Abstract

This paper concerns the domain of flexible manufacturing systems (FMS) and focuses on the scheduling problems encountered in these systems. We have chosen the cyclic behaviour to study this problem, to reduce its complexity. This cyclic scheduling problem, whose complexity is NP-hard in the general case, aims to minimise the work in process (WIP) to satisfy economic constraints. We first recall and discuss the best known cyclic scheduling heuristics. Then, we present a two-step resolution approach. In the first step, a performance analysis is carried out; it is based on the Petri net modelling of the production process. This analysis resolves some indeterminism due to the system’s flexibility and allows a lower bound of the WIP to be obtained. In the second step, after a formal model of the scheduling problem has been given, we describe a genetic algorithm approach to find a schedule which can reach the optimal production speed while minimizing the WIP. Finally, our genetic approach is validated and compared with known heuristics on a set of test problems.

Keywords: Flexible manufacturing systems; Cyclic scheduling; Genetic algorithm; Petri nets

1. Introduction

Flexible manufacturing systems (FMSs) aim to produce a wide range of products in a short time without loss of manufacturing efficiency. The scheduling of production in flexible manufacturing systems has been extensively researched over the past years and it continues to attract the interest of both academic researchers and practitioners (Jain and Elmaraghy, 1997). In this paper, we study the problem of FMS control by a predictive approach to compute a cyclic and deterministic schedule. This repetitive command must be optimised
in regard to different criteria which we choose. These criteria are the throughput and the number of parts in the system; this number of parts represents the intermediate stock and is thus an economic criterion. These two criteria are antagonistic: to maximise the throughput, we have to use enough parts (work in process or WIP for short) and if we use few WIP’s (one for example) the machines will wait for parts and the throughput is lower. Hence, to take into account these two criteria, we opted for optimising first the more important of the two: the throughput. This allows the cycle time to be minimised and to be considered as a hard constraint while minimising the WIP. This hierarchical resolution (see Camus et al., 1996) is presented in a phased and structured method for performance evaluation of the production system.

The resolution of the problem was divided into two steps. The first one makes a trade-off between these two contradictory criteria. It computes a horizon and its associated optimal cycle time (CT) as well as resolving some indeterminism due to the system’s flexibility. During this step, the indeterminism is progressively resolved in order to obtain a Petri net model where the only remaining indeterminism is related to the shared resources and the WIP. The second step computes a schedule respecting the results of the first step (CT and Horizon) and minimises the last criterion: the WIP. As the remaining problem is still NP-hard, we are obliged to use a heuristic or meta-heuristic method in order to obtain a near-optimal solution in a reasonable computation time. Indeed, it has been proved that tasks planning problems have a polynomial complexity and that cyclic scheduling problem is NP complete (Serafini and Ukovich, 1989). The introduction of the (transformation) resources makes the first type of problems (i.e., planning) NP-hard in most cases and remains the second one (i.e., cyclic) in the NP class of problems (Serafini and Ukovich, 1989).

The problem we will focus on in this paper is the determination of an algorithm for the computation of a cyclic schedule respecting a hard constraint (the optimal cycle time CT) and minimising the work in process. Several scheduling heuristics can be found in the literature (Ershler et al., 1982; Hillion 1987,1988,1989; Valentin, 1994; Ohl et al., 1995; Korbaa et al., 1997, 2002; Lee and Korbaa, 2004), etc. The problem is that these algorithms are based on different assumptions, or use different optimising criteria, or have a very long resolution time. Erschler et al. studied the problem of scheduling the critical machine in a flow-shop. Hillion, Valentin and Lee do not consider flexibility in operation sequences. Ohl’s algorithm does not take into account the cyclic aspect of the problem (operations must start and end at the same cycle!). As for Korbaa’s algorithm, it requires a long time for the resolution to obtain a nearly optimal solution. Lee’s approach gives a quick but not “guaranteed” good solution with a Petri nets based resolution. All these heuristics need an entire run to give the final solution (no partial solution!).

This paper is made up of four main parts. First, we present the performance analysis method that solves some indeterminism due to system flexibility. In the second part, a formal model of the cyclic scheduling problem is specified. The genetic approach to solve this scheduling problem is given in part three. Lastly, a validation is performed on a set of cyclic benchmarks.

2. Performance analysis

It is assumed that the reader is familiar with a modelling tool of the discrete event dynamic systems: the Petri nets and its main properties, presented for instance in Murata (1989). Our first assumption is that the operating times are deterministic, because we want to study the behaviour of the system and to compute a predictive command.

In addition, we assume that:

- Products remain clamped to their transport resource (pallet for example) during their entire sojourn in the FMS.
- Empty transport resources are reloaded with raw parts as soon as possible.
- These transport resources are dedicated to a given part type (job); a pallet used for a given part type cannot be reused for another part type.

These assumptions induce equivalence between WIP, which we want to minimise, and the number of transport resources in the system.
For better comprehension of the following sections we introduce an illustrative example (see Fig. 1). Three part types (P1, P2 and P3) have to be produced on six machines M1–M6. For each cycle, we have to produce three parts with the production ratios $R_1 = 1/3$, $R_2 = 1/3$ and $R_3 = 1/3$. The flexibility of operating sequences appears in this figure, as there are multiple routes for each type of part. For instance, the P1 type part can be obtained with the operating sequence M1, M2, M3, M4, M5, M1 or it can be obtained with the operating sequence M1, M2, M6, M4, M5, M1. A routing ratio labelled $\alpha$, respectively $(1/C0_{\alpha})$ is associated with each branch of this flexibility. The operating sequences are linearised by the method developed by Camus et al. (1996).

The first step to carry out is the performance analysis, see Ramamoorthy and Ho (1980). As we want to compute a deterministic schedule, we can evaluate the optimal cycle time and manage to obtain a cyclic command respecting this optimal reference.

Hence, the workload of each machine can be obtained (for a virtual average part) as follows: for instance, $Z_2$ the workload of machine M2 is: $Z_2 = 5R_1 + (1 - \gamma)5R_2 + \gamma5R_2 = 5R_1 + 5R_2$. The whole set of workflows is:

$$Z_1 = (1 + 1)R_1 + (1 + 1)R_2 + (1 + 1)R_3,$$
$$Z_2 = 5R_1 + 5R_2,$$
$$Z_3 = 4R_1(1 - \alpha) + 4R_3(1 - \beta),$$
$$Z_4 = 3R_1 + 3R_2 + 3R_3,$$
$$Z_5 = 2R_1 + 2R_2 + 2R_3 \beta,$$
$$Z_6 = 1R_1 \alpha.$$

With the values of $R_1$, $R_2$, and $R_3$:

- $Z_1 = 2 \times 1/3 + 2 \times 1/3 + 2 \times 1/3 = 2$,
- $Z_2 = 5 \times 1/3 + 5 \times 1/3 = 10/3$,
- $Z_3 = 4 \times 1/3(1 - \alpha) + 4 \times 1/3(1 - \beta) \leq 8/3 < 3$,
- $Z_4 = 3 \times 1/3 + 3 \times 1/3 + 3 \times 1/3 = 3$,
- $Z_5 = 2 \times 1/3 + 2 \times 1/3 + 2 \times 1/3 \beta \leq 6/3 \leq 2$,
- $Z_6 = 1 \times 1/3 \times \alpha \leq 1/3 < 1$.

---

1 The resource sharing constraints are not represented in this figure to facilitate comprehension.
The maximum workload is $Z_2(\forall \alpha, \beta \text{ and } \gamma)$. Hence, the bottleneck machine$^2$ is $M_2$ and the cycle time $CT^3$ is equal to 10 (for three parts per cycle). Therefore, we can arbitrarily choose the values of $\alpha$ and $\beta$ (for example $\alpha = 0$, $\beta = 1$ and $\gamma = 1$). So the operating sequences become linear as shown in Fig. 2. Indeed, as the values of $\alpha$, $\beta$ and $\gamma$ are either 0 or 1, one of the two branches of each flexibility is not used and the operating sequences obtained are linear. For instance, for the first part, as $\alpha$ is null, the linear sequence obtained corresponds to the inferior route of the general sequence of this part. If the values are not integers, the operating sequences have to be duplicated to respect these quotients, see Camus et al. (1996).

Now it is possible to ask if this cycle time is attainable and, if so, to evaluate the work in process to be used to reach this reference of optimality. The reachability of the “discrete”$^4$ cycle time was proven in Hillion and Proth (1989). We only have to use enough WIP to ensure that each machine which finishes its job will “find” a part in its input. Hence, the bottleneck machine will not wait for parts and will work at 100% of its capacity. The optimal cycle time is thus attainable while using “enough” WIP. This problem was solved in Lafit et al. (1992, 1993). Now, under the optimal cycle time as a hard constraint we will minimise the WIP, allowing work at maximum speed.

To compute a lower bound for this WIP, we know the cycle time $CT$ (already computed) and the total duration of each operating sequence (no more flexibilities). Suppose that an operating sequence has $T_i$ as a total duration. If $T_i$ is superior to $CT$, the operating sequence is longer than the cycle and has to be “cut” into several cycles. This is done by introducing several parts of this operating sequence: WIP. This number has to be at least equal to the integer superior or equal to $T_i$ by $CT$, see Campos et al. (1992). For example, for operating sequence 1, $T_1$ is equal to the sum of operating durations = $1 + 5 + 4 + 3 + 2 + 1 = 16$ t.u. For this operating sequence, we need the integer which is at least superior or equal to 16/10 (=1.6) = 2 pallets.

Hence, a work in process lower bound is: $B = \lceil\frac{16}{10}\rceil + \lceil\frac{12}{10}\rceil + \lceil\frac{7}{10}\rceil = 5.5$

This value is just a lower bound: the waiting time due to resource conflicts can make the operating sequence longer and the work in process number higher. Nevertheless, if this lower bound is attained, we are sure that the solution obtained is optimal.

$^2$ Machine with the maximum workload.
$^3$ Is given by the workload of the bottleneck machine and cannot be improved using this data (number of parts and machines).
$^4$ Corresponding to an integer value of parts per cycle.
$^5$ where $\lceil x \rceil$ denotes the integer superior or equal to $x$. 
3. Problem formulation

We focus now on the second step of the resolution which computes a cyclic schedule respecting the optimal cycle time (CT) and minimising the WIP. It should be noticed that an overview of cyclic scheduling was given by Hanen and Munier (1995). Moreover, a logical formulation was proposed by Draper et al. (1999) for a cyclic job shop scheduling problem.

Prior to the presentation of our mathematical model, it’s important to note that we consider the following hypotheses in the modelling of this problem:

- We focused on a cyclic scheduling whose property is to be periodic: there is a periodic factor between all consecutive occurrences of operations. Hence, we can restrict the scheduling to a unique cycle time, the same scheduling pattern is repeated in the subsequent cycles. At each cycle time and for each part type (job), a new pallet is used to start a new occurrence of job. Assuming the time to complete each individual occurrence of job is greater than the cycle time, the result is a pipeline in which multiple instances are under construction at any cycle time.
- A pallet released at instant time $t$, can be reused at the same instant for a new part using this type of pallet.
- As mentioned in Section 2, the pallets are dedicated (or non-standardized pallets).

It should be noted that in cyclic scheduling, WIP allows to “break” the operations sequencing constraints during the cycle. Let us consider the example of Fig. 3 which contains one part type, three operations (A1–A2–A3) and two pallets (P1, P2). If we look at each pallet, the operations sequencing is respected. Nevertheless, during each cycle, operation A3 starts before A1 and A2! This is due to the fact that operation A3 of cycle ‘$n$’ belongs to the part launched in cycle ’$n − 1$’ (see Fig. 4).

So, at the time we make up a schedule within a single cycle (i.e., “pattern schedule”), the precedence constraints can be omitted but the price to pay is the added pallets we need to unfold the “pattern schedule” over several cycles in order to obtain a feasible schedule occurrence. Roughly speaking, each time a precedence constraint is violated, one more cycle (and thus a added pallet) is necessary to build a feasible schedule.

![Fig. 3. The operation sequencing constraints in the case of cyclic scheduling.](image)

![Fig. 4a. Disjunctive constraints (2) and (3) case $o_j$ before $o_{kl}$.](image)

![Fig. 4b. Disjunctive constraints (2) and (3) case $o_{kl}$ before $o_j$: cross is true.](image)
Let us introduce the notations used in the scheduling:

Let \( m \) be the number of machines,
Let \( n \) be the number of parts,
Let \( n_i \) be the number of operations of part \( i \) (or job \( i \)),
Let \( O_{ij} \) be the \( j \)th operation of part \( i \),
Let \( d_{ij} \) be the duration of operation \( O_{ij} \),
Let \( s(i,j) \) be the index of the successor of operation \( O_{ij} \): \( s(i,j) = (j + 1) \mod n_i \), \( i = 1, \ldots, n; j = 1, \ldots, n_i \),
Let \( m_{ij} \) be the machine assigned to operations \( O_{ij} \).

The set of operations to be processed \( O = \{ O_{ij} | i = 1, \ldots, n; j = 1, \ldots, n_i \} \). This set contains all the operations to do each cycle. We also define \( E_r \) the set of the operations processed by machine \( r \). These sets \((E_r, r = 1, \ldots, m)\) are a partition of set \( O \):

\[
\bigcup_{r=1}^{m} E_r = O \quad \text{and} \quad \bigcup_{r=1}^{m} E_r = \emptyset.
\]

The minimal cycle time (CT) is given by:

\[
CT = \max_{k=1, \ldots, m}(\sum_{i=1}^{n_i} \sum_{j=1}^{n} \delta_{ij} \cdot d_{ij}) \quad \text{with} \quad \delta_{ij} = 1 \quad \text{if} \quad m_{ij} = k \quad \text{and} \quad \delta_{ij} = 0 \quad \text{if} \quad m_{ij} \neq k.
\]

Let \( E_r^* \) be the set of the pairs \((O_{ij}, O_{kl})\) belonging to the same machine \( r \). \( E_r^* \) is equal to \( E_r \times E_r \) from which we delete the pairs of identical operations.

\[
E_r^* = \{(O_{ij}, O_{kl}) | \exists r \in \{1, \ldots, m\}, \quad O_{ij} \in E_r, \quad O_{kl} \in E_r, \quad (i, j) \neq (k, l)\}.
\]

3.1. Decision variables

The first decision variable is the starting date of the operations. We aim to obtain the Gantt diagram of cyclic schedule:

\[
t_{ij} \quad \text{is the starting date of operation} \quad O_{ij}; \quad t_{ij} \in \{0, \ldots, CT - 1\},
\]

\( t_{11} \) is arbitrarily fixed to 0.

Two operations belonging to the same machine cannot be overlapping (i.e., disjunctive constraint). Hence, one is before the other. The second variable is used to decide which operation is before the other in the cycle:

\[
y_{ijkl} \quad \text{is a binary variable of the operations sequence between} \quad O_{ij} \quad \text{and} \quad O_{kl} \in E_r.
\]

\( y_{ijkl} = 1 \) if \( O_{ij} \) is scheduled before \( O_{kl} \).

\( y_{ijkl} = 0 \) else.

3.2. Mathematical model

Minimize WIP,

\[
\text{(1)}
\]

where

\[
t_{ij} - t_{kl} \geq d_{kl} - y_{ijkl} \cdot CT \quad \text{if} \quad (O_{ij}, O_{kl}) \in E_r^* \quad \text{(2)}
\]

\[
t_{kl} - t_{ij} \geq d_{ij} - (1 - y_{ijkl}) \cdot CT \quad \text{if} \quad (O_{ij}, O_{kl}) \in E_r^* \quad \text{(3)}
\]

\[
t_{ij} \in \{0, \ldots, CT - 1\} \quad \text{if} \quad O_{ij} \in O \quad \text{(4)}
\]

\[
y_{ijkl} \in \{0, 1\} \quad \text{if} \quad (O_{ij}, O_{kl}) \in E_r^* \quad \text{(5)}
\]

The aim of the linear model is to minimize the work in process (1).

Constraints (4) and (5) define the definition domain of the decision variables.

The disjunctive constraints (2) and (3) define the mutual exclusion between two operations belonging to the same machine. Indeed, if \((O_{ij}, O_{kl})\) belong to the same machine, either \( O_{ij} \) is before \( O_{kl} \) or \( O_{kl} \) is before \( O_{ij} \). In
the first case (see Fig. 3a), \( y_{ijkl} \) is equal to 1 and \( t_{ij} = d_{kl} = CT \) (2). At the same time, \( t_{kl} - t_{ij} \geq d_{ij} \) (3). The second case corresponds to the Fig. 3b. \( y_{ijkl} \) is equal to 0 and \( t_{ij} - t_{kl} \geq d_{kl} \) (2). At the same time, \( t_{kl} - t_{ij} \geq d_{ij} - CT \) ((3)).

Let us now consider that the above set of constraints of our model does not include the precedence constraint between the operations of the same operating sequence (see the beginning of this Section). This constraint is relaxed and integrated into the objective function. The choice of relaxing this constraint has the advantage of simplifying coding and genetic operators for the genetic algorithm which is described in Section 4.

The objective function detailed below is based on a penalty principle. When a precedence constraint cannot be satisfied in the current cycle, it will be satisfied in subsequent cycles but the penalty assigned to the schedule is increased. This penalty increase is directly related to the WIP increase expressed in the pallet unit.

### 3.3. Objective function

The objective function corresponds to the minimum of the necessary WIP. The WIP of a job \( i \) is expressed as the integer number of cycle times necessary to achieve all the operations in order to produce one part, respecting the precedence constraints of the corresponding operating sequence and also respecting the resource constraints.

Hence, the WIP of job \( i \) is defined as follows: \( \text{WIP}_i = \left[ \frac{\sum_{j=1}^{n_i} d_{ij}}{CT} + \sum_{j=1}^{n_i - 1} W_j \right] \) with \( W_j \) being the waiting time between operation \( O_{ij} \) and its successor \( O_{ij+1} \) for the same occurrence of part. So, minimizing \( \text{WIP}_i \) is equivalent to the minimisation of the sum of all waiting times between the operations of job \( i \).

In order to express this objective and obtain the values of waiting times, we use auxiliary variables, labelled ‘\( X \)’. In the evaluation of the waiting time between operation \( O_{ij} \) and its successor \( O_{ij+1} \), \( X_{ij} \) represents the number of cycle times that the successor has to be shifted in order to have a positive waiting time. In fact, it corresponds to a penalty, that is each time the next operation is scheduled “before” the current one in the ‘pattern schedule’, we need one more pallet (work in process) to produce a feasible schedule:

\[
X_{ij} \in \{0, 1, 2\} \quad O_{ij} \in O.
\]

In order to express the variable \( X \), the crossing cycle time boundary predicate, labelled “Cross”, must be defined first. This predicate, described in Fig. 5, is defined as follows:

- Cross \( (O_{ij}) \) is true \( \Rightarrow t_{ij} + d_{ij} > CT \) (see Fig. 5a);
- Cross \( (O_{ij}) \) is false \( \Rightarrow t_{ij} + d_{ij} \leq CT \) (see Fig. 5b);

Now, the values of \( X_{ij} \) can be defined according to this crossing predicate; two cases have to be considered:

Case 1: Cross \( (O_{ij}) \) is false: \( O_{ij} \) does not cross the cycle time boundary.

In this case represented in Fig. 6, \( X_{ij} \) is defined as follows:

- if \( t_{ij} \geq t_{ij} + d_{ij} \) then \( X_{ij} = 0 \);
- if \( t_{ij} < t_{ij} + d_{ij} \) then \( X_{ij} = 1 \).

---

![Fig. 5. Predicate of crossing cycle time boundary.](image-url)
Case 2: Cross $(O_{ij})$ is true: $O_{ij}$ crosses the cycle time boundary.

In this case, represented in Fig. 7, $X_{ijs(i,j)}$ is defined as follows:

- if $t_{ij} + d_{ij} \geq (t_{ij} + d_{ij}) \mod CT$ then $X_{ijs(i,j)} = 1$;
- if $t_{ij} + d_{ij} < (t_{ij} + d_{ij}) \mod CT$ then $X_{ijs(i,j)} = 2$.

Using these variables $X_{ijs(i,j)}$, the WIP of job $i$ can be rewritten as follows:

$$WIP_i = \left( \sum_{j=1}^{n_i} d_{ij} + \sum_{j=1}^{n_i-1} (t_{ij}(i,j) + X_{ijs(i,j)} \cdot CT - (t_{ij} + d_{ij})) \right) / CT.$$  \hfill (7)

The global WIP, for all jobs, is given by:

$$WIP = \sum_{i=1}^{n} WIP_i.$$  \hfill (8)

Let us consider for example the objective function of the problem given in Fig. 8. It contains one job made up of four operations$^6$: A1, A2, A3 and A4, each one having a processing time of four UT that equals the optimal cycle time duration (CT). In this figure, the definition domain of the decision variables is “CT\(n\)”; the corresponding operations used to calculate WIP, are asterisked: A1*, A2*, A3*, A4*.

The values of the variables $X$ and $W$ are as follows:

- $X_{A1A2} = 1; w_1 = t_{A2} + X_{A1A2} \cdot CT - (t_{A1} + d_{A1}) = 3 + 1 \cdot 4 - (0 + 4) = 3$
- $X_{A2A3} = 2; w_2 = t_{A3} + X_{A2A3} \cdot CT - (t_{A2} + d_{A2}) = 2 + 2 \cdot 4 - (3 + 4) = 3$
- $X_{A3A4} = 2; w_3 = t_{A4} + X_{A3A4} \cdot CT - (t_{A3} + d_{A3}) = 1 + 2 \cdot 4 - (2 + 4) = 3$.

$^6$ In order to keep the presentation clear, operating sequences are now labelled with letters (A, B, C, etc.) for the rest of the paper.
For instance, $X_{A1A1}$ equals one because the positions of $A1/C3$ and $A2/C3$ correspond to the case $Z = 1$ of Fig. 6.

Then, for the calculation of $w_1$, the values of "$t_{A1}$" and "$t_{A2}$" are the starting dates of $A1^*$ and $A2^*$ whose values are, respectively, 0 and 3; the processing time of $A1^*$, labelled $d_1$, being equal to 4 ut.

\[
\begin{align*}
\text{WIP}_{A} &= d_{A1} + d_{A2} + d_{A3} + d_{A4} + (w_1 + w_2 + w_3) \\
&= \frac{(16) + (9)}{CT} = \frac{25}{4} = 7.
\end{align*}
\]

### 3.4. Lower bound

A work in process lower bound, labelled B, can be formalised by generalising the bound given in Section 2. It is the sum of the work in process lower bounds of each job $i$ ($B_i$);

\[
B_i = \frac{\sum_{j=1}^{n_j} d_{ij}}{CT} \quad B = \sum_{i=1}^{n} B_i.
\]

In general, cyclic scheduling problems with resource constraints are NP-hard (Serafini and Ukovich, 1989). That is the reason why, the use of exact methods to solve instances of large size is excluded. A first possible approach to tackle this complexity is to search for approximation algorithms with performance ratios. These approaches are relatively rare for cyclic scheduling except the works of Chrétienne (1996) about the extension of list scheduling to cyclic scheduling.

The previous approaches were confronted with this problem on combinatorial complexity. Indeed, the algorithms of Hillion et al. (1987) and Valentin (1994) use a two-step approach with a heuristic at each step. Lee and Korbaa (2004) method is no guarantee as to the quality of the result. As for Korbaa’s algorithms (1997), it takes too long a computational time (several hours) for resolution.

The alternative approach we have chosen in this paper is the use of meta-heuristics. These approaches are known to give good or near-optimal solutions within a reasonable computation time, which is an important criterion in the industrial context from which FMS comes.

The following parts of this paper concern the description and validation of a genetic algorithm approach for solving this cyclic scheduling problem.

### 4. A genetic algorithm to solve the problem

In this Section, we develop the genetic approach used to solve the scheduling problem formulated in the previous paragraph. The coding of solutions, the genetic operators proposed and elitism are then presented.
4.1. Introduction to genetic algorithm resolution

The paradigm of genetic algorithms (Goldberg, 1989; Michalewicz, 1996) is inspired by biological evolution. Population of individuals is caused to evolve by altering parent chromosomes or genotypes using cross-over and mutation operators. The behavioural expression of the genotype in a specific environment is the phenotype. Genetic algorithms have been successfully used in the field of optimisation for many years.

The general pattern of genetic algorithm must be customised to the problem to be solved. In particular, it is necessary to define the coding of solutions and the genetic operators. An important characteristic is the proportion of search space that can be explored via this coding. If the coding covers only a restricted part of the search space (phenotypes), there is no guarantee that this part contains the optimal or near-optimal solutions. Hence, it is better to have a coding that covers the entire search space of the problem. These algorithms have been widely applied to scheduling problems and particularly to the job shop problem (Cheng et al., 1999).

Within these works, two approaches can be distinguished:

- Direct coding: a chromosome represents a schedule directly (Yamada and Nakano, 1992). This approach is efficient but its main drawback is that it generally necessitates specific and complex genetic operators in order to produce feasible solutions.
- Indirect coding: a chromosome contains rules for the construction of a schedule (Mattfeld, 1999); this coding requires a transformation of the chromosome into a real schedule. The transformation is carried out by a scheduler. This type of coding avoids designing complex genetic operators but it is generally difficult to cover all the search space with this form of coding. In the following section, the genetic algorithm approach we used for solving the cyclic scheduling problem in FMS is presented.

4.2. Coding of solutions

Our resolution approach is based on direct coding using a discrete time representation (Hsu et al., 2002). Each chromosome defines the schedule of operations over a period restricted to a cycle time. This chromosome is represented by a matrix. Each element of this matrix corresponds to an operation of the operating sequence of a particular part. Each line of this matrix contains the operations to be processed on a particular machine. The column numbers define the order of operations on a machine; i.e., the operations in each line are ranked by increasing order of processing starting date. Each gene is an operation characterised by the operating sequence concerned, the order number and the starting date.

Let us consider the same illustrative example as that in Section 2, with operating sequences described in Fig. 2. An example of coding corresponding to a possible solution of this problem is presented in Fig. 9. In this figure, value “A2,5” for instance, corresponds to the second operation of operating sequence A which starts at time 5.

This coding has the following advantages:

- the chromosome represents a feasible schedule and there is no resource conflict possible;
- the corresponding Gantt diagram, represented in Fig. 9b, can easily be deduced from the chromosome and the processing times of operations. The complete Gantt diagram, covering two cycle times, with the occurrences of operations is given in Fig. 10;
- the precedence constraints of the operating sequences need not be respected in the coding adopted (for the same reasons described in Section 3). When these constraints are unverified, the operations of the operating

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>M1</td>
<td>A1.0</td>
<td>C1.1</td>
<td>B1.2</td>
<td>A6.7</td>
<td>B5.8</td>
<td>C4.9</td>
</tr>
<tr>
<td>M2</td>
<td>B3.0</td>
<td>A2.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>M3</td>
<td>A3.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>M4</td>
<td>A4.0</td>
<td>B2.3</td>
<td>C3.6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>M5</td>
<td>A5.1</td>
<td>C2.3</td>
<td>B4.5</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Fig. 9a. Representing a chromosome.
sequence concerned correspond to several occurrences of the same type of part. More precisely, each time two consecutive operations of the operating sequence do not respect the precedence constraints, the WIP of the corresponding part is increased by one unit. For instance in Fig. 10, occurrence number \(i\) of part \(B\) requires two cycle times to be achieved (cycle \(i\) and cycle \(i + 1\)): operations \(B_{1i}\) and \(B_{2i}\) are processed in cycle \(i\) and operations \(B_{3i} - B_{5i}\) are processed in cycle \(i + 1\).

It is worth noticing that in Fig. 10:

- The idle times of the machines are not explicitly coded but appear in dark grey in the Gantt diagram associated with the chromosome.
- There is no idle time on the line of machine \(M_2\), as this is the critical machine i.e., the bottleneck.

### 4.2.1. Coding the cycle overlap

An overlapping operation begins in a given cycle and ends in the next cycle: this operation crosses the cycle time boundary. In some cases, the overlapping of operations enables the WIP to decrease by one unit. The overlapping of operation \(o_{ij}\) decreases the current WIP (without overlapping) only if WIP is strictly greater than the sum of times of operations of the operating sequence of job \(i\) which corresponds to the bound \(B_i\) (see bound (9)); hence the following condition has to be verified \(WIP > B_i\) (see bound (9)).

The example of Fig. 11a represents a schedule without overlapping operations that generates the following WIP:
WIPOS_A = 3; WIPOS_B = 3; WIPOS_C = 1; WIPGlobal = WIPOS_A + WIPOS_B + WIPOS_C = 7.

Fig. 11b indicates that if operation B_3 overlaps two cycles, then each WIP of operating sequences A and B is decreased by one unit, so the global WIP (WIPGlobal) equals 5.

In our coding, an operation is characterised by its starting date of processing on the machine. Hence, there is no need to modify it in order to represent overlapping operations. For instance, operation B_3 starts at time 6 of cycle i and ends at time 1 of cycle i+1. The schedule represented in Fig. 12a is simply coded by the following chromosome of Fig. 12b.

This coding makes it possible to represent overlapping operations over two cycles as well as solutions integrating idle times. Hence, the complete search space is covered by the proposed coding.

4.3. Genetic operators

Crossover and mutation operators are detailed below. The main property of these operators is that they never produce an unfeasible solution; there is no need to repair chromosomes.

4.3.1. Crossover

The principle of this operator is to exchange genetic material of sub-matrix obtained from a horizontal cut of parental matrix. At first, the crossover operator randomly selects a horizontal line in the matrix. This line delimits the superior and inferior parts of genetic material in each randomly selected parents labelled Par1 and Par2. Then the inferior part of parent Par1 is exchanged with the inferior part of parent Par2, in order to pro-
duce offspring Ofs1. A similar exchange is carried out in order to produce offspring Ofs2, with the superior parts of genetic materials. There is no need to calculate and modify the starting date of the offspring produced: the exchange operates on the lines of the matrix i.e., machines – and preserves the schedule of operations of the parent chromosomes. Only the WIP is affected by this crossover operation. Fig. 13 shows a crossover between two parents Par1 and Par2; the selected cutting point of crossover is the upper boundary of the row of machine M3.

The offspring labelled Ofs1, (respectively Ofs2) is obtained by merging the schedule lines associated to machines M1 and M2 of the parent chromosome labelled Par1, (respectively Par2), with the schedule line of machines M3 to M5 of the parent chromosome Par2, (respectively Par1).

This crossover operator is described in Algorithm 1.

**Algorithm 1. Crossover operator**

- Randomly select two parents chromosomes Par1 et Par2
- Randomly select a machine \( m_r \), \( r \in [1, \ldots, m] \)
- Create an offspring Ofs1, (respectively Ofs2) by merging:
  - schedule of machines \( m_1 \) to \( m_{r-1} \) of Par1, (respectively Par2)
  - schedule of machines \( m_r \) to \( m_m \) of Par2, (respectively Par1)

Notice again that all the solutions produced by this crossover are feasible: there is no possibility of resource conflict since they are inherently avoided by the matrix representation; neither is there a potential precedence constraint violation since these constraints are ignored in the coding but integrated and satisfied in the objective function. The good properties of our coding would easily allow us to use extension of this crossover such as multi-point crossover operator.

4.3.2. Mutation

The mutation aims to explore the search space. More precisely, the mutation operator used is a mutation by insertion. First, two random choices are made: an operation \( o_{ij} \) assigned to machine \( m_r \) (\( o_{ij} \in E_r \) and \( r \in [1, \ldots, m] \)), and a start processing date \( t \). The mutation consists in modifying the schedule of machine \( m_r \) by making operation \( o_{ij} \) start at date \( t \). This modification necessitates the moving of certain operations around date \( t \). The starting dates of these operations have to be modified in order to make this new schedule feasible.
Fig. 14 shows a mutation on machine M5 (see chromosome Fig. 7) that moves operation A5 to start at time 2. Operations B4 and C2 are moved so as to avoid conflict on machine M5: operation B4 is advanced by one time unit and operation C2 is delayed by one time unit.

This mutation operator is formalised in Algorithm 2.

Algorithm 2. Mutation operator

<table>
<thead>
<tr>
<th>Mutation</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Randomly select:</td>
</tr>
<tr>
<td>- a chromosome</td>
</tr>
<tr>
<td>- a machine ( m_r ), ( r \in [1, \ldots, m] ), of this chromosome</td>
</tr>
<tr>
<td>- an operation ( o_{i,j} ) schedule on machine ( m_r )</td>
</tr>
<tr>
<td>- a date ( t ) in the interval ( [0, \ldots, CT - 1] )</td>
</tr>
<tr>
<td>- Save operation ( o_{i,j} ) and remove it from ( m_r )</td>
</tr>
<tr>
<td>- In the event of conflict between the new starting position of ( o_{i,j} ) and other operations, circularly shift backward operations in conflict (for example B4 is shifted backward, by one time unit, in Fig. 10b)</td>
</tr>
<tr>
<td>- In the event of conflict between the new ending position of ( o_{i,j} ) and other operations, circularly shift forward operations in conflict (C2 is shifted forward, by one time unit, in Fig. 10b)</td>
</tr>
<tr>
<td>- Set operation ( o_{i,j} ) in its new position</td>
</tr>
</tbody>
</table>
4.3.3. Initial population

This population is generated by a uniform random selection. It contains individuals with or without idle times in the sequence of operations on machines and also individuals who do or do not overlap two production cycles.

4.4. Elitism

An important characteristic of the problem studied is that the fitness is an integer value (i.e., WIP) and that its domain of variation is narrow. The maximum fitness of the worst individuals is generally inferior to twice the value of the near-optimal solution found. Moreover it appears in many runs that the population contains many individuals with similar genotypes i.e., clones. Consequently, an elitism mechanism has been developed to improve the performance of the genetic approach. This mechanism is presented in Algorithm 3.

Algorithm 3. Elitism

Elitism

Let $P(n)$ (size $\lambda$) the population at generation number $n$

Let $P'(n + 1)$ (size $\lambda$) the population obtained after selection and recombination operators (crossover and mutation)

Let $P(n + 1)$ (size $\lambda$) the new population at generation number $n + 1$

- Merge population $P(n)$ and $P'(n + 1)$ into a new population $P'$ (size $2\lambda$)
- Sort all individuals of $P'$ by decreasing fitness
- Delete all clones of $P'$
- Create $P(n + 1)$ from $\beta\%$ of best individuals of $P'$
- Complete $P(n + 1)$ with random individuals from $P'$

This elitism mechanism aims to preserve the diversity of population. First, all clones of individuals are deleted. Secondly, the diversity is controlled by parameter $\beta$ which represents the percentage of the best individuals transferred from mating population to the new population. This parameter is increased with an exponential growth during the generation process. At the start of the evolution process (first generation) its values is set to a constant $\beta_0$ percentage until $100\%$ for the last generation ($\beta_0 = 20\%$ in the experiments reported below).

5. Validation

This validation is based on a set of height test problems of the FMS literature. The first six problems are fully described in Appendix 1. The last two, labelled FT06 (Fisher and Thompson, 1963) and LA04
(Lawrence, 1984), are classical instances of job shop. The results presented in this Section were obtained with the following values for the parameters of genetic algorithm:

- Population size: 200
- Crossover probability: 0.7
- Swap mutation rate: 0.3
- Selection method: Tournament
- Number of runs: 30
- Initialisation of the population: Random
- Stopping condition: 30 to 150 Generations depending on the test problem

The results are the mean of 30 runs for a set of genetic operators. This gives a certain robustness to the results obtained through these genetic algorithms, the comportment of which is stochastic. Height test problems were used to validate our genetic approach. The results obtained are shown in Table 1. For each test problem, this table contains the following columns:

- the number of operations of the test problem;
- the cycle time value of the test problem;
- the number of feasible solutions calculated with formula in Appendix 2. This information is only an indicator of the size of the search space but should not be considered as a complexity proof;
- the theoretical lower bound of WIP;
- the result found by the best heuristic;
- the result obtained with the genetic algorithm;
- the GA computing time;
- the optimal solution found with the XPRESS-MP solver implementing the mathematical model of the paragraph three under the form of linear programming.

The main conclusions of this evaluation are as follows:

- the genetic algorithm (GA) performs as well as the best known heuristic for the six test problems;
- the optimal solutions are reached by the GA in six of the eight problems, except for the problems of larger size labelled FT06 and LA04;
- the exact resolution approach based on linear programming enables to show, with problem instances (Ohl et al., 1995; and Korbaa, 1998) that the lower bound is not always reachable;
- the lower bound quality is good since the difference between optimal and bound values is no more than one unit and furthermore it is no more than two units between the best known solution and the bound.

The computing time of the genetic algorithm which is written in C language, ranges (GA) from 1–60 seconds on a Pentium III 677 MHz under Linux for the first six problems. For the FT06 and LA04 larger problems, the computing times are, respectively 48 and 72 hours. As an indication, one can also mention that: first, the Korbaa’s heuristic used to solve the problem labelled (Korbaa, 1998; instance b) requires 17 hours of computing time; secondly the computing time of the exact resolution approach (based on XPRESS MP) is about 3 weeks for FT06 problem.

6. Conclusions

In this paper, we have proposed a genetic approach to the cyclic scheduling problem of flexible manufacturing systems. This approach was combined with a performance evaluation to take into account operating
<table>
<thead>
<tr>
<th>Test problems</th>
<th>Number of operations</th>
<th>Cycle time length</th>
<th>Number of feasible solutions</th>
<th>Lower bound</th>
<th>Best heuristic</th>
<th>Genetic algorithm</th>
<th>GA computing time</th>
<th>Optimum (PLNE XPRESS-IVE)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hillion et al. (1987)</td>
<td>15</td>
<td>8</td>
<td>1.6E+7</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>~1 second</td>
<td>5</td>
</tr>
<tr>
<td>Hillion and Proth (1988)</td>
<td>9</td>
<td>6</td>
<td>1.1E+4</td>
<td>5</td>
<td>6</td>
<td>5</td>
<td>~1 second</td>
<td>5</td>
</tr>
<tr>
<td>Valentin (1994)</td>
<td>13</td>
<td>11</td>
<td>7.9E+7</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>~1 second</td>
<td>5</td>
</tr>
<tr>
<td>Ohl et al. (1995)</td>
<td>10</td>
<td>28</td>
<td>3.5E+10</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>~1 second</td>
<td>5</td>
</tr>
<tr>
<td>Korbka (1998) instance a</td>
<td>23</td>
<td>24</td>
<td>8.6E+10</td>
<td>12</td>
<td>13</td>
<td>13</td>
<td>~10 seconds</td>
<td>13*</td>
</tr>
<tr>
<td>Korbka (1998) instance b</td>
<td>23</td>
<td>24</td>
<td>8.6E+10</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>~60 seconds</td>
<td>9</td>
</tr>
<tr>
<td>Fisher and Thompson (1963) FT06</td>
<td>36</td>
<td>43</td>
<td>4.4E+38</td>
<td>7</td>
<td>NK*</td>
<td>8</td>
<td>~48 hours</td>
<td>7</td>
</tr>
<tr>
<td>Lawrence (1984) LA04</td>
<td>50</td>
<td>537</td>
<td>3.9E+71</td>
<td>10</td>
<td>NK*</td>
<td>12</td>
<td>~72 hours</td>
<td>NK*</td>
</tr>
</tbody>
</table>

* Non-reachable bound.

* Not known.
sequence flexibility and guarantee maximal flow while minimising work in process. An efficient direct coding of this scheduling problem has been proposed. This work extends the applications of genetic algorithms to the field of cyclic scheduling.

The results enable us to validate the approach proposed in this paper. Future developments of this work will concern the following points:

- the improvement of coding and genetic operators to increase the efficiency of the resolution. A first possibility would be to develop a crossover operator that performs other recombination of parent chromosomes; for instance other solutions of the bi-partition of the machine set have to be considered instead of the simple one we use here. Another possibility to be studied is the hybridisation of the genetic algorithm with other meta-heuristics or local search approaches. A work is under progress concerning the hybridisation of GA with a neural networks approach that would improve the solving of large size instances.
- a theoretical research would consists in searching structural properties of optimal solutions that would help solve this problem even more efficiently.

Appendix 1. Test problems

Problem description format:

Problem reference

Number of machines/set of machines

Number of operating sequences/set of operating sequences (operating sequence label, number of operations)

List of operating sequences

For each operating sequence all operations are ordered and separated by a semicolon; each operation is given under the following form: machine number processing time

(Hillion et al., 1987)

4 / {M1, M2, M3, M4}
4 / {(A, 4); (B, 4); (C; 3); (D, 4)}
A: M1 1; M2 4; M3 3; M4 3;
B: M4 1; M2 2; M1 3; M3 1;
C: M1 2; M3 1; M4 1;
D: M3 3; M1 2; M2 1; M4 2;

(Hillion and Proth, 1988)

3 / {M1, M2, M3}
4 / {(A, 3); (B, 2); (C, 2); (D, 2)}
A: M1 1; M2 3; M3 3;
B: M3 1; M2 2;
C: M1 2; M3 1;
D: M1 2; M3 1;

(Ohl et al., 1995)

6 / {M1, M2, M3, M4, M5, M6}
2 / {(A, 6); (B, 4)}
A: M1 18; M3 14; M2 12; M3 14; M2 10; M4 14;
B: M5 5; M6 4; M5 5; M6 4;
Appendix 2. Number of admissible solutions

The number of admissible solutions explored by a naïve search can be calculated in the following way:

- let $n_k$ be the number of operations assigned to machine $k$: $n_k = \sum_{j=1}^{n} \sum_{i=1}^{n} \delta_{ij}$, with $\delta_{ij} = 1$ if $m_{ij} = k$ and else $\delta_{ij} = 0$;
- let $n_{fk} = (CT - W_k)$ be the duration of the idle time of machine $k$, in the cycle time $CT$, with $W_k$ being the workload of machine ‘$k$’ defined as follows $W_k = \sum_{i=1}^{n} \sum_{j=1}^{n} \delta_{ij} * p_{ij}$, with $\delta_{ij} = 1$ if $m_{ij} = k$ and else $\delta_{ij} = 0$. $n_{fk}$ corresponds to the number of fictive operations of unitary duration that can be inserted in a cycle time $CT$.

At first, the number of possible schedules on machine “$k$” without overlapping can be evaluated; it is labelled $\phi_k$. To that purpose, all the permutations of fictive and real operations are considered as taking into account that one must withdraw the permutations of fictive operations that produce the same solution. So $\phi_k$ is equal to: $\phi_k = (n_{fk} + n_k)!/n_{fk}!$.

Secondly, all possible overlapping of previously obtained solutions must be counted. This number of schedules, labelled $\varphi_k$, is obtained as follows: for each operation whose duration is strictly superior to one, overlapping is possible (see below the term $(p_{ij} - 1)$); for each overlapping solution, all the operations (fictive or not) except the one considered can be permuted as in previous case without overlapping (see the term: $(n_{fk} + n_k - 1)!$).

So $\varphi_k = \sum_{i=1}^{n} \sum_{j=1}^{n} ((p_{ij} - 1) \cdot (n_{fk} + n_k - 1)!))m_{ij} = k.$
Consequently, the total number of possible schedules, labelled $N_s$, taking into account all the machines of the problem is:

$$N_s = \prod_{k=1}^{m} \varphi_k.$$ 

For instance, the number of admissible solutions ($N_a$) for the test problem Korbaa, 1998 (instance b) equals $8.6E+10$.

References


