Information Sciences 329 (2016) 491–502

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

# Control of Flexible Manufacturing Systems under model uncertainty using Supervisory Control Theory and evolutionary computation schedule synthesis

Patrícia N. Pena [a,*], Tatiana A. Costa [b], Regiane S. Silva [c], Ricardo H.C. Takahashi [d]

[a] Department of Electronics Engineering, Universidade Federal de Minas Gerais, Belo Horizonte, MG, Brazil
[b] Department of Exact and Applied Sciences, Universidade Federal de Ouro Preto, João Monlevade, MG, Brazil
[c] Department of Automation and Control Engineering and Fundamental Techniques, Universidade Federal de Ouro Preto, Ouro Preto, MG, Brazil
[d] Department of Mathematics, Universidade Federal de Minas Gerais, Belo Horizonte, MG, Brazil

ARTICLE INFO

*Article history:*
Received 31 December 2014
Revised 27 July 2015
Accepted 5 August 2015
Available online 28 September 2015

*Keywords:*
Scheduling
Optimization
Metaheuristics
Supervisory Control Theory
Discrete Event Systems

ABSTRACT

A new approach for the problem of optimal task scheduling in flexible manufacturing systems is proposed in this work, as a combination of metaheuristic optimization techniques with the supervisory control theory of discrete-event systems. A specific encoding, the word-shuffling encoding, which avoids the generation of a large number of infeasible sequences, is employed. A metaheuristic method based on a Variable Neighborhood Search is then built using such an encoding. The optimization algorithm performs the search for the optimal schedules, while the supervisory control has the role of codifying all the problem constraints, allowing an efficient feasibility correction procedure, and avoiding schedules that are sensitive to uncertainties in the execution times associated with the plant operation. In this way, the proposed methodology achieves a system performance which is typical from model-predictive scheduling, combined with the robustness which is required from a structural control.



## 1. Introduction

The task scheduling problem in a flexible manufacturing system (FMS) is of great importance in industry. A suitable solution for this problem would achieve an operation free of faults and high efficiency on the use of the resources available in the plant. Without the use of systematic techniques to create the production programs it is not possible to guarantee such features. For that reason, there has been a significant research effort in the last few years to develop systematic tools to deal with such a problem. The scheduling of FMS is a problem far more difficult than the ordinary scheduling of simpler manufacturing systems, due to the difficulty of building reliable models which are able to predict exactly the system behavior. This means that the model uncertainty should be considered in the design of schedules.

With regard to the uncertainty, scheduling problems are traditionally divided in two broad classes [3]: (i) deterministic scheduling problems, also called *model-predictive scheduling* [5,10,36], for which the system behavior is so predictable that it is possible to build models that, given a sequence of inputs, describe the sequence of system states with negligible error; and (ii) stochastic scheduling problems [23,27], in which the system is subject to unpredictable disturbances, rendering the system states predictable only in a statistical sense.

* Corresponding author. Tel.: +55 31 34094848.
*E-mail addresses:* ppena@ufmg.br (P.N. Pena), tatiana@decea.ufop.br (T.A. Costa), regiane.ufop@gmail.com (R.S. Silva), taka@mat.ufmg.br (R.H.C. Takahashi).

http://dx.doi.org/10.1016/j.ins.2015.08.056
0020-0255/

In recent years many authors have recognized that there are several manufacturing environments which are mostly deterministic but, notwithstanding, are unlikely to be predictable to the extent that would be required by the deterministic scheduling approaches. Some authors have even stated that the inability of much scheduling research to address the general issue of uncertainty may be considered as a major reason for the lack of influence of scheduling research on industrial practice. Those observations suggested that some efforts should be developed in order to extend deterministic approaches (or model-predictive approaches) to situations in which there is some type of uncertainty. For a review on those efforts, see reference [3].

In particular, FMS tend to be inherently susceptible to some uncertainty, due to their characteristic of involving a complex coordination of different operations. This tends to give rise to small differences between the states predicted by the model and the actual deployment of the plant operations, which accumulates over time – eventually leading to a loss of synchronism in the schedule [44]. However, the simple adoption of stochastic approaches would render the scheduling of operations in those systems rather inefficient, due to their complex structure that cannot be captured by simple rules.

Different approaches have been employed in order to solve scheduling problems in FMS in an efficient way. For instance, there are works based on Neural Networks [33], Fuzzy Logic [15] and Heuristic and Metaheuristic methods [2,19,22,35,40]. These approaches use artificial intelligence techniques in order to search for the optimal scheduling. However, in most cases, these approaches do not employ any formal technique that model the behavior of the FMS and assure the structural control[1] of it.

Usually, structural controllers are designed in a conservative way, leading to the rejection of some safe states, in order to enhance their ability to avoid forbidden states [25]. There is a class of works that implements the structural control of the FMS using some modeling technique of discrete-event systems, and uses together some optimization technique to search for optimal scheduling, for instance [16,21,42,45], among others. Those approaches rely on the Petri Nets to implement restrictions on the behavior of the system. In most cases, there is no hint on how to obtain such correct models. The focus of such works is on the integration of the information from the model into the algorithm.

The Supervisory Control of Discrete Event Systems (SCDES) has been developed to design structural controllers for discrete-event systems, automatically determining the commands that can be applied to the system avoiding its evolution to prohibited states, where the system can be damaged or enters in deadlock, while also ensuring that the resulting controller is minimally restrictive – allowing all possible legal command sequences [34]. There are some former works that deal with the problem of scheduling under the SCDES framework, such as [20,32,38], among others. Those works translate optimization problems into the framework of SCDES and propose theoretical solutions and algorithms for the problem. However, those translation procedures are problem-specific, and it is not known if they could be extended to generic FMS scheduling problems. Those procedures also may lead to a substantial increase of the computational burden of the controller design problem.

The problem to be solved in this paper is to find a scheduling that minimizes the makespan for the production of a batch of products, while ensuring the system robustness properties that are associated to the SCDES structural control. The purpose here is to perform an integration of a predictive scheduling (executed by optimization algorithms over a detailed discrete-event model of the plant) with a minimally-restrictive structural control (implemented by the SCDES methodology).

Due to the dynamic nature of the problem under consideration, in which the choices performed in each step determine the available choices in the subsequent steps, the formal setting for solving it up to optimality would be the dynamic programming methodology [6,28]. However, it is well known that such a methodology incurs in computational costs that grow rapidly as the problem size grows, becoming non-practical even for problems of moderate size [7,28]. Heuristic optimization techniques, on the other hand, have been developed as computational tools that can promote the enhancement of the solutions of combinatorial problems for which greedy heuristics lead to solutions far from the optimum and exact methods are excessively costly. Such techniques rely on stochastic procedures which are performed on candidate solutions, progressively converging to the regions of the solution space which feature the most promising solutions [17,37].

In the approach developed in this paper, a heuristic optimization method is applied on a model that implements the problem physical structure and the security and non-blocking constraints, using the SCDES paradigm. The model implements the least restrictive legal behavior using a systematic way to achieve it, based on the application of the Local Modular Supervisory Control of Discrete Event Systems [11], an extension of the classical Ramadge and Wonham Supervisory Control Theory [34]. A specific version of the metaheuristic technique Variable Neighborhood Search (VNS) is employed. This technique has been chosen, from eleven different metaheuristic techniques[2] which composed the portfolio of algorithms that were considered initially.

The methodology presented in this paper is tested on an example problem introduced in [12], and significant improvements have been observed in relation to strategies that are not optimization-oriented, which are typically produced by the greedy approach suggested in [8]. It is also shown that the resulting controller is robust, allowing significant errors in the system model times without compromising the correct functioning of the system.

This paper is organized as follows. The optimization problem is formally presented in Section 2, along with a summary of the proposed methodology. Section 3 presents the example plant that is used in this paper in order to illustrate the methodology. In Section 4, the modeling of the constraints is discussed. In Section 5, the optimization algorithm using the SCDES solution to encode the problem constraints is presented. Section 6 presents the results of some tests that indicate that the approach is promising. The last section presents some conclusions. A much larger set of experiments, involving the testing on some other

[1] *Structural control* stands for the control layer that prevents the system entering an illegal (unsafe or blocking) state [3,25].

[2] Eleven algorithms have been considered including, in addition to VNS: two other versions of VNS algorithms, two Clonal Selection Algorithms, an Ant Colony algorithm, two versions of Iterated Local Search algorithms, two versions of a Variable Neighborhood Descent algorithm and a Tabu Search algorithm.

metaheuristic algorithms and the application of the proposed procedure to more example plants, is presented as a companion supplementary material which is available from the journal website.

## 2. Optimization problem statement

The optimization problem to be considered in this paper is the minimization of the total time required for the production of a batch of products in a manufacturing plant. Formally:

- Let $\Sigma$ be the set of events associated to the plant (commands and responses), which is partitioned into controllable events $\Sigma_c$ and uncontrollable events $\Sigma_{uc}$;
- Let $\mathcal{A}$ be the set of instances of events from $\Sigma$ which are associated to the production of the complete batch, and let $\mathcal{A}^c = \mathcal{A} \cap \Sigma_c$ be the subset of controllable events of $\mathcal{A}$ which are associated to the production of the complete batch;
- Let $A_k$ be an ordered set of the events of $\mathcal{A}$, representing a production schedule candidate, $|\mathcal{A}| = |A_k|$, and let $A_k^c$ denote the ordered subset of controllable events in $A_k$;
- Let $\aleph = \left\{A_1^c, A_2^c, \ldots, A_{|\aleph|}^c\right\}$ be the set of all permutations of the elements of $\mathcal{A}^c$, i.e., all ordered sets composed with the elements of $\mathcal{A}^c$;
- Let $T_k$ denote the time elapsed while the plant processes the sequence $A_k$. If the sequence is infeasible, $T_k = \infty$.

The optimization problem can be stated as follows.

$$A^* = \arg \min_{k \in \{1,\ldots,|\aleph|\}} T_k \tag{1}$$

Notice that there is no need to define constraints in this setting, since an infinite $T_k$ is assigned to any infeasible sequence. The objective function $T_k$ is usually named the system *makespan*.

### 2.1. Summary of the proposed methodology

The proposed methodology for the synthesis of production schedules for manufacturing systems consists of the following steps:

1. The plant is modeled and the *supervisors* are synthesized using the Local Modular Supervisory Control theory [11].
2. The plant plus supervisors are computationally implemented, as described in Section 4.1.
3. The decision variable encoding procedure (word-shuffling encoding) is performed, as described in Section 5.1.
4. An approximate solution for the optimization problem (Section 2) is obtained using a metaheuristic optimization algorithm, the VNS presented in Section 5.3. This solution is called the *controller*, which actually defines the sequence of commands that will be applied to the plant in operation time.

The problem might be decomposed into Minimum Part Sets (MPS), each one involving a number of products which is approximately equal to the number of items which can be processed simultaneously by the plant. After the optimization of each subproblem, the complete schedule for the batch can be obtained by the concatenation of the schedules for the sub-problems. This procedure ends with the determination of a schedule for the production of a pre-determined batch of products.

## 3. Description of the example manufacturing system

The case study discussed in this paper is the Flexible Manufacturing System (FMS) presented in Fig. 1 and introduced in [12]. This system produces two types of products from raw blocks and raw pegs: a block with a conical pin on top (Product A) and a block with a cylindrical painted pin (Product B). The FMS consists of eight devices. The lathe, the mill, the painting device and the assembling machine perform tasks over the pieces. The pieces are taken from buffers and, after the end of each operation, they are deposited in buffers. The three conveyors and the robot move the pieces from buffers to machines and from machines to buffers. There are 8 buffers that act as intermediate deposits for the pieces, $B_i$, $i = 1, \ldots, 8$, each one with capacity for one piece. The arrows in Fig. 1 indicate the events that represent the flow of unfinished parts through the FMS. More details can be found in [12]. The specifications that should be attained by the SCDES solution are to avoid overflow and underflow of pieces in all of the buffers.

The open loop behavior of the system is modeled by the automata of Fig. 2. The set of events used to model the plant is partitioned into two subsets: the set of controllable events (the ones that can be disabled by the controller), $\Sigma_c$, and the set of uncontrollable events (the ones that cannot be disabled, usually representing the spontaneous events of the plant), $\Sigma_{uc}$. In the case of the FMS, the events labeled with odd numbers are controllable and the ones labeled by even numbers are uncontrollable. The control specifications are to avoid overflow and underflow of the buffers [29].

## 4. Using supervisory control to model the constraints

In order to synthesize supervisors that guarantee the legal execution of the system, the models of the plant and the specifications for the closed-loop behavior have to be developed. The action of the supervisor on the plant is to inhibit the occurrence of controllable events that might lead the system to an illegal behavior [34].

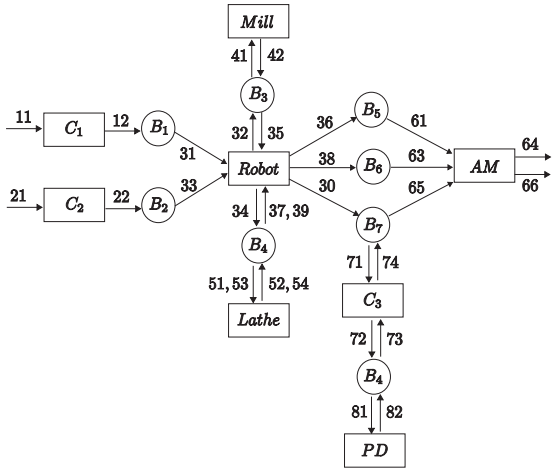


**Fig. 1.** Flexible Manufacturing System.

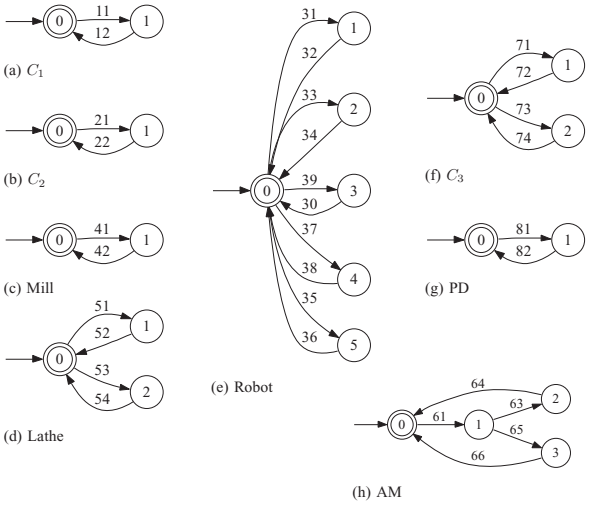


**Fig. 2.** Modeling the FMS.

There are extensions to the classical supervisory control of discrete event systems (DES) [34] that provide the same quality of the solution but with less complexity [11,43]. Among them, the choice of the Local Modular Supervisory Control of Discrete Event Systems (LMSCDES) [11] is supported by the fact that the resulting local supervisors are much smaller thus easier to implement. This solution provides the same minimally restrictive behavior if the local supervisors are verified to be nonconflicting. This test has high complexity but there are abstraction-based tests that reduce significantly that complexity [30].

After nonconflicting local supervisors are obtained, they are reduced in order to remove redundant information. The reduced supervisors carry only the action of each supervisor over the plant [39,41]. In the case of the FMS, a set of reduced supervisors is presented in Fig. 3.

More details on the solution using LMSCDES for the FMS can be found in [29].

The supervisor is obtained and verified for nonconflict only once. The same structure is used to generate schedules (controllers) for different production batches. So, the size of the production batch does not impact on the complexity of the LMSCDES solution.

### *4.1. Legal Sequence Generator and Evaluator - LSGE*

The supervised system, which is the conjunction of the automata of the plant and of the reduced supervisors, is represented in a computational program that can be used to generate and evaluate legal sequences. By definition, time is not part of the DES model, that deals only with the logical part. However, in real systems time is inherent and the idea is to be able to evaluate a sequence in terms of its makespan.

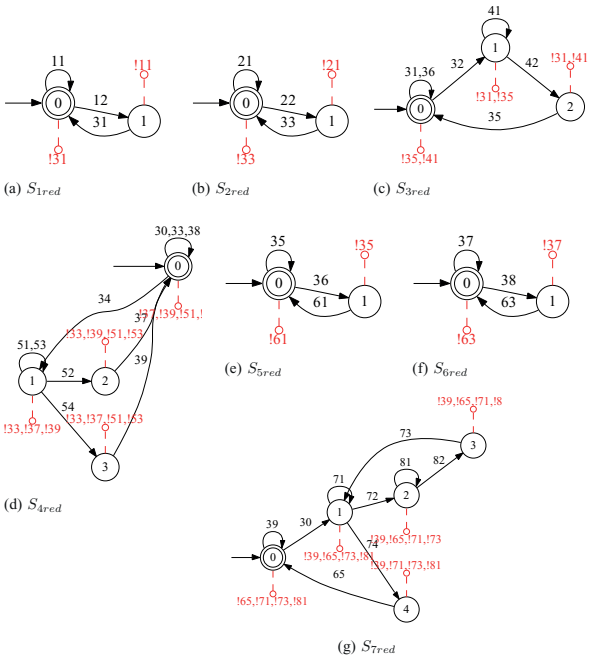


**Fig. 3.** Reduced supervisors for the FMS.

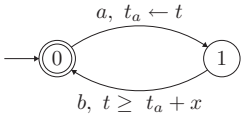


**Fig. 4.** Automaton: model with guards.

This evaluation is performed via system simulation. For this purpose, the automata that model the system are transformed into automata with parameters (FSMwP, such as in [9]). Guards and function actions are added to the transitions of the automata in order to model not only the logical behavior but also the duration of each operation performed by each equipment. The reduced supervisors are kept the same. At first, for simplicity, such durations (represented by $x$) are considered to be fixed and known. So, every pair of events (command, response) is related by guards such as in the model presented in Fig. 4, where $a$ represents the controllable event, $t_a$ represents the simulation time ($t$) in which event $a$ was executed (stored by doing $t_a = t$), $b$ represents the uncontrollable event that relates to $a$ and the guard for this transition represents that the event can only occur $x$ t.u. after $t_a$. There is no temporal condition for the controllable events, meaning that if a controllable event is logically feasible and allowed by the supervisor there is no temporal condition that will postpone its execution.

Two assumptions are made on the model. The first is that every uncontrollable event models a response to a specific controllable event and the second assumption is that an uncontrollable event will never be prevented to happen, neither by the action of the supervisor nor by the model of the plant. These assumptions fit to manufacturing systems modeled in an abstraction level such that each uncontrollable event denotes the end of an operation that was initiated by the corresponding controllable event.

For each equipment, the duration $x$ of the operation should be measured and the corresponding guard introduced in the model. This guard, which works as a condition for the execution of the event, should be added to every uncontrollable event of the model, relating its occurrence to the simulation time of the corresponding controllable event.

The values for the durations of the operations of the equipment in the example FMS are defined in Table 1, where each $x$ is associated to the controllable event that triggers the operation.

As an example, the automaton for the Lathe is presented in Fig. 5.

The uncontrollable events are the ones that cannot be disabled by the control. Typically, they represent the activation of a sensor, the end of the operation of a machine, and so on. Those signals cannot be controlled; therefore one cannot impose any order on them. The supervisor imposes an order to the controllable events and the uncontrollable events occur within some time after them, as a consequence of the physical arrangement of the system. In order to represent the fact that an uncontrollable event will always occur when the process that leads to it is completed, the following rule is implemented: if two or more events can occur at the same time, the uncontrollable one is always executed first.

**Table 1**
Operation times, $x$.

| Subplants | Event | $x$ (t.u.) | Subplants | Event | $x$ (t.u.) |
|---|---|---|---|---|---|
| $C_1$ | 11 | 25 | $C_2$ | 21 | 25 |
| Robot | 31 | 21 | Mill | 41 | 30 |
| | 33 | 19 | $C_3$ | 71 | 25 |
| | 35 | 16 | | 73 | 25 |
| | 37 | 24 | AM | 61 | 15 |
| | 39 | 20 | | 63 | 25 |
| Lathe | 51 | 38 | | 65 | 25 |
| | 53 | 32 | PD | 81 | 24 |

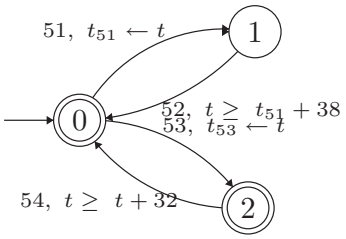


**Fig. 5.** Automaton with guards to model the Lathe.

The LSGE is a computational program that jointly executes the extended automata that model the system and the reduced supervisors obtained from the LMSCDES. It operates in two modes, the autonomous mode (LSGE-AM) and the evaluator mode (LSGE-EM).

In the LSGE-AM, the sequence to produce a batch is generated by executing the automata and choosing the next event of the sequence using a greedy decision policy, such as in [8]. The input is the batch size and the outputs are: the time elapsed from the beginning of the batch processing until the end of the batch production and the production schedule, $A_k$. This autonomous operation mode is employed to provide a baseline for comparison purposes.

The core of the autonomous simulation is used also in the LSGE-EM for the purpose of candidate solution evaluation, within the main loop of the optimization procedure, as a tool to encode the problem constraints. In this mode of operation, the input is a sequence of controllable events (the uncontrollable ones can be left out for this type of modeling strategy), provided by the optimization procedure, which should be evaluated. The LSGE-EM processes the sequence and in case the sequence is processed until the end it provides, as outputs, the information that the sequence is feasible and the total time elapsed for it execution. In case the sequence is not executable up to its end, the algorithm returns the position of the sequence in which the evaluation stopped and the set of feasible continuations at that point. This information is employed in a correction procedure (part of the optimization algorithm) that is performed until the sequence becomes feasible.

## 5. Optimization algorithm

Given a batch size, the idea is to search for a production schedule that minimizes the time necessary to produce the batch (the makespan). The following subsections describe the solution encoding and the search algorithm.

### 5.1. Representation of a solution - the word-shuffling encoding

In the plant considered, the fabrication of each product can be modeled by a specific sequence of events. Suppose that the following sequence of $n$ commands applied to the plant results in the production of product $P$:

$$\mathcal{E} = \{e_1, e_2, e_3, \ldots, e_{n-1}, e_n\}. \tag{2}$$

Suppose also that some subsequences of that sequence need to be executed according to the order in the sequence – otherwise the product would not be fabricated correctly, possibly causing the plant to enter a forbidden state or causing it to try the execution of an impossible operation. These subsequences are called *invariant subsequences*. A change in the order of events between two different invariant subsequences may still lead to the correct completion of the fabrication process of $P$, provided that the relative order between the elements within the invariant subsequences is preserved.

The role of the metaheuristic algorithms is to search for an optimal permutation of sequence $\mathcal{E}$. Clearly, if the algorithm adopts an encoding that always preserves the internal order of the invariant subsequences, this avoids lots of unfruitful function evaluations that are bound to result infeasible, and the computational effort can be spent more efficiently in the evaluation of permutations that possibly are feasible. With this purpose, a specific encoding procedure is proposed. Each invariant subsequence is encoded by a string of repeated symbols from an alphabet, with the same length of the subsequence, which identifies uniquely

this subsequence. In this way, let the sequence $\mathcal{E}$ be composed of $m$ invariant subsequences:

$$\mathcal{E}_1 \rightarrow a_1 a_1 \ldots a_1 \quad \mathcal{E}_2 \rightarrow a_2 a_2 \ldots a_2 \quad \ldots \quad \mathcal{E}_m \rightarrow a_m a_m \ldots a_m \tag{3}$$

The candidate solutions for the problem are now represented by permutations of those symbols, for instance:

$$a_1 a_m a_1 a_2 a_2 a_m a_2 a_3 a_1 \ldots a_2 \tag{4}$$

This string is decoded in the following way: the first occurrence of symbol $a_1$ is decoded as the first event in $\mathcal{E}_1$; the second occurrence of symbol $a_1$ is decoded as the second event in $\mathcal{E}_1$, and so forth. The first occurrence of symbol $a_2$ is decoded as the first event in $\mathcal{E}_2$, and so forth. The internal ordering of the events in all invariant subsequences is preserved, and all other permutations are allowed. This encoding procedure is similar to the job-based representations, employed in job-shop scheduling [13] and will be called the *word-shuffling* encoding.

#### 5.1.1. Large batch sizes and minimum part set

When the batch size is smaller than or nearly equal to the number of products that can be processed simultaneously in the plant, the events related to the manufacturing of different instances of the same product should be labeled using different symbols. This is necessary in order to ensure that there is a unique decoding, with any symbol representing a specific event. For instance, if two products A are to be manufactured in the plant, the string $a_0^1 a_0^1 a_0^1 a_0^1 a_0^1$ should represent the sequence of events to produce the base of the first product A, while another string $a_0^2 a_0^2 a_0^2 a_0^2 a_0^2$ should represent the events to produce the base of the second product A.

However, when the batch size is much larger than the number of products that can be processed simultaneously in the plant, another concern arises for the encoding: the sequences that lead the plant to start the manufacturing of some product that will be finished much later, therefore occupying some buffers in the plant for a long time, tend to be inefficient. Such sequences also tend to become infeasible often. In order to avoid the occurrence of such sequences, the encoding scheme adopts the following strategy:

- A small set of products to be manufactured is defined, such that the concatenation of several sets corresponds to the batch. This set is called the *minimum part set* (MPS). The MPS should not have much more products than the number which can be processed simultaneously by the plant.
- Only the MPS is encoded according to the word-shuffling encoding, resulting in the *MPS-string*.

The evaluation of the total makespan is performed by processing the whole batch, which consists of the concatenation of MPS-strings.

**Remarks.** (i) In addition to avoiding infeasibility, this encoding using MPS-strings keeps the decision space much smaller than would be the case of a direct encoding of the whole batch. This improves substantially the performance of the optimization algorithms when searching in such a space. (ii) Although this strategy works by the concatenation of several pieces of MPS-strings, this does not mean that the production related to a set of events encoded in an MPS-string should be finished after the production related to the events encoded in the next MPS-string can start. Instead, the end of an MPS-string will be finished in parallel with the start of the next MPS-string, by the simple application of the same decoding procedure already described. (iii) There is no need to define new symbols for the same events that take part on new instances of the same product under manufacturing, if the events belong to different MPS-strings. Now, it becomes meaningful the simple procedure of making the pointer refer to the events that belong to the first MPS-string before starting the events that belong to the next MPS-string, in the case of a repeated symbol.

### 5.2. Function evaluation and feasibility correction

The optimization algorithm will perform the search on the space of candidate solutions which are represented in a word-shuffling encoding. Each time a candidate solution is to be evaluated, it must be decoded into the event sequence encoding (represented by the sequence of event numbers), which is then fed into the LSGE-EM.

The proposed optimization methodology always employs, jointly with the function evaluation procedure, a feasibility correction procedure when an infeasible individual is found, in order to work with feasible individuals only. Assume that the function evaluation procedure **eval** is defined such that

$$[feas_k, T_k, j, \mathcal{F}_j] \leftarrow \textbf{eval}(\mathcal{W}_k)$$

$\mathcal{W}_k$ represents a sequence that contains exactly the set of controllable events that are necessary to produce the batch (which means that $\mathcal{W}_k \in \aleph$), $feas_k$ returns 1 if the input sequence is feasible and 0 otherwise and $T_k$ returns the makespan if the sequence is feasible and $\infty$ otherwise. If the sequence is infeasible, $j$ returns the index of the first controllable event $e_j$ that cannot be executed by the LSGE-EM, and $\mathcal{F}_j$ returns the list of events that represent the possible one-step-ahead feasible continuations after $e_{j-1}$ (this list is provided directly by the supervisor). Let also $W_{k,j}$ represent a subsequence of events in $\mathcal{W}_k$, starting from $e_j$, going up to the end of the sequence. The evaluation procedure with feasibility correction is presented below.

Evaluation Procedure with Feasibility Correction

**input:** $\mathcal{W}_k$
1 $[feas_k, T_k, j, \mathcal{F}_j] \leftarrow$ **eval**$(\mathcal{W}_k)$;
2 **while** $feas_k = 0$ **do:**
3 starting from $e_j$ to the end of $\mathcal{W}_k$, find the first event $e_i \in \mathcal{E}_j \cap \mathcal{W}_{k,j}$;
4 swap $e_i$ and $e_j$ in $\mathcal{W}_k$;
5 $[feas_k, T_k, j, \mathcal{F}_j] \leftarrow$ **eval**$(\mathcal{W}_k)$;
6 **end-while**

At the end of this procedure, the sequence $\mathcal{W}_k$ will be feasible and the variable $T_k$ will carry its makespan, as stated in the next proposition.

**Proposition.** *Let n represent the number of controlled events that are necessary to produce the batch. The Evaluation Procedure with Feasibility Correction converges to a feasible sequence after at most* $n - 1$ *calls of the* ***eval*** *function.*

**Proof.** This result is due to the property of the supervisor, which will return, in the set $\mathcal{F}_j$, only the events which have some feasible continuation up to the completion of the batch fabrication; see details in [34]. This implies that after each iteration there will be at least one new feasible event which is added to the sequence of feasible events that was available before that step. After $n - 1$ steps, the whole sequence must be feasible because the last controllable event is the only possible continuation of the penultimate controllable event, which means that it must be feasible. □

As this result constitutes a central point in this work, it deserves some further discussion. In general, a sequence may be infeasible for three different reasons:

- **(a)** The next event may represent the execution of an impossible operation, for instance the attempt to paint a piece which is not in the painting machine, or the execution of a damaging operation, for instance the attempt to put a piece in a place which is already occupied by another piece;
- **(b)** The next event may be immediately possible and non-damaging but leading to future blocking, which becomes unavoidable if that event occurs at that point;
- **(c)** The next event may be possible and non-damaging and still not lead to blocking, but may be unsafe, allowing operations that depend on the correct synchronization between operations for being successful. For instance, the event may start a machine without its output buffer being empty, counting on that another event will empty that buffer before the machine finishes its operation.

In conventional optimization-based approaches for scheduling, repairing a tentative solution when an infeasibility of case (a) occurs is an easy task: the algorithm should simply replace the stopping event by another one that belongs to a list of possible and non-damaging events, which is easy to determine. However, repairing a tentative solution for an infeasibility of case (b) is much more involved: in first place, the infeasibility will not be detected until the point of the sequence which effectively blocks is identified (for instance, when it enters a deadlock). In such a case, the repairing procedure should go back in the sequence, up to an unknown point from which it is possible to proceed without blocking, and then change the sequence in that point, restarting the sequence evaluation towards its end. This procedure may involve an unknown number of backward and forward searches before a feasible sequence is found. Finally, in the conventional approaches, the case (c) may even be considered feasible, although delivering solutions that may be sensitive to plant uncertainties and disturbances. This is a main source of difficulties in the usual deterministic scheduling approaches, which can prevent their practical applicability in real-world situations.

In the proposed approach, all such sources of infeasibility are reported by the supervisor (generated by the LMSCDES) in a suitable way to perform solution repairing. In cases (a) and (b), the supervisor directly avoids the start of any sequence of events which is bound to finish in a blocking condition, which means that dealing with case (b) is as easy as dealing with case (a). The supervisor, instead of allowing the simulation to continue until there is no possible continuation, already indicates the infeasibility of the solution as soon as the starting event of the blocking sequence appears. Therefore, it is enough to choose another command which is feasible, from the set of commands of the encoded sequence that appear after the blocking command, and perform the swap. The sequence of commands, up to this point, is necessarily an initial sub-sequence of a feasible solution, and therefore there is no need to perform a backward–forward search for correcting such a problem.

An important distinctive property of the proposed approach arises in case (c): the supervisor will not allow such unsafe events, instead indicating their infeasibility. Therefore, the correction of this kind of infeasibility is performed in the same way as in the cases (a) and (b). So, this kind of infeasible solutions will no longer exist after the feasibility correction procedure is performed.

In any case, the correction procedure for the whole sequence requires a number of iterations which is less than the length of the sequence, which represents an important advantage in relation to the usual correction approaches employed in most heuristic search algorithms. Due to the word-shuffling encoding, the number of iterations in the proposed Evaluation Procedure

**Table 2**
Results from VNS and greedy search - FMS Plant.

| | VNS | | | | Greedy | | |
|---|---|---|---|---|---|---|---|
| (nA,nB) | Min. | Avg. | Dev % | Eval. | Min. | Avg. | Dev % |
| (0,1) | 204 | 204.03 | 0.18 | 4600 | 206 | 217.00 | 11.17 |
| (1,0) | 160 | 160.00 | 0.00 | 2570 | 161 | 164.60 | 4.48 |
| (1,1) | 250 | 250.20 | 0.41 | 74600 | 254 | 294.33 | 26.67 |
| (0,2) | 303 | 303.07 | 0.25 | 66460 | 305 | 323.47 | 19.60 |
| (2,0) | 254 | 254.03 | 0.18 | 32840 | 258 | 261.50 | 4.18 |
| (1,2) | 349 | 349.17 | 0.38 | 426360 | 353 | 398.70 | 22.01 |
| (2,1) | 334 | 334.53 | 1.28 | 287940 | 340 | 370.67 | 29.57 |
| (2,2) | 414 | 416.13 | 5.20 | 1300100 | 425 | 470.57 | 28.34 |
| (0,3) | 402 | 402.13 | 0.35 | 395460 | 405 | 464.03 | 45.43 |
| (3,0) | 339 | 339.13 | 0.35 | 147800 | 346 | 366.77 | 13.34 |
| (1,3) | 448 | 464.97 | 8.56 | 1281500 | 476 | 540.83 | 41.13 |
| (3,1) | 419 | 421.10 | 3.32 | 844400 | 428 | 467.37 | 26.60 |
| (0,4) | 501 | 501.00 | 0.31 | 1207200 | 505 | 534.83 | 12.91 |
| (4,0) | 424 | 424.00 | 0.0 | 423600 | 434 | 439.36 | 9.53 |

with Feasibility Correction is usually much less than that upper bound, as a consequence of the previous assessment of the precedence constraints.

A remark should be stated here: in the case of large-size batches, which are equivalent to a concatenation of several replicas of the MPS, if the MPS-string is feasible then the whole batch is feasible too.

### 5.3. VNS algorithm

In this paper, a Variable Neighborhood Search (VNS) algorithm is employed as the optimization machinery. The VNS is a local search method which explores the decision space through systematic changes in the neighborhood structure [18,26]. In recent years, the VNS metaheuristic has been studied intensively to solve scheduling problems [18], reaching favorable results in comparison with other heuristic methods [1,14,24,31].

In this work, the VNS employs the block change and event swap moves, defined below as M1 and M2 moves, in order to generate different neighborhood structures:

**M1:** Block change - 2 blocks of events of the candidate solution are swapped. The blocks can have from 1 event to at most 20% of the size of the sequence;
**M2:** Event swap with higher probability at the beginning - events are exchanged in the candidate solution such that events in the beginning of the sequence have more probability of being swapped.

The following neighborhoods are induced by varying the intensity of the moves M1 and M2:

- $N^{Swap-20}(s)$, $N^{Swap-30}(s)$, $N^{Swap-40}(s)$: Consisting of the set of the neighbors obtained from a solution $s$ when a number $n$ of swap moves are applied on $s$, with $n = 20$, 30 and 40% of the size of the sequence.
- $N^{BSwap-20}(s)$, $N^{BSwap-30}(s)$, $N^{BSwap-40}(s)$: Consisting of all neighbors that are reached from a solution $s$, when a number $n$ of block swap moves is applied with $n = 20$, 30 and 40% of the size of the sequence.

In the proposed VNS methodology, the initial solution for the problem is generated by the LSGE-AM, according to Section 4.1. In the case of infeasible neighbors being generated, the feasibility correction procedure described in Section 5.2 is employed. The VNS algorithm is interrupted when a total of 20 consecutive iterations without enhancement in the objective function value is reached.

## 6. Results

This section presents the results of the tests that have been performed in order to analyze the performance of the proposed methodology, when applied to the FMS. A set of 14 problem instances, that provides the manufacturing of up to four products, is used. Table 2 presents the results obtained by VNS when applied to the FMS. The first column indicates the number of products that are produced. The four columns under VNS indicate the best result, the average, the standard deviation and the number of evaluations of objective function of the solutions found by the VNS algorithm, from 30 runs. The three columns labeled Greedy indicate the best result, the average and the standard deviation found by the LSGE-AM, from 30 runs.

The first point to be noticed from these results is concerned with the comparison of the greedy policy with the proposed VNS. Although the minimum value of makespan for the greedy policy is in some cases close to the minimum found by VNS, the average values of the solutions provided by the VNS is of the order of 5–15% better than the average values provided by the greedy policy.

**Table 3**
Resulting operation times for the optimal scheduling in some instances (Inst.) of FMS Plant, undisturbed (undist.) and when the execution time of tasks is disturbed with a Gaussian noise with different standard deviations.

| | | | Disturbed parameters (std. dev.) | | | | | |
|---|---|---|---|---|---|---|---|---|
| Inst. | Undist. | | 5% | 10% | 15% | 20% | 25% | 30% |
| | | Min. | 350 | 345 | 344 | 342 | 333 | 328 |
| (1,2) | 349 | Mean | 353.24 | 354.38 | 355.97 | 357.55 | 357.79 | 358.62 |
| | | Max. | 359 | 364 | 368 | 371 | 379 | 395 |
| | | Min. | 334 | 334 | 329 | 327 | 319 | 320 |
| (2,1) | 334 | Mean | 338.38 | 340.31 | 338.90 | 341.69 | 341.83 | 343.14 |
| | | Max. | 342 | 348 | 352 | 358 | 370 | 366 |
| | | Min. | 403 | 396 | 395 | 389 | 385 | 391 |
| (0,3) | 402 | Mean | 407.86 | 409.10 | 410.45 | 406.86 | 407.04 | 411.07 |
| | | Max. | 414 | 429 | 426 | 432 | 435 | 436 |
| | | Min. | 338 | 335 | 327 | 326 | 315 | 324 |
| (3,0) | 339 | Mean | 342.93 | 343.75 | 342.93 | 343.62 | 342.55 | 350.41 |
| | | Max. | 348 | 354 | 361 | 366 | 363 | 384 |

**Table 4**
Results from concatenating MPS of different sizes for producing a batch of 12 products of type A.

| MPS size | MPS makespan | Number of MPS | Batch makespan |
|---|---|---|---|
| (1) | 160 | 12 | 1634 |
| (2) | 254 | 6 | 1394 |
| (3) | 339 | 4 | 1278 |
| (4) | 424 | 3 | 1220 |

The VNS algorithm requires, in some cases, a large number of function evaluations, which may lead to computation times of the order of one hour for some problem instances. However, it should be noticed that the production schedule can be stored in a pre-processed form, as a small number of different elementary Minimum Part Sets (MPS) which are simply combined in order to compose batches of any size. If this procedure is performed, there is no relevant constraint on the processing times in order to generate production schedules. Table 2 presents the list of relevant MPS to be pre-processed, since the FMS may process simultaneously up to four products – which means that MPS with more products tend to be irrelevant.

### 6.1. Performance under disturbances

In order to assess the behavior of the optimal schedules that were synthesized, an experiment has been conducted as follows: The best schedules that were found for the instances (1,2), (2,1), (3,0) and (0,3) of the FMS were considered. For each instance, 30 independent runs of simulation of that optimal schedules were executed with the task times presented in Table 1 perturbed by a Gaussian noise of 5%, other 30 simulations for a noise of 10%, and so forth, up to a noise of 30%. The resulting total times for completing the batch production is displayed in Table 3. The single most important conclusion that might be extracted from this experiment is: the proposed methodology shows robustness against system parameter disturbances. As the controller that commands the system is event-driven and only safe operations are allowed at any time, any change of the order in which some uncontrollable events occur does not cause the system to fail.

As the perturbation in the task times were Gaussian, both negative and positive perturbations occurred – which sometimes caused the total production time to become smaller than the time in the undisturbed case. The differences between the minimum time and the maximum time increased as the perturbation size increased, as expected. The mean production time also increased as the perturbation size increased, what can be explained by the fact that the optimal scheduling was calculated for the non-perturbed system, making it non-optimal for the perturbed system.

Looking at the degradation of the mean production time as the perturbation size grows, a desirable property can be observed: (i) for small perturbations the system performance remains nearly optimal; and (ii) the performance degradation increases gradually, as the perturbation size grows. This means that the model-predictive schedule which is obtained for the disturbance-free model still anchors the system behavior in the situations in which disturbances arise.

### 6.2. Large batches and minimum part sets

As mentioned before, the FMS may process simultaneously up to four products, and the batch sizes analyzed have up to this size. In the case of larger batches, the procedure to be followed would be to concatenate the MPS-strings, generated for those smaller numbers of products, in order to form the batch. As an example, Table 4 presents the results of the concatenation of MPS-strings of different sizes in the FMS, in order to produce a batch of 12 products of type A.

An MPS with only one product has, of course, the smallest number of degrees of freedom for the optimization, since the production of each product is performed without the simultaneous manufacturing of any other product. Even in this case, the total makespan of producing 12 products is equal to 1634 time units, much less than the sum of 12 makespans of the production of a single product, which would be 1920 time units. This gain in the total makespan is due to the event-driven operation of the controller, which naturally allows the starting of the first tasks of a new MPS before the tasks related to the former MPS are completed; the only requirement is that the last event that starts a task in the former MPS have already been commanded.

For larger MPS sizes, there are significant gains in the total makespan for the same batch of 12 products. This is due to the removal of the constraint that makes the last event related to the manufacturing of a product to be commanded before the start of the events related to another product. In this way, an exploitation of the possibility of simultaneous processing of several pieces is performed, leading to a more efficient system operation, and much smaller makespans. Considering the largest MPS, which refers to the simultaneous manufacturing of four products of type A, the total makespan for a batch size of 12 products becomes 1220 time units, which is 36.4% less than the time that would be required to produce each piece, from the beginning to the end, before starting the next piece, and 25.3% smaller than the time that would be spent by the simple concatenation of the solutions (MPS-strings) for one product.

It becomes clear that: (i) Better results are achieved when the MPS is of greater size, which is reasonable, since the optimization, in this case, is performed with more degrees of freedom. (ii) The total makespan is always smaller than the sum of the makespans of MPS. This is due to the possibility of starting a sequence without finishing the former one – which is a consequence of the employment of the SCDES as an interface layer between the solution encoding and the actual generation of a sequence of commands to the plant.

## 7. Conclusions

In this work, a new approach for the optimal sequencing of tasks in manufacturing systems was presented, based on the combination of metaheuristic algorithms that try to optimize the solution makespan, using Supervisory Control Theory of Discrete Event Systems to model the constraints. Concerning the metaheuristic search, the word-shuffling encoding induces a decision variable space in which efficient searches can be performed. With this encoding, a version of the Variable Neighborhood Search (VNS) was developed.

The specific roles of the supervisor in the proposed methodology are:

- To encode the problem constraints, allowing a more elegant and efficient search. Concerning elegance, the supervisor becomes a transparent and systematic way to express all the problem constraints related to feasibility and safety. Concerning efficiency, the supervisor allows the feasibility correction of tentative solutions to be performed in a forward direction only, avoiding both a forward-backward search and the discarding of feasible solutions, which would be performed in most usual approaches. In this way, the search complexity is reduced and the possibility of finding the optimal solutions increases.
- To provide robustness to the solutions, guaranteeing that the resulting controller, which defines the plant operation schedule, will not lead to infeasible operations if there is any mismatch between the time associated with the execution of a task in the model and the effective time observed in real-time operation. The supervisor therefore provides the necessary accommodation of the effects of plant disturbances and model uncertainty.

In the case of large batch sizes, when the batch is decomposed into several MPS, it becomes possible to synthesize solutions by a simple concatenation of the MPS-strings. The resulting event-driven controller provides a meaningful interpretation of those concatenations, allowing a parallel execution of some tasks encoded in consecutive MPS-strings.

The results obtained indicate that the proposed methodology achieves a significant reduction of the total production time, when compared with the greedy randomized algorithm. The robustness of the proposed methodology against disturbances in the model parameters indicates that the main obstacle for the application of model-predictive scheduling in real FMS, which is the sensitivity to error accumulation, does not arise in the proposed methodology. Due to the high costs involved in the acquisition of flexible manufacturing systems, the obtained improvement in operation times may represent important economic gains for industry. It is also noticeable that the proposed methodology can be useful for dealing with other scheduling problems with complex dynamics, for instance in logistics [4].

## Acknowledgments

The authors gratefully acknowledge the support of the Brazilian agencies CNPq, Fapemig and Capes. This work was also supported by a Marie Curie International Research Staff Exchange Scheme Fellowship within the 7th European Community Framework Programme.

## Supplementary material

Supplementary material associated with this article can be found, in the online version, at 10.1016/j.ins.2015.08.056.

## References


[1] C. Almeder, L. Monch, Metaheuristics for scheduling jobs with incompatible families on parallel batching machines, J. Oper. Res. Soc. 62 (12) (2011) 2083–2096.
[2] H. Aytug, S. Bhattacharyya, G.J. Koehler, Genetic learning through simulation: an investigation in shop floor scheduling, Ann. Oper. Res. 78 (1998) 1–29.
[3] H. Aytug, M.A. Lawley, K. McKay, S. Mohan, R. Uzsoy, Executing production schedules in the face of uncertainties: a review and some future directions, Eur. J. Oper. Res. 161 (2005) 86–110.
[4] S.A.K.I. Babu, S. Pratap, G. Lahoti, K.J. Fernandes, M.K. Tiwari, M. Mount, Y. Xiong, Minimizing delay of ships in bulk terminals by simultaneous ship scheduling, stockyard planning and train scheduling, Marit. Econ. Logist. (2014) 1–29.
[5] R.L. Becerra, C.A.C. Coello, A cultural algorithm for solving the job shop scheduling problem, in: Y. Jin (Ed.), Knowledge Incorporation in Evolutionary Computation, Springer, 2005, pp. 37–55.
[6] R.E. Bellman, Dynamic Programming, Princeton University Press, 1957.
[7] D.P. Bertsekas, Dynamic Programming - Deterministic and Stochastic Models, Prentice-Hall, 1985.
[8] C.G. Cassandras, S. Lafortune, Introduction to Discrete Event Systems, Kluwer Academic Publishers, 2008.
[9] Y.-L. Chen, F. Lin, Modeling of discrete event systems using finite state machines with parameters, in: Proceedings of the 2000 IEEE International Conference on Control Applications, 2000, pp. 941–946.
[10] R. Cheng, M. Gen, Y. Tsujimura, A tutorial survey of job-shop scheduling problems using genetic algorithms - I. Representation, Comput. Ind. Eng. 30 (4) (1996) 983–997.
[11] M.H. de Queiroz, J.E.R. Cury, Modular supervisory control of large scale discrete event systems, in: Proceedings of the 5th Workshop on Discrete Event Systems, vol. 1, 2000, pp. 103–110.
[12] M.H. de Queiroz, J.E.R. Cury, W.M. Wonham, Multitasking supervisory control of discrete-event systems, Discret. Event Dyn. Syst.: Theory Appl. 15 (2005) 375–395.
[13] C. Dimopoulos, A.M.S. Zalzala, Recent developments in evolutionary computation for manufacturing optimization: problems, solutions, and comparisons, IEEE Trans. Evol. Comput. 4 (2) (2000) 93–113.
[14] R. Driessel, L. Mnch, Variable neighborhood search approaches for scheduling jobs on parallel machines with sequence-dependent setup times, precedence constraints and ready times, Comput. Ind. Eng. 61 (2010) 336–345.
[15] A.M. Erkmen, M. Erbudak, O. Anlagan, O. Unver, Genetically tuned fuzzy scheduling for flexible manufacturing systems, in: Proceedings of the 1997 IEEE International Conference on Robotics and Automation, vol. 2, 1997, pp. 951–956.
[16] X. Gang, Z. Wu, Deadlock-free scheduling strategy for automated production cell, IEEE Trans. Syst., Man Cybernet., Part A: Syst. Hum. 34 (1) (2004) 113–122.
[17] D.E. Goldberg, Genetic Algorithms in Search, Optimization, and Machine Learning, Addison-Wesley, 1989.
[18] P. Hansen, N. Mladenovic, J.A.M. Perez, Variable neighbourhood search: methods and applications, Ann. Oper. Res. 175 (1) (2010) 367–407.
[19] N.B. Ho, J.C. Tay, E.M.K. Lai, An effective architecture for learning and evolving flexible job-shop schedules, Eur. J. Oper. Res. 179 (2007) 316–333.
[20] J. Huang, R. Kumar, Optimal nonblocking directed control of discrete event systems, IEEE Trans. Autom. Control 53 (7) (2008) 1592–1603.
[21] H.-J. Kim, J.-H. Lee, T.-E. Lee, A petri net-based modeling and scheduling with a branch and bound algorithm, in: Proceedings of the 2012 IEEE International Conference on Systems, Man, and Cybernetics, 2012, pp. 1779–1784.
[22] M.H. Kim, Y.-D. Kim, Simulation-based real-time scheduling in a flexible manufacturing system, J. Manuf. Syst. 13 (2) (1994) 85–93.
[23] C.-H. Kuo, C.-S. Huang, Dispatching of overhead hoist vehicles in a fab intrabay using a multimission-oriented controller, International Journal of Advanced Manufacturing Technology 27 (2006) 824–832.
[24] N. Labadie, R. Mansini, J. Melechovsky, R.W. Calvo, The team orienteering problem with time windows: An LP-based granular Variable Neighborhood Search, European Journal of Operational Research 220 (1) (2012) 15–27.
[25] M. Lawley, S. Reveliotis, P. Ferreira, Flexible manufacturing system structural control and the neighborhood policy, part 1. Correctness and scalability, IIE Trans. 29 (10) (1997) 877–887.
[26] N. Maldenovic, P. Hansen, Variable neighborhood search, Comput. Oper. Res. 24 (1997) 1097–1100.
[27] T.Y.E. Mekkawy, H.A.E. Maraghy, Real-time scheduling with deadlock avoidance in flexible manufacturing systems, Int. J. Adv. Manuf. Technol. 22 (2003) 259–270.
[28] Y. Pan, L. Shi, New hybrid optimization algorithms for machine scheduling problems, IEEE Trans. Autom. Sci. Eng. 5 (2) (2008) 337–348.
[29] P.N. Pena, A.E.C. Cunha, J.E.R. Cury, S. Lafortune, New results on the nonconflict test of modular supervisors, in: Proceedings of the 9th International Workshop on Discrete Event Systems, 2008, pp. 468–473.
[30] P.N. Pena, J.E.R. Cury, S. Lafortune, Verification of nonconflict of supervisors using abstractions, IEEE Trans. Autom. Control 54 (12) (2009) 2803–2815.
[31] P. Perez-Gonzalez, J.M. Framinan, Setting a common due date in a constrained flowshop: a variable neighbourhood search approach, Comput. Oper. Res. 37 (10) (2010) 1740–1748.
[32] D.C. Pinha, M.H. de Queiroz, J.E.R. Cury, Optimal scheduling of a repair shipyard based on supervisory control theory, in: Proceedings of the 2011 IEEE Conference on Automation Science and Engineering, 2011, pp. 39–44.
[33] L. Rabelo, Y. Yih, A. Jones, J.-S. Tsai, Intelligent scheduling for flexible manufacturing systems, in: Proceedings of the 1993 IEEE International Conference on Robotics and Automation, 1993, pp. 810–815.
[34] P.J.G. Ramadge, W.M. Wonham, The control of discrete event systems, in: Proceedings of IEEE, Special Issue on Discrete Event Dynamic Systems, vol. 77, 1989, pp. 81–98.
[35] K. Saitou, S. Malpathak, H. Qvam, Robust design of flexible manufacturing systems using colored Petri net and genetic algorithm, J. Intell. Manuf. 13 (2002) 339–351.
[36] Y. Song, M.T. Zhang, J. Yi, L. Zhang, L. Zheng, Bottleneck station scheduling in semiconductor assembly and test manufacturing using ant colony optimization, IEEE Trans. Autom. Sci. Eng. 4 (4) (2007) 569–578.
[37] W.M. Spears, K.A. De Jong, T. Back, D.B. Fogel, H. de Garis, An overview of evolutionary computation, in: Machine Learning: ECML-93, volume 667 in LNCS, Springer, 1993, pp. 442–459.
[38] R. Su, Abstraction-based synthesis of timed supervisors for time-weighted systems, in: Proceedings of the 11th International Workshop on Discrete Event Systems, 2012, pp. 128–134.
[39] R. Su, W.M. Wonham, Supervisor reduction for discrete-event systems, Discret. Event Dyn. Syst.: Theory Appl. 14 (2004) 31–53.
[40] M.K. Tiwari, S. Kumar, S. Kumar, Prakash, R. Shankar, Solving part-type selection and operation allocation problems in an fms: an approach using constraints-based fast simulated annealing algorithm, IEEE Trans. Syst. Man Cybernet., Part A: Syst. Hum. 36 (6) (2006) 1170–1184.
[41] A.F. Vaz, W.M. Wonham, On supervisor reduction in discrete-event systems, Int. J. Control 44 (1996) 475–491.
[42] U. Wikborg, T.-E. Lee, Scheduling of petri nets as a multi-objective shortest path problem, in: Proceedings of the 2012 IEEE International Conference on Automation Science and Engineering, 2012, pp. 212–217.
[43] W.M. Wonham, P.J. Ramadge, Modular supervisory control of discrete event systems, Math. Control, Signals Syst. 1 (1988) 13–30.
[44] N. Wu, M. Zhou, Schedulability analysis and optimal scheduling of dual-arm cluster tools with residency time constraint and activity time variation, IEEE Trans. Autom. Sci. Eng. 9 (1) (2012) 203–209.
[45] A.W.L. Yao, Y.M. Pan, A petri nets and genetic algorithm based optimal scheduling for job shop manufacturing systems, in: Proceedings of the 2013 IEEE International Conference on System Science and Engineering, 2013, pp. 99–104.