that $[augMin, augMax] \leq [36, 48]$ and $nbBreak = 0, 1, 2, 3$. For dispersed job arrivals ($\rho = 1.00$), our approach is very superior when $nbBreak \geq 2$.

Figure 11. Results for $(p_{\min}, p_{\max}) = (1, 11)$, $\rho = 0.50$, $(a, b) = (1, 3)$, $nbArcs = 680$, $nbDelay = 8$ et $nbBreak = 0$.

Figure 12. Results for $(p_{\min}, p_{\max}) = (1, 11)$, $\rho = 0.50$, $(a, b) = (1, 3)$, $nbArcs = 680$, $nbDelay = 8$ et $nbBreak = 1$.

Figure 13. Results for $(p_{\min}, p_{\max}) = (1, 11)$, $\rho = 0.50$, $(a, b) = (1, 3)$, $nbArcs = 680$, $nbDelay = 8$ et $nbBreak = 2$.

Figure 14. Results for $(p_{\min}, p_{\max}) = (1, 11)$, $\rho = 0.50$, $(a, b) = (1, 3)$, $nbArcs = 680$, $nbDelay = 8$ et $nbBreak = 3$.

When $nbBreak < 2$, it also outperforms the other approach for disruptions on job arrival with low amplitude.

In conclusion, according to the experiments presented in this paper, our proactive reactive approach outperforms a predictive reactive approach for low and medium disruption amplitude.

5. Conclusion

We considered in this paper the single machine scheduling problem with dynamic job arrival and total weighted tardiness and makespan as objective functions. The shop environment is subject to perturbations related to late raw material arrival and to machine breakdowns. We proposed a proactive reactive approach to solve the problem. The proactive algorithm is a genetic algorithm which computes partial orders providing a good trade-off between flexibility and performance. These solutions are built up while taking into account the shop constraints, the decision maker preferences and some knowledge about possible future perturbations. The reactive algorithm is used to guide on-line the execution of the jobs. In the presence of disruptions, the reactive algorithm is used to absorb their effects by exploiting the flexibility introduced in the proactive algorithm. We made several experimentations to compare our approach to a predictive reactive approach in stochastic shop conditions. The results showed that our approach outperform the predictive reactive approach for low and medium disruption amplitude. This confirms that anticipating the presence of disruptions proactively allows to obtain good realized schedules and to plan other shop activities.

The results obtained for single machine scheduling problems gave us some insight to extend our approach to more complex scheduling problems. Currently, we are adapting the proposed proactive algorithm for flow shop scheduling problem with maximum cost functions. A solution to this problem is characterized by a partial order on each machine. This extension is motivated by a polynomial time algorithm developed in [1] to compute the worst case performance for the flow shop case.

References

- [1] M.A. Aloulou. *Structure flexible d'ordonnements à performances contrôlées pour le pilotage d'atelier en présence de perturbations*. PhD thesis, Institut National Polytechnique de Lorraine, December 2002.
- [2] M.A. Aloulou, M.Y. Kovalyov, and M.C. Portmann. Maximization in single machine scheduling. *Annals of Operations Research*, accepted for publication by the guest editors, 2003.
- [3] M.A. Aloulou and M.C. Portmann. Incorporating flexibility in job sequencing for the single machine total weighted tardiness problem with release dates. In

- 10th Annual Industrial Engineering Research Conference*, pages in CD-ROM. IIE, May 2001.
- [4] 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, Applications and Practice*, 6:112–122, 1999.
 - [5] K.R. Baker. *Introduction to Sequencing and Scheduling*. John Wiley and Sons, 1974.
 - [6] J.C. Billaut and F. Roubellat. A new method for workshop real time scheduling. *International Journal of Production Research*, 34(6):1555–1579, june 1996.
 - [7] G. Brightwell and P. Winkler. Counting linear extensions. *Order*, 8:225–242, 1991.
 - [8] L.K. Church and R. Uzsoy. Analysis of periodic and event-drive rescheduling policies in dynamic shops. *International Journal of Computer Integrated Manufacturing*, 5:153–163, 1992.
 - [9] A.J. Davenport and J.C. Beck. A survey of techniques for scheduling with uncertainty. <http://www.eil.utoronto.ca/profiles/chris/chris.papers.html>, 2000.
 - [10] L. Djerid and M.C. Portmann. Genetic algorithm operators restricted to precedent constraint sets : genetic algorithm designs with or without branch and bound approach for solving scheduling problems with disjunctive constraints. In *Proceedings of IEEE International Conference on Systems, Man and Cybernetics*, volume 4, pages 2922–2927, October 14-17 1996.
 - [11] W. Herroelen and R. Leus. Project scheduling under uncertainty survey and research potentials. *Invited paper in a special issue of the European Journal of Operational Research*, 2003. to appear.
 - [12] A. Jouglet. *Ordonnancer sur une machine pour minimiser la somme des coût*. PhD thesis, Université de Technologie de Compiègne, November 2002.
 - [13] E.L. Lawler. Optimal sequencing of a single machine subject to precedence constraints. *Management Science*, 1973.
 - [14] E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, and D.B. Shmoys. *Logistics of Production and Inventory, Handbook in Operations Research and Management Science 4*, chapter Sequencing and scheduling: algorithms and complexity, pages 445–452. 1993.
 - [15] V.J. Leon, S.D. Wu, and R.H. Storer. Robustness measures and robust scheduling for job shops. *IIE Transactions*, 26(5):32–43, 1994.
 - [16] S.V. Mehta and R. Uzsoy. Predictable scheduling of a single machine subject to breakdowns. *International Journal of Computer Integrated Manufacturing*, 12:15–38, 1999.
 - [17] T.E. Morton and D.W. Pentico. *Heuristic Scheduling with Applications to Production Systems and Project Management*. Wiley, New York, 1993.
 - [18] M.C. Portmann, A. Vignier, C.D. Dardilhac, and D. Dezalay. Branch and bound crossed with ga to solve hybrid flowshops. *European Journal of Operational Research*, 107:389–400, 1998.
 - [19] G.E. Vieira, J.W. Herrmann, and E. Lin. Rescheduling manufacturing systems: a framework of strategies, policies, and methods. *Journal of Scheduling*, 6(1):35–58, 2003.

- [20] 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.